

Master's Thesis



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

Automatic event recognition for Higgs boson detection

Bc. Jakub Malý

**Supervisor: Prof. Dr. Ing. Jan Kybic
Supervisor–specialist: doc. Dr. André Sopczak
May 2020**



Acknowledgements

I would like to thank both of my supervisors for their continuous support and patience. It was an honour to work under such academics and their joint lead greatly motivated this work.

Thanks also to my university, the Czech Technical University (CTU) in Prague, for giving me a great education which served as a solid basis during the writing of this thesis and providing access to a computational grid.

Lastly, I would like to thank my family and relatives for their great support. This thesis was written during turbulent times, which greatly challenged my mental health and forced the work to happen remotely. Without their support and care, finishing would not have been possible.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 22. May 2020

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 22. Května 2020

Abstract

Several groups of researches have tried, and are still trying, to improve Higgs boson detection [PB04], [Col18a], [Col18b]. Machine Learning (ML) appears as one of the most promising ways, namely Boosted Decision Trees (BDT), Shallow Neural Networks (SNN), and Deep Neural Networks (DNN). The great advantage of such classifiers is that they can be trained once and then reused several times without needing any significant computational power. Also, they can be free of knowing the physical background, or the meaning of the features. The main aim of this project is to test recently developed ML libraries on data provided by CERN.

Keywords: Higgs boson, $t\bar{t}H$, $t\bar{t}W$, $t\bar{t}Z$, Significance, ROOT, UpRoot, Scikit-learn, RFC, KNC, GNB, ADA, GBC, MLPC, SVC

Supervisor: Prof. Dr. Ing. Jan Kybic
Department of Cybernetics

Supervisor–specialist: doc. Dr. Andr  Sopczak
Institute of Experimental and Applied Physics

Abstrakt

N kolik skupin v zkumn k  se zab valo a st le zaj ma vylep en m detekce Higgsova bosonu [PB04], [Col18a], [Col18b]. Strojov  u en  se uk zalo, jako jedna z nejlep  ch metod. Jmenovit  to byly pos len  stromov  struktury, m lk  neuronov  s t , a hlubok  neuronov  s t . Velkou v hodou takov chto model  je, jejich schopnost se nau it pouze jednou a pot  b t nespo etn kr t pou ity bez pot by dal  ho velk ho v po etn ho v konu. D le mohou b t nezávisl  na fyzik ln m pozad  nebo v znamu jednotliv ch vlastnost . Hlavn m c lem tohoto projektu je otestovat ned vno vyvinut  knihovny strojov ho u en  na datech, je  byly poskytnuty organizac  CERN.

Kl  ov  slova: Higgs boson, $t\bar{t}H$, $t\bar{t}W$, $t\bar{t}Z$, Significance, ROOT, UpRoot, Scikit-learn, RFC, KNC, GNB, ADA, GBC, MLPC, SVC

Contents

Machine Learning

Project Specification	iii
------------------------------	------------

1 Introduction	1
-----------------------	----------

Part I Particle Physics

2 The Large Hadron Collider	5
------------------------------------	----------

2.1 Introduction	5
------------------------	---

2.2 Area of interest	5
----------------------------	---

3 Signal in the presence of background	9
---	----------

3.1 Signal structure	9
----------------------------	---

3.2 Backgrounds	10
-----------------------	----

3.2.1 $t\bar{t}W$	10
-------------------------	----

3.2.2 $t\bar{t}Z$	10
-------------------------	----

3.2.3 Others	11
--------------------	----

3.3 Evaluation	12
----------------------	----

3.3.1 Significance computation	12
-------------------------------------	----

Part II

4 Data	17
---------------	-----------

4.1 The ROOT File system	17
--------------------------------	----

4.2 UpRoot Library	18
--------------------------	----

4.3 Pre-processing	19
--------------------------	----

4.3.1 Inputs	19
--------------------	----

4.3.2 Scaling	20
---------------------	----

4.3.3 Compression	20
-------------------------	----

4.4 Processing scripts	21
------------------------------	----

4.5 Used data	21
---------------------	----

4.5.1 Working channel	22
-----------------------------	----

4.5.2 Histograms	22
------------------------	----

5 Classification	23
-------------------------	-----------

5.1 Introduction	23
------------------------	----

5.2 General types of probabilities ..	23
---------------------------------------	----

5.2.1 Empirical	23
-----------------------	----

5.2.2 Subjective	24
------------------------	----

5.2.3 Apriori	24	6.5.1 Introduction.....	37
5.3 Bayesian decision task	25	6.5.2 Parameters.....	37
5.3.1 Bayes' theorem	25	6.5.3 Feature importances.....	40
5.3.2 Bayesian Risk	26	6.5.4 Additional tests.....	42
5.3.3 Decision strategy	27	6.5.5 Working Point	43
6 The Scikit-learn library	29	6.5.6 Conclusion	46
6.1 Splits	29	6.6 The K-Neighbors Classifier.....	48
6.1.1 Test Split Validation	30	6.6.1 Introduction.....	48
6.1.2 K-Fold Cross-Validation	30	6.6.2 Dimension reduction	48
6.1.3 Stratified K-Fold Cross-Validation.....	31	6.6.3 Parameters.....	51
6.2 Pipeline	31	6.6.4 Additional tests.....	52
6.3 Classifiers.....	32	6.6.5 Working Point	53
6.4 Scoring	33	6.6.6 Conclusion	55
6.4.1 F1-score	33	6.7 Gaussian Naïve Bayes	57
6.4.2 Mean accuracy score	35	6.7.1 Introduction.....	57
6.4.3 ROC AUC score	36	6.7.2 Parameters.....	57
6.5 The Random Forest Classifier ..	37	6.7.3 Additional tests.....	59
		6.7.4 Working Point	60

6.7.5 Conclusion	61	6.10.5 Conclusion	85
6.8 AdaBoost Classifier	64	6.11 Support Vector Classifier	87
6.8.1 Introduction	64	6.11.1 Introduction	87
6.8.2 Parameters	64	6.11.2 Parameters	88
6.8.3 Additional tests	66	6.11.3 Additional tests	90
6.8.4 Working Point	67	6.11.4 Working Point	91
6.8.5 Conclusion	68	6.11.5 Conclusion	92
6.9 Gradient Boosting Classifier	72	7 Real data	95
6.9.1 Introduction	72	8 Additional Changes	99
6.9.2 Parameters	72	8.1 Cheating	99
6.9.3 Additional tests	75	8.2 New data training	99
6.9.4 Working Point	76	8.2.1 RFC	101
6.9.5 Conclusion	77	8.2.2 KNC	102
6.10 Multi-Layer Perceptron Classifier	79	8.2.3 GNB	103
6.10.1 Introduction	79	8.2.4 ADA	104
6.10.2 Parameters	80	8.2.5 GBC	105
6.10.3 Additional tests	84	8.2.6 MLPC	106
6.10.4 Working Point	84		

8.2.7 SVC.....	107	C.2.2 The K-Neighbors Classifier	160
8.2.8 Summary	107	C.2.3 Gaussian Naïve Bayes	166
8.2.9 Real data	108	C.2.4 AdaBoost Classifier	170
8.3 Feature importances	109	C.2.5 Gradient Boosting Classifier	174
8.4 $t\bar{t}Z$ weighting	110	C.2.6 Multi-Layer Perceptron Classifier	184
9 Conclusion	111	C.2.7 Support Vector Classifier ..	196
Appendices			
A Computations	115	D Code	203
A.1 Significance computation example	115	D.1 Utils.....	203
B Tables	119	D.1.1 converter.py	203
B.1 Data.....	119	D.1.2 reader.py	207
C Figures	125	D.1.3 ml_utils.py	211
C.1 Data Histograms	125	D.2 Scripts	230
C.1.1 Older mc16a set	125	D.2.1 data_convert.py.....	230
C.1.2 Real mc16a & mc16d set ..	143	D.2.2 data_filter.py	231
C.2 Tuning.....	152	D.2.3 data_prepare.py	234
C.2.1 The Random Forest Classifier	152	D.2.4 data_split.py	236
		D.2.5 data_view.py	241

D.2.6 tuning scripts (only RFC listed)	244
D.2.7 tuning results view scripts (only RFC listed)	247
D.2.8 train scripts (only RFC listed)	251
D.2.9 tuning_wp.py	255
D.2.10 eval_real.py	257
E Bibliography	261

Figures

2.1 Aerial view of the LHC with the locations of the four biggest detectors cita	6	6.5 Performance of RFC with tuned parameters	40
2.2 ATLAS detector with crucial parts marked citb	7	6.6 Performance of RFC (channel) with default parameters	40
2.3 ATLAS inner detector with its components citc	8	6.7 Performance of RFC (channel) with tuned parameters	41
3.1 Higgs boson production with a pair of top quarks cited	10	6.8 Performance of RFC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	41
3.2 $t\bar{t}H$ (a) and $t\bar{t}W$ (b) Freyman diagrams PB14	11	6.9 Feature importances for full-data model (left) and channel-data model (right)	43
3.3 Example $t\bar{t}Z$ Freyman diagram with trilepton signature Byl16 . .	11	6.10 Performance of RFC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data . .	43
4.1 Converter and Reader flowchart diagram	19	6.11 Dependence of significance on the threshold, vertical lines denote maximums	44
6.1 K-Fold Cross Validation procedure QR19	30	6.12 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold	45
6.2 Pipeline flowchart diagram	31	6.13 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	46
6.3 Elemental decision tree classification Yiu19	38		
6.4 Performance of RFC with default parameters	39		

6.14 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	47	6.24 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	55
6.15 KNC steps visualization for binary classification in two-dimensional space [Nav18b] ..	49	6.25 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	56
6.16 Performance of KNC with default parameters	50	6.26 Performance of GNB with default parameters	58
6.17 Performance of KNC with tuned parameters	50	6.27 Performance of GNB with tuned parameters	58
6.18 Performance of KNC (channel) with default parameters	52	6.28 Performance of GNB (channel) with default parameters	59
6.19 Performance of KNC (channel) with tuned parameters	52	6.29 Performance of GNB (channel) with tuned parameters	59
6.20 Performance of KNC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	53	6.30 Performance of GNB with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	60
6.21 Performance of KNC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data ...	53	6.31 Performance of GNB with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data ...	60
6.22 Dependence of significance on the threshold, vertical lines denote maximums	54	6.32 Dependence of significance on the threshold, vertical lines denote maximums	61
6.23 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold	54		

6.33 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold.....	61	6.42 Performance of ADA with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data ...	69
6.34 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data.....	62	6.43 Dependence of significance on the threshold, vertical lines denote maximums	69
6.35 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	63	6.44 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold.....	70
6.36 AdaBoost steps visualization [Nav18a]	65	6.45 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	70
6.37 Performance of ADA with default parameters.....	66	6.46 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	71
6.38 Performance of ADA with tuned parameters.....	67	6.47 Performance of GBC with default parameters.....	73
6.39 Performance of ADA (channel) with default parameters	67	6.48 Performance of GBC with tuned parameters.....	73
6.40 Performance of ADA (channel) with tuned parameters	68	6.49 Performance of GBC (channel) with default parameters	74
6.41 Performance of ADA with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	68	6.50 Performance of GBC (channel) with tuned parameters	74

6.51 Performance of GBC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	75	6.60 Performance of MLPC with tuned parameters	82
6.52 Performance of GBC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data . . .	75	6.61 Performance of MLPC (channel) with default parameters	83
6.53 Dependence of significance on the threshold, vertical lines denote maximums	76	6.62 Performance of MLPC (channel) with tuned parameters	83
6.54 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold	76	6.63 Performance of MLPC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data	83
6.55 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	77	6.64 Performance of MLPC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data . . .	84
6.56 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	78	6.65 Dependence of significance on the threshold, vertical lines denote maximums	85
6.57 Artificial neuron visualization [Roo19]	79	6.66 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold	85
6.58 Artificial neuron network visualization [Sap19]	80	6.67 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	86
6.59 Performance of MLPC with default parameters	82	6.68 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	86

6.69 Binary classification problem solved by Support Vector Machine [Was19]	87
6.70 Performance of SVC with default parameters	88
6.71 Performance of SVC with tuned parameters	89
6.72 Performance of SVC (channel) with default parameters	90
6.73 Performance of SVC (channel) with tuned parameters	90
6.74 Performance of SVC with tuned parameters, trained on 10% of full-data and tested on 80% and 20% of channel-data	91
6.75 Performance of SVC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data ...	91
6.76 Dependence of significance on the threshold, vertical lines denote maximums	92
6.77 Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold	92
6.78 Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data	93

6.79 a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data	94
7.1 Overview of individual real-data predictions	96
7.2 Individual real-data predictions .	96
7.2 Individual real-data predictions .	97
8.1 Performance of newly trained RFC	101
8.2 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	101
8.3 Performance of newly trained KNC	102
8.4 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	102
8.5 Performance of newly trained GNB	103

8.6 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	103	8.15 Overview of individual real-data predictions of newly trained classifiers	108
8.7 Performance of newly trained ADA	104	8.16 Individual real-data predictions of newly trained classifiers.....	109
8.8 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	104	8.16 Individual real-data predictions of newly trained classifiers.....	110
8.9 Performance of newly trained GBC	105	8.17 Feature importances of newly trained RFC	110
8.10 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	105	A.1 Example of 3-class classification results, $t\bar{t}H$ class is expressed as 0, $t\bar{t}W$ as 1 and $t\bar{t}Z$ as 2	116
8.11 Performance of newly trained MLPC	106	C.1 Full-data histograms (older mc16a set)	125
8.12 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	106	C.1 Full-data histograms (older mc16a set)	126
8.13 Performance of newly trained SVC	107	C.1 Full-data histograms (older mc16a set)	127
8.14 Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold	107	C.1 Full-data histograms (older mc16a set)	128
		C.1 Full-data histograms (older mc16a set)	129
		C.1 Full-data histograms (older mc16a set)	130
		C.1 Full-data histograms (older mc16a set)	131

C.1 Full-data histograms (older mc16a set)	132	C.3 Real-data histograms (mc16a & mc16d set)	145
C.1 Full-data histograms (older mc16a set)	133	C.3 Real-data histograms (mc16a & mc16d set)	146
C.2 Channel-data histograms (older mc16a set)	134	C.3 Real-data histograms (mc16a & mc16d set)	147
C.2 Channel-data histograms (older mc16a set)	135	C.3 Real-data histograms (mc16a & mc16d set)	148
C.2 Channel-data histograms (older mc16a set)	136	C.3 Real-data histograms (mc16a & mc16d set)	149
C.2 Channel-data histograms (older mc16a set)	137	C.3 Real-data histograms (mc16a & mc16d set)	150
C.2 Channel-data histograms (older mc16a set)	138	C.3 Real-data histograms (mc16a & mc16d set)	151
C.2 Channel-data histograms (older mc16a set)	139	C.4 n_estimators parameter tuning	152
C.2 Channel-data histograms (older mc16a set)	140	C.5 max_depth parameter tuning .	152
C.2 Channel-data histograms (older mc16a set)	141	C.6 min_samples_split parameter tuning	153
C.2 Channel-data histograms (older mc16a set)	142	C.7 max_features parameter tuning	153
C.3 Real-data histograms (mc16a & mc16d set)	143	C.8 max_leaf_nodes parameter tuning	153
C.3 Real-data histograms (mc16a & mc16d set)	144	C.9 min_impurity_decrease parameter tuning	154
		C.10 ccp_alpha parameter tuning .	154

C.11 max_samples parameter tuning	154	C.25 n_components TSVD parameter tuning	160
C.12 oob_score parameter tuning (0: False, 1: True)	155	C.26 n_neighbors parameter tuning	161
C.13 criterion parameter tuning (0: 'gini', 1: 'entropy')	155	C.27 leaf_size parameter tuning ..	161
C.14 n_estimators parameter tuning	156	C.28 p parameter tuning	161
C.15 max_depth parameter tuning	156	C.29 weights parameter tuning (0: 'uniform', 1: 'distance')	162
C.16 min_samples_split parameter tuning	157	C.30 algorithm parameter tuning (0: 'auto', 1: 'ball_tree', 2: 'kd_tree', 3: 'brute')	162
C.17 max_features parameter tuning	157	C.31 n_components PCA parameter tuning	163
C.18 max_leaf_nodes parameter tuning	157	C.32 n_components TSVD parameter tuning	163
C.19 min_impurity_decrease parameter tuning	158	C.33 n_neighbors parameter tuning	164
C.20 ccp_alpha parameter tuning .	158	C.34 leaf_size parameter tuning ..	164
C.21 max_samples parameter tuning	158	C.35 p parameter tuning	164
C.22 oob_score parameter tuning (0: False, 1: True)	159	C.36 weights parameter tuning (0: 'uniform', 1: 'distance')	165
C.23 criterion parameter tuning (0: 'gini', 1: 'entropy')	159	C.37 algorithm parameter tuning (0: 'auto', 1: 'ball_tree', 2: 'kd_tree', 3: 'brute')	165
C.24 n_components PCA parameter tuning	160	C.38 n_components PCA parameter tuning	166

C.39 n_components TSVD parameter tuning.....	166	C.53 n_estimators parameter tuning	173
C.40 priors parameter tuning	167	C.54 learning_rate parameter tuning	173
C.41 var_smoothing parameter tuning.....	167	C.55 base_estimator parameter tuning (depth parameter of DT)	173
C.42 n_components PCA parameter tuning.....	168	C.56 n_components PCA parameter tuning.....	174
C.43 n_components TSVD parameter tuning.....	168	C.57 n_components TSVD parameter tuning.....	174
C.44 priors parameter tuning	169	C.58 learning_rate parameter tuning	175
C.45 var_smoothing parameter tuning.....	169	C.59 subsample parameter tuning .	175
C.46 n_components PCA parameter tuning.....	170	C.60 criterion parameter tuning (0: 'friedman_mse', 1: 'mse')	175
C.47 n_components TSVD parameter tuning.....	170	C.61 n_estimators parameter tuning	176
C.48 n_estimators parameter tuning	171	C.62 min_samples_split parameter tuning.....	176
C.49 learning_rate parameter tuning	171	C.63 max_depth parameter tuning	176
C.50 base_estimator parameter tuning (depth parameter of DT)	171	C.64 min_impurity_decrease parameter tuning	177
C.51 n_components PCA parameter tuning.....	172	C.65 max_features parameter tuning.....	177
C.52 n_components TSVD parameter tuning.....	172	C.66 max_leaf_nodes parameter tuning.....	177
		C.67 ccp_alpha parameter tuning .	178

C.68 n_components PCA parameter tuning	179
C.69 n_components TSVD parameter tuning	179
C.70 learning_rate parameter tuning	180
C.71 subsample parameter tuning .	180
C.72 criterion parameter tuning (0: 'friedman_mse', 1: 'mse')	180
C.73 n_estimators parameter tuning	181
C.74 min_samples_split parameter tuning	181
C.75 max_depth parameter tuning	181
C.76 min_impurity_decrease parameter tuning	182
C.77 max_features parameter tuning	182
C.78 max_leaf_nodes parameter tuning	182
C.79 ccp_alpha parameter tuning .	183
C.80 n_components PCA parameter tuning	184
C.81 n_components TSVD parameter tuning	184

C.82 hidden_layer_sizes parameter tuning (Table 6.13 contains mapping of tuned values to numbers)	185
C.83 activation parameter tuning (0: 'identity', 1: 'logistic', 2: 'tanh', 3: 'relu')	185
C.84 solver parameter tuning (0: 'lbfgs', 1: 'sgd', 2: 'adam')	185
C.85 alpha parameter tuning	186
C.86 batch_size parameter tuning	186
C.87 learning_rate parameter tuning (0: 'constant', 1: 'invscaling', 2: 'adaptive')	186
C.88 learning-rate-init parameter tuning	187
C.89 max_iter parameter tuning ..	187
C.90 tol parameter tuning	187
C.91 beta_1 parameter tuning	188
C.92 beta_2 parameter tuning	188
C.93 epsilon parameter tuning	188
C.94 n_iter_no_change parameter tuning (early_stopping = 'True')	189
C.95 n_components PCA parameter tuning	190

C.96 n_components TSVD parameter tuning.....	190	C.110 kernel parameter tuning (0: 'linear', 1: 'poly', 2: 'rbf', 3: 'sigmoid')	196
C.97 hidden_layer_sizes parameter tuning (Table 6.13 contains mapping of tuned values to numbers)	191	C.111 C parameter tuning	196
C.98 activation parameter tuning (0: 'identity', 1: 'logistic', 2: 'tanh', 3: 'relu')	191	C.112 gamma parameter tuning...	197
C.99 solver parameter tuning (0: 'lbfgs', 1: 'sgd', 2: 'adam')	191	C.113 tol parameter tuning.....	197
C.100 alpha parameter tuning	192	C.114 shrinking parameter tuning (0: False, 1: True)	197
C.101 batch_size parameter tuning	192	C.115 break_ties parameter tuning (0: False, 1: True)	198
C.102 learning_rate parameter tuning (0: 'constant', 1: 'invscaling', 2: 'adaptive')	192	C.116 kernel parameter tuning (0: 'linear', 1: 'poly', 2: 'rbf', 3: 'sigmoid')	199
C.103 learning-rate-init parameter tuning.....	193	C.117 C parameter tuning	199
C.104 max_iter parameter tuning.	193	C.118 gamma parameter tuning...	200
C.105 tol parameter tuning.....	193	C.119 tol parameter tuning.....	200
C.106 beta_1 parameter tuning...	194	C.120 shrinking parameter tuning (0: False, 1: True)	200
C.107 beta_2 parameter tuning...	194	C.121 break_ties parameter tuning (0: False, 1: True)	201
C.108 epsilon parameter tuning ...	194		
C.109 n_iter_no_change parameter tuning (early_stopping = 'True')	195		

Tables

3.1 Relation between significance and probability [Kor08]	12	6.8 List of tuned parameters of ADA with results	66
4.1 Structure of input matrix $\mathbf{X}$	19	6.9 List of tuned dimension reduction algorithms for GBC with results . .	72
4.2 Structure of truth vector $\vec{y}$	20	6.10 List of tuned parameters of GBC with results	72
4.3 Comparison of performance for different compression methods	20	6.11 List of tuned dimension reduction algorithms for MLPC with results .	81
4.4 List of possible channels containing τ and light leptons (x - non-existing, o - existing, $\checkmark$ - ours)	22	6.12 List of tuned parameters of MLPC with results	81
6.1 List of tuned parameters of RFC with results	39	6.13 List of tuned values for 'hidden_layer_sizes' parameter . . .	82
6.2 List of the most important features with descriptions	42	6.14 List of tuned parameters of SVC with results	88
6.3 List of tuned dimension reduction algorithms for KNC with results . .	51	8.1 List of classifiers and their significance scores	108
6.4 List of tuned parameters of KNC with results	51	8.2 List of the most important features with descriptions for newly trained RFC	109
6.5 List of tuned dimension reduction algorithms for GNB with results . .	57	B.1 List of used features	120
6.6 List of tuned parameters of KNC with results	57	B.2 List of features after removing additional 'simulation-only related' features	121
6.7 List of tuned dimension reduction algorithms for ADA with results . .	64	B.3 List of root files used for training (80%) and testing (20%) splits - older data set (mc16a)	122

B.4 List of root files used for additional
testing - older data set (mc16d) . 122

B.5 List of used root files for
comparison (100%), re-training
(80%), and validation (20%) - newer
data set containing older $t\bar{t}Z$ files
(both mc16a & mc16d) 123

B.6 List of used real data root files for
testing 123



Chapter 1

Introduction

The Higgs boson, notoriously known as the God's particle by the general public, is a key particle in physics. Without it, no atoms would be created and all particles would remain at the speed of light. In this thesis, enhancements with the aim of boosting the efficiency of Higgs boson classification are proposed. They are based on many experiments, carried out during a ten-month cooperation between two CTU subsections: the Department of Cybernetics and the Institute of Experimental and Applied Physics.

This paper applies well known machine learning techniques to achieve a higher significance score. The results are discussed in detail and are sometimes surprising. It was discovered that having a higher efficiency for background than for signal can actually yield a better significance score than in the opposite case. Of course, the performance mainly depends on the quality of the classification. To boost this, many classifier parameters were tuned. Later, a custom working point selection function was introduced for significance maximization.

The thesis is divided into two parts: in the first, particle physics is shortly discussed providing an elementary understanding to the problem; and the second discusses machine learning and all related topics. This is followed by the conclusion with a summary and a discussion of the pitfalls.



Part I

Particle Physics



Chapter 2

The Large Hadron Collider



2.1 Introduction

The Large Hadron Collider (LHC) is the biggest collider in the world, built and operated by the European Organization for Nuclear Research (CERN). The France-Switzerland border near Geneva was chosen as the construction site. The construction itself took 10 years (1998-2008). Currently, more than 10,000 scientists are users of the CERN infrastructure. By the year 2010 the first collision was made at an energy of 3.5 TeV (teraelectronvolts) per beam. This number exceeded the previous world record by about four times [CER12]. The current record held by the LHC is 6.5 TeV per beam [Web15]. However, in the year 2018 a two-year shut-down was introduced to allow additional adjustments for slightly higher energy output and a much more intense beam. So in near future, the record will be probably exceeded once more, 175 meters under the ground near Geneva.



2.2 Area of interest

There are many reasons why the LHC was constructed, including the testing of theories and the discovery of physics beyond the Standard Model of particle physics. Like any research fields, particle physics needs to validate its theoretical conclusions. For this thesis, and for the team led by my supervisor-

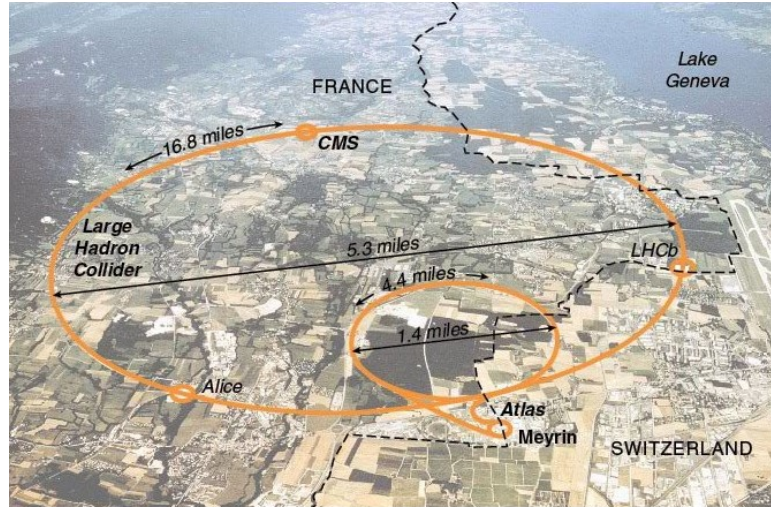


Figure 2.1: Aerial view of the LHC with the locations of the four biggest detectors [cita]

specialist, 2012 became an important date due to the confirmation of the Higgs boson [BBC12], 48 years after Peter Higgs and his team proposed its existence (1964) [TUoEA12]. This discovery yielded two Nobel prizes, one for Peter Higgs and the other for his colleague François Englert [Amo13].

Existence confirmation is a complicated process in which the key elements are detectors. The ATLAS and CMS are general-purpose detectors, constructed to look for clues of new physics, and confirmed the existence of a Higgs boson [ERN12] [Tay12].

It is not within the scope of this thesis to explain in detail how such a detector works, however a short description may be of use. A basic diagram can be seen in Figure 2.2. At 7,000 tonnes, this cylindrical machine with a length of 44 metres and a diameter of 25 metres is the biggest detector ever constructed [CER08]. Inside, many layers precisely measure the trajectory, momentum, and energy of particles, which can then be identified. Over a billion particle interactions per second are detected by ATLAS. But only one in a million is interesting, and the study of the Higgs boson is no exception. In order to process such a huge amount of data, a special system was developed, called the trigger and data acquisition system. This multi-level computing system analyses data at 130 computing centres worldwide [Sca10].

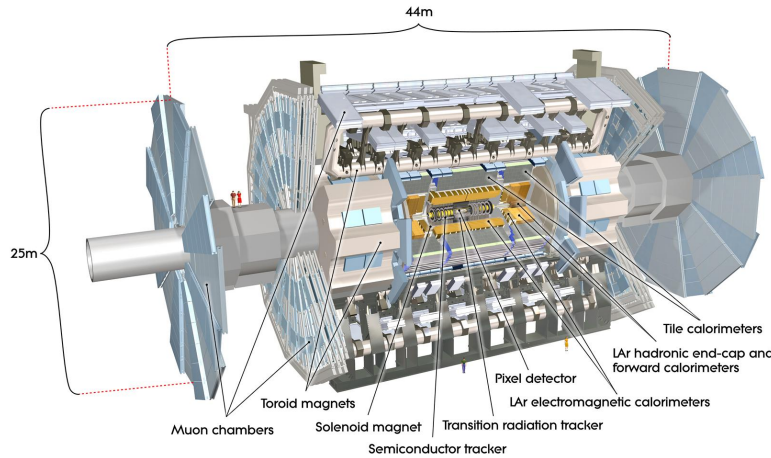


Figure 2.2: ATLAS detector with crucial parts marked citb

There are several sub-detectors inside the ATLAS detector:

- an inner detector
- calorimeters
- muon spectrometers
- a magnet system

The first sub-detector, a sensor system called an inner detector, Figure 2.3, is located in the very heart of the detector. In total three sub-components (a pixel detector, a semi-conductor tracker, and a transition radiation tracker) help to detect direction, momentum, and electrical charge for the decay products of the proton-proton collision CERb.

The second component, a group of calorimeters, measures the energy lost by particles during passing. In general there are two layers in a calorimeter. The "passive" material, which has a high density, and the "active" material for evaluation. The most common mediums used for these layers are lead (passive) and lead-gas or liquid argon (active) respectively. In ATLAS, the Liquid Argon Calorimeter (LAr) and the Tile Hadronic Calorimeter (TileCal) are used CERa. LAr measures the energy of electrons and photons. TileCal measures the energy of hadrons (protons and neutrons). The design is usually chosen to deposit all energy of the passing particles into a detector. However, there are two exceptions which cannot be stopped: muons and neutrinos.

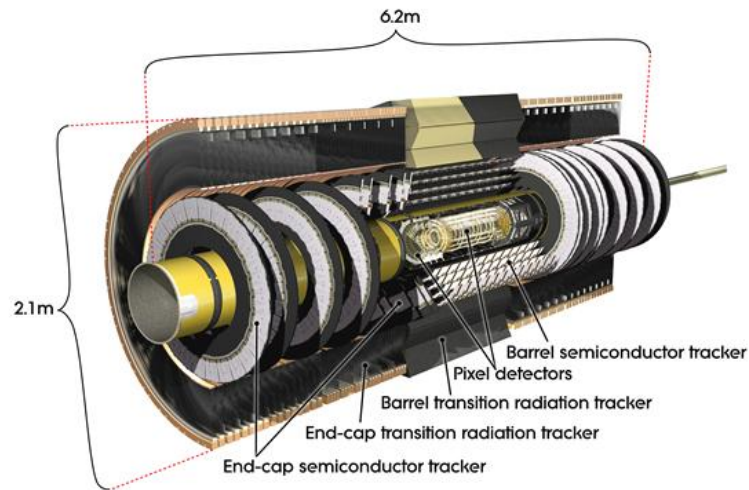


Figure 2.3: ATLAS inner detector with its components [citc](#)

Muon spectrometers, which measure passing muons, are therefore necessary. They are a special system made of many muon chambers. In ATLAS, four different designs are used. For more information please see [CERd](#).

The last important component mentioned here is the magnet system. Shortly explained, this is used to bend particles and makes containing their tracks easier. A more detailed description can be found in [CERc](#).

Chapter 3

Signal in the presence of background

3.1 Signal structure

In the Standard Model (SM), the Higgs particle is a boson with zero spin, no electric charge, and no colour charge. By SM predictions there is a number of possible ways for the Higgs particle to be produced. However, the probability of producing the Higgs boson in a collision is expected to be very small [Str12]. In this thesis we are observing only one method of production, called $t\bar{t}H$. As can be seen in Figure 3.1, two colliding gluons decay into a top quark-antiquark pair. This pair will form one of a number of particles, one possibility being the Higgs boson. These are discussed in Section 3.2.

Data for both signal and background that was available to us originates from Monte-Carlo random sampling simulations. For the years between 2015 and 2016, the dataset is called mc16a. For recorded 2017 data, the simulated dataset is called mc16d. Each dataset contributes differently to the total number of events. These samples are distinguished by the decay products of H as all-hadronic (both tops decay hadronically), semi-leptonic (one top hadronic, the other leptonic), and di-leptonic (both tops decay leptonically).

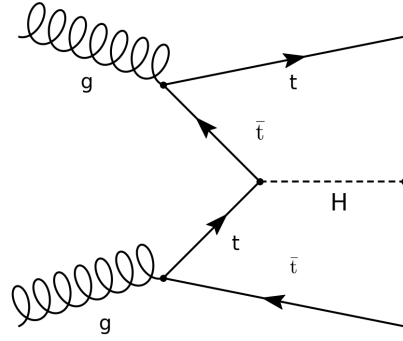


Figure 3.1: Higgs boson production with a pair of top quarks citd

3.2 Backgrounds

Several backgrounds can be mistaken for the production and decay of Higgs bosons as the input is the same: two gluons interacting with each other. And similarly for the output, the same particles are observed. To tell whether Higgs boson production has occurred, a knowledge of backgrounds is important. For example in the case of invisible particle production, like a neutrino, momentum and energy loss are both present. This helps us to distinguish between signal and background PB14.

3.2.1 $t\bar{t}W$

When two W bosons and two b-quarks are detected as decay products, $t\bar{t}W$ background can be involved. Two Feynman diagrams, one for $t\bar{t}H$ and one for $t\bar{t}W$ which both result in the same decay products, are shown in Figure 3.2.

3.2.2 $t\bar{t}Z$

$t\bar{t}Z$ background was available in five different decay modes. For us $Z \rightarrow ee$ and $Z \rightarrow \mu\mu$ are the most critical, because they can lead to the same final state as $t\bar{t}H$. $Z \rightarrow \tau\tau$ is also very important due to further work with a

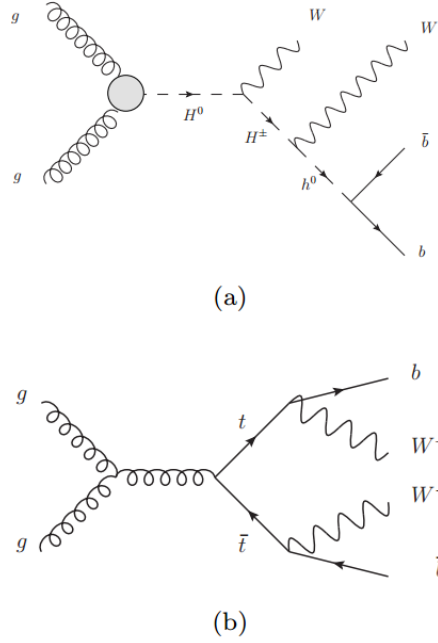


Figure 3.2: $t\bar{t}H$ (a) and $t\bar{t}W$ (b) Freyman diagrams [PB14]

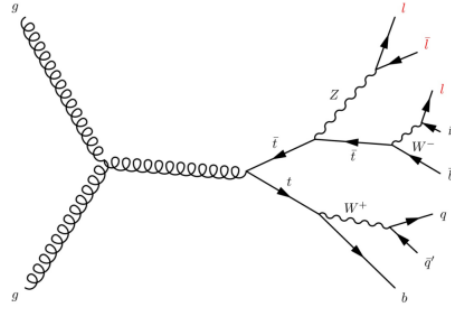


Figure 3.3: Example $t\bar{t}Z$ Freyman diagram with trilepton signature [Byl16]

channel called '2LSS1Tau'. It requires two same-sign light leptons (e or μ) and one hadronically decaying τ . The remaining two decay modes are $Z \rightarrow \nu\nu$, and $Z \rightarrow qq$. The example of $t\bar{t}Z$ background giving a trilepton signature is shown in Figure 3.3.

3.2.3 Others

There are many other backgrounds which contribute to the uncertainty of detecting the Higgs boson. Over the last few years in the $t\bar{t}H$ search, other backgrounds have been reduced, and thus the $t\bar{t}W/Z$ became relatively more

significance	1	2	3	4	5
probability (p-value)	16%	2.3%	0.14%	$3 \cdot 10^{-5}$	$3 \cdot 10^{-7}$

Table 3.1: Relation between significance and probability [Kor08]

important [Col19a]. However, for a proper comparison with existing results in [Col19b], four other types of backgrounds need to be taken into account:

- $t\bar{t}bar$
- $t\bar{t}\gamma$
- VV
- *rare*

■ 3.3 Evaluation

Many techniques can be used to determine how well our classifiers are performing. The basic techniques (such as F1 score, mean accuracy, and ROC/AUC), are good indicators for machine learning. However, for particle physics, another score is more important: the maximization of the special case of statistical significance which maps the probability of a statistical fluctuation into a number of Gaussian sigmas.

■ 3.3.1 Significance computation

Many CERN papers use general approximation formulas (poor man solutions), such as:

$$significance = \frac{S}{\sqrt{S+B}}, \quad (3.1)$$

or in its simplified form:

$$significance = \frac{S}{\sqrt{B}}. \quad (3.2)$$

We use these for a direct comparison when comparing the significances. It is good to note that these formulas only hold for $(B+S) > 100$ or $B > 100$ in the case of the simplified form [Kor08]. As we encounter much smaller values, the approximation is very rough.

In the equations, S and B are the expected numbers of selected signal and background events in the real data. They can be expressed as:

$$S = L_{t\bar{t}H} \cdot \sigma_{t\bar{t}H} \cdot \epsilon_{t\bar{t}H} = N_{t\bar{t}H} \cdot \omega_{t\bar{t}H} \cdot \epsilon_{t\bar{t}H}, \quad (3.3)$$

$$B = \sum_{i \in (t\bar{t}W, t\bar{t}Z)} L_i \cdot \sigma_i \cdot \epsilon_i + B_{other} = \sum_{i \in (t\bar{t}W, t\bar{t}Z)} N_i \cdot \omega_i \cdot \epsilon_i + B_{other}, \quad (3.4)$$

where L - luminosity [pb^{-1}] and σ - cross-section [pb] are file-related constants. Together they can be expressed in the form of a weighting factor ω as:

$$\omega_i = \frac{1}{\frac{N_i}{E_i}} = \frac{E_i}{N_i} = \frac{L_i \cdot \sigma_i}{N_i}, \forall i \in (t\bar{t}H, t\bar{t}W, t\bar{t}Z), \quad (3.5)$$

where N is the number of events in simulated data and E is the expected number of events in real data.

B_{other} stands for the estimated B of all other backgrounds and the variable ϵ - efficiency is the evaluation of classification performance.

It can be expressed as:

$$\epsilon_i = \frac{N s_i}{N t_i}, \forall i \in (t\bar{t}H, t\bar{t}W, t\bar{t}Z), \quad (3.6)$$

where N_s denotes the number of class events which were selected as $t\bar{t}H$ (signal) events and N_t the number of total events in a given class.

This can be expressed in machine learning terms as:

$$\epsilon_{t\bar{t}H} = \frac{TP}{TP + \sum_i FN_i} \forall i \in (t\bar{t}W, t\bar{t}Z), \quad (3.7)$$

for the $t\bar{t}H$ class. And as:

$$\epsilon_i = \frac{FP_i}{FP_i + \sum_j TN_{i,j}}, \forall i, j \in (t\bar{t}W, t\bar{t}Z), \quad (3.8)$$

for the background classes. TP is the number of signals classified as a signal, FN is the number of signals classified as background i, FP is the number of background i's classified as a signal, and TN is the number of background i's classified as background j.

A step-by-step example of the computation is shown in Appendix [A.1](#).

Part II

Machine Learning

Chapter 4

Data

4.1 The ROOT File system

ROOT is a scientific tool kit developed by CERN and mainly written as an object-oriented program in C++. There are also less frequently used implementations for Python and R. The purpose of this program is mainly to provide functionalities such as big data processing, visualization, or analysis. On one hand it is accepted by a wide range of scientists, but on the other it is criticized for its complexity, bugs, and not being user-friendly [Buc07].

The basic challenge was to understand the format of the data. Generated in simulators, the datasets contain many branches which consist of tuples. One can imagine this type of storage as a Python dictionary. The tree itself is the dictionary; the branches are the dictionary's keys; and the events are key values. The ROOT file system, however, is far more complicated: for example, it also allows the storing of data in time and uses trees to travel in the data time line, as well as the creation of many more elements than just branches. After a discussion, all data types apart from 'TBranch' were discarded due to the assumption that they are of only minor importance for our work.

4.2 UpRoot Library

The UpRoot library is a lighter version of ROOT written in Python, which works directly with the NumPy library [\[Oli\]](#). It is independent of the original C++ ROOT and was created primarily to stream data into Python's machine learning libraries. Like many other C++ programs, ROOT is very efficient and fast. However, it has been proven that UpRoot handles big data more efficiently [\[Piv\]](#).

A new Python class called Converter (Appendix [D.1.1](#), [converter.py](#)), located inside the 'utils' library folder, was created. This class uses the UpRoot Application Programmable Interface (API) to allow users to load .root files, query them by the original ROOT syntax, convert the results into NumPy arrays, and save them. It was decided to save the arrays in Python's pickle library. Saved files contain a dictionary with branch names as keys and branch data as the keys' data. As discussed, this structure appears closest to the original.

For loading and reading these files, another class called Reader (Appendix [D.1.2](#), [reader.py](#)) was built. Its purpose is to collect information such as the criterion used to obtain data, or the names of branches (features in machine learning terms) and data itself. Everything can be directly accessed via getter methods. There are also two filter methods to satisfy the preprocessing needs: the first returns data with only desired features and channels, and the second returns data with desired features with a specified criterion applied (for example channel selection).

One could propose to joining these two classes and working directly with the data in the converter without saving. The main reason for having two separate classes is the time it takes to convert a .root file into a .pkl file. It is very convenient to convert once and save the result. Later, multiple tasks can be done on the data, which is already converted and ready to be loaded within a few seconds. A flowchart of the complete procedure can be seen in Figure [4.1](#).

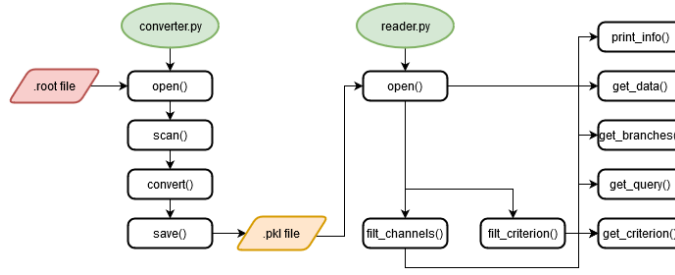


Figure 4.1: Converter and Reader flowchart diagram

	feature1	feature2	...	featureN
event1	0	0.1		0
event2	0	0.2		1
...				
eventN	0	0		1

Table 4.1: Structure of input matrix $\mathbf{X}$

4.3 Pre-processing

Before data can be used in any machine-learning library, several steps are required. First, "cheating" needs to be eliminated. In our case this term signifies features which are simulation-related and do not occur in real data. A filter was created, constructed under the advisement of my particle physics advisor and included only well known features with a relatively good separation power. Also, all simulation-related features were discarded. This resulted in a final number of 83. These features are listed in Appendix B.1. They are discussed further in Chapter 5, where those with the greatest importance to the classification are described in detail.

4.3.1 Inputs

Another step was to prepare the data in the correct format. Due to already being stored in the NumPy arrays format, this library was used and provided a fast solution. A new file called `ml_utils.py` was created inside the 'utils' folder, with the purpose of containing all machine-learning related support methods in one file. Using the originally created methods `get_X_y_f()` and `get_X_y_f_multiclass()`, data for multiple files was concatenated by features to form the input matrix $\mathbf{X}$. Vector $\vec{y}$ recorded to which file (or more precisely, class) a given set of events belonged. The format of these two elements is shown in Tables 4.1 and 4.2.

	0/1/2 (ttH/ttW/ttZ)
event1	0
event2	2
...	
eventN	1

Table 4.2: Structure of truth vector $\vec{y}$

method	size of $\mathbf{X}$ [KB]	decompress time [s]
normal	1,945,184	2.131
gzip	364,387	8.635
bz2	326,777	48.859
lzma	264,997	43.110

Table 4.3: Comparison of performance for different compression methods

4.3.2 Scaling

Because of unspecified data distribution, it is advisable to create a pre-processor to re-distribute the data. Generally, all classifiers assume data in the form of comparable numerical magnitudes. Failure to do so may lead to major problems. There are several common ways of scaling data. For example, scaling data to look like standard normally-distributed data: Gaussian with zero mean and unit variance. Or scaling features to a range, usually between 0 and 1. This process is discussed further in Subsection 6.2, where the pipeline process is described in detail.

4.3.3 Compression

A large amount of data from input matrix $\mathbf{X}$ needed to be stored. In order to reduce the storage space needed, the CompressPickle [Paz] library was used, which joins the default pickle library with Python's compression packages. This results in a very efficient and user-friendly library with almost identical calls. In Table 4.3, the performance of the compression methods is listed. All tests were performed on a system with an HDD with 7200 RPM and Intel® Core™ i7 4720HQ 2.6 GHz - 3.6 GHz Processor. Taking into account both the size and the decompression time, a 'gzip' algorithm was selected for further work with the data. Four new methods in total were added inside `ml_utils.py`: two for saving and loading files with compression and two without.

4.4 Processing scripts

Everything discussed above (except scaling) is demonstrated in five Python scripts. The running order of these scripts is very important, as script n is dependent on script $n - 1$, $\forall n \in (1, 5)$. First, starting with `data_convert.py`, data is converted from the root format into the python dictionary and saved as a .pkl file. Second, `data_filter.py` demonstrates one possible way of filtering. The desired channel is required together with pre-selected features. Third, `data_prepare.py` reshapes the data into input matrix $\mathbf{X}$ and truth label vector $\vec{y}$. Fourth, `data_split.py` performs basic splits and channel feature removals, as channels are still present between features. For this purpose, two new methods were added to `ml_utils.py`: `apply_channel()` and `remove_channel()`. Finally, `data_view.py` observes the data obtained in the fourth step. These are all listed in Appendix [D.2](#).

4.5 Used data

At the beginning of this thesis it was mentioned that the mc16a and mc16d datasets from the years 2015-2017 were obtained from CERN. This data is located inside the folder 'older'. However, for mc16d, only one signal file with a small number of events was present. Therefore, mc16a was used mainly for training and testing, and mc16d only for additional testing. Another folder called 'newer' can also be found, which includes the latest data (both mc16a and mc16d) and which was used for comparison with the existing results in [Col19b](#) and later for re-training the classifiers. Tables with all data used are shown in Appendix [B.3](#). Firstly, 80% of the older mc16a dataset was used for training and 20% for testing. At the end of the thesis, training on the newer set is also discussed, where 80% and 20% splits were also used. The only exception is the SVC, which was trained on only 10% of the data to significantly speed up the training process. However, it was also tested on only 20%.

To distinguish between multiple data sets, a naming convention was implemented. Data obtained earlier is referred to as 'older', and the latest data as 'newer'. In both cases a suffix in the form of 'mc16a', 'mc16d', or 'mc16a & mc16d' usually follows, to provide information about what files are included.

Other terms used frequently are 'full-data' and 'channel-data' which refer to cases where an experiment was performed on the data without any preceding

τ/N_l	1	2	3
2	o	o	x
1	o	✓	o
0	x	o	o

Table 4.4: List of possible channels containing τ and light leptons (x - non-existing, o - existing, ✓ - ours)

restrictions in channel selection. As discussed below, in the case of channel 2LSS1Tau, the data is referred to as 'channel-data'.

4.5.1 Working channel

Shortly mentioned in Part I, a specific channel is observed alongside the data. This channel is called 2LSS1Tau and requires two same-sign light leptons and one hadronically decaying τ . A visualization of possible channels is demonstrated in Table 4.4.

The data for our working channel, 2LSS1Tau, contained a total of 6,729 events. 3,844 belonged to $t\bar{t}H$, 684 $t\bar{t}W$, and 2,201 $t\bar{t}Z$ respectively. Clearly this is not much data and training any classifier will be difficult.

4.5.2 Histograms

Before moving to classification, the distribution of data inside the features was observed. The results are listed in Appendix C.1.1 in the form of normalized histograms with automatic bin size selection and a kernel density estimate. Small differences between classes can be seen there. But it is clear that an automatic classification will be a challenging task.



Chapter 5

Classification



5.1 Introduction

In machine learning, a classification is a process of identifying to which category an observation belongs. It is done using a model which previously learned using training data. There are two common ways of learning: supervised, where the correct categories for the training data are well known (our case); and unsupervised, where they are not.



5.2 General types of probabilities

To be able to predict the future, several probabilities can be used, all of which require some kind of observation, intuition, or other data-related information.



5.2.1 Empirical

The basic form of probability is empirical. Having a given observation of the past, a probability can be made as:

$$p(k) = \frac{N_k}{N}, \forall k \in \mathbf{K}, \quad (5.1)$$

where N denotes the number of events.

For example, knowing that $t\bar{t}H$ was present 70 times in 100 selected events, the probability of it being present in the 101-st will be 70%.

■ 5.2.2 Subjective

Another type of probability which can be used is subjective probability. In this an opinion is formed and decisions are made on it. For example, as previously, we observe 70 $t\bar{t}H$ events in 100. However, we know that $t\bar{t}H$ is much rarer in complete data. The final prediction for the 101-st event belonging to the $t\bar{t}H$ class could be around 40%. This strategy can lead to better results but also to worse. Subjective probability was not used in this work at all.

■ 5.2.3 Apriori

To obtain good information about the data, apriori probability is a better option. In the case of balanced classes (where all classes have the same number of events), apriori probabilities would be the same as empirical probability. But in our case, as noted in Section 4.5, the data is not balanced at all. There are many ways to define apriori probabilities in such a case. The one we use is the observation of results. From Table 152 in [Col19b], the expected number of Higgs events for all classes was taken and apriori probabilities were computed as:

$$p(k) = \frac{E_k}{\sum_k E_k}, \forall k \in \mathbf{K}, \quad (5.2)$$

where E is the number of expected events for a given class.

Knowing that there were 5.5650 $t\bar{t}H$, 4.9338 $t\bar{t}W$, and 3.9677 $t\bar{t}Z$ events, the apriori probabilities are $p(k) = (0.3847, 0.3411, 0.2742)$.

5.3 Bayesian decision task

For supervised learning and well known probabilities, the Bayesian theorem can be used. This strategy is called the Bayesian decision task and is a key element in machine-learning classification.

The Bayesian classifier (sometimes called model or estimator) can be expressed as:

$$k = q(x), \quad (5.3)$$

where k denotes class, q is the strategy, and x is event data (row of matrix $\mathbf{X}$).

5.3.1 Bayes' theorem

Theorem 5.1.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (5.4)$$

Bayes' theorem is named after Reverend Thomas Bayes and describes the probability of an event, based on prior knowledge of the conditions which could be related to the event Joy. For example, knowing that the probability of having a car is related to age. More accurate information about the probability of having a car can be obtained knowing the age as opposed to not knowing.

In our case, the probability of event x belonging to class k (posterior probability) can be expressed as:

$$p(k|x) = \frac{p(x|k)p(k)}{p(x)}, \quad (5.5)$$

where $p(x|k)$ is the probability of evidence x given k is true, $p(k)$ is apriori probability, and the probability of evidence $p(x)$ can be computed using the law of total probability as:

$$p(k|x) = \frac{p(x|k)p(k)}{\sum_{k \in \mathbf{K}} p(x|k)p(k)}. \quad (5.6)$$

Clearly there are several limitations in order for these formulas to hold. The first is a knowledge of apriori probabilities. And the second is knowing the conditional probabilities. Without either of them, alternative decision tasks can be used.

■ 5.3.2 Bayesian Risk

Given a set of observations $\mathbf{X}$, a set of hidden states $\mathbf{K}$, and a penalty function $W(k, d)$, where $d \in \mathbf{D}$ is some possible decision, the strategy which minimizes the expectation of W is:

$$R(q) = \sum_{x \in \mathbf{X}} \sum_{k \in \mathbf{K}} p(x, k) W(k, q(x)), \quad (5.7)$$

which is often rewritten as:

$$R(q) = \sum_{x \in \mathbf{X}} \sum_{k \in \mathbf{K}} p(x|k)p(k) W(k, q(x)), \quad (5.8)$$

where $p(x, k)$ is a joint probability and $R(q)$ is called Bayesian risk. The solution is a Bayesian strategy q^* minimizing the risk.

■ 5.3.3 Decision strategy

Based on this strategy, a class will be returned. However, as a default, a different expression is chosen as a working point. It is the maximization of posterior probability. A classifier which uses such a strategy is often called a Maximum Posterior (MAP) estimator [Pos]. The strategy equation is obtained using Eq. 5.7 as:

$$q(x) = \arg \min_{d \in \mathbf{D}} \sum_{k \in \mathbf{K}} p(x, k) W(k, d) = \arg \max_{d \in \mathbf{D}} p(k|x). \quad (5.9)$$

For our problem, another approach was proposed, which is discussed in Section 6.5.5.

Chapter 6

The Scikit-learn library

Scikit-learn (also known as sklearn) is a popular machine-learning library written in the Python programming language. It provides an easy Application Programmable Interface (API) for many classification, regression, and clustering algorithms including support vector machines and random forests. It also comes with useful functions: for example, for preprocessing, augmentation, and scoring. A basic support of neural networks is also present. However, Graphic Processing Unit (GPU) accelerated training is not supported and the use of additional libraries is recommended [PVG⁺](#).

6.1 Splits

Usually, data is split into subsets to ensure unbiased evaluation of the prediction. There are many ways to split the data; each dataset requires observations to determine split sizes; there is no fixed split ratio and training each of the classifiers can demand different set sizes. Several functions in the scikit-learn library can serve this purpose: each splits data differently and is used in individual cases.

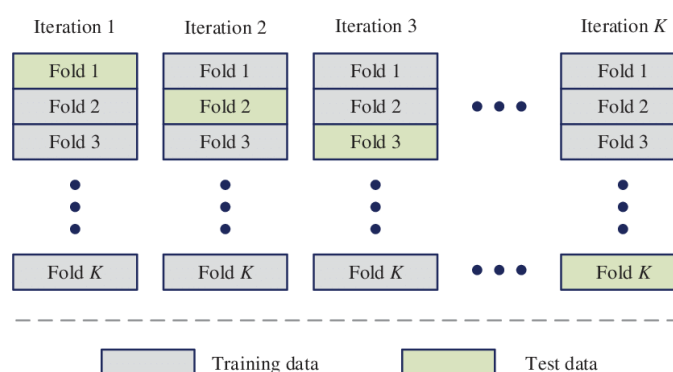


Figure 6.1: K-Fold Cross Validation procedure QR19

6.1.1 Test Split Validation

`sklearn.model_selection.train_test_split()` is a basic function for shuffling and splitting data into two subsets, one for training and one for testing. The parameters are: the data with labels in an array format; the size of the training and testing split; and two booleans, one for enabling shuffling and one for enabling stratifying. In general, this split function is used for final evaluations, when a classifier's parameters are already tuned. For the training process and tuning of parameters, two other splitting functions are often used.

6.1.2 K-Fold Cross-Validation

K-Fold Cross-Validation (CV) is a re-sampling procedure for evaluating models on K groups of original data. A common starting number of folds is 5, which is referred to as 5-Fold Cross-Validation. CV is mainly used to evaluate the performance of a freshly trained classifier on unseen data. This helps to estimate how well can it perform in general. Often, general estimations are less optimistic with predictions, although they are more accurate as they yield unbiased generalized evaluation. CV is widely used for tuning parameters of classifiers and observing their impact on the performance.

Let us divide $\mathbf{X}$ into K sets. Each set will be used once for evaluation and K-1 times for the model training. Figure 6.1 illustrates this process. For 5-Fold CV there will be 5 models trained separately on four different sets and tested on the remaining one. In scikit-learn, the `sklearn.model_selection.KFold()` function can be used for splitting data into the desired number of K-Folds.

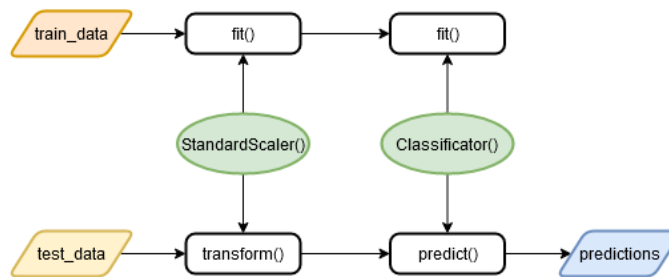


Figure 6.2: Pipeline flowchart diagram

6.1.3 Stratified K-Fold Cross-Validation

A more advanced version of K-Fold CV is its stratified version. The splitting process may be governed by criteria. Such folds are made by preserving the percentage of samples for each class. This helps the training and test splits to have almost the same distribution of data which significantly helps in boosting the performance of the test classification. The `sklearn.model_selection.StratifiedKFold()` function can serve this purpose.

6.2 Pipeline

A pipeline, `sklearn.pipeline.Pipeline()`, is a handy tool which can assemble several steps together, allowing the setting of different parameters separately and the Cross-Validating to happen together. It sequentially applies to all transformations.

As discussed in Subsection 4.3.2, it is advisable to create a pre-processor for re-distributing the data around a mean value suitable for the classifiers. Scikit-learn's Standard Scaler, `sklearn.preprocessing.StandardScaler()`, performing Gaussian with zero mean and unit variance transformation, was used for our needs and was set as the first step in the pipeline.

The second and final step in the pipeline was the introduction of a class of classifier. The whole process diagram is illustrated in Figure 6.2. One could argue that creating a pipeline just for two steps might be unnecessary. It was mainly done to ensure the modularity of code. In the case of further demand in the future (for example an addition of feature augmentation, decomposition, or model stacking) only a small change in steps would need to be made.

6.3 Classifiers

A short introduction to the format of Scikit-learn classifiers representation will be given before moving on to the experiments. For each of the classifiers, there are several common methods:

- `fit()` - performs training
- `predict()` - performs predictions (returns $\arg \max p(k|x)$)
- `predict_proba()` - performs predictions (returns $p(k|x)$)
- `score()` - returns mean accuracy on given test data and labels

Additional methods, such as `get_params()`, can appear with respect to the type of classifier and its algorithm implementations.

Each classifier also has many parameters (sometimes called hyper-parameters). These parameters need to be specified during class initialization and cannot be changed during the training process. It is important not to confuse these parameters with variables inside the algorithms. The adjustments invoked by the training are done only on the variables.

The larger the amount of data, the longer the training time. Therefore, it is very costly to train hundreds of classifiers just to tune the parameters. A process for tuning is thus desired. A group of such processes is often called Automated Machine Learning (AML) and is still under rapid development [CT], [MF]. Scikit-learn provides two methods, which can be used for tuning. The first, `sklearn.model_selection.RandomizedSearchCV()`, randomly searches the area of the parameters in a specified number of iterations and samples classifiers from them. It is useful for classification problems, where basic parameters are not working.

In contrast, the second method, `sklearn.model_selection.GridSearchCV()`, tries all value combinations inside the area of the parameters. The problem is, of course, time and complexity, and it is advised to use this grid search only for fine-tuning, when relatively good parameters have already been found.

6.4 Scoring

Before finally moving to the classifiers themselves, a few last things remain to be explained. As noted in Section 3.3, significance maximization will mainly be used for final evaluations. But in the search for optimal parameters, different scores can be observed.

6.4.1 F1-score

The F1-score is interpreted as a weighted average (harmonic mean) of the precision and recall, and reaches its best value at 1 and worst at 0. The computation for binary classification can be expressed as:

$$F_1 = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}, \quad (6.1)$$

where precision is:

$$\textit{precision} = \frac{TP}{TP + FP}, \quad (6.2)$$

and recall is:

$$\textit{recall} = \frac{TP}{TP + FN}. \quad (6.3)$$

However, it is recommended to approach the F1-score with precautions. Higher score values do not necessarily mean a better classifier because there is a large trade-off between precision and recall. During tuning, improving precision often results in a deterioration in recall and vice versa. The F1-score is useful for evaluating two or more classifiers with their own precision and recall. It combines these two computed parameters into a single number and helps to determine which of the classifiers is better. But the F1-score gives a larger weight to lower numbers. Given an example of a classifier with

precision 100% and recall 0%, the F1-score will be 0%, not 50%. Also, as mentioned, the arithmetical mean is not used. Therefore, two classifiers, one with precision 80% and recall 70%, and the other with precision 90% and recall 60%, will not have the same score. The first will score 75% and the second 72%. The low recall score resulted in a penalization for the second classifier.

In the multi-class case, the value of the F1-score depends on the average parameter. Precision and recall are computed per-class and from them the per-class F1-score is computed. This is often called the one-vs-all approach. In the next step these numbers need to be combined. There are several ways of doing so. The most common way is 'macro-F1' which uses the arithmetic mean of the per-class F1-scores as:

$$MacroF1 = \frac{F1_1 + F1_2 + \dots + F1_n}{n}, \quad (6.4)$$

where n denotes the number of classes.

Another variant is called 'micro-F1' and computes the F1-score from its micro-averaged precision and recall. All correctly predicted samples are considered as true positives. Any other predictions which put samples into the wrong class are considered as false positives but also as false negatives. The precision and recall are computed from these values. But there is no reason to compute both of them. Looking at Eq. 6.2, 6.3 and after substituting FP=FN, the precision and recall are equal to each other in the micro-averaging case. This means they are also equal to their harmonic mean. Therefore, the micro-F1 score is:

$$MicroF1 = MicroPrecision = MicroRecall. \quad (6.5)$$

These two methods would be sufficient where the classes are balanced, but in our case, the data is unbalanced. A weighted-F1 score is therefore introduced to count the weight of a given class. The general approach is to weight the F1-score of each class by the number of samples (events in our case) from that class. This can be expressed as:

$$WeightedF1 = \frac{N_1 \cdot F1_1 + N_2 \cdot F1_2 + \dots + N_n \cdot F1_n}{N}, \quad (6.6)$$

where N is the total number of events.

In scikit-learn, All variants are implemented inside function `sklearn.metrics.f1_score()`.

6.4.2 Mean accuracy score

Another method is called mean accuracy and is the default scoring method of any classifier inside a library. Again, the values can be between 0 and 1, where 1 is the best. The computation for a binary classification problem is calculated as:

$$MeanAccuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N}. \quad (6.7)$$

In a multi-class case, the formula is a little more complicated:

$$MeanAccuracy = \frac{1}{N} \sum_{k=0}^{|\mathbf{K}|} \sum_{l=0}^{|\mathbf{K}|} I(p_k = p_l). \quad (6.8)$$

$|\mathbf{K}|$ denotes the total number of classes, p is the number of predicted events for a given class, and I is a function, which returns 1 if the classes match and 0 otherwise.

However, as mentioned earlier, weighting needs to be implemented due to unbalanced classes in the data. For this, the formula can be adjusted to:

$$WeightedMeanAccuracy = \sum_{k=0}^{|\mathbf{K}|} w_k \sum_{l=0}^{|\mathbf{K}|} I(p_k = p_l). \quad (6.9)$$

Sadly, the scikit-learn library does not support custom weights for the `sklearn.metrics.accuracy_score()` function. Judging by my observations, a mean weighting is used as:

$$w_k = \frac{1}{|\mathbf{K}|}. \quad (6.10)$$

6.4.3 ROC AUC score

The area under the ROC curve (AUC) is a useful tool for evaluating the quality of class separation for most types of classifier. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. Again, 0 denotes the worst, and 1 the best score. For the multi-class problematic a one-vs-all method, similar to the precision/recall one-vs-all computation, or one-vs-one method can be used. Also, similar averaging methods (macro, micro, weighted) for the F1-score are used in the case of multi-class classification. Documentation of the [sklearn.metrics.roc_auc_score\(\)](#) function can be seen for further information.

For binary (or one-vs-all/one) classification:

$$TPR = \frac{TP}{P}, \quad (6.11)$$

$$FPR = \frac{FP}{N}, \quad (6.12)$$

where P is the number of all positive and N of all negative samples.

6.5 The Random Forest Classifier

6.5.1 Introduction

Theorem 6.1. *A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.* [Yiu19](#)

Before moving to Random Forest (RF) it is crucial to understand Decision Trees (DT). The visualization in Figure [6.3](#) can be used as an example. In the diagram, a binary classification problem is solved using a single decision tree. Firstly, the feature 'colour' is observed. Based on this, the samples are split into two groups. However, there are none present in the left split. Therefore, another feature, 'underline' is observed, and the final estimation is made.

Random forest or random decision forest is an ensemble classification and regression method. Multiple decision trees are built during the learning process. Randomness is introduced to suppress the habit of decision trees to over-fit on a training set, and there are two common random processes used to combat this over-fit. Bagging (also known as Bootstrap) reduces the impact of small changes on the training set and allows each individual tree to randomly sample from the dataset which results in different trees. The other, Feature Randomness, reduces the number of features considered during the split. By default, in DT every feature is considered and the one with the biggest separation power between the left and right node is selected. Not doing so forces more variation amongst the trees and results in a better classification on unseen data.

The trees also protect each other from their individual errors. While some trees can be wrong, many others will be right, and so a group move in the correct direction will be possible. The final output is decided in the form of voting between the trees.

6.5.2 Parameters

There are many parameters to be tuned for RFC scikit-learn library implementation. Firstly, the performance of the classifier trained on default parameters

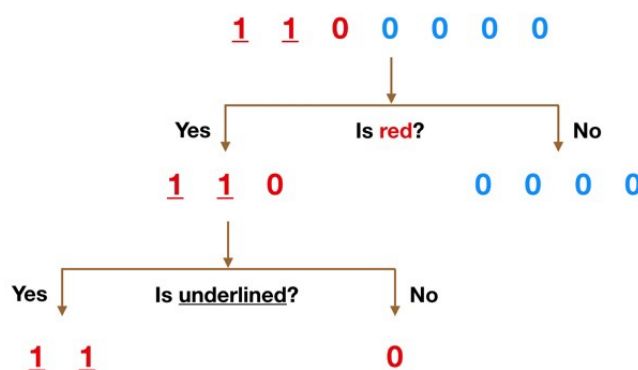


Figure 6.3: Elemental decision tree classification [Yiu19]

was obtained using the script `train_rfc.py`. In Figure 6.4, ROC curves can be seen for the training and test splits of 80% and 20% respectively. As discussed in Subsection 6.4, three scores are listed to evaluate performance.

Judging by Figure 6.4, default parameters can serve as a good basis. However, an over-fitting problem occurred. To reduce this and smooth the classification, a Grid Search with 5-Fold Stratified CV was chosen for fine tuning the parameters. Script `tuning_rfc.py` can be viewed for implementation. Five splits allowed consistency in the split ration. Classifiers were always trained on 80% and tested on 20% of the older mc16a data. Later, the results are saved by date and constant identifier in the form of 'script_name_year_month_day_constant'. The console outputs are located in the 'logs' folder, and the tuning results in the form of a Pandas `M+` table are in the folder 'tuning'.

The searched space and parameter definition are listed in Table 6.1. Note that each parameter was tuned independently. In the same table the best parameters obtained from the evaluating score and time performances are listed for both full-data and channel-data models. These results were obtained from the script `tuning_rfc_results.py` and are listed in Appendix C.2.1.

The job file `tuning_rfc.job` was created to submit a tuning script to the Sun Grid Engine-governed (SGE) [Cor10] CTU cluster servers. The training/tuning processes were done on 8 cores (K8 or E5, 2000 - 2400, depended on cluster) with 32 GB of dedicated memory.

Several parameters were not tuned. One of them was the 'bootstrap'. Forcing a 'False' value would undermine the randomness of the RFC and lower the performance on general data. 'min_sample_leaf' was also not tuned due to having a similar effect on the classifier as a tuned 'min_sample_split'.

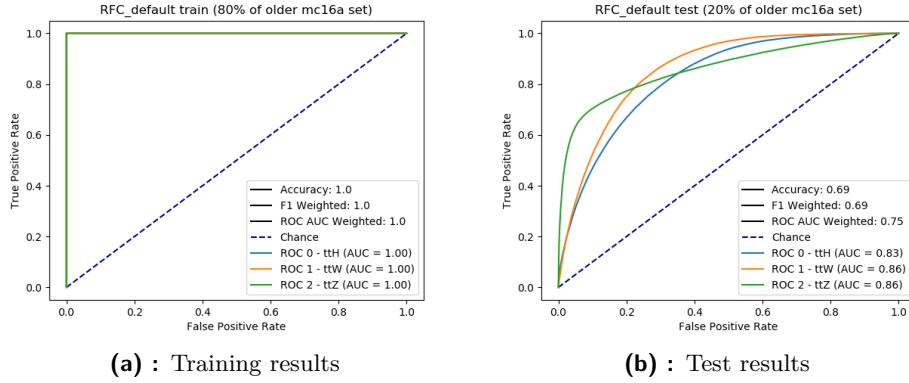


Figure 6.4: Performance of RFC with default parameters

parameter	definition	search space	best value	best value (channel)
n_estimators	the number of trees in the forest	range(50, 601, 20)	540	50
max_depth	the maximum depth of the tree	range(10, 101, 5)	15	10
min_samples_split	minimum number of samples required to split an internal node	range(2, 203, 5)	117	122
max_features	the number of features to consider when looking for the best split	linSPACE(0.05, 1, 20)	0.3	0.55
max_leaf_nodes	grow trees in best-first fashion (best - relative reduction in impurity)	range(2, 603, 5)	522	47
min_impurity_decrease	a node will be split if this split induces a decrease of the impurity greater than or equal to this value	linSPACE(0, 0.1, 21)	0.0	0.01 (0.0)*
ccp_alpha	complexity parameter used for Minimal Cost-Complexity Pruning	linSPACE(0, 0.1, 21)	0.0	0.005 (0.0)*
max_samples	the number of samples to draw (using bootstrap) from X to train each base estimator	linSPACE(0.05, 1, 20)	0.85	0.85
oob_score	whether to use out-of-bag samples to estimate the generalization accuracy	['False', 'True']	'True'	'True'
criterion	the function to measure the quality of a split	['gini', 'entropy']	'gini'	'gini'

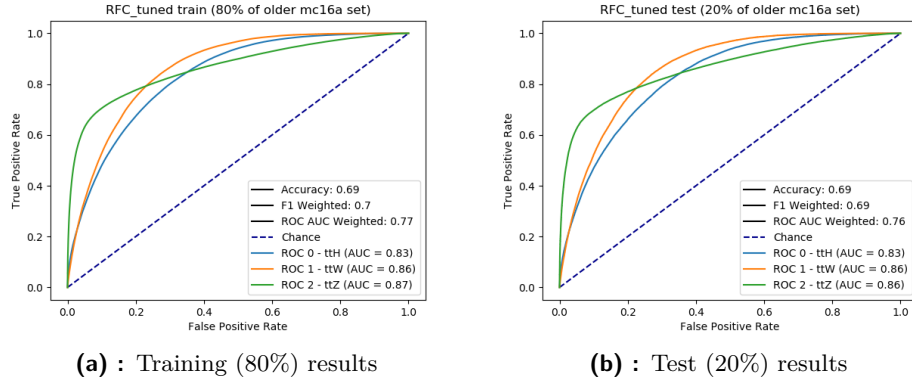
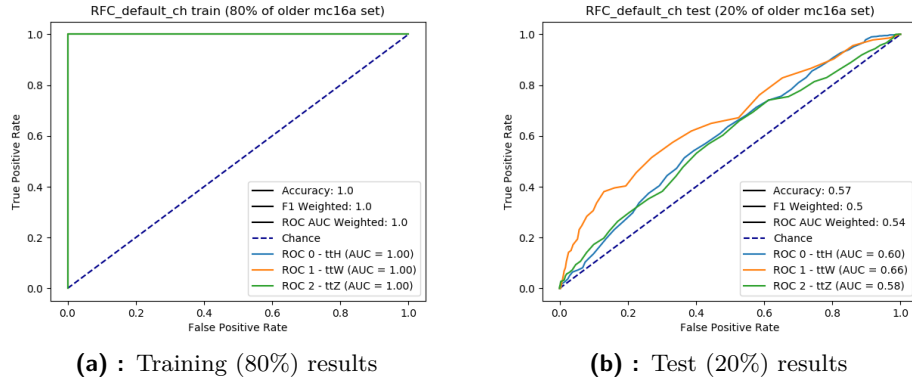
Table 6.1: List of tuned parameters of RFC with results

Due to not having weighted samples, 'min_weight_fraction_leaf' (which depends on sample weights) was dropped. The 'warm_start' parameter was also rejected for tuning. It can significantly speed up the training process for an increase of certain parameters (such as 'n_estimators'). However, it is advised to approach this parameter with caution as it uses previous training results. There was also no reason to tune the 'class_weights' parameter as the apriori probabilities from Eq. 5.2 were used.

From the tuning results, the best parameters were chosen and a new classifier trained. Figure 6.5 contains ROC curves with scores included. Compared to Figure 6.4, the over-fit was clearly reduced. However, there is almost no improvement in performance. To boost it, a working point setting was observed.

Channel

But before moving there, experiments were also carried out for our working channel 2LSS1Tau. Again, 80% of the data was selected for training and 20% left for test/cross-validation purposes. Figure 6.6 contains ROC curves for the model trained using default parameters and Figure 6.7 the one trained with parameters obtained from tuning. Apart from the achievement of over-fit

**Figure 6.5:** Performance of RFC with tuned parameters**Figure 6.6:** Performance of RFC (channel) with default parameters

reduction, no improvement is seen. This is probably due to not having enough data for training. The performance of the classifier trained on 80% of the full-data and tested on the channel-data was observed to prove this fact. The results are shown in Figure 6.8.

It is good to note that many parameters depend on each other. This is a well known problematic and one of the possible tuning strategies is tuning them together. However, the computation demand and time of such an operation is great.

6.5.3 Feature importances

Also, feature importances were observed. In Figure 6.9, 15 features with the highest importances are listed along with their std. Table B.1 in Appendix

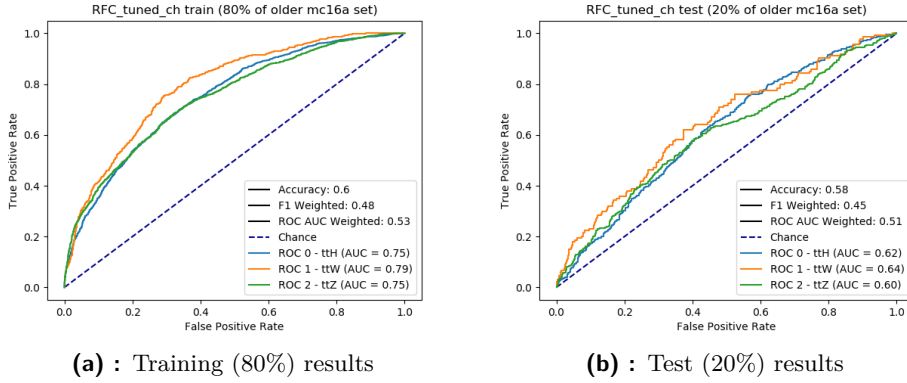


Figure 6.7: Performance of RFC (channel) with tuned parameters

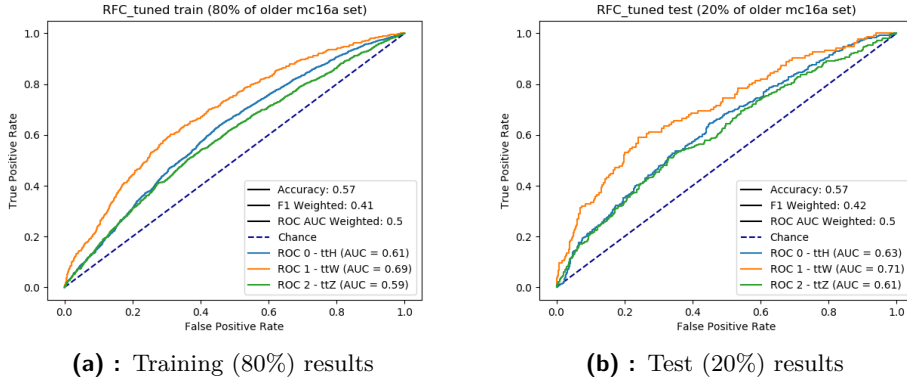


Figure 6.8: Performance of RFC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

[B.1](#) can be used to translate the feature number into a name and [Table 6.2](#) provides more information about selected features. During the training of the channel classifier with tuned parameters, a feature importance drop was observed. Only one (number 5) was present and had an importance of 1. After further observations, two parameters causing this behaviour were discovered: 'min_impurity_decrease' and 'ccp_alpha'. Both were set to 0.0, which is denoted with * in the tuning table.

For the full-data model, the 'best_Z_Mll' feature took a lead, followed by many other features with lesser importance. For the channel-data model 'Mll01' is in the lead, again followed by many others. Looking at histograms ([Appendix C.1.1](#)), it is not a surprise that only plots with a lot of information were listed as important features.

ID	name	definition
13	best_Z_Mll	smallest mass difference between Z mass and invariant mass of lepton pair
5	Mll01	invariant mass of lepton pair (leading and subleading)
11	Mll012	invariant mass of leptons (leading, subleading, and sub-subleading)
6	Mll02	invariant mass of leptons (leading and sub-subleading)
44	lep_isPrompt_2	
43	lep_isPrompt_1	
0	DRll01	special distance of jet and lepton
77	total_charge	total electric charge of leptons
8	lep_isTightLH_2	lepton is fulfills high quality criteria
58	nJets_OR	number of jets
60	nJets_OR_T	number of jets after overlap removal
31	lep_Pt_2	transverse momentum of sub-subleading lepton
81	HT_lep	sum of transverse momentum of all leptons
68	tau_fromPV_0	
79	total_leptons	
15	lep_Eta_0	Pseudo-rapidity of leading lepton
80	HT	sum of transverse momentum of all objects
2	MET_RefFinal_et	missing transverse energy
16	lep_Eta_1	
56	lep_promptLeptonVeto_TagWeight_1	prompt lepton veto
63	Tau_MV2c10_0	tau isolation
4	MV2c10_70_EventWeight	tau isolation
30	lep_Pt_1	transverse momentum of subleading lepton
82	HT_jets	sum of transverse momenta of jets
29	lep_Pt_0	transverse momentum of leading lepton
35	lep_chargeIDBDTTight_0	charge of leading lepton with high quality

Table 6.2: List of the most important features with descriptions

6.5.4 Additional tests

Two extra tests were carried out. The first on the older mc16d data, which consists of just one signal file and the second on the newer mc16a and mc16d set. The results are shown in Figure 6.10. Only small differences were observed. For the older mc16d data, the performance was slightly better. For the newer data, it was almost exactly the same as the training data in Figure 6.5 (b). The over-fit was suppressed successfully and general classification yielded approximately the same results.

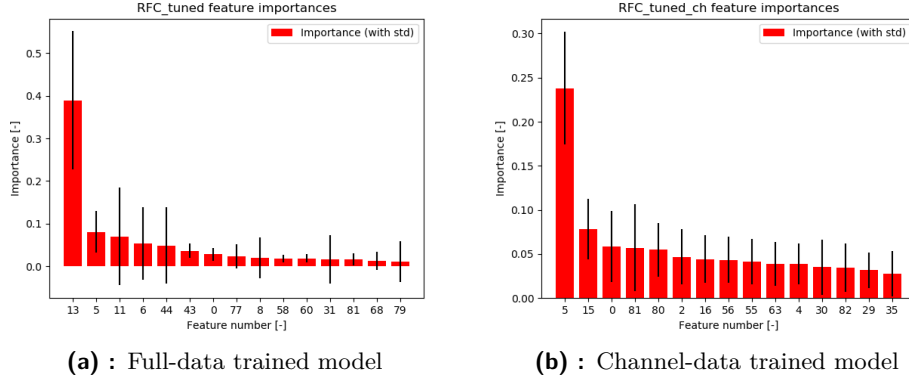


Figure 6.9: Feature importances for full-data model (left) and channel-data model (right)

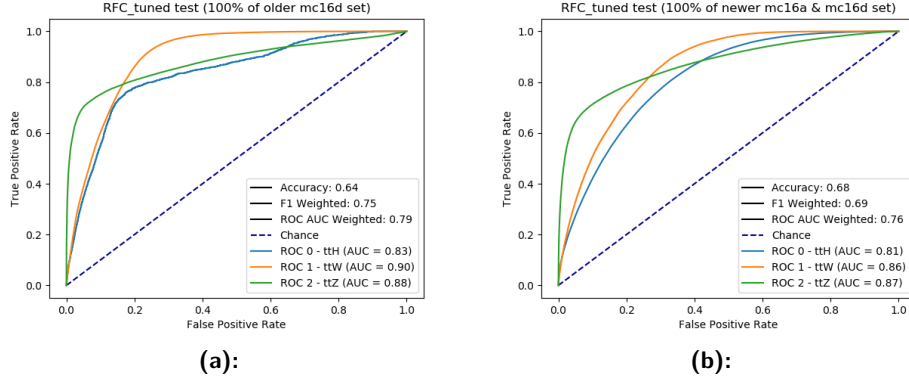


Figure 6.10: Performance of RFC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

6.5.5 Working Point

Significance maximization

To maximize the significance score, a new decision strategy was introduced:

$$\phi(x, t) = \begin{cases} t\bar{t}H, & \text{for } p(t\bar{t}H|x) \geq t, \\ t\bar{t}W, & \text{for } p(t\bar{t}W|x) \geq p(t\bar{t}Z|x), \\ t\bar{t}Z, & \text{for } p(t\bar{t}W|x) < p(t\bar{t}Z|x), \end{cases} \quad (6.13)$$

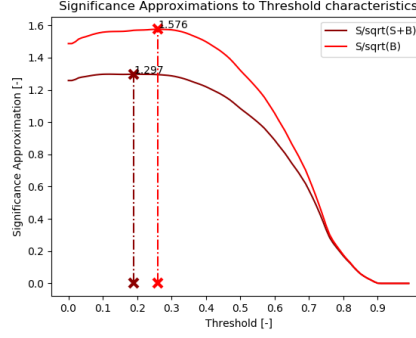


Figure 6.11: Dependence of significance on the threshold, vertical lines denote maximums

where t is the threshold constant.

This is basically the threshold $t\bar{t}H$ class vs. all other backgrounds. If the threshold is not met, the background class with maximum probability is returned.

Weighting factors, the number of events, and other background elements of the computations were obtained using Table 152 from [Col19b], as shown in Appendix A.1.

Figures 6.11 and 6.12 contain the significance approximations and characteristics of efficiencies S and B up to the change of the threshold. In the first figure, vertical lines denote the maximum value of any given approximation. In the second, vertical lines denote values of the given characteristics at the point of the selected threshold from the first figure. Note that all predictions were done on the newer data.

As a working point, a threshold of 0.26 was selected, as the simplified significance approximation was highest for this value.

In Figure 6.13, confusion matrices for the default (Eq. 5.9) and new (Eq. 6.13) decision functions are located. Clearly, setting a lower threshold caused more background to be classed as a signal.

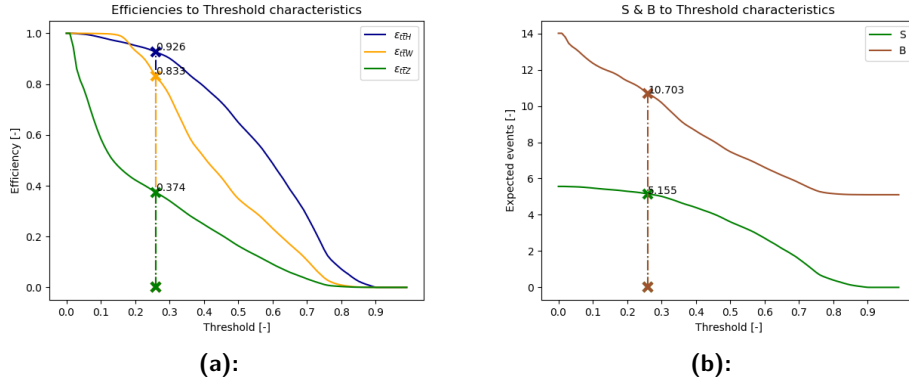


Figure 6.12: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

■ Sensitivity maximization

There is also another approach to set a working point: recall (also known as sensitivity) maximization. The same decision function (Eq. 6.13) as significance maximization tuning was used. The sensitivity was observed at each interval of the false discovery rate ($FDR = 1 - \text{precision}$) value. The maximum value was again selected as the working point. Characteristics with a confusion matrix are shown in Figure 6.14. An FDR interval which does not exceed 0.4 was selected. For this interval, the maximum sensitivity of magnitude 0.681 was found at a threshold value of 0.47. Of course, a working point tuned this way will not outperform the previous one, when it comes to significance. But for general classification evaluation, it yields more accurate results.

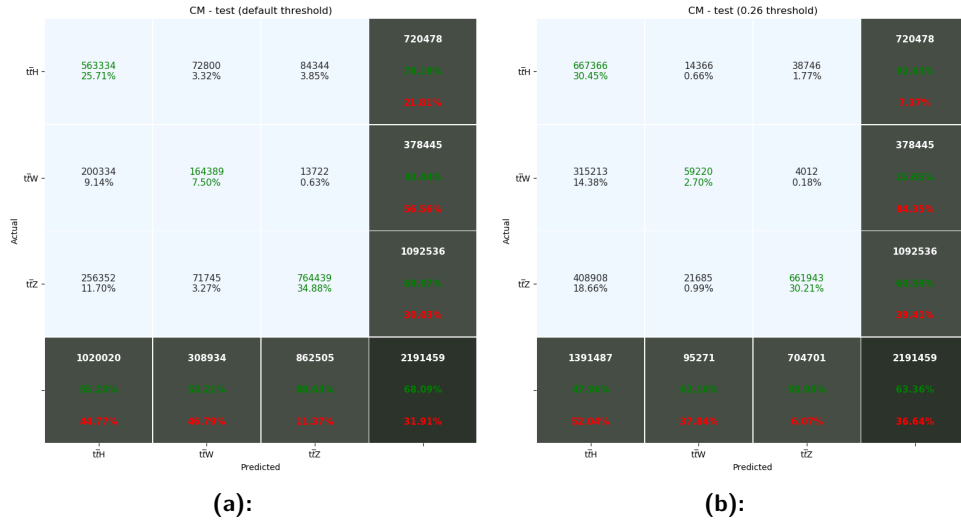


Figure 6.13: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

Both significance and sensitivity maximization tuning can be found in the script [tuning_wp.py](#).

6.5.6 Conclusion

The RFC served as a good starting point in our classification problem. However, even after tuning, the classifier is not ideal. From observing histograms, it can be seen that finding differences between classes is very difficult. This was confirmed during the evaluation of performance on channel-data. The amount of data in it is not efficient enough to form strong classification boundaries. As noted, only a few features out of 83 were used to train the classifier. The rest were not used due to low information gain. Several solutions can be proposed. The first is to add more features: we are working with only a small fraction of the possible 2000 features. However, it is very difficult to select a good feature and ensure that it is not only simulation related. Several experiments were carried out to select good features automatically. However, they always yielded unusable results which were only available in the simulation data. Another solution could be to add more data. As is shown by the performance of the full-data model, more data boosts the classification and every feature has at least some importance. Adding data to the full-data would also result in more data in the channel-data, since the channel selection is based on the full-data. The last possibility would be to augment the features to yield more information. This is very difficult and

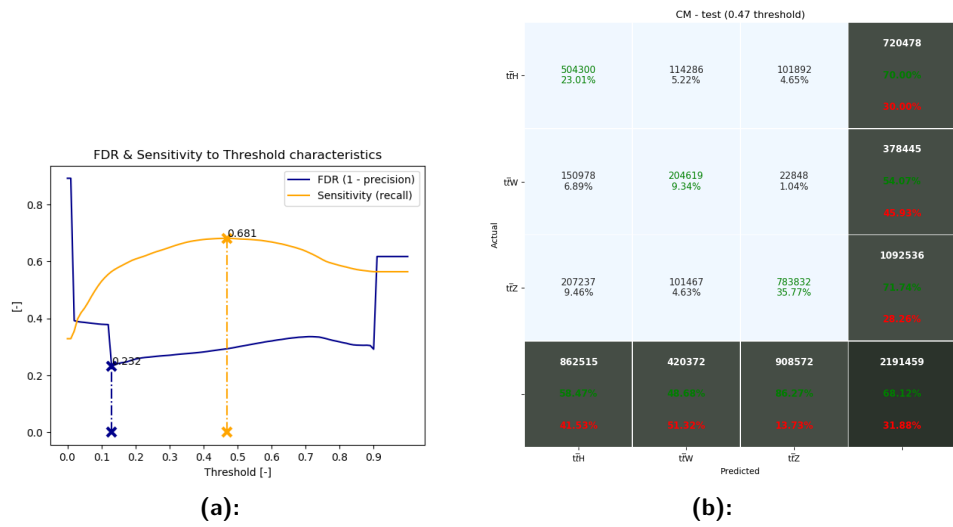


Figure 6.14: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

would require in-depth data analysis.

6.6 The K-Neighbors Classifier

6.6.1 Introduction

The K-Neighbors Classifier (KNC) is one of the lazy learning algorithms. Lazy means that any generalization on training data is delayed until a test query is invoked. This is the exact opposite of the RFC (an eager learning algorithm, which generalizes directly on the training data). This of course makes training faster but testing slower. In the worst case, testing requires the scanning of all data points. This can require much more memory than any of the eager learning algorithms do.

In the KNC, K is the number of the nearest neighbors, and the algorithm can be summed up in the following steps, which are also visualized in Figure 6.15:

- store training points
- calculate distances for test point
- find K-nearest neighbors
- vote for the output class

6.6.2 Dimension reduction

Being a lazy learning algorithm, the KNC performs better with a smaller number of features. A larger number of features usually leads to an over-fitting problem. To avoid this, an exponential increase in data for each of the features is needed as we increase the number of dimensions (number of features). Such a problem is called the curse of dimensionality.

PCA

To reduce the dimension of the data, several approaches can be used. One of them is Principal Component Analysis (PCA). In general, this method works

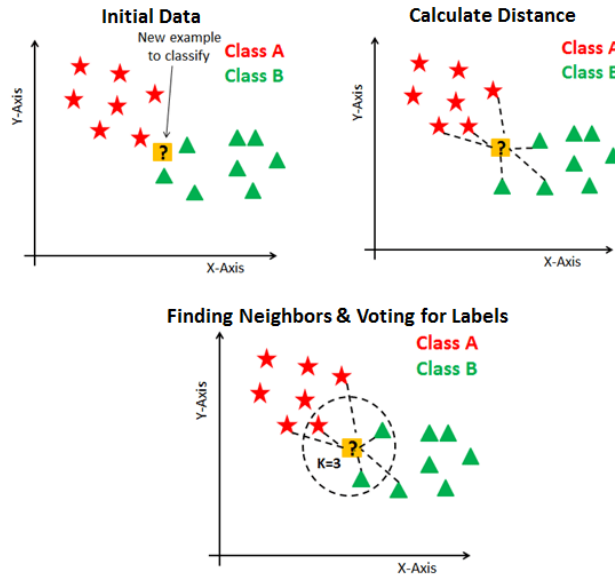


Figure 6.15: KNC steps visualization for binary classification in two-dimensional space [Nav18b]

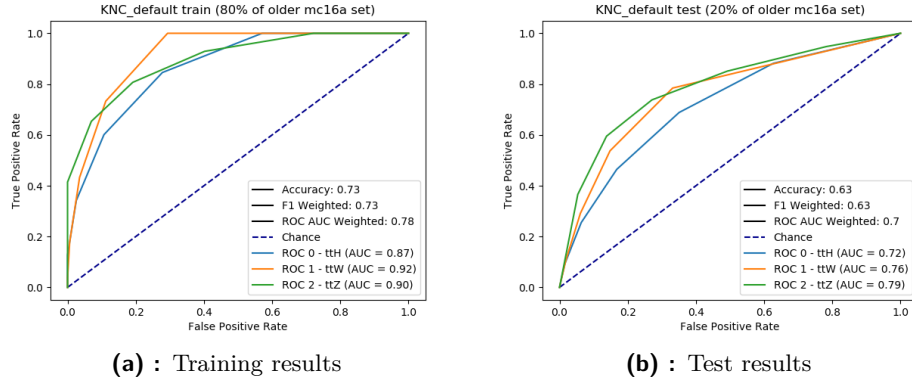
by mapping the original dataset into a new space. In this space, columns of the matrix are orthogonal. This can be viewed as a transformation, which creates a new covariance matrix. In this matrix the first component can explain certain percentages of the variance on whole data. Scikit-learn implementation is in the class `sklearn.decomposition.PCA()`.

■ Truncated SVD

Singular Value Decomposition (SVD) factors matrix M into the three matrices U , Σ , and V^T . For SVD, the factorization is done on the data matrix, whereas for PCA, the factorization is done on the covariance matrix.

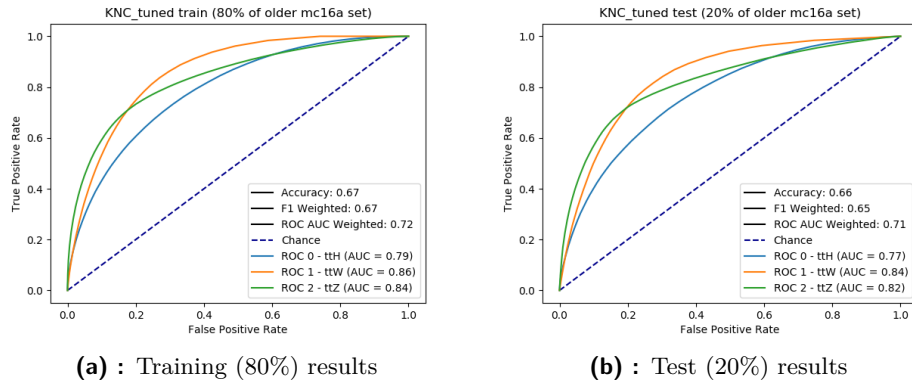
Truncated SVD adds another step into the factorization. It produces it only when the number of columns is equal to the specified truncation. It therefore produces a specified number of columns in result data. In comparison, normal SVD would produce n columns, where n is number of columns in the original data. More information on scikit-learn implementation can be found in `sklearn.decomposition.TruncatedSVD()`.

One of the big advantages of Truncated SVD over PCA is that it can operate on sparse matrices. For PCA, the covariance matrix must be computed on the entire matrix.



(a) : Training results

(b) : Test results

Figure 6.16: Performance of KNC with default parameters

(a) : Training (80%) results

(b) : Test (20%) results

Figure 6.17: Performance of KNC with tuned parameters

Metric

It is good to note that the metric used within the algorithm can also suffer from higher dimensionality. It is well known that in large dimensions, Euclidean distance is no longer useful, and so Mikowski distance is often used. Its power parameter allows us to directly set the best distance for any given problem.

6.6.3 Parameters

Both PCA and Truncated SVD algorithms were observed. Their instances were added into the pipeline, between the scaler and the classifier. For PCA, setting the percentage of the variance which can be explained was observed. For Truncated SVD, it was the number of columns. These parameters are

algorithm	parameter	search space	best value	best value (channel)
PCA	n_components	linspace(0.9, 1, 20)	0.9 (would need further tuning)	0.957 (w.n.f.t.)
Truncated SVD	n_components	range(2, 41, 1)	13	24

Table 6.3: List of tuned dimension reduction algorithms for KNC with results

parameter	definition	search space	best value	best value (channel)
n_neighbors	number of neighbors used by queries	range(1, 102, 2)	53	5
leaf_size	leaf size passed to BallTree or KDTree	range(1, 102, 2)	23	1
p	power parameter for the Minkowski metric	2**linspace(-2, 10, 25)	2**0.5	2**1
weights	weight function used in prediction	['uniform', 'distance']	'uniform'	'uniform'
algorithm	algorithm used to compute the nearest neighbors	['auto', 'ball_tree', 'kd_tree', 'brute']	'kd_tree'	'auto'

Table 6.4: List of tuned parameters of KNC with results

called 'n_components' for both classes. However, for PCA a float value between zero and one is accepted. In the case of Truncated SVD, an integer can be passed. For Truncated SVD, 13 was selected as a working point: lower values resulted in worse performance, and higher only increased learning time exponentially. To determine if the score is close to the results which can be obtained for the original data, PCA for the 'n_components' parameter between values 0.9 and 1 was observed. It proved that reducing the dimension does not result in worse performance of marginal magnitude. Instead it is almost negligible. Table 6.3 lists the best values for both algorithms.

Comparing the performance and predict time, Truncated SVD was selected and used for tuning the parameters of the classifier. Table 6.4 contains the results of this tuning, and Appendix C.2.2 shows tuning plots.

Comparing the results for the default parameters in Figure 6.16 with tuned parameters in Figure 6.17 shows a decrease in over-fit and also small improvements in the scores. However, the results are not as good as when using the RFC.

Channel

Again, the channel was also observed. Dimension reduction was again tuned and a Truncated SVD of 24 components yielded the best results. After setting this, other parameters were then tuned, and the results are shown in Table 6.4. Figure 6.18 contains ROCs for the model trained on the channel-data with the default KNC's parameters and Figure 6.19 with tuned parameters.

The performance of the classifier trained on 80% of full-data and tested on the channel-data was also observed again, and the results are shown in Figure 6.20. A small improvement is visible for the test set ROC curves.

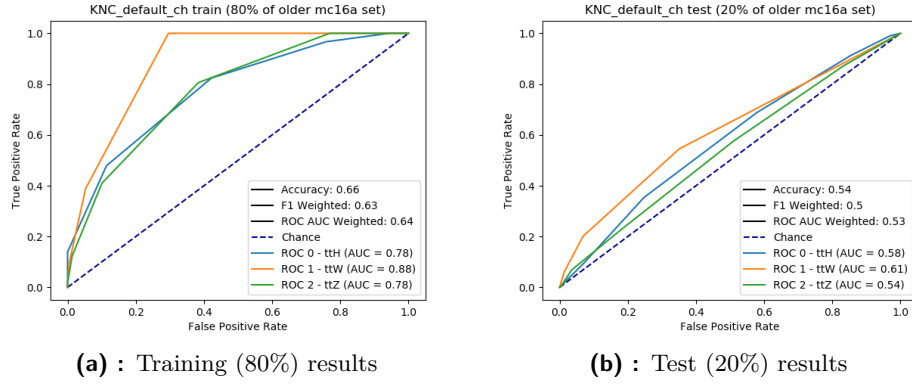


Figure 6.18: Performance of KNC (channel) with default parameters

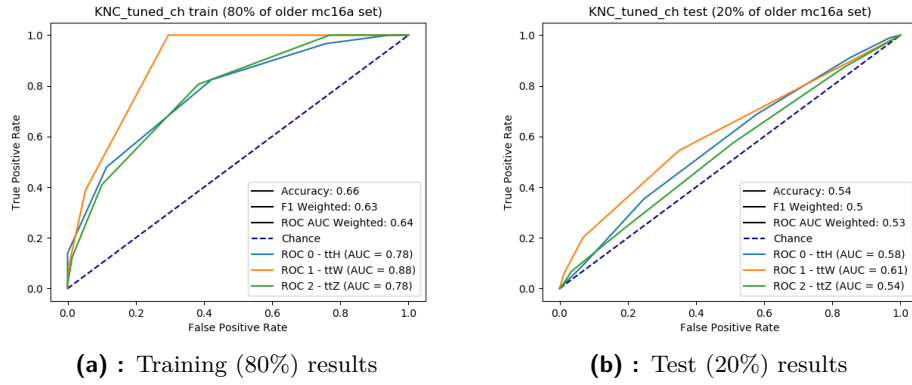


Figure 6.19: Performance of KNC (channel) with tuned parameters

6.6.4 Additional tests

For additional tests, no large deviation from Figure 6.17 was observed. Once again, the older mc16d set did a little better compared to any other. Figure 6.21 contains the results of the tests.

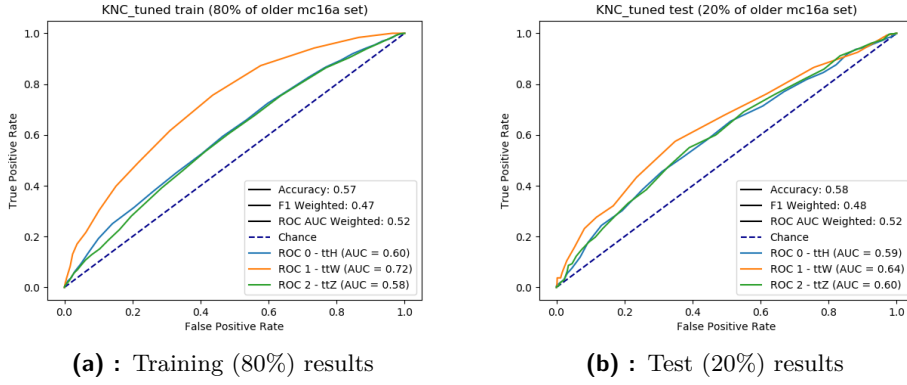


Figure 6.20: Performance of KNC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

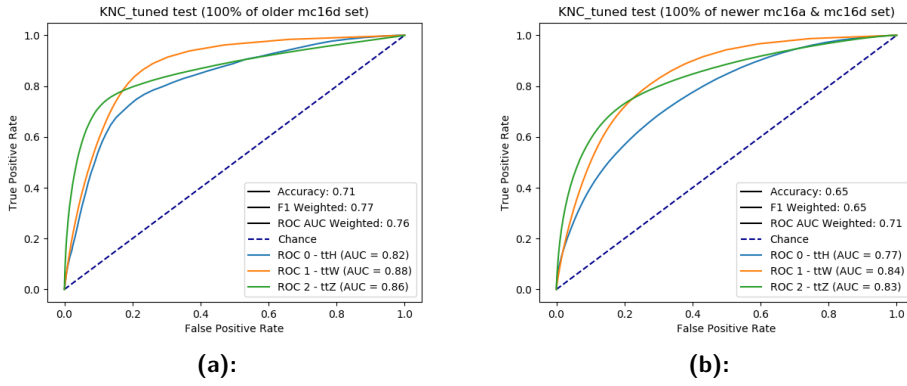


Figure 6.21: Performance of KNC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

6.6.5 Working Point

Significance maximization

The previously derived Eq. [6.13](#) was used to find an optimal threshold to maximize significance. Figures [6.22](#) and [6.23](#) contain characteristics of efficiencies, S , B , and significance approximations for the values of the threshold.

For the threshold, 0.08 was selected as the working point as it yielded the maximum significance. In Figure [6.24](#), confusion matrices for the default (Eq. [5.9](#)) and new decision functions (Eq. [6.13](#)) are located. Again, with a higher significance, more background is classified as a signal. Surprisingly, in

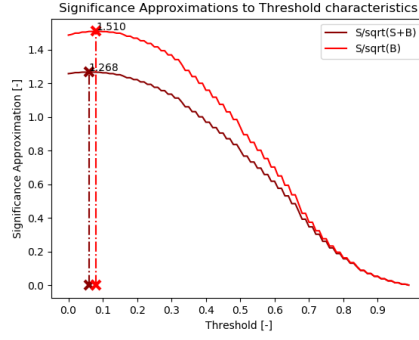


Figure 6.22: Dependence of significance on the threshold, vertical lines denote maximums

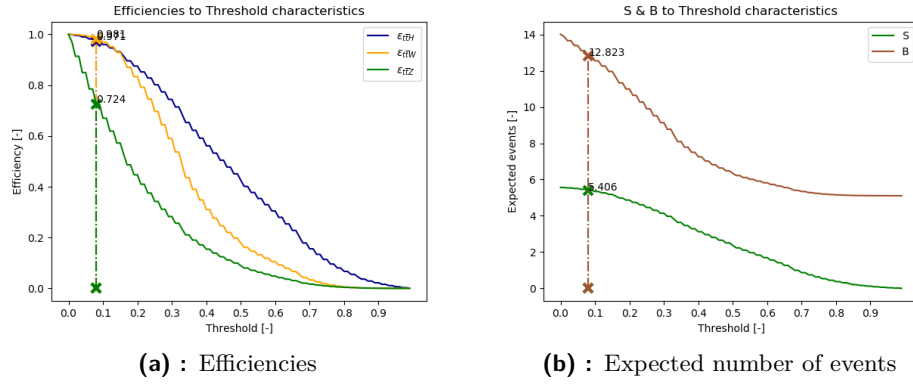


Figure 6.23: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

the case of the KNC, the highest simplified significance is yielded for $t\bar{t}W$ efficiency (0.9111), which is higher than for $t\bar{t}H$ (0.9714).

■ Sensitivity maximization

The second tuning method, which was proposed in the RFC section was observed as well. In the case of the KNC, the optimal threshold was found to be 0.4, which is equal to a sensitivity of 0.652. Figure 6.25 contains the results of the tuning.

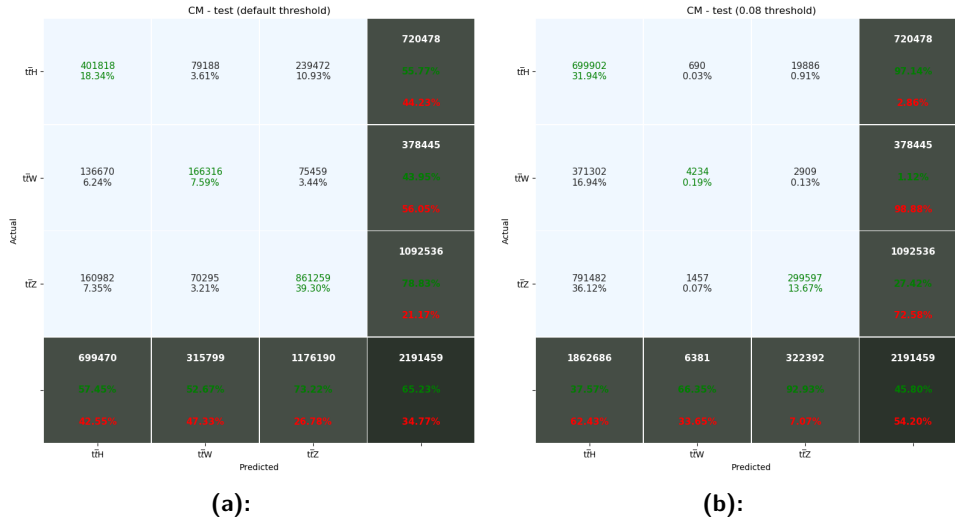


Figure 6.24: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

6.6.6 Conclusion

The KNC did perform a little worse compared to the RFC. It was raised that the KNC suffers from the curse of dimensionality. But even reductions using decomposition methods could not achieve the results of the RFC. Also, again the data itself was shown to be a great problem. The points stored during the learning phase were very close to each other for the signal and the background classes. This resulted in a large number of false classifications, even with a great number of neighbors from which the output was computed.

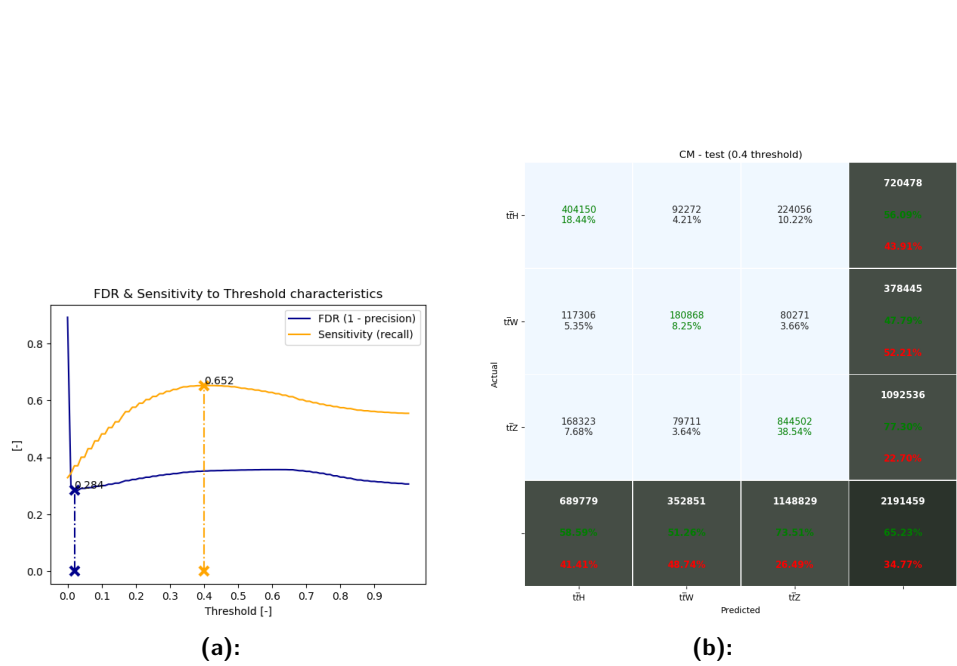


Figure 6.25: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

algorithm	parameter	search space	best value	best value (channel)
PCA	n_components	linspace(0.1, 0.9, 81)	0.77	0.78
Truncated SVD	n_components	range(2, 41, 1)	23	14

Table 6.5: List of tuned dimension reduction algorithms for GNB with results

parameter	definition	search space	best value	best value (channel)
priors	aprior probabilities of the classes	['None' (Eq. 5.1), 'Custom' (Eq. 5.2)]	'Custom'	'Custom'
var_smoothing	portion of the largest variance of all features that is added to variances for calculation stability	logspace(-15, -3, 13)	1e-13	1e-3

Table 6.6: List of tuned parameters of KNC with results

6.7 Gaussian Naïve Bayes

6.7.1 Introduction

Naïve Bayes (or shortly NB) is type of classifier which implies Bayes' theorem 5.1 with the naïve assumption of features being independent from each other. An NB can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution. Such an extension is called a Gaussian NB. Our data is already pre-scaled using a standard scaler to have the form of a Gaussian (or normal) distribution. Therefore, it was proposed that we test the performance of this elementary but sometimes surprisingly powerful tool.

6.7.2 Parameters

Similar to the KNC, dimension reduction can significantly speed up, and help with, the classification process. Table 6.5 contains the results of the dimension reduction algorithms' tuning. Truncated SVD with 23 components yielded the best results.

The GNB is a simple classifier: not many parameters can be tuned. Only two of them, called 'priors' and 'var_smoothing', are accessible. The former represents apriori probabilities: by default the GNB computes them as empirical probabilities. The latter denotes the portion of the largest variance of features from which stability is calculated.

Plots of tuning processes can be found in Appendix C.2.3. Table 6.6 contains the best parameters for both full-data and channel-data models. Introducing custom apriori probabilities yielded negligibly worse results compared to the

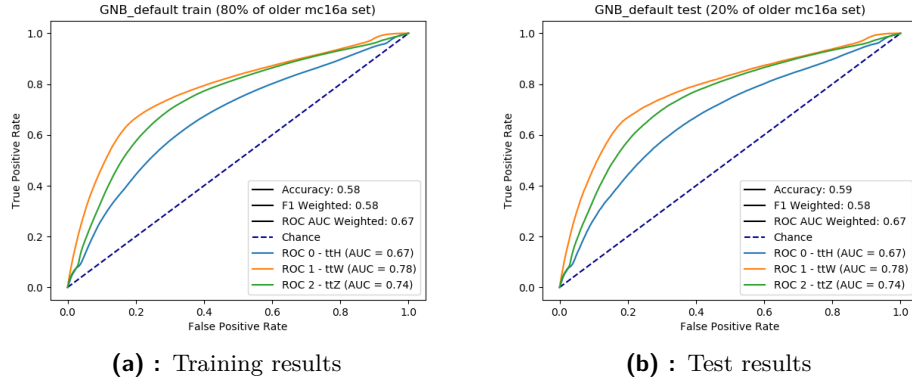


Figure 6.26: Performance of GNB with default parameters

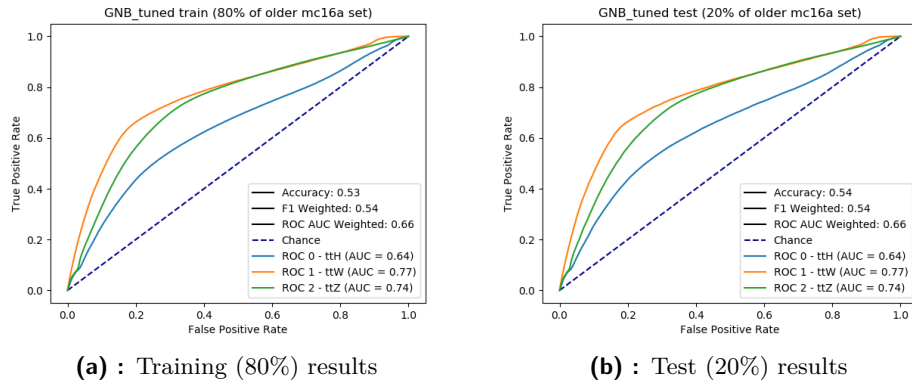


Figure 6.27: Performance of GNB with tuned parameters

empirical one (Figures 6.26 and 6.27). However, in the future, when real data will be tested, the apriori probabilities will surely outperform the empirical.

Channel

Dimension reduction was also observed for our working channel. It yielded slightly better scores. Therefore, it was decided that Truncated SVD with 14 components will be used. ROCs for both default and tuned models are shown in Figures 6.28 and 6.29. Again, the performance was a little lowered due to the introduction of custom apriori probabilities.

The performance of the full-data model on 20% and the channel-data was observed once again. Results are shown in Figure 6.30.

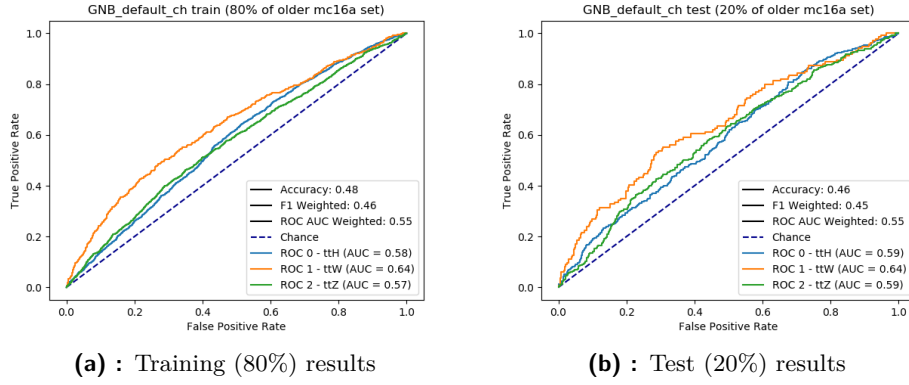


Figure 6.28: Performance of GNB (channel) with default parameters

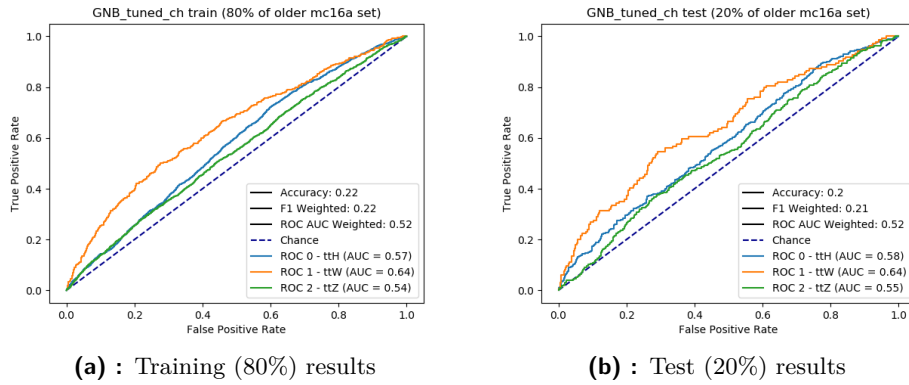


Figure 6.29: Performance of GNB (channel) with tuned parameters

6.7.3 Additional tests

The results of the tests are shown in Figure 6.31. Once again, the classifier which performed on the older mc16d set did a little better in comparison to the others. For the newer mc16a and mc16d data, the results are almost identical to Figure 6.27 (b).

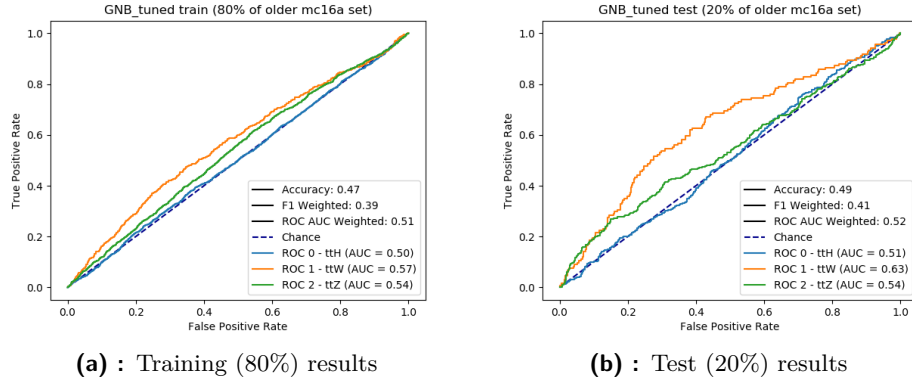


Figure 6.30: Performance of GNB with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

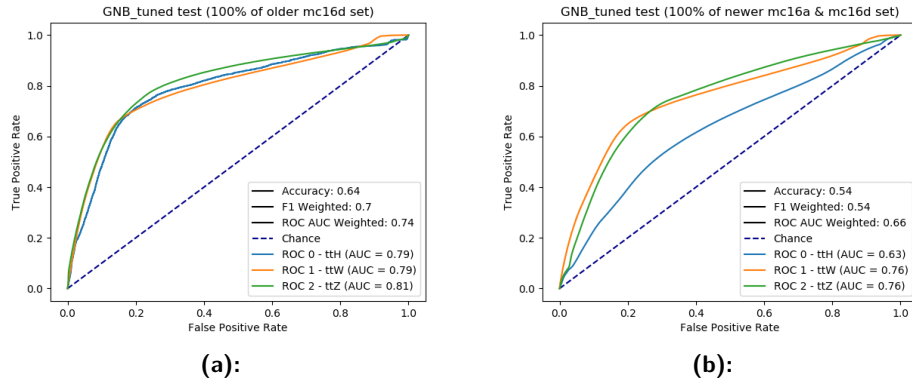


Figure 6.31: Performance of GNB with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

6.7.4 Working Point

Significance maximization

The threshold was again tuned to maximize the simplified significance approximation. Figures 6.32 and 6.33 contains the results.

For the threshold, 0.0 was selected by the algorithm as it yielded the best significance. This resulted in all events being classified in the $t\bar{t}H$ class (Figure 6.34), completely undermining the purpose of the classification.

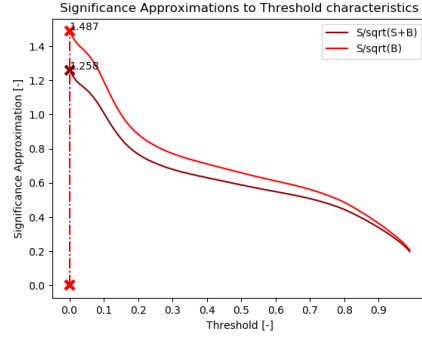


Figure 6.32: Dependence of significance on the threshold, vertical lines denote maximums

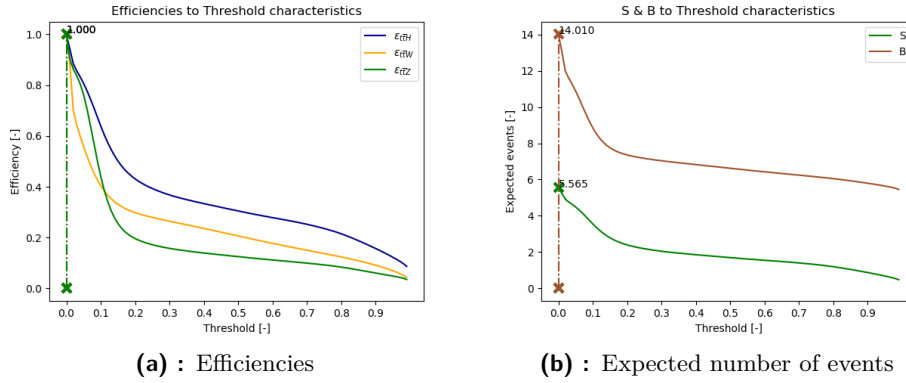


Figure 6.33: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

■ Sensitivity maximization

More reasonable results were obtained for the second type of tuning and are shown in Figure 6.35. A threshold of 0.2 was selected and yielded a relatively good result (0.541) for a simple classifier such as the GNB.

■ 6.7.5 Conclusion

A quick diversion from complex ML algorithms was a nice variation from this difficult task. As expected, poorer results were obtained. This resulted in instability during the significance maximization tuning. Instead of the significance of the threshold curve being a concave function with local maximum between the 0 and 1, its shape was more like a convex function. On one hand

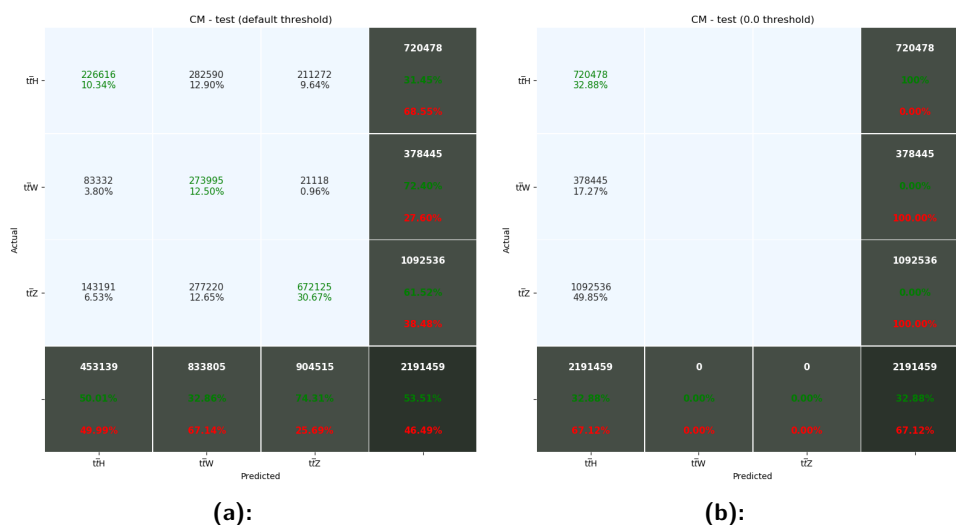
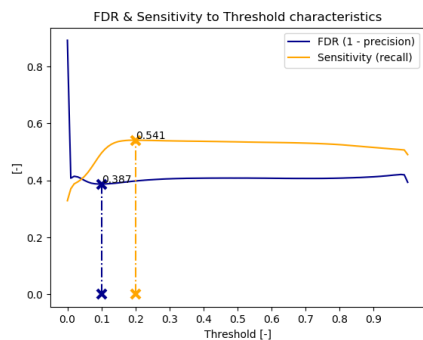


Figure 6.34: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

the significance was maximized, but on the other, the classification saturated and was completely unnecessary. The same results could be achieved directly by not performing the classification at all.



(a):

CM - test (0.2 threshold)				
Actual	tH -	tW -	tZ -	
	310352 14.16%	238226 10.87%	171900 7.84%	720478 43.08% 36.82%
	112726 5.14%	250412 11.43%	15307 0.70%	378445 66.17% 35.63%
	214084 9.77%	253276 11.56%	625176 28.53%	1092536 57.22% 42.78%
	637162 48.71% 31.28%	741914 33.78% 66.21%	812383 76.96% 23.04%	2191459 88.13% 45.88%
		Predicted		

(b):

Figure 6.35: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

algorithm	parameter	search space	best value	best value (channel)
PCA	n_components	linspace(0.1, 0.9, 81)	0.89	0.87
Truncated SVD	n_components	range(2, 83, 1)	76	24

Table 6.7: List of tuned dimension reduction algorithms for ADA with results

6.8 AdaBoost Classifier

6.8.1 Introduction

In the past, boosting algorithms were developed to achieve higher accuracy at a reasonable cost, and became a very popular tool. They combine multiple weak classifiers into a more powerful ensemble. As opposed to the RFC, no bagging or bootstrapping is present. The main process behind boosting is setting the weights, of which there are two types: the 'observations' and classifiers' weights. In general, they help to track hard-to-learn observations and classifiers with the highest accuracy. This also allows the boosting algorithms to avoid suffering from over-fitting. AdaBoost (ADA) stands for Adaptive Boosting and is one type of boosting algorithm. The other type, used later in this thesis, is Gradient Boosting. The steps of a general adaptive boosting algorithm are:

- randomly select a training subset
- iteratively train
- assign weights to the observations (a high weight means a less accurate observation)
- assign weights to the classifiers (a high weight means a better score)
- repeat until convergence or a number of estimators is reached
- vote for output

6.8.2 Parameters

To speed up the learning process, dimension reduction was once again observed. Table 6.7 contains the results for both PCA and Truncated SVD methods.

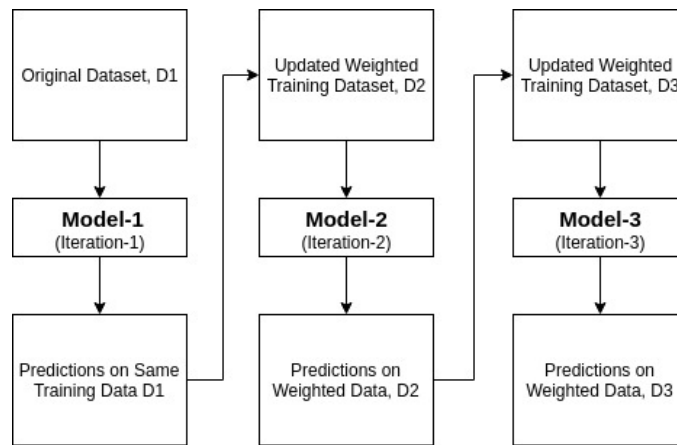


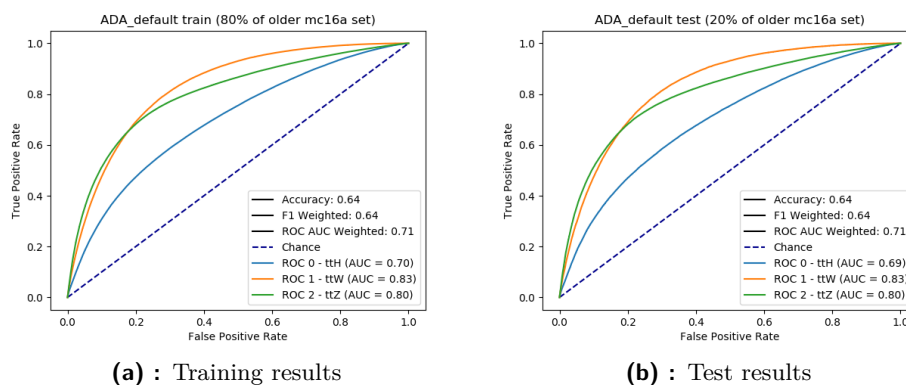
Figure 6.36: AdaBoost steps visualization Nav18a

There are a total of five parameters which can be changed. Two of them, 'algorithm' and 'random_state' are set by default to values which yield the best results. This leaves only three parameters to be tuned. 'n_estimators' is the maximum number of classifiers (estimators) used inside the ensemble, in case the convergence is not reached in advance. 'learning_rate' reduces the contribution of classifiers. And 'base_estimator' is the weak classifier object used as a key element in the ensemble. There are several limitations on these, including sample weights and classes support. More information can be obtained by viewing [sklearn.ensemble.AdaBoostClassifier\(\)](#). Table 6.8 can be referred to for tuning results.

As a weak estimator, an elementary Decision Tree (DT) of depth 1 is used as a default. To achieve proper classification, the DT parameter 'class_weight' was additionally set to the apriori probabilities from Eq. 5.2. The depth was later observed and the results listed in Table 6.8 are noted with an asterisk. Using a Support Vector Classifier (SVC) was also proposed. However, due to the computational complexity of the SVC algorithm for high dimensional data and the requirement of vast numbers of trained classifiers, this method was abandoned and experiments using a separate group of SVCs were carried out later in Section 6.11.

Due to being limited by time, only one parameter change was able to be tuned at a time. This formed as a crucial problem when deciding between the number of estimators, and the depth of the base estimators. In the end, based on Figures C.48 and C.50, setting the depth to 6 was selected as an optimal approach, as it yields better values for both scores and time complexity compared to a great number of estimators tuned for a DT of depth 1.

parameter	definition	search space	best value	best value (channel)
<code>n_estimators</code>	the maximum number of estimators at which boosting is terminated	<code>range(50, 501, 50)</code>	500 (50*)	150
<code>learning_rate</code>	learning rate shrinks the contribution of each classifier by its value	<code>logspace(-20, 1, 22)</code>	1e0	1e0
<code>base_estimator</code>	the base estimator from which the boosted ensemble is built	DT of depth 1 to 10	1 (6*)	1

Table 6.8: List of tuned parameters of ADA with results**(a) :** Training results**(b) :** Test results**Figure 6.37:** Performance of ADA with default parameters

Channel

Once again, our working channel was observed. Truncated SVD with 14 components yielded better results compared to PCA and was selected for further tuning. ROCs for both the default and tuned model are shown in Figures 6.28 and 6.29.

The performance of the full-data model on 20% and 80% of channel-data is shown in Figure 6.30.

6.8.3 Additional tests

The results of the tests are shown in Figure 6.42. The older mc16d test set once again performed slightly better. For the newer set, almost the same results as in Figure 6.27 (b) were obtained.

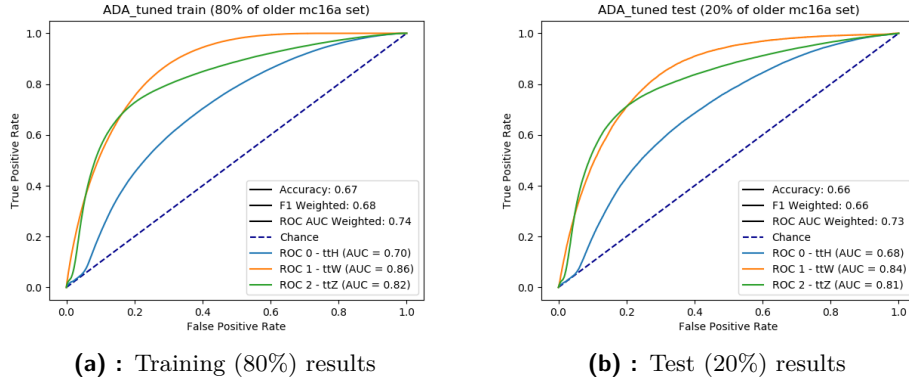


Figure 6.38: Performance of ADA with tuned parameters

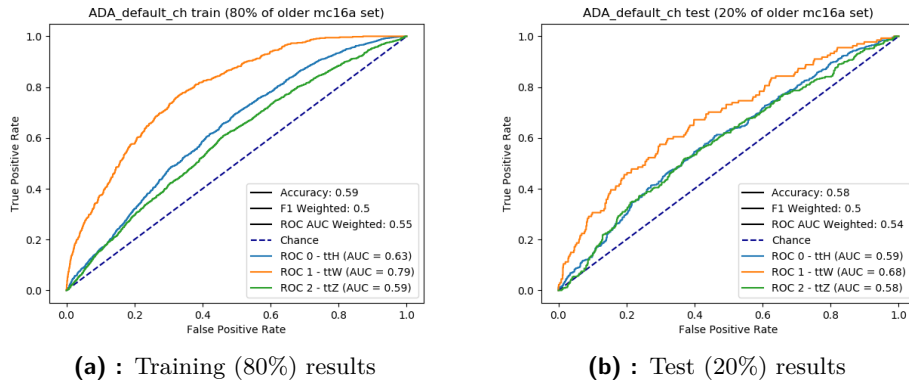


Figure 6.39: Performance of ADA (channel) with default parameters

6.8.4 Working Point

Significance maximization

To maximize the simplified significance approximation, the threshold value was once again tuned, and the results are shown in Figures 6.43 and 6.44.

The value 0.33 was selected for the threshold as it yielded the best significance. In Figure 6.45, confusion matrices for the default and new threshold are shown.

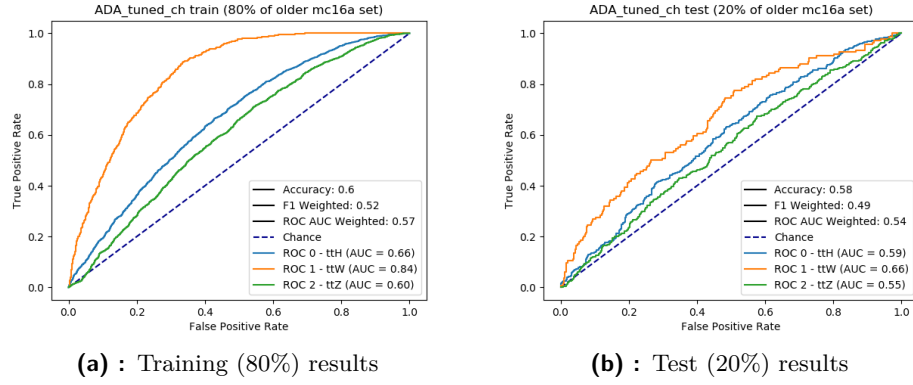


Figure 6.40: Performance of ADA (channel) with tuned parameters

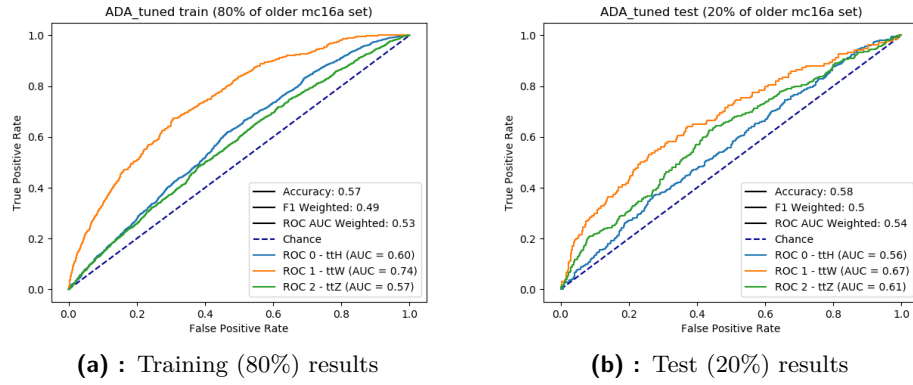


Figure 6.41: Performance of ADA with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

■ Sensitivity maximization

The results were also obtained for the second type of tuning and are shown in Figure 6.46. A threshold of 0.34 was selected, yielding a sensitivity of magnitude of 0.582.

■ 6.8.5 Conclusion

The ADA performed slightly better compared to the GNB. However, both the RFC and the KNC achieved even higher scores. Further tuning of the ADA is certainly an option for boosting its performance. But due to the computational demand and time constraints, it was not performed. All the

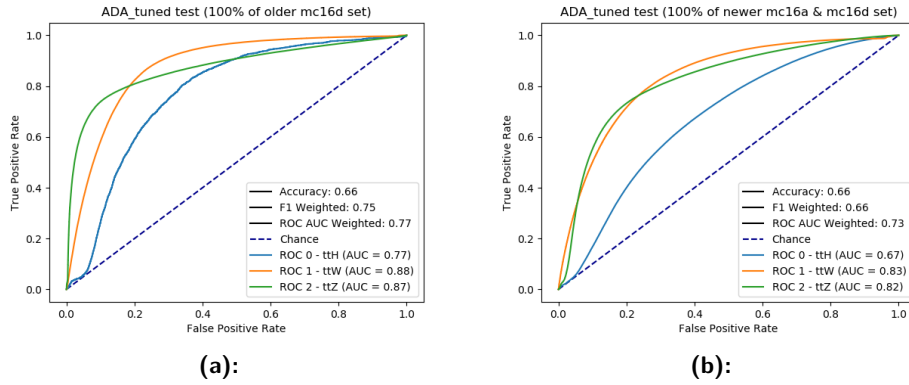


Figure 6.42: Performance of ADA with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

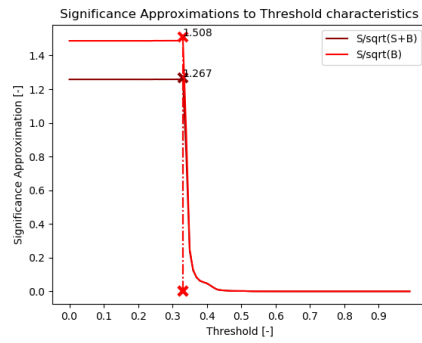


Figure 6.43: Dependence of significance on the threshold, vertical lines denote maximums

characteristics in the working point tuning sections were very interesting. Instead of concave, the curves were in the form of rectangular waves, possibly due to the elementary classifiers inside the ensemble.

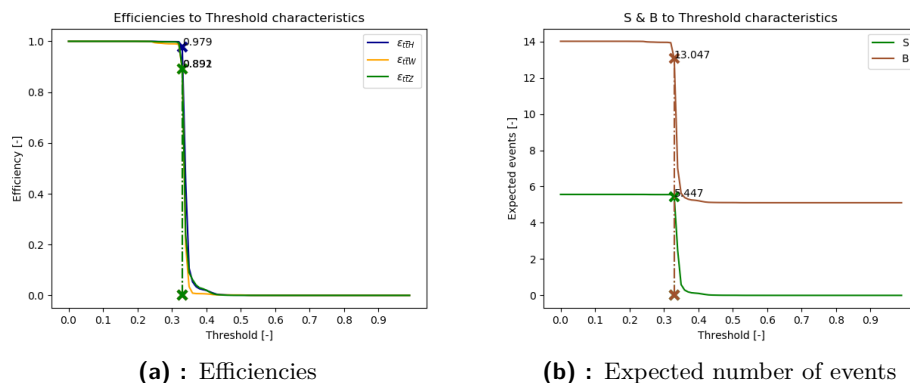


Figure 6.44: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

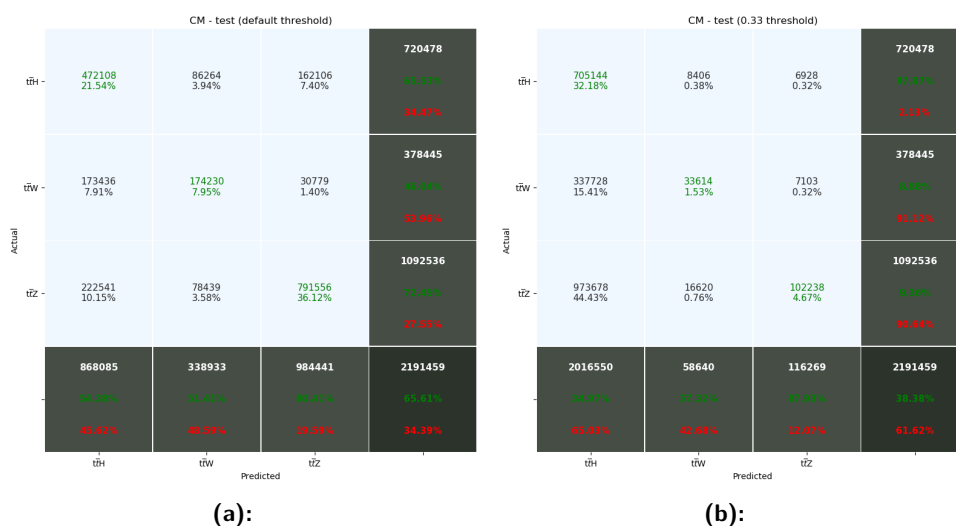
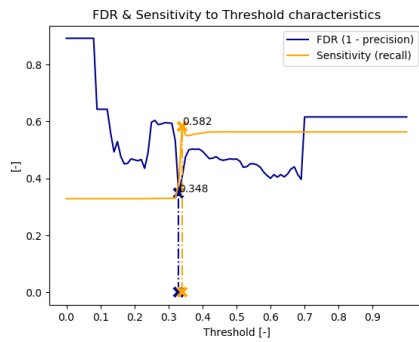


Figure 6.45: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data



(a):

CM - test (0.34 threshold)					
Actual	tH -	322208 14.70%	179554 8.19%	218716 9.98%	720478 44.72% 30.38%
	tW -	73701 3.36%	243588 11.12%	61156 2.79%	378445 64.37% 25.43%
	tZ -	258682 11.80%	124538 5.68%	709316 32.37%	1092536 64.92% 35.08%
		654591 49.22% 30.74%	547680 44.48% 38.33%	989188 71.71% 38.19%	2191459 84.81% 41.81%
	tH	tW	tZ		
	Predicted				

(b):

Figure 6.46: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

algorithm	parameter	search space	best value	best value (channel)
PCA	n_components	linspace(0.1, 0.9, 81)	0.9*	0.9*
Truncated SVD	n_components	range(2, 61, 1)	38	21

Table 6.9: List of tuned dimension reduction algorithms for GBC with results

parameter	definition	search space	best value	best value (channel)
learning_rate	learning rate shrinks the contribution of each classifier by its value	logspace(-5, 1, 7)	1	1
subsample	the fraction of samples to be used for fitting the individual base learners	linspace(0.1, 1, 10)	1	1
criterion	the function to measure the quality of a split	['friedman_mse', 'mse']	'friedman_mse'	'friedman_mse'
n_estimators	the number of boosting stages to perform	range(100, 201, 10)	200*	100
min_samples_split	the minimum number of samples required to split an internal node	range(2, 103, 5)	2**	2**
max_depth	maximum depth of the individual regression estimators	range(3, 11, 1)	10*	6
min_impurity_decrease	a node will be split if this split induces a decrease of the impurity greater than or equal to this value	linspace(0, 0.1, 21)	0**	0**
max_features	the number of features to consider when looking for the best split	linspace(0.05, 1, 20)	1	1
max_leaf_nodes	grow trees with max_leaf_nodes in best-first fashion	range(2, 103, 5)	'None**'	'None**'
ccp_alpha	complexity parameter used for Minimal Cost-Complexity Pruning	linspace(0, 0.1, 21)	0	0

Table 6.10: List of tuned parameters of GBC with results

6.9 Gradient Boosting Classifier

6.9.1 Introduction

Alongside adaptive, gradient boosting is another technique for boosting an ensemble of weak classifiers, although the approach is completely different. Every time a new weak classifier is added, the weights of the previous one are frozen. Due to this, the already existing weak classifiers inside the ensemble remain at their fixed values. The boosting itself depends on a loss function. Any custom function can be given, but it needs to be differentiable. However, in general, a logarithmic loss function is mainly used and can satisfy the majority of the ML problems. In this thesis, GBC is used as an abbreviation for the gradient boosted classifier.

6.9.2 Parameters

PCA and SVD were observed and once again, SVD with Truncation to r-first elements scored higher than PCA. Table 6.9 contains the dimension reduction results.

Contrary to the ADA, for the GBC there are many tunable parameters. In Table 6.10, the majority are shown with the best values found during tuning. However, several parameters were not tuned. The first of them: 'loss' was desired to be set to a default value, in order not to perform adaptive boosting once again. The second: 'min_samples_leaf', as in the case of the RFC is very similar to 'min_samples_split'. Another parameter,

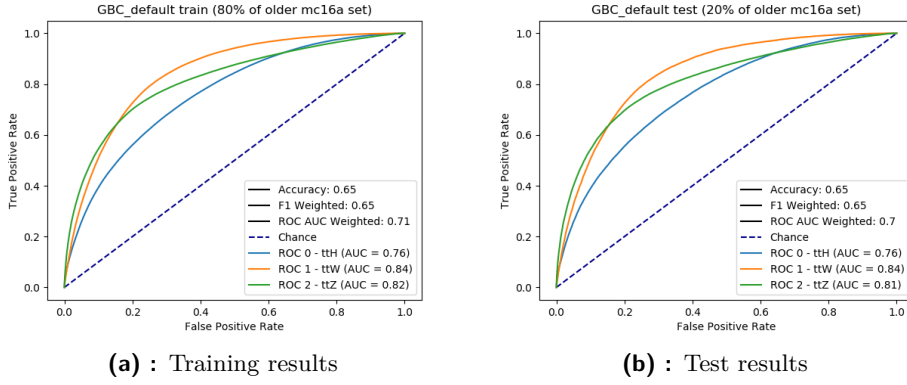


Figure 6.47: Performance of GBC with default parameters

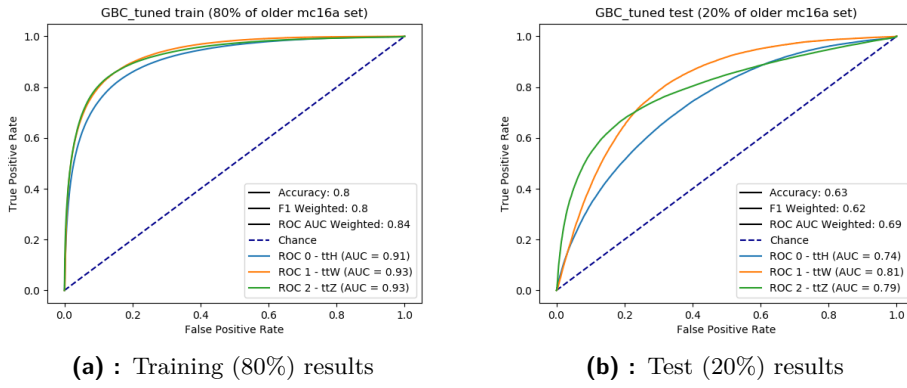


Figure 6.48: Performance of GBC with tuned parameters

'min_weight_fraction_leaf', was not used due to not having weighted samples. It was decided that the parameter 'init' would not be tuned due to its default setting to a Dummy Classifier, which is used to compute the initial predictions. Also, the 'random_state' default value was preserved, using the random function of the NumPy library. As discussed previously for the RFC, 'warm_start' can help improve the training, however great precaution is recommended, and so it was not tuned. The last parameters which were not tuned were related to early stopping behavior: namely, 'n_iter_no_change', 'validation_fraction', and 'tol'. As we aspired for the best results possible, default values were preserved and early stopping was disabled to allow longer, but uninterrupted, learning.

An asterisk (*) in the tables denotes that the parameter would require further tuning to be perfectly tuned but that there was not enough time to continue the tuning. A double asterisk (**) means that default value was preserved because tuning had negligible results.

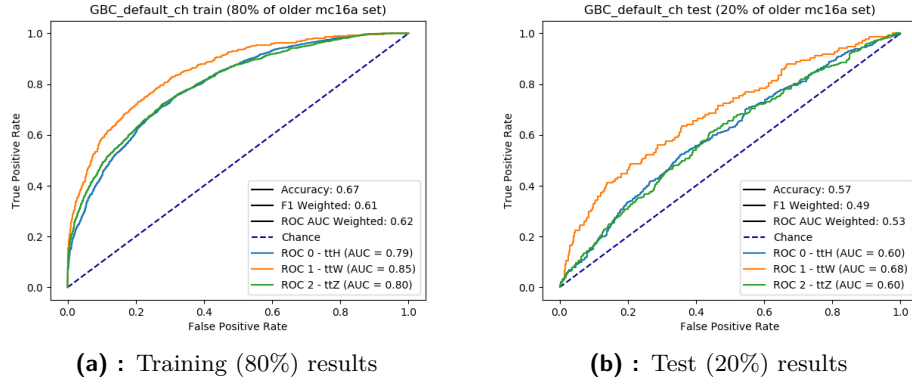


Figure 6.49: Performance of GBC (channel) with default parameters

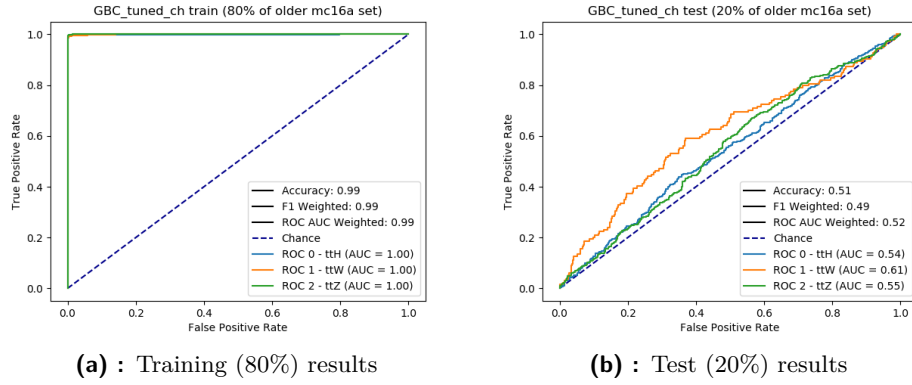


Figure 6.50: Performance of GBC (channel) with tuned parameters

Channel

For the channel, the results are also shown in Table 6.10. Figure 6.49 contains ROCs for the model trained on the channel-data with default GBC parameters. For the tuned model, see Figure 6.50.

The performance of the classifier trained on 80% of full-data and tested on the channel-data is shown in Figure 6.20.

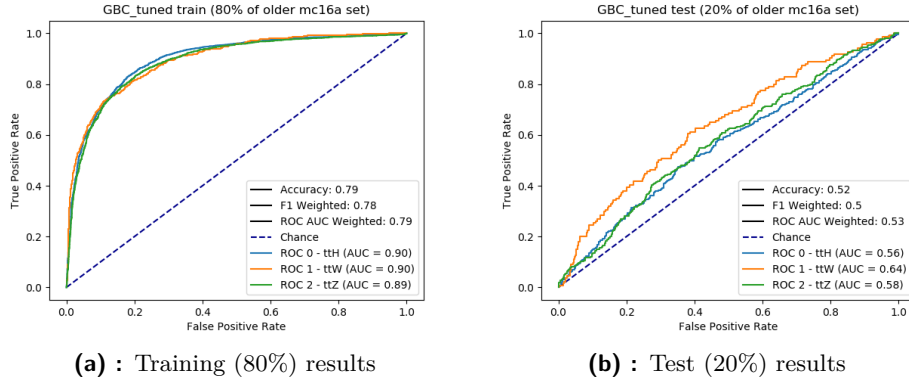


Figure 6.51: Performance of GBC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

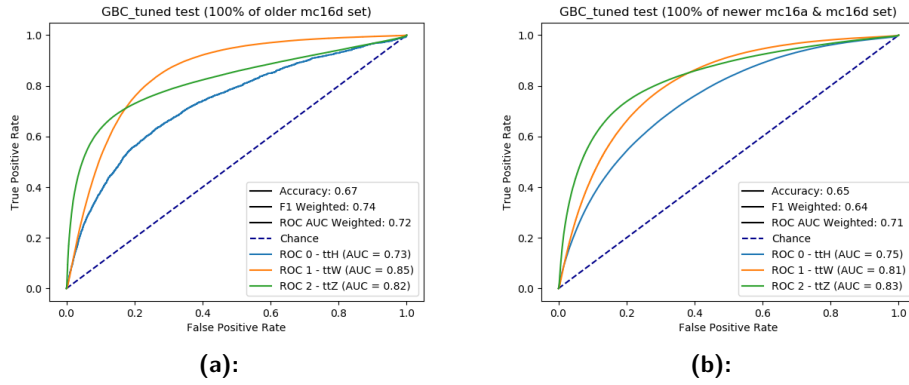


Figure 6.52: Performance of GBC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

6.9.3 Additional tests

No large deviation from Figure 6.48 was observed for any of the additional tests. The older mc16d set did a little better once again and the newer set yielded almost identical results to the default test set. The results are shown in Figure 6.52.

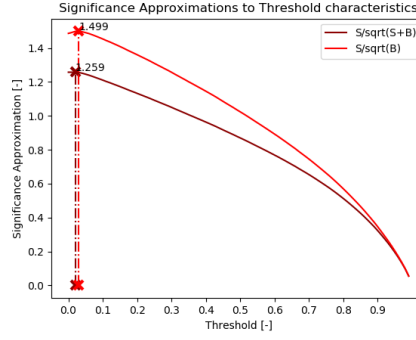


Figure 6.53: Dependence of significance on the threshold, vertical lines denote maximums

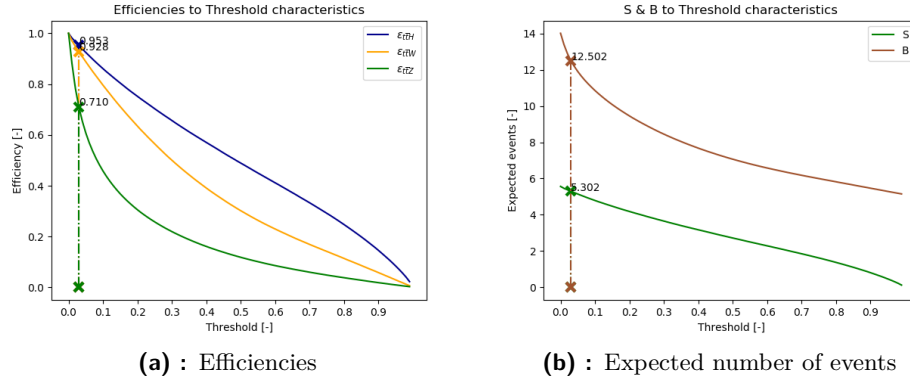


Figure 6.54: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

6.9.4 Working Point

Significance maximization

A threshold value of 0.03 was found to be optimal for maximizing the simplified significance approximation, yielding a result of 1.499. Significance, efficiencies, and S, B parameter plots are shown in Figures 6.53 and 6.54.

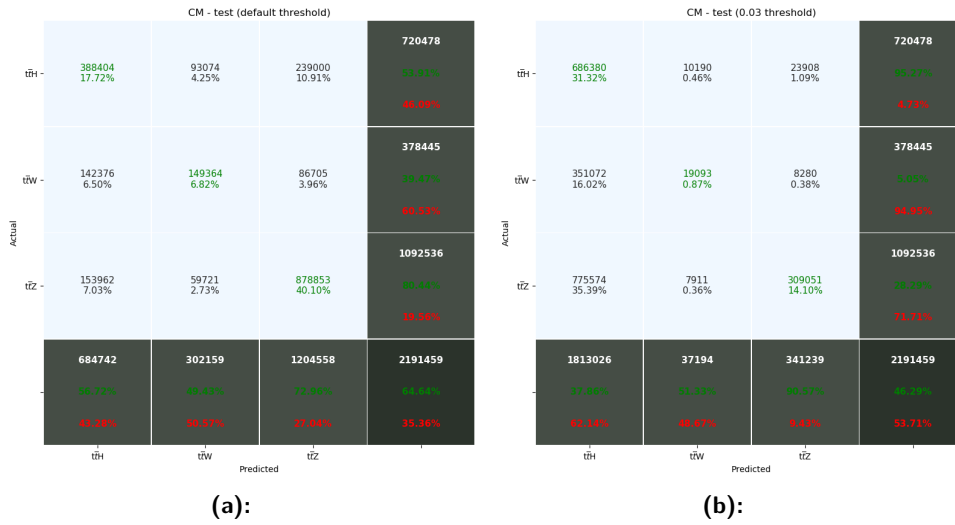


Figure 6.55: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

■ Sensitivity maximization

In the case of the second type of tuning, 0.43 was found to maximize the sensitivity at a given level of the FDR. The maximum value is 0.646 and, together with a confusion matrix, can be found in Figure 6.46.

■ 6.9.5 Conclusion

Requiring the longest time for training, the GBC did not perform spectacularly. Yes, a higher score than the default in Col19b was achieved, however the improvement is very small. The RFC, KNC, and ADA achieved higher scores in shorter learning times. One could argue that the KNC is a lazy learner and that its time of prediction should be compared. But even then, the GBC is more than four-times less efficient. The performance could maybe be improved by further tuning of the parameters which are denoted by an asterisk in Table 6.10. Nevertheless, this would require even more time, which was rather dedicated to other two classification algorithms.

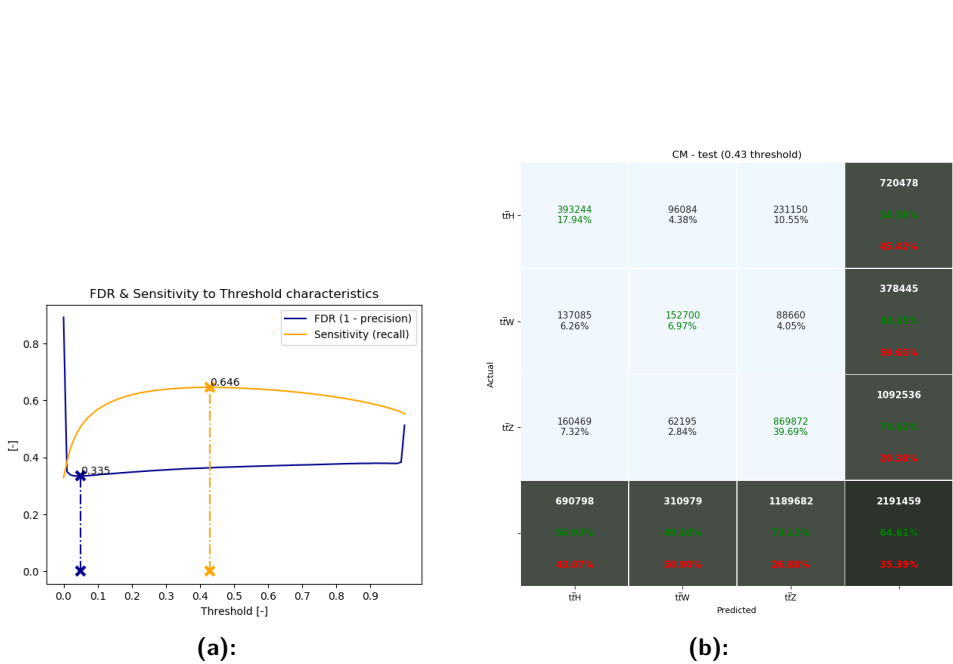


Figure 6.56: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

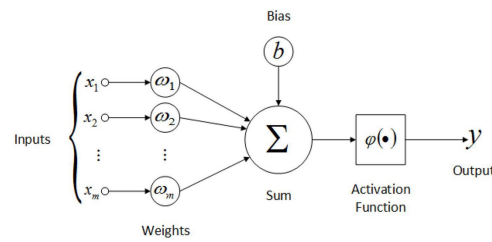


Figure 6.57: Artificial neuron visualization [Roo19]

6.10 Multi-Layer Perceptron Classifier

6.10.1 Introduction

In recent years, a rapid development in Artificial Neural Networks (ANN) has been achieved. Similar to the human brain, the key element of each ANN is the neuron (sometimes called perceptron due to historical reasons and to distinguish it from the real tissue neurons), which is shown in Figure 6.57. At first, it receives inputs (also called activations) from the other neurons, or directly from the data, depending on the position of the neuron in the network. The inputs are multiplied by weights and added together with bias. In this form they are passed to the activation function (also known as non-linearity) which describes the non-linear input-output characteristic and gives the neuron more power to describe the relations in the inputs. Many activation functions exist and can be used. Several were later observed during the tuning process.

As noted, a formation of neurons can be called an ANN. There are many types of them. The one we observe is called a Multi-Layer Perceptron Classifier (MLPC) and is an example of a general feedforward ANN with backpropagation supervised learning technique. There are, in total, n layers and $n-2$ hidden layers inside the network. Two are subtracted due to the first and the last layer being visible. In Figure 6.58 a demonstration of such a network is shown.

The training process of a general MLPC can be summed in three steps as:

- Forward pass
- Loss calculation
- Backward pass

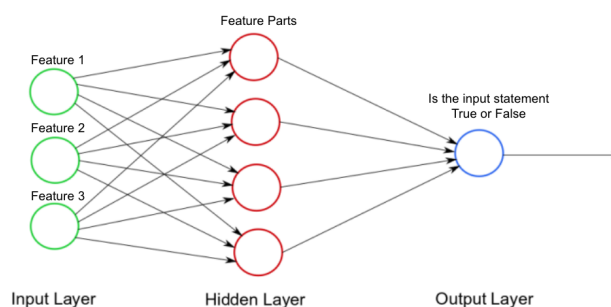


Figure 6.58: Artificial neuron network visualization Sap19

This method is called a Stochastic Gradient Descent (SGD). In the forward pass, the fraction of data is directly introduced at the input and is multiplied by weights and added together with biases at each layer until the output is computed. After this, the total loss of such a network is computed as the difference between the ground truth and the output. This loss is later used in the last step, when it propagates backwards through the network (one layer at a time). During this, weights are updated proportional to the error. This whole process is often referred to as an epoch.

6.10.2 Parameters

For both PCA and Truncated SVD dimension reductions the score and time was compared. Truncated SVD was the winner, with 50 components and 72 components in the case of an observed channel: see Table 6.11 for the results.

In the case of the MLPC, many parameters can be tuned. Some have a smaller impact on the performance, and others a bigger impact, and some even have none. Between these parameters are: 'momentum', 'nesterovs_momentum', 'max_fun', and 'power_t'. None of these were tuned as they are parameters for a different solver than the used 'adam' (Adaptive Movement Estimation - an advanced type of SGD). Several others parameters were also not tuned for many reasons. The 'shuffle', 'random_state', and 'warm_start', were left as default, as they were already set properly. The last parameter which was not tuned was 'validation_fraction'. This parameter is related to early stopping behaviour and denotes how much of the training data should be reserved for validation. After observing early stopping, it was decided that it should be kept turned-off to maximize the performance event at the cost of more computational time. One more parameter which was tuned, but is only active when early stopping is active as well, is denoted by an asterisk in Table 6.12 and its tuned value was not used for later tuning.

algorithm	parameter	search space	best value	best value (channel)
PCA	n_components	linspace(0.1, 0.9, 81)	0.89	0.8
Truncated SVD	n_components	range(2, 83, 1)	52	72

Table 6.11: List of tuned dimension reduction algorithms for MLPC with results

parameter	definition	search space	best value	best value (channel)
hidden_layer_sizes	the number of neurons in the i-th hidden layer	Table 6.13	(100, 100)	(25, 50, 100, 50, 25)
activation	activation function for the hidden layer	['identity', 'logistic', 'tanh', 'relu']	'relu'	'relu'
solver	the solver for weight optimization	['lbfgs', 'sgd', 'adam']	'adam'	'adam'
alpha	L2 penalty (regularization term) parameter	logspace(-12, 0, 13)	1e-6	1e-12
batch_size	size of minibatches for stochastic optimizers	range(100, 401, 25)	200	225
learning_rate	learning rate schedule for weight updates	['constant', 'invscaling', 'adaptive']	'constant'	'constant'
learning_rate_init	the initial learning rate used	logspace(-6, 0, 7)	1e-3	1e-3
max_iter	maximum number of iterations (epochs)	range(100, 401, 25)	250	100
tol	tolerance for the optimization	logspace(-8, -2, 7)	1e-4	1e-4
beta_1	exponential decay rate for estimates of first moment vector in adam	linspace(0.8, 0.99, 20)	0.99	0.85
beta_2	exponential decay rate for estimates of second moment vector in adam	linspace(0.9, 0.999, 10)	0.965	0.91
epsilon	value for numerical stability in adam	logspace(-12, -4, 9)	1e-11	1e-8
early_stopping	whether to use early stopping to terminate training when validation score is not improving	['False', 'True']	'False'	'False'
n_iter_no_change	maximum number of epochs to not meet tol improvement (requires early_stopping)	range(10, 100, 10)	40*	80*

Table 6.12: List of tuned parameters of MLPC with results

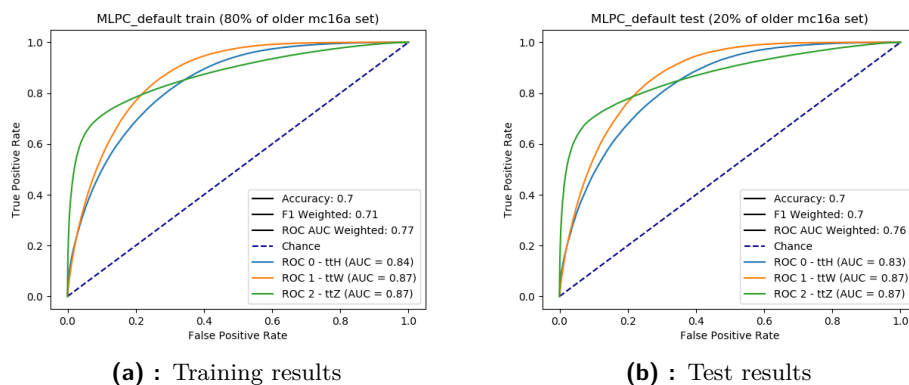
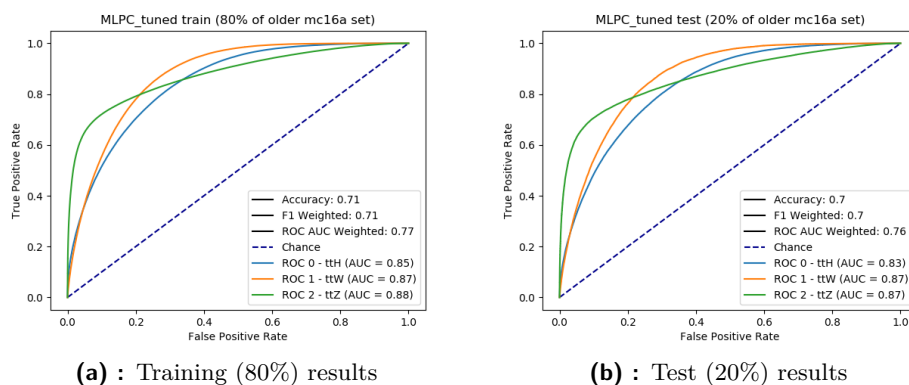
The parameter 'hidden_layer_sizes' turned out to be very complicated to tune. As a value, a 2D array of numbers can be passed. Several values were randomly tested to acknowledge the direction of further changes. Surprisingly, the performance did not increase with the addition of more hidden layers. The best results were achieved with 2 layers, both containing 100 perceptrons.

Training results for the classifier with the default and tuned parameters are shown in Figures 6.59 and 6.60 respectively.

Channel

In the case of channel, Table 6.12 contains the tuning results as well. Figure 6.61 contains characteristics for the model with the default parameters, and Figure 6.62 for the model with the tuned parameters. And in Figure 6.63, the performance of the classifier trained on 80% of full-data and tested on channel-data is shown.

tuning number	hidden_layer_sizes
0	(100,)
1	(100, 100,)
2	(100, 200, 100,)
3	(100, 200, 200, 100,)
4	(100, 200, 300, 200, 100,)
5	(100,)
6	(50, 100,)
7	(50, 100, 50,)
8	(50, 100, 100, 50,)
9	(25, 50, 100, 50, 25,)
10	(200,)
11	(200, 50,)
12	(50, 200,)
13	(200, 100,)
14	(100, 200,)
15	(300,)
16	(400,)

Table 6.13: List of tuned values for 'hidden_layer_sizes' parameter**Figure 6.59:** Performance of MLPC with default parameters**Figure 6.60:** Performance of MLPC with tuned parameters

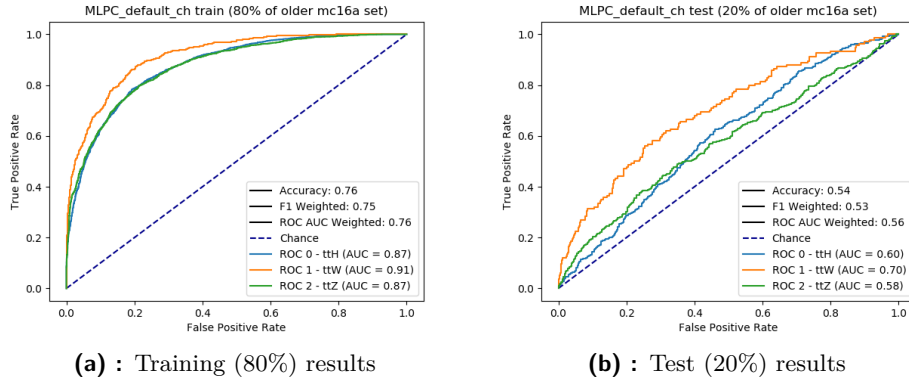


Figure 6.61: Performance of MLPC (channel) with default parameters

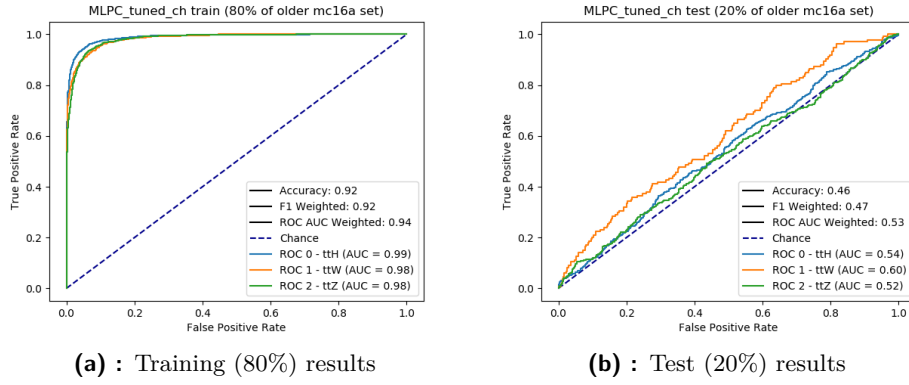


Figure 6.62: Performance of MLPC (channel) with tuned parameters

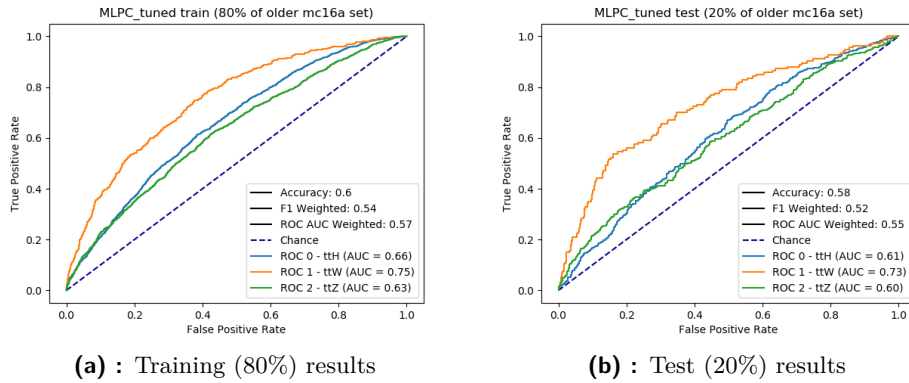


Figure 6.63: Performance of MLPC with tuned parameters, trained on 80% of full-data and tested on 80% and 20% of channel-data

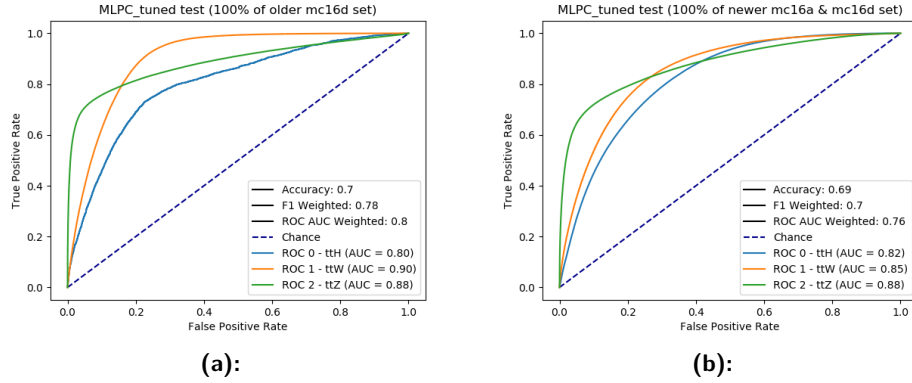


Figure 6.64: Performance of MLPC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

6.10.3 Additional tests

In the case of additional tests, shown in Figure 6.64, once again no large deviations were observed compared to the results on the default test set (Figure 6.60). For the newer set containing both mc16a and mc16d, the results were the same. For the older set with only mc16d, little improvement in $t\bar{t}W$ classification was observed.

6.10.4 Working Point

Significance maximization

The optimal value of the threshold, in order to maximize the significance, was found to be 0.16. This value yielded a score of 1.586. Due to this, the MLPC became the new leading classifiers with the best score. Significance, efficiencies, and S, B parameter plots are shown in Figures 6.65 and 6.66.

Sensitivity maximization

For the second type of tuning, a threshold of value of 0.4 was found to be optimal and maximized the sensitivity to the value of 0.692. In Figure 6.68 both tuning plot and confusion matrix are shown.

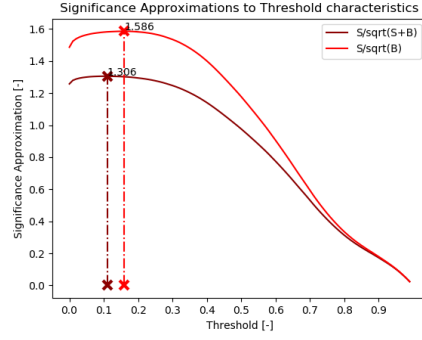


Figure 6.65: Dependence of significance on the threshold, vertical lines denote maximums

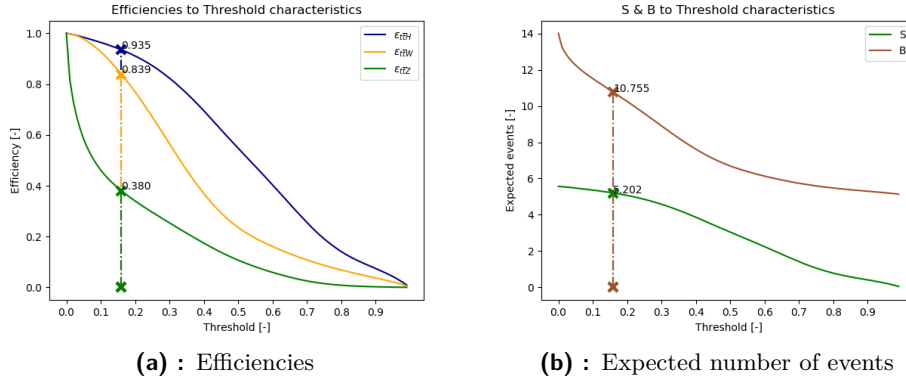


Figure 6.66: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

6.10.5 Conclusion

The MLPC was shown to be the most effective classifier of all which were tested. As an elementary ANN it served as a good demonstration of the power these tools have and proved that it requires more complex algorithms to achieve higher classification scores on our data. In the future, with enough time, other types of ANN could also be tested: such as a Convolutional Neural Network (CNN), a Deep Neural Network (DNN), a Recurrent Neural Network (RNN), and many others.

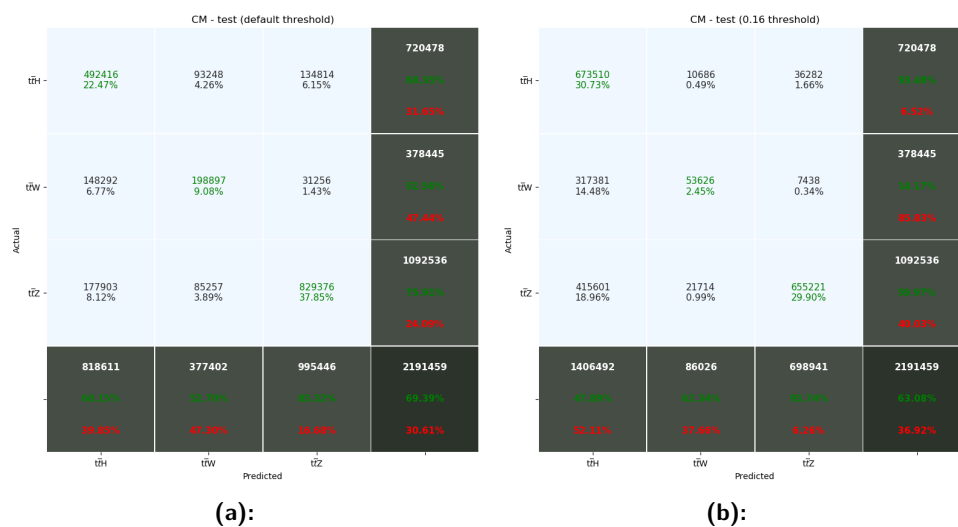


Figure 6.67: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

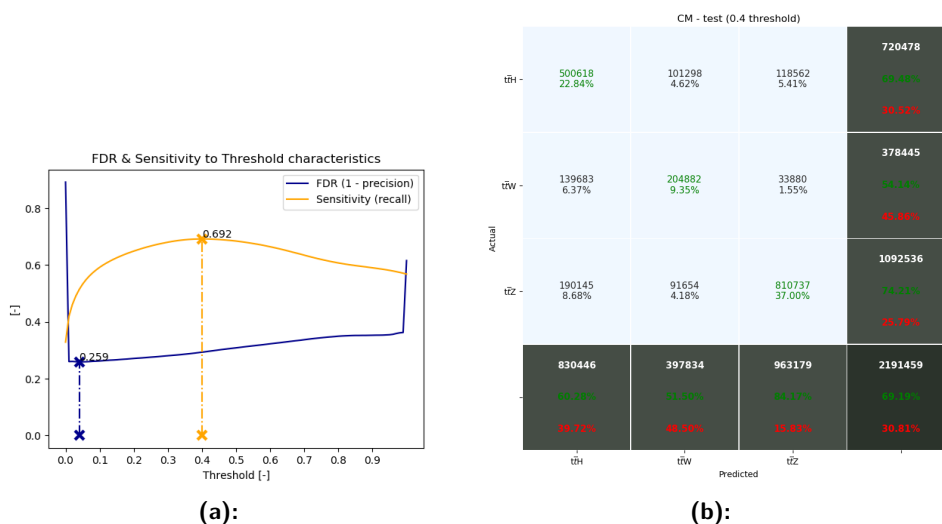


Figure 6.68: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

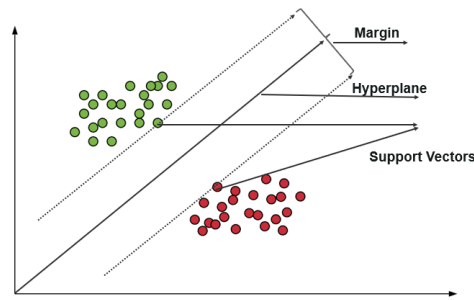


Figure 6.69: Binary classification problem solved by Support Vector Machine

Was19

6.11 Support Vector Classifier

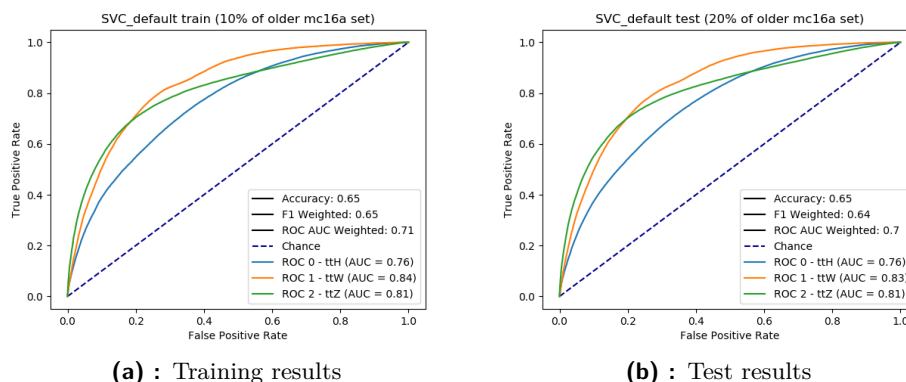
6.11.1 Introduction

The Support Vector Machine (SVM), or the Support Vector Classifier (SVC) in the case of classification, is a very powerful and popular machine learning algorithm. However, it can only be used for supervised learning. A key element of every SVM, as the name denotes, is the support vectors which are the datapoints (events) closest to the separation hyperplane. The hyperplane itself is defined using these vectors and is taken as a decision boundary between the classes. A margin is introduced to fine-tune this boundary. The centering of the hyperplane between the closest datapoints of each class is desired. A good margin not only increases the separation power, it also reduces the number of wrong classifications and is often called the Maximum Marginal Hyperplane (MMH). In Figure 6.69 a binary classification 2D problem solution by the SVM, with all previously mentioned terms, is demonstrated.

Another important element which allows the rapid developments of the SVM is a kernel trick, in which the kernel function maps low dimensional space and transforms it into high dimensional space. This allows the SVM to transform non-separable problems into separable problems and solve them very efficiently. At a later stage, the most famous kernel functions (such as linear, poly, rbf, and sigmoid) were tuned.

However, there is a large problem with the SVM: the length of training time. Training it is a quadratic programming problem and it was proven in Cha06 to be equal to $O(\max(n, d)\min(n, d)^2)$, where n is the number of datapoints (events) and d is the number of dimensions (features).

parameter	definition	search space	best value	best value (channel)
kernel	kernel type used in the algorithm	['linear', 'poly', 'rbf', 'sigmoid']	'rbf'	
C	regularization parameter	[logspace(-5, 0, 6), range(1, 41, 1)]	7	
gamma	kernel coefficient	logspace(-20, 0, 21)	1e-2	
tol	tolerance for stopping criterion	logspace(-6, 0, 7)	1e-2	
shrinking	whether to use the shrinking heuristic	[False, True]	True*	
break_ties	whether 'predict' method will break ties according to the confidence values of decision function	[False, True]	False*	

Table 6.14: List of tuned parameters of SVC with results**(a) :** Training results**(b) :** Test results**Figure 6.70:** Performance of SVC with default parameters

To reduce the training time but still present a large amount of training data, a new wrapper called a bagging classifier ([sklearn.ensemble.BaggingClassifier](#)) was introduced. Similar to any other ensemble classifier, it takes a specified number of weak classifiers and trains each of them on a fraction of the data, usually adequately divided. This allows the parallelization of training which therefore allows the ensemble to learn on more data in the training time of an individual classifier. The bagging classifier itself does not provide any supervision over the ensemble and only collects final votes from it. Such an approach allows an easy tuning of the underlying classifiers. Experimentally, the number of SVC classifiers used inside the bagging classifier was set to 10.

Later the training data was additionally shrunk to contain up to approximately 100,000 events (10% of the older mc16a data). This was mainly done to increase training speed due to the short time limit left for the SVC tuning task. Other dimension reductions, such as PCA or Truncated SVD were discarded as $n \gg d$.

6.11.2 Parameters

For SVC implementation in the scikit-learn library there are just a few parameters which require tuning. However, their importance is great. The 'kernel' parameter sets the kernel function. To be able to tell how much

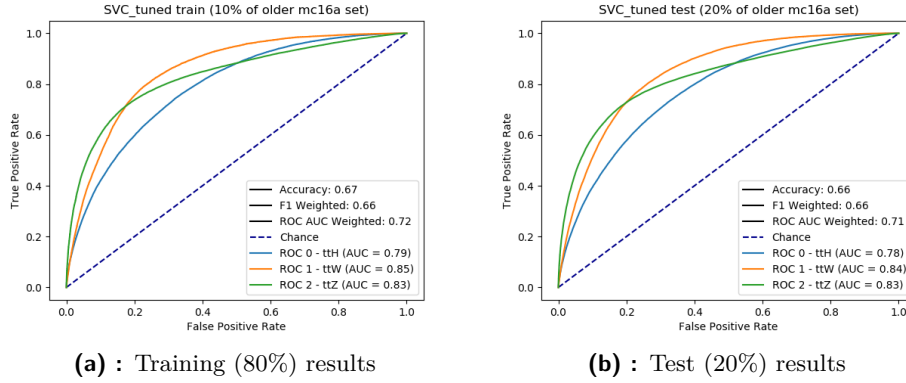
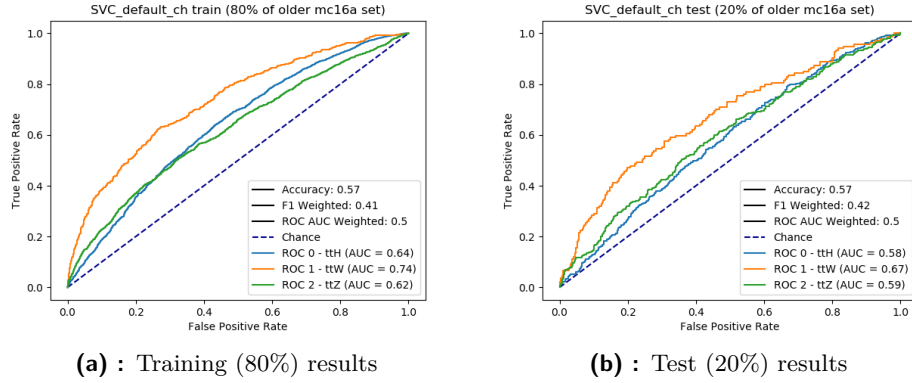
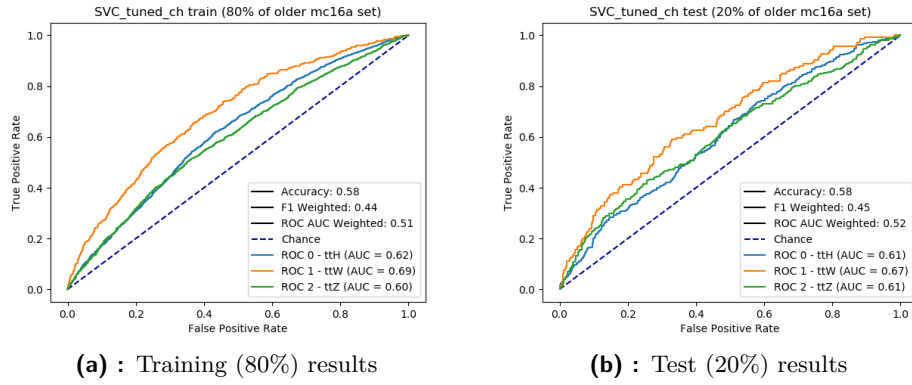


Figure 6.71: Performance of SVC with tuned parameters

misclassification is penalized, a regularization parameter 'C' can be used. And to describe the reach of the influences of a single datapoint, parameter 'gamma' can be tuned. The 'tol' parameter (also called cost) describes the tolerance of wrong classification. The behaviour of the predict method in the case of 'ovr' (one vs the rest) classification can be set by specifying the 'break_ties' parameter. If true, the predict method will break ties according to the confidence values of the decision function. The shrinking heuristic which speeds up the optimization can be turned on or off by specifying the parameter 'shrinking'. This sometimes helped, and sometimes did not, and therefore this parameter was also tuned.

Some parameters were directly changed without tuning as the optimal value was known in advance, including the 'class_weight' which was set to apriori probabilities (5.2). The 'probability' parameter was set to True, since WP tuning requires posteriori probabilities. Lastly, several parameters were left as their default. The first, 'decision_function_shape' allows us to specify how the multiclass classification will be handled. The default value of 'ovr' was what we desired. Two other parameters, 'degree' and 'coef0', were related to a different kernel function than the Radial Basis Function (RBF), which achieved the highest score of all kernel functions. The last parameter not tuned was the 'random_state', which, as usual, was left to be set to a random function of the NumPy library.

The training and test results for the SVC with default parameters are shown in Figure 6.70. Table 6.14 shows the results of the tuning and in Figure 6.71, the results of the SVC with tuned parameters can be seen.

**Figure 6.72:** Performance of SVC (channel) with default parameters**Figure 6.73:** Performance of SVC (channel) with tuned parameters

Channel

In the case of the channel, the same data as the other classifiers was used, as only 6,729 events were present and reducing this number was not necessary. Table 6.14 contains the results and in Figures 6.72 and 6.73 the scores and ROC for the default and tuned channel models are shown respectively. Figure 6.74 shows the evaluation of the performance of the classifier trained on 10% of the full-data and tested on the channel-data.

6.11.3 Additional tests

For two additional testing sets: the older mc16d and the newer mc16a & mc16d, improvements were observed in both cases. The results are shown in

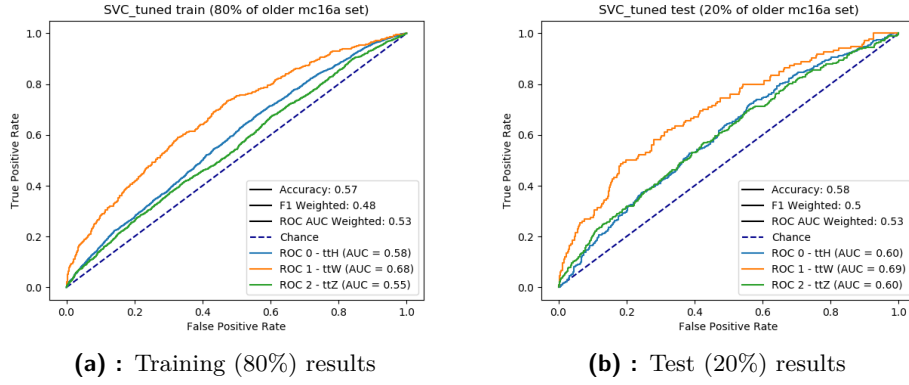


Figure 6.74: Performance of SVC with tuned parameters, trained on 10% of full-data and tested on 80% and 20% of channel-data

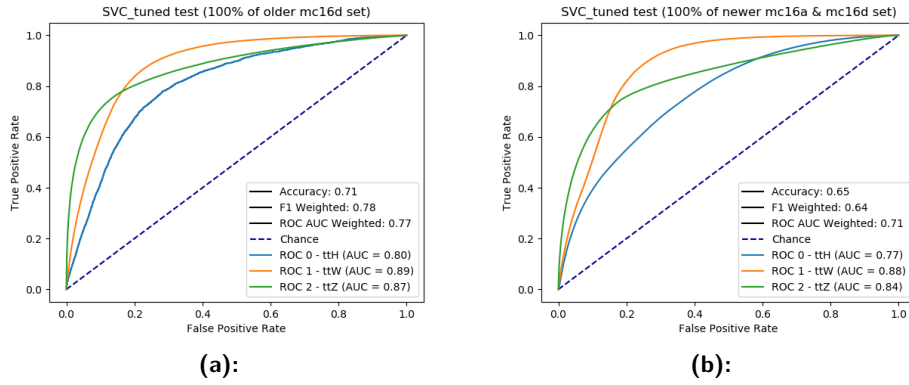


Figure 6.75: Performance of SVC with tuned parameters, tested on a) older mc16d b) newer mc16a & mc16d data

Figure 6.75 and were compared with results from Figure 6.71 (b)).

6.11.4 Working Point

Significance maximization

A threshold value of 0.14 was found to maximize the simplified significance to a value of 1.524, which is a good score. However, there are two other classifiers, RFC and MLPC, which performed better. For the tuning please refer to Figures 6.65 and 6.66.

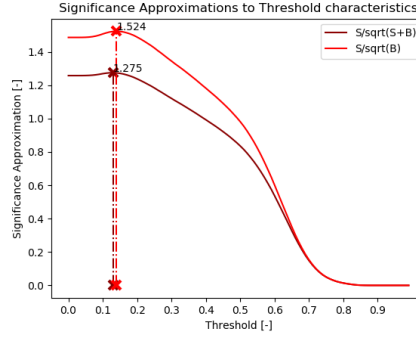


Figure 6.76: Dependence of significance on the threshold, vertical lines denote maximums

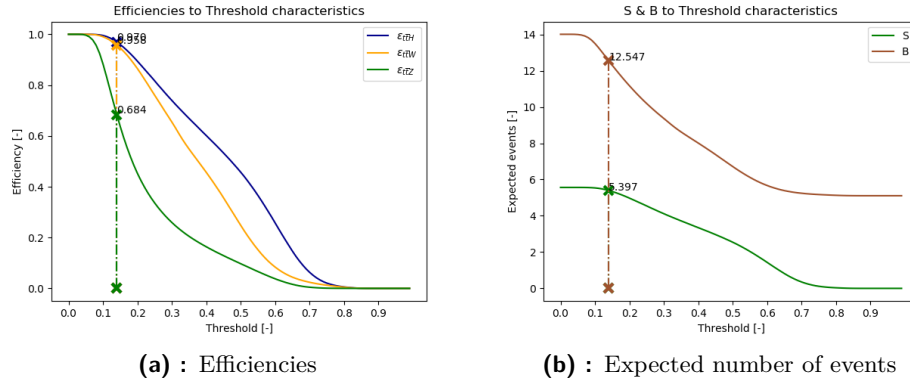


Figure 6.77: Dependences of a) the efficiencies, b) the expected number of events on the threshold, vertical lines denote values for found threshold

Sensitivity maximization

In the second type of tuning, a threshold value of 0.41 was found to maximize the sensitivity to 0.655 at a given FDR. The results are shown in Figure 6.79.

6.11.5 Conclusion

The SVC performed fairly: only two other classifiers achieved higher scores. The performance could probably be boosted by adding more classifiers inside an ensemble or training on data with more events. However, this would mean very long training times. In the case of the SVC, the number of events in the full-data training set was reduced. After this reduction, the data was roughly equal to the channel-data multiplied by ten. Observing results for

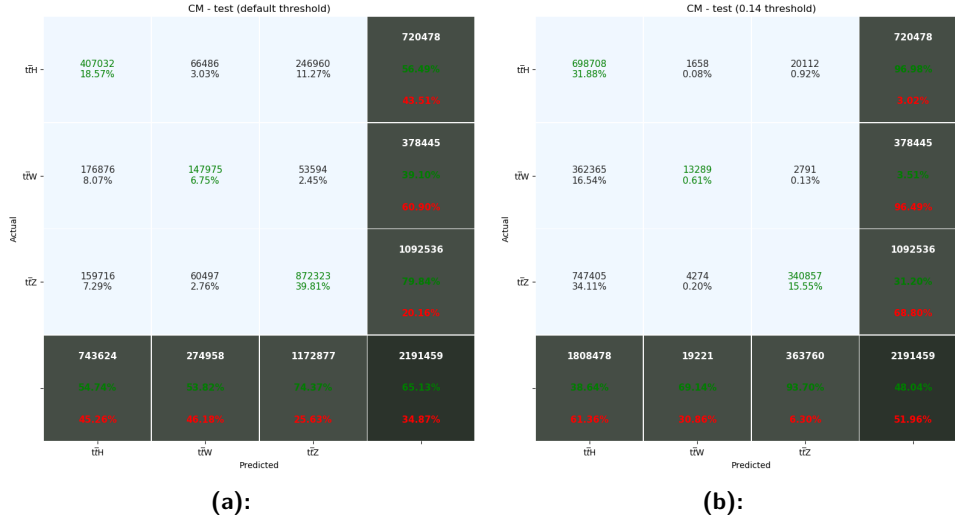


Figure 6.78: Confusion matrices for a) default decision and b) significance maximization decision function for the tuned classifier performing on the newer data

both reduced full-data and channel-data trained models, it is clear that the number of events is not the cause of a channel not performing well. It is probably the 'is2LSS1Tau' cut itself, dropping important information in the channel-data.

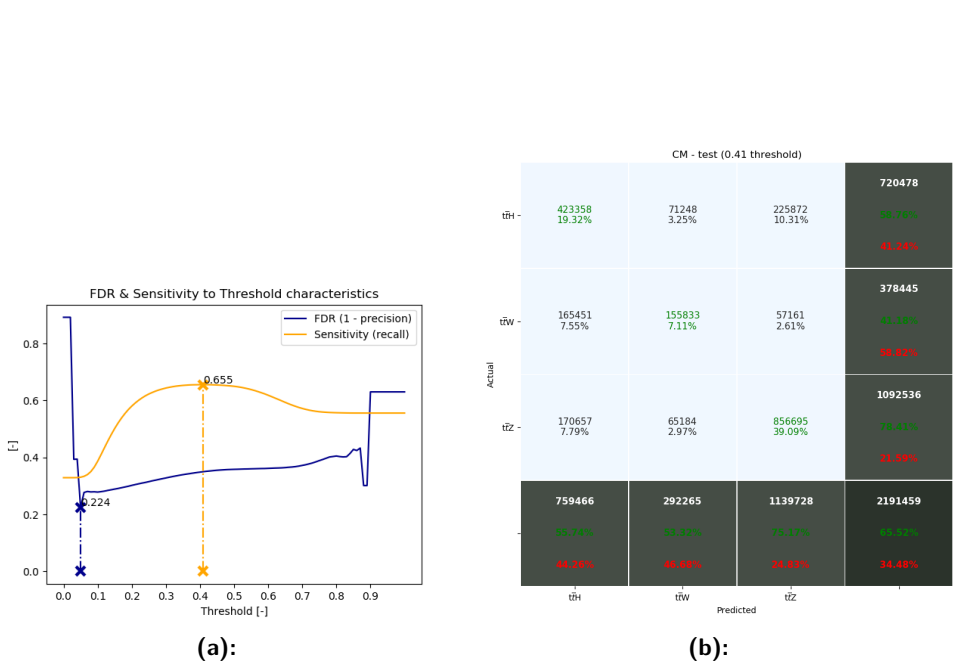


Figure 6.79: a) parameters dependence on the threshold and b) confusion matrix for the WP maximizing sensitivity for a given value of FDR for the tuned classifier performing on the newer data

Chapter 7

Real data

During the final weeks, real data was obtained to evaluate the performance of the tuned classifiers. Table [B.6](#) in the Appendices contains more information about used files. Appendix [C.1.2](#) can be viewed for histograms of the data and can be compared with Appendix [C.1.1](#), where the histograms of the training data are shown.

All classifiers were tested and their results compared. In Figure [7.1](#) the predictions of each classifier are listed in the form of 1D heat maps, where light blue denotes the $t\bar{t}H$, orange $t\bar{t}W$, and green $t\bar{t}Z$ classes. Figure [7.2](#) shows the individual results in the form of bar graphs.

Most classifiers found a very small amount of background. Only the GBC was more sceptical and classified many events as background. Whether or not this is correct can be decided using the classifier with the highest significance score for the test data, the MLPC, as a reference to evaluate the remaining classifiers. The MLPC detected only a few background events, and thus the GBC results are wrong. A second determining method is to use those classifiers which have approximately the same predictions. From Figure [7.1](#), two groups can be seen: one which predicts all events belonging to a signal (KNC, GNB, and SVC); and the other which has some identical predictions for background (RFC, ADA, and MLPC). The GBC does not share prediction results with any of these classifiers, and therefore its results are wrong.

No more experiments were carried out with real-data. Their purpose was mainly to demonstrate the capabilities of individual classifiers and to observe whether they will make the same predictions. The lack of ground-truth labels

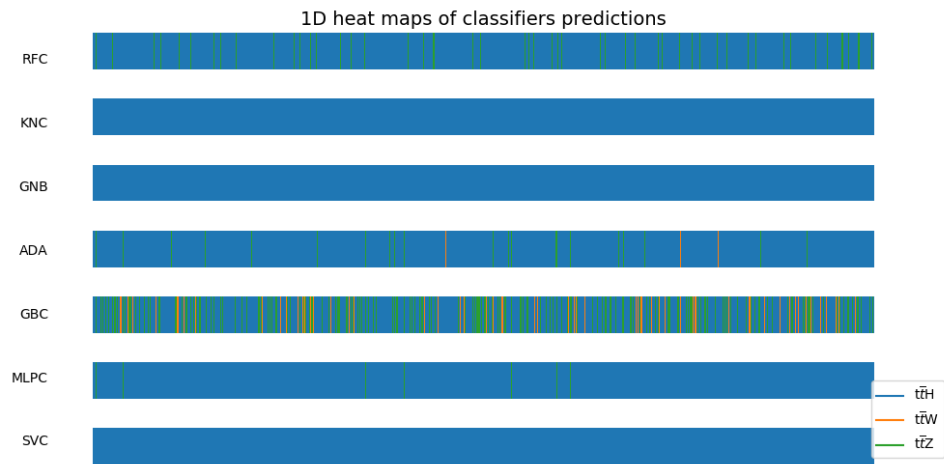


Figure 7.1: Overview of individual real-data predictions

prevents any score computation or ROC plot evaluation.

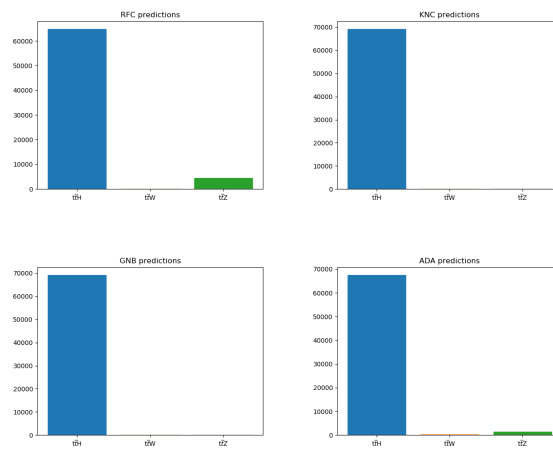


Figure 7.2: Individual real-data predictions

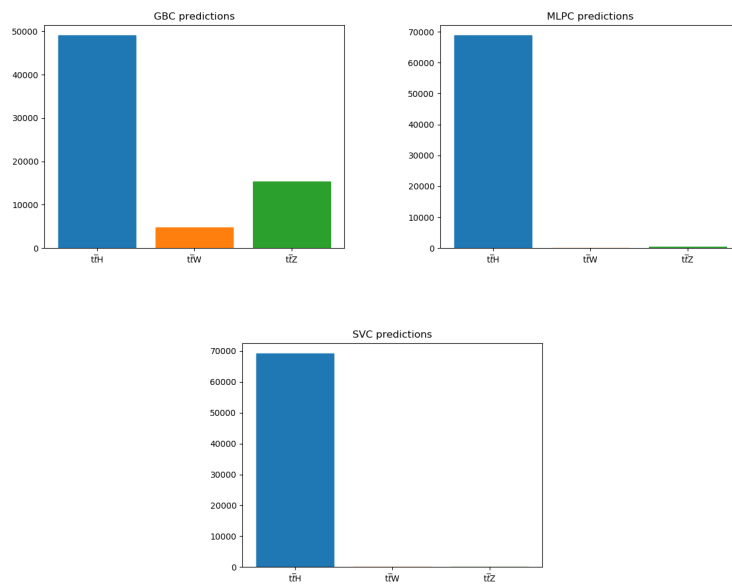


Figure 7.2: Individual real-data predictions

Chapter 8

Additional Changes

8.1 Cheating

Several additional changes were made in the final weeks, the first of which was triggered by the discovery of 'cheating'. As described at the beginning of the thesis, such a term is used for features which are only available for simulation data or, more precisely, which are related only to simulations. Namely: 'lep_isPrompt_0', 'lep_isPrompt_1', 'lep_isPrompt_2', 'lep_isQMisID_0', 'lep_isQMisID_1', and 'lep_isQMisID_2'. They were all removed in conversion scripts and new files were generated, with only 77 features listed in Table B.2. Any converted files containing the reduced number of features are denoted by the suffix 'new'.

8.2 New data training

Another change was introduced into the training process. Before the change, the older mc16a data set was used, but it was explained later that any newer sets of simulated data would surpass the previous one. Therefore, it is desirable both to train and evaluate on the newest data set as the older data would be contained along with the new. The use of both mc16a and mc16d sets was also recommended by my supervisors. All classifiers were re-trained on 80% of the newer data and tested on 20% respectively.

The working point was retuned based on this new test split. However, the parameters of the classifiers were not retuned due to the short time limit. In the following subsections, the ROC plots, together with the significance tuning and confusion matrices are shown for each classifier. The significance score was improved in many cases. However, the testing and computation was newly done on only 20% of newer data, instead of the previous 100%. This was due to the classifiers being trained on this set as well, taking the remaining 80% for training.

In the previous experiments, full-data models significantly outperformed channel-data models during training, and so in this case the training and tests were performed only on full-datasets (the newer mc16a & mc16d set).

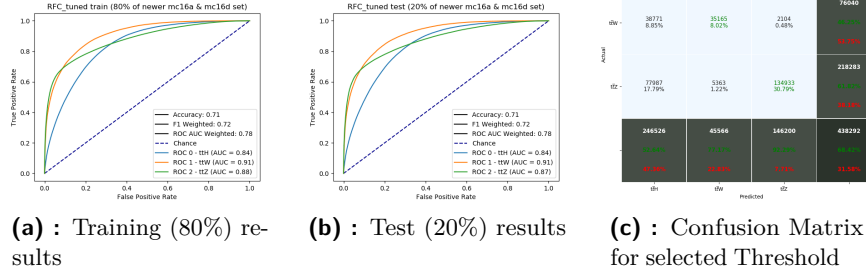


Figure 8.1: Performance of newly trained RFC

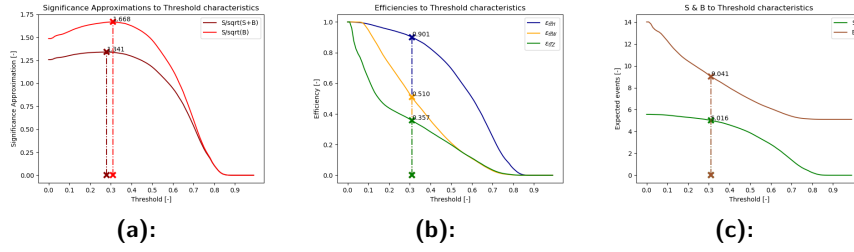


Figure 8.2: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.1 RFC

In the case of the RFC, the score was drastically improved: from 1.576 (Figure 6.11) to 1.668. This was a result of both background efficiencies being reduced while signal efficiency still being high.

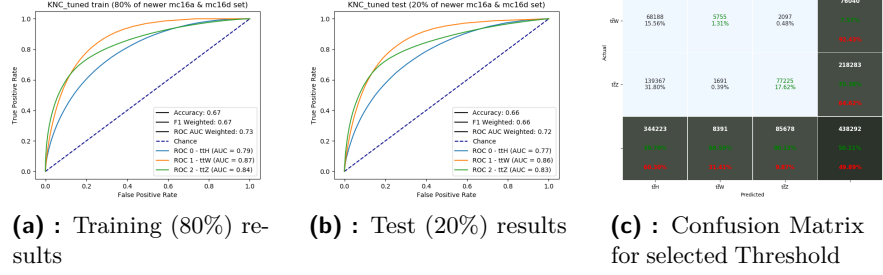


Figure 8.3: Performance of newly trained KNC

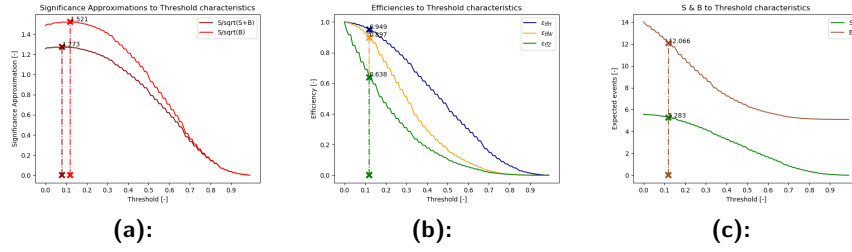


Figure 8.4: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.2 KNC

A small improvement was also observed for the KNC. From 1.51 (Figure 6.22), the significance increased to 1.521. The efficiency of the $t\bar{t}W$ background was still too high to achieve higher scores. However, after re-training, for an optimal working point which maximizes the significance, the efficiency of the background was no longer greater than that of the signal.

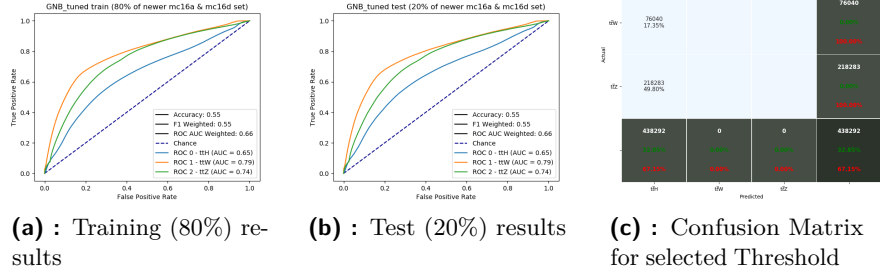


Figure 8.5: Performance of newly trained GNB

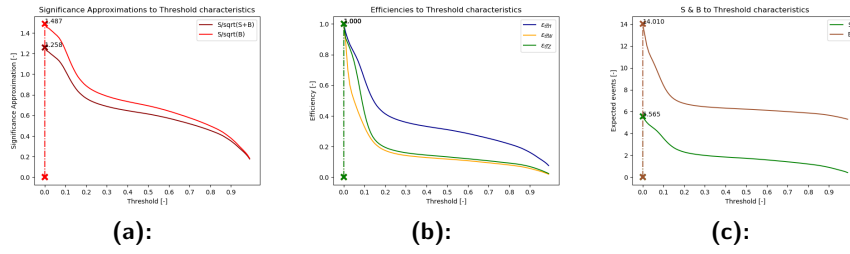


Figure 8.6: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.3 GNB

In the case of the GNB, no change was observed. This elementary classification method again classified every single event into the $t\bar{t}H$ class.

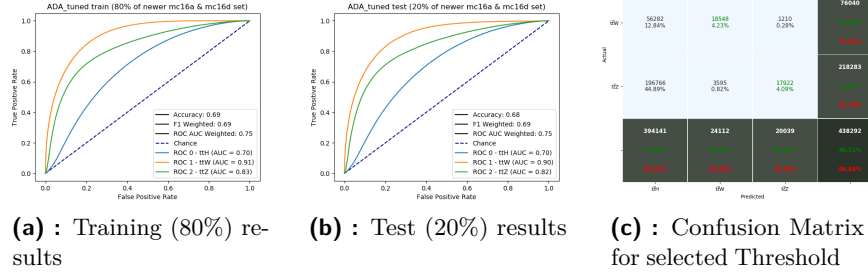


Figure 8.7: Performance of newly trained ADA

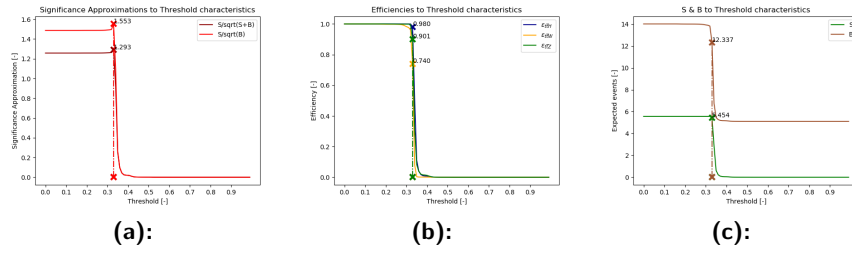


Figure 8.8: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.4 ADA

A small increase from 1.508 (Figure 6.43) to 1.553 'was observed for the ADA, although the original rectangular wave shape is retained.

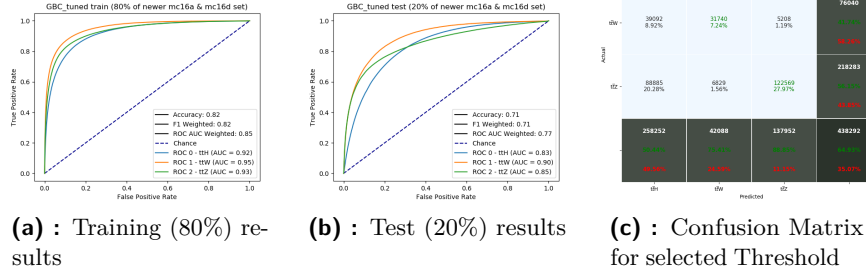


Figure 8.9: Performance of newly trained GBC

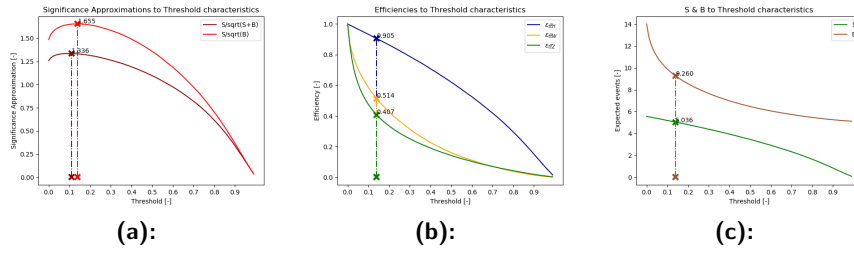


Figure 8.10: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.5 GBC

One of the biggest score increases was observed for the GBC. From its original 1.499 (Figure 6.53) it leapt to 1.655. The reasons for this improvement are again due to the suppression of background efficiencies while keeping a high efficiency for signal.

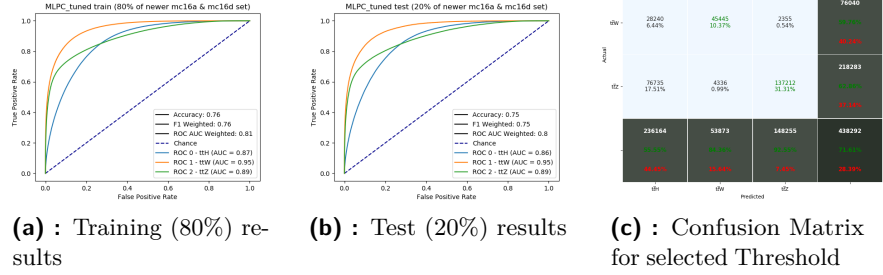


Figure 8.11: Performance of newly trained MLPC

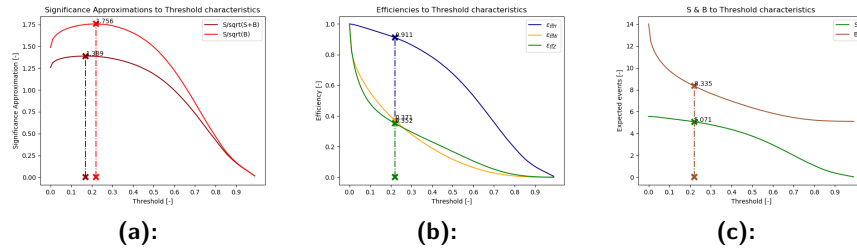


Figure 8.12: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

8.2.6 MLPC

The biggest score increase was achieved for the MLPC. From a previously obtained 1.586 (Figure 6.65) a jump to an astonishing 1.756 was made. Both background efficiencies were reduced to below 0.4, while keeping the signal efficiency higher than 0.9. This was a new record and the best performance achieved.

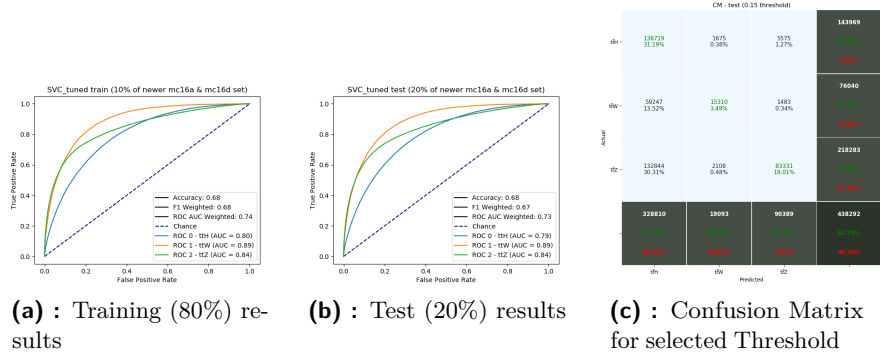


Figure 8.13: Performance of newly trained SVC

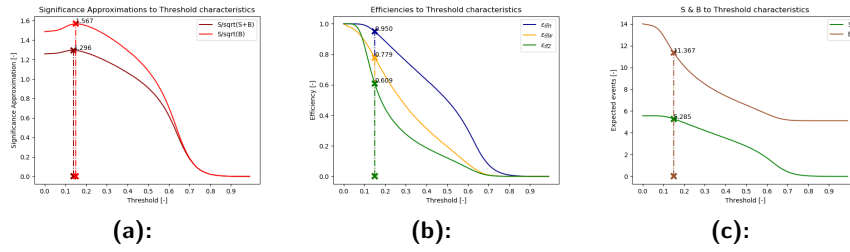


Figure 8.14: Dependences of a) the significance, b) the efficiencies, c) the expected number of events on the threshold, vertical lines denote values for found threshold

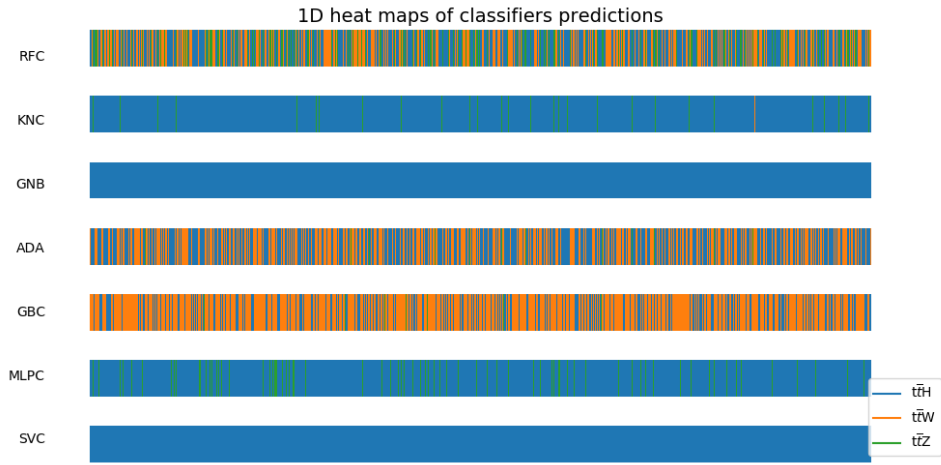
8.2.7 SVC

A small increase in the significance score from 1.524 (Figure 6.76) to 1.567 was also observed for the SVC. The main reason for such a small improvement is likely due to the amount of training data, which in the case of the SVC was again reduced to only 10%. Increasing the number of training events up to 80% to be equal with other classifiers would certainly produce better results. However, as mentioned earlier, the length of time taken by the SVC is dependent on the amount of data as $O(\max(n, d)\min(n, d)^2)$, where n is the number of events and d is the number of dimensions.

8.2.8 Summary

In Table 8.1 the old and new significance scores are listed for each of the classifiers. An improvement was achieved in almost all cases except the GNB. The highest was obtained for the MLPC, and the lowest for the KNC.

classifier	old score	new score	difference
RFC	1.576	1.668	0.092
KNC	1.51	1.521	0.011
GNB	1.487	1.487	0
ADA	1.508	1.553	0.045
GBC	1.499	1.655	0.156
MLPC	1.586	1.756	0.17
SVC	1.524	1.567	0.043

Table 8.1: List of classifiers and their significance scores**Figure 8.15:** Overview of individual real-data predictions of newly trained classifiers

8.2.9 Real data

With newly trained classifiers, the performance on the real data was observed again. An increase in the number of $t\bar{t}W$ class predictions was detected and is shown in Figures 8.15 and 8.16, although the MLPC and the KNC predictions were almost identical with previously obtained predictions for the old models (Figure 7.1, 7.2). Using the MLPC as a reference to evaluate the remaining classifiers, it can be seen that the results of the RFC, ADA, and GBC classifiers are wrong. Looking at the new figures, three groups were formed: one which predicts that all events belong to signal (GNB and SVC); one which predicts only a little background would be present (MLPC and KNC); and a third which predicts a lot of background in the data (RFC, ADA, and GBC). There is no way to determine which classifier (or group) performed the best on real data.

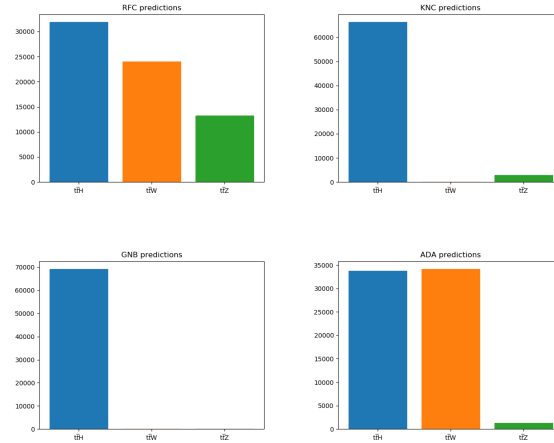


Figure 8.16: Individual real-data predictions of newly trained classifiers

8.3 Feature importances

The feature importances were also updated. In Figure 6.9, the newly discovered cheating features were present, and so its veracity is questionable. In Figure 8.17, the new importances are shown.

ID	name	definition
13	best_Z_Mll	smallest mass difference between Z mass and invariant mass of lepton pair
5	Mll01	invariant mass of lepton pair (leading and subleading)
11	Mll012	invariant mass of leptons (leading, subleading, and sub-subleading)
4	MV2c10_70_EventWeight	tau isolation
6	Mll02	invariant mass of leptons (leading and sub-subleading)
52	nJets_OR	number of jets
8	MLL12	
54	nJets_OR_T	number of jets after overlap removal
0	DRll01	special distance of jet and lepton
55	nJets_OR_T_MV2c10_70	
31	lep_Pt_2	transverse momentum of sub-subleading lepton
53	nJets_OR_MV2c10_70	
71	total_charge	
50	lep_promptLeptonVeto_TagWeight_0	prompt lepton veto
75	HT_lep	sum of transverse momentum of all leptons

Table 8.2: List of the most important features with descriptions for newly trained RFC

Compared to Figure 6.9, no large change was observed. Feature ID 13 (best_Z_Mll) once again achieved the highest importance. Between the first five, a new feature with ID 4 is seen (MV2c10_70_EventWeight) which in the previous test had a much smaller importance. Due to the additional removal of several simulation-only related features, the numbers shifted a little. Table B.2 can be referred to for updated IDs. In Table 8.2, the most important features are again listed with definitions.

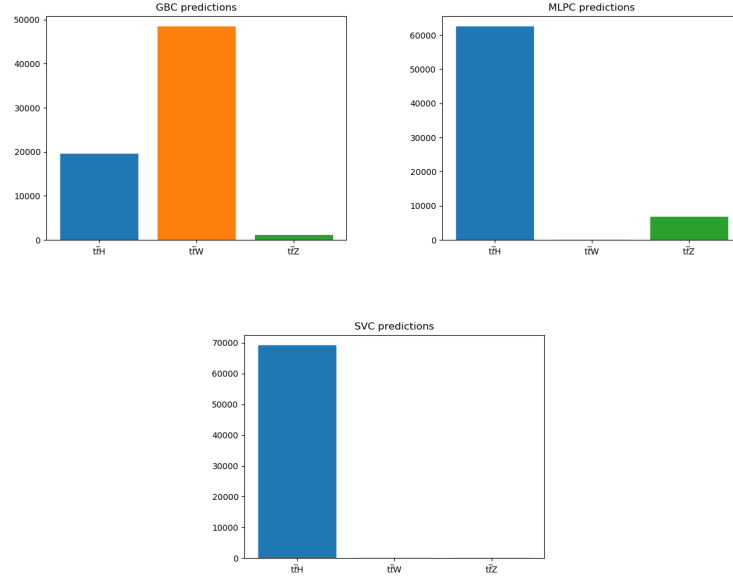


Figure 8.16: Individual real-data predictions of newly trained classifiers

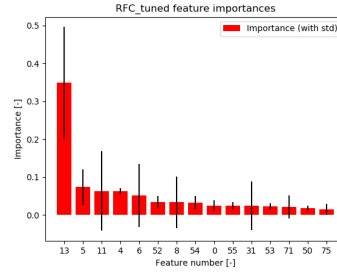


Figure 8.17: Feature importances of newly trained RFC

8.4 $t\bar{t}Z$ weighting

Another action which took place during the final weeks was a presentation regarding our progress at a weekly CERN $t\bar{t}H$ collaboration meeting. From this meeting, very valuable feedback was received. There was a suggestion to use individual weights for the $t\bar{t}Z$ backgrounds, not an overall weight as we currently do. The weight is defined as the ratio of the total number of $t\bar{t}Z$ events divided by the total number of expected $t\bar{t}Z$ events. The suggestion was regarding using different weights for each of the $t\bar{t}Z$ decay mode: in our case there are five in total. There should be a separate weight for each of them. This last suggestion was not implemented due both to the short time limit and not having enough information about the individual $t\bar{t}Z$ decay modes.



Chapter 9

Conclusion

The original goal was to test XGBoost, TensorFlow, and other libraries alongside Scikit-learn. However, the tuning process took a significant portion of the allocated time for this project, and therefore only one library with seven classifiers was demonstrated.

As mentioned earlier, Automatic Machine Learning (AML) was proposed to fine-tune the parameters of the classifiers, after the parameter search space is reduced. Sadly there was no time for the implementation of AML libraries, such as Auto-sklearn or Neuraxle. Keeping in mind the training and predict times of the classifiers, such an automated process would require a vast number of computation machines. It is questionable whether this would be the right approach, or whether trying other classifiers (such as Convolutional Neural Networks (CNN)) would be a more lucrative option.

However, despite being lighter than the original plan, this thesis brought forward some new discoveries. The major one being the effectiveness (or, rather, the ineffectiveness) of the 'is2LSS1Tau' cut for obtaining channel-data. Models trained on the full-data significantly surpassed the channel trained model. Another interesting discovery was related to efficiencies: as shown in the case of the KNC and its WP tuning, the best significance was achieved with signal efficiency being lower than the background. Lastly an evaluation of several classifiers revealed the most powerful for a given problem. Further work with the RFC and the MLPC is recommended, as their performances were astonishing with regards to both the score and time.

To conclude I would like to say a few words about the thesis in general. The

time given for this task was roughly ten months, the first few of which were dedicated to understanding the basics of particle physics. During this time, many articles were read and many meetings with the supervisors were held. The output of the first five months were several reports, demonstrating the successful conversion from root format as well as the first ML attempts. In the second half of the devoted time, classifiers were studied in detail and the most prospective were selected for training. At that moment, the dimensionality of the data became a problem. Decomposition methods were studied and two of them (PCA and Truncated SVD) were further observed for several classifiers with dimensionality problems.

Also, during this time, strict measures were accepted by nearly the entire world to prevent a newly discovered virus from spreading. Personal meetings became impossible, and were partially replaced by online contact although the frequency significantly decreased. A major part of the blame for the cancellation of many of the original goals for this thesis can be placed on my own mental health and productivity during this crisis. For a person having many relatives on the other side of the world, a country-wide lockdown with no lift-date was a big uncertainty in my life. Being the only person working on this thesis, there was no additional push from collaborators, and so my productivity decreased and the time limit grew nearer. For me, this was a time of discovery regarding my personality and inability to work alone: during my student life I have had many projects and internships which ended well. I now know that this was due, in part, to the pressure and accountability from colleagues, working in great teams, and the knowledge that I was not alone. I hope that this thesis can serve as a basis for future research and that these new discoveries will prove correct and important.

Downloaded from <https://www.cambridge.org/core>. University of Cambridge, on 01 Jun 2018 at 10:00:00, subject to the Cambridge Core terms of use, available at <https://www.cambridge.org/core/terms>. <https://doi.org/10.1017/9781315336435.008>

Appendix A

Computations

A.1 Significance computation example

The following classification results in the form of a Confusion Matrix in Figure A.1 as a result of sensitivity maximization (RFC). The efficiencies can be computed (using Eq. 3.7 and 3.8) as:

$$\epsilon_{t\bar{t}H} = \frac{TP}{TP + \sum_{i \in (t\bar{t}W, t\bar{t}Z)} FN_i} = \frac{CM[0][0]}{\sum CM[0][:]} = \frac{667366}{667366 + 14366 + 38746} \approx 0.9263 \quad (\text{A.1})$$

$$\epsilon_{t\bar{t}W} = \frac{FP_{t\bar{t}W}}{FP_{t\bar{t}W} + \sum_{j \in (t\bar{t}W, t\bar{t}Z)} TN_{t\bar{t}W,j}} = \frac{CM[1][0]}{\sum CM[1][:]} = \frac{315213}{315213 + 59220 + 4012} \approx 0.8329 \quad (\text{A.2})$$

$$\epsilon_{t\bar{t}Z} = \frac{FP_{t\bar{t}Z}}{FP_{t\bar{t}Z} + \sum_{j \in (t\bar{t}W, t\bar{t}Z)} TN_{t\bar{t}Z,j}} = \frac{CM[2][0]}{\sum CM[2][:]} = \frac{408908}{408908 + 24685 + 661943} \approx 0.3732 \quad (\text{A.3})$$

CM - test (0.26 threshold)

Actual	$t\bar{t}H$	667366 30.45%	14366 0.66%	38746 1.77%	720478 92.63% 7.37%
	$t\bar{t}W$	315213 14.38%	59220 2.70%	4012 0.18%	378445 15.65% 84.35%
	$t\bar{t}Z$	408908 18.66%	21685 0.99%	661943 30.21%	1092536 60.59% 39.41%
		1391487 47.96% 52.04%	95271 62.16% 37.84%	704701 93.93% 6.07%	2191459 44.34% 36.64%
		Predicted			
		$t\bar{t}H$	$t\bar{t}W$	$t\bar{t}Z$	

Figure A.1: Example of 3-class classification results, $t\bar{t}H$ class is expressed as 0, $t\bar{t}W$ as 1 and $t\bar{t}Z$ as 2

For comparison with existing results, the weighting factors can be computed from [Col19b], Table 152, using Eq. [3.5] and the number of events from Table [B.5] as:

$$\omega_{t\bar{t}H} = \frac{1}{\frac{7674}{5.5650}} = \frac{1}{1378.9757} \quad (\text{A.4})$$

$$\omega_{t\bar{t}W} = \frac{1}{\frac{1178}{4.9338}} = \frac{1}{238.7612} \quad (\text{A.5})$$

$$\omega_{t\bar{t}Z} = \frac{1}{\frac{4253}{3.9677}} = \frac{1}{1071.9056} \quad (\text{A.6})$$

From table 152, B_{other} can also be computed as the sum of the rest of the expected background events:

$$B_{other} = B_{t\bar{t}b\bar{a}r} + B_{t\bar{t}\gamma} + B_{VV} + B_{rare} = 1.7113 + 0.2165 + 0.7311 + 2.4493 = 5.1082 \quad (\text{A.7})$$

With this knowledge, a substitution to a simplified form of Eq. 3.7 and 3.8 is possible. The final number of expected events are:

$$S = N_{t\bar{t}W} \cdot \omega_{t\bar{t}H} \cdot \epsilon_{t\bar{t}H} = 7674 \cdot \frac{1}{1378.9757} \cdot 0.9263 \approx 5.1549, \quad (\text{A.8})$$

for signal, and:

$$B = \sum_{i \in (t\bar{t}W, t\bar{t}Z)} N_i \cdot \omega_i \cdot \epsilon_i + B_{other} = 1178 \cdot \frac{1}{238.7612} \cdot 0.8329 + 4253 \cdot \frac{1}{1071.9056} \cdot 0.3732 + 5.1082 \approx 10.6983, \quad (\text{A.9})$$

for the background.

Significance approximation in its simplified form (Eq. 3.2) is therefore:

$$significance = \frac{S}{\sqrt{B}} = \frac{5.1549}{\sqrt{10.6983}} \approx 1.576. \quad (\text{A.10})$$

This is an improvement, compared to 1.487 in [Col19b].



Appendix B

Tables



B.1 Data

ID	name	ID	name
0	DRll01	42	lep_isPrompt_0
1	DRll12	43	lep_isPrompt_1
2	MET_RefFinal_et	44	lep_isPrompt_2
3	MET_RefFinal_phi	45	lep_isQMisID_0
4	MV2c10_70_EventWeight	46	lep_isQMisID_1
5	Mll01	47	lep_isQMisID_2
6	Mll02	48	lep_isTightLH_0
7	Mll03	49	lep_isTightLH_1
8	Mll12	50	lep_isTightLH_2
9	Mll13	51	lep_isolationFixedCutLoose_0
10	Mll23	52	lep_isolationFixedCutLoose_1
11	Mlll012	53	lep_isolationFixedCutLoose_2
12	Mllll0123	54	lep_isolationFixedCutLoose_3
13	best_Z_Mll	55	lep_promptLeptonVeto_TagWeight_0
14	dilep_type	56	lep_promptLeptonVeto_TagWeight_1
15	lep_Eta_0	57	lep_promptLeptonVeto_TagWeight_2
16	lep_Eta_1	58	nJets_OR
17	lep_Eta_2	59	nJets_OR_MV2c10_70
18	lep_ID_0	60	nJets_OR_T
19	lep_ID_1	61	nJets_OR_T_MV2c10_70
20	lep_ID_2	62	nTaus_OR_Pt25
21	lep_ID_2_new	63	tau_MV2c10_0
22	lep_ID_3	64	tau_btag70_0
23	lep_Mtrktrk_atConvV_CO_0	65	tau_btag70_1
24	lep_Mtrktrk_atConvV_CO_1	66	tau_charge_0
25	lep_Mtrktrk_atConvV_CO_2	67	tau_charge_1
26	lep_Mtrktrk_atPV_CO_0	68	tau_fromPV_0
27	lep_Mtrktrk_atPV_CO_1	69	tau_passEleBDT_0
28	lep_Mtrktrk_atPV_CO_2	70	tau_passEleBDT_1
29	lep_Pt_0	71	tau_passMuonOLR_0
30	lep_Pt_1	72	tau_passMuonOLR_1
31	lep_Pt_2	73	tau_tagWeightBin_0
32	lep_RadiusCO_0	74	tau_tagWeightBin_1
33	lep_RadiusCO_1	75	tau_tight_0
34	lep_RadiusCO_2	76	tau_tight_1
35	lep_chargeIDBDTTight_0	77	total_charge
36	lep_chargeIDBDTTight_1	78	trilep_type
37	lep_chargeIDBDTTight_2	79	total_leptons
38	lep_flavour	80	HT
39	lep_isMedium_0	81	HT_lep
40	lep_isMedium_1	82	HT_jets
41	lep_isMedium_2		

Table B.1: List of used features

ID	name	ID	name
0	DRll01	39	lep_isMedium_0
1	DRll12	40	lep_isMedium_1
2	MET_RefFinal_et	41	lep_isMedium_2
3	MET_RefFinal_phi	42	lep_isTightLH_0
4	MV2c10_70_EventWeight	43	lep_isTightLH_1
5	Mll01	44	lep_isTightLH_2
6	Mll02	45	lep_isolationFixedCutLoose_0
7	Mll03	46	lep_isolationFixedCutLoose_1
8	Mll12	47	lep_isolationFixedCutLoose_2
9	Mll13	48	lep_isolationFixedCutLoose_3
10	Mll23	49	lep_promptLeptonVeto_TagWeight_0
11	Mlll012	50	lep_promptLeptonVeto_TagWeight_1
12	Mllll0123	51	lep_promptLeptonVeto_TagWeight_2
13	best_Z_Mll	52	nJets_OR
14	dilep_type	53	nJets_OR_MV2c10_70
15	lep_Eta_0	54	nJets_OR_T
16	lep_Eta_1	55	nJets_OR_T_MV2c10_70
17	lep_Eta_2	56	nTaus_OR_Pt25
18	lep_ID_0	57	tau_MV2c10_0
19	lep_ID_1	58	tau_btag70_0
20	lep_ID_2	59	tau_btag70_1
21	lep_ID_2_new	60	tau_charge_0
22	lep_ID_3	61	tau_charge_1
23	lep_Mtrktrk_atConvV_CO_0	62	tau_fromPV_0
24	lep_Mtrktrk_atConvV_CO_1	63	tau_passEleBDT_0
25	lep_Mtrktrk_atConvV_CO_2	64	tau_passEleBDT_1
26	lep_Mtrktrk_atPV_CO_0	65	tau_passMuonOLR_0
27	lep_Mtrktrk_atPV_CO_1	66	tau_passMuonOLR_1
28	lep_Mtrktrk_atPV_CO_2	67	tau_tagWeightBin_0
29	lep_Pt_0	68	tau_tagWeightBin_1
30	lep_Pt_1	69	tau_tight_0
31	lep_Pt_2	70	tau_tight_1
32	lep_RadiusCO_0	71	total_charge
33	lep_RadiusCO_1	72	trilep_type
34	lep_RadiusCO_2	73	total_leptons
35	lep_chargeIDBDTTight_0	74	HT
36	lep_chargeIDBDTTight_1	75	HT_lep
37	lep_chargeIDBDTTight_2	76	HT_jets
38	lep_flavour		

Table B.2: List of features after removing additional 'simulation-only related' features

folder	file ID	events	2LSS1Tau events
mc16a/older	345672	2,211 $t\bar{t}H$	2 $t\bar{t}H$
mc16a/older	345673	99,790 $t\bar{t}H$	1,729 $t\bar{t}H$
mc16a/older	345674	255,663 $t\bar{t}H$	2,113 $t\bar{t}H$
mc16a/older	410155	163,406 $t\bar{t}W$	684 $t\bar{t}W$
mc16a/older	410156	1,333 $t\bar{t}Z$	0 $t\bar{t}Z$
mc16a/older	410157	4,713 $t\bar{t}Z$	1 $t\bar{t}Z$
mc16a/older	410218	244,768 $t\bar{t}Z$	550 $t\bar{t}Z$
mc16a/older	410219	286,557 $t\bar{t}Z$	379 $t\bar{t}Z$
mc16a/older	410220	34,675 $t\bar{t}Z$	1,271 $t\bar{t}Z$
total		357,664 $t\bar{t}H$	3,844 $t\bar{t}H$
		163,406 $t\bar{t}W$	684 $t\bar{t}W$
		572,046 $t\bar{t}Z$	2,201 $t\bar{t}Z$

Table B.3: List of root files used for training (80%) and testing (20%) splits - older data set (mc16a)

folder	file ID	events	2LSS1Tau events
mc16d/older	345672	2,531 $t\bar{t}H$	3 $t\bar{t}H$
mc16d/older	410155	155,594 $t\bar{t}W$	680 $t\bar{t}W$
mc16d/older	410156	1,178 $t\bar{t}Z$	0 $t\bar{t}Z$
mc16d/older	410157	4,234 $t\bar{t}Z$	2 $t\bar{t}Z$
mc16d/older	410218	216,848 $t\bar{t}Z$	586 $t\bar{t}Z$
mc16d/older	410219	266,987 $t\bar{t}Z$	342 $t\bar{t}Z$
mc16d/older	410220	31,243 $t\bar{t}Z$	1,122 $t\bar{t}Z$
total		2,531 $t\bar{t}H$	3 $t\bar{t}H$
		155,594 $t\bar{t}W$	680 $t\bar{t}W$
		520,490 $t\bar{t}Z$	2,052 $t\bar{t}Z$

Table B.4: List of root files used for additional testing - older data set (mc16d)

folder	file ID	events	2LSS1Tau events
mc16a/newer	345873	2,315 $t\bar{t}H$	3 $t\bar{t}H$
mc16a/newer	345874	100,860 $t\bar{t}H$	1,653 $t\bar{t}H$
mc16a/newer	345875	257,064 $t\bar{t}H$	2,181 $t\bar{t}H$
mc16d/newer	345873	2,315 $t\bar{t}H$	3 $t\bar{t}H$
mc16d/newer	345874	100,860 $t\bar{t}H$	1,653 $t\bar{t}H$
mc16d/newer	345875	257,064 $t\bar{t}H$	2,181 $t\bar{t}H$
mc16a/newer	413008_wCharge	172,937 $t\bar{t}W$	511 $t\bar{t}W$
mc16d/newer	413008_wCharge	205,508 $t\bar{t}W$	667 $t\bar{t}W$
mc16a/older	410156	1,333 $t\bar{t}Z$	0 $t\bar{t}Z$
mc16a/older	410157	4,713 $t\bar{t}Z$	1 $t\bar{t}Z$
mc16a/older	410218	244,768 $t\bar{t}Z$	550 $t\bar{t}Z$
mc16a/older	410219	286,557 $t\bar{t}Z$	379 $t\bar{t}Z$
mc16a/older	410220	34,675 $t\bar{t}Z$	1,271 $t\bar{t}Z$
mc16d/older	410156	1,178 $t\bar{t}Z$	0 $t\bar{t}Z$
mc16d/older	410157	4,234 $t\bar{t}Z$	2 $t\bar{t}Z$
mc16d/older	410218	216,848 $t\bar{t}Z$	586 $t\bar{t}Z$
mc16d/older	410219	266,987 $t\bar{t}Z$	342 $t\bar{t}Z$
mc16d/older	410220	31,243 $t\bar{t}Z$	1,122 $t\bar{t}Z$
total		720,478 $t\bar{t}H$	7,674 $t\bar{t}H$
		378,445 $t\bar{t}W$	1,178 $t\bar{t}W$
		1,092,536 $t\bar{t}Z$	4,253 $t\bar{t}Z$

Table B.5: List of used root files for comparison (100%), re-training (80%), and validation (20%) - newer data set containing older $t\bar{t}Z$ files (both mc16a & mc16d)

folder	file ID	events	2LSS1Tau events
mc16a/real	data15	3,286	1
mc16a/real	data16	30,090	13
mc16d/real	data17	35,788	9
total		69,164	23

Table B.6: List of used real data root files for testing

Appendix C

Figures

C.1 Data Histograms

C.1.1 Older mc16a set

Full-data

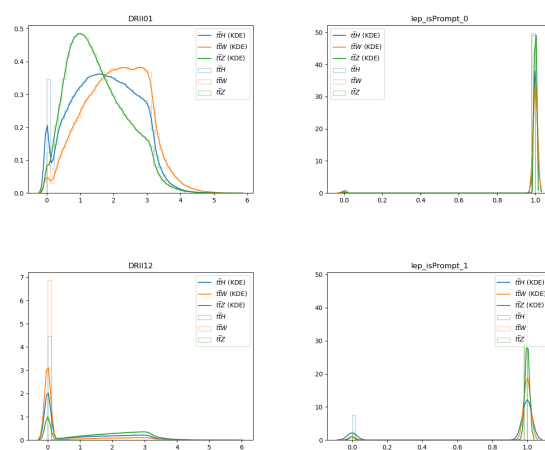


Figure C.1: Full-data histograms (older mc16a set)

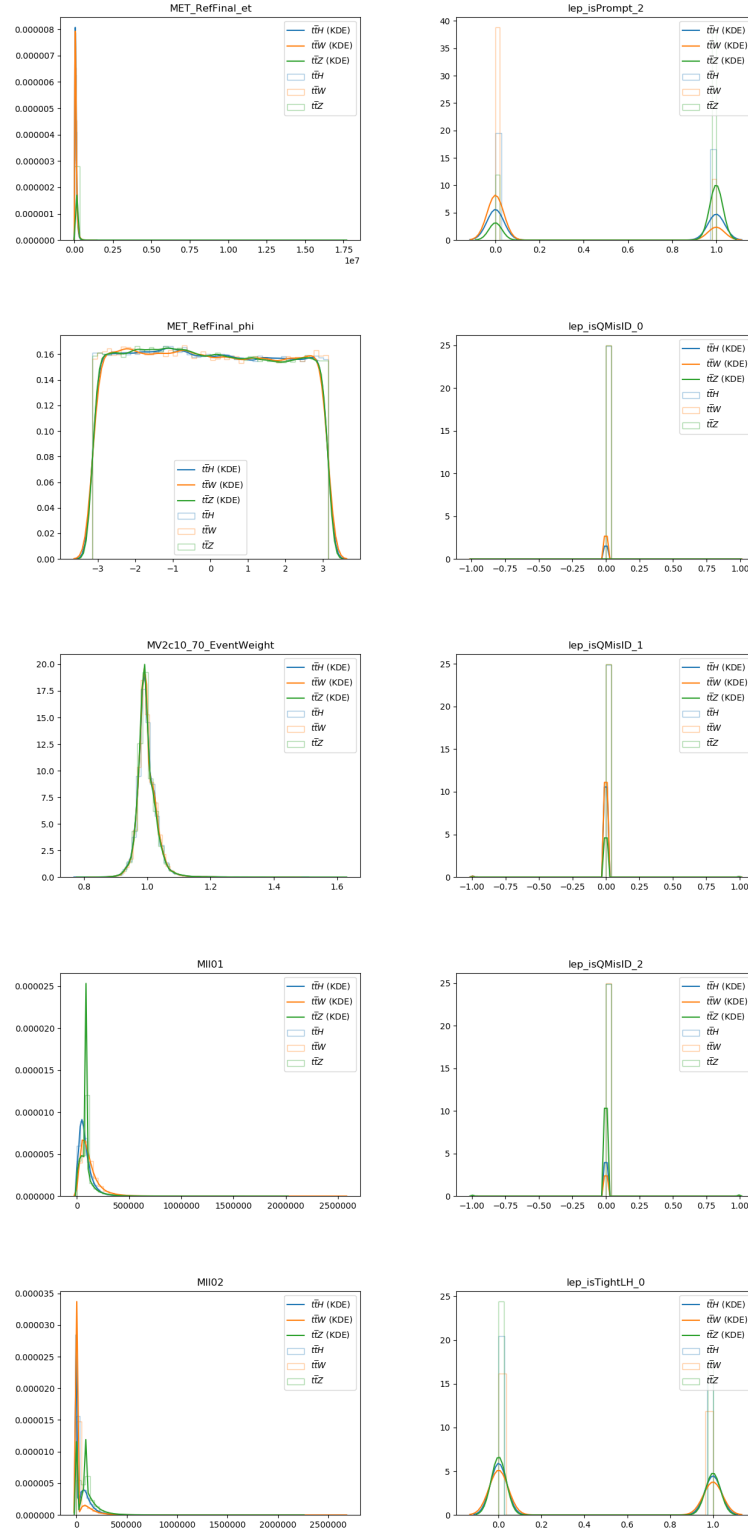


Figure C.1: Full-data histograms (older mc16a set)

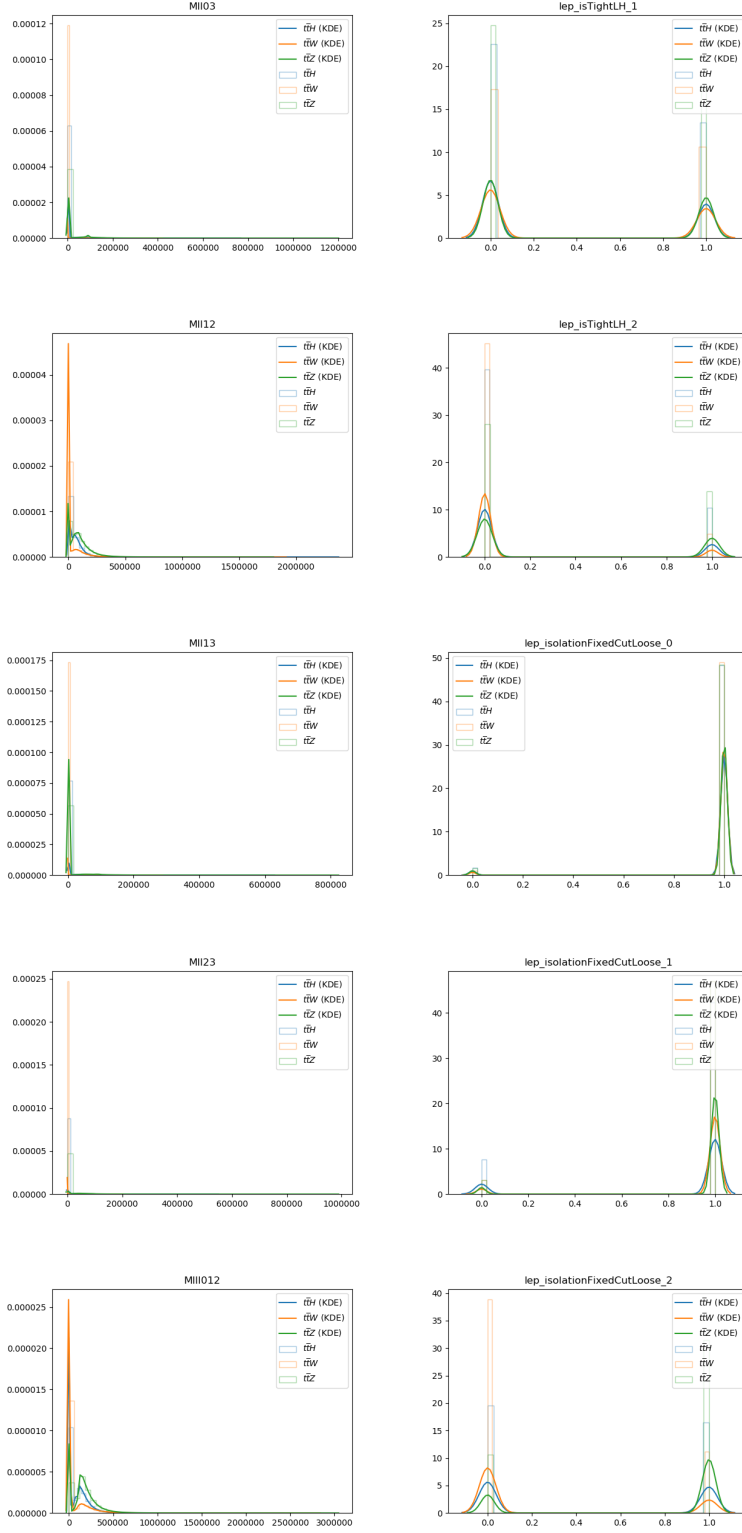


Figure C.1: Full-data histograms (older mc16a set)

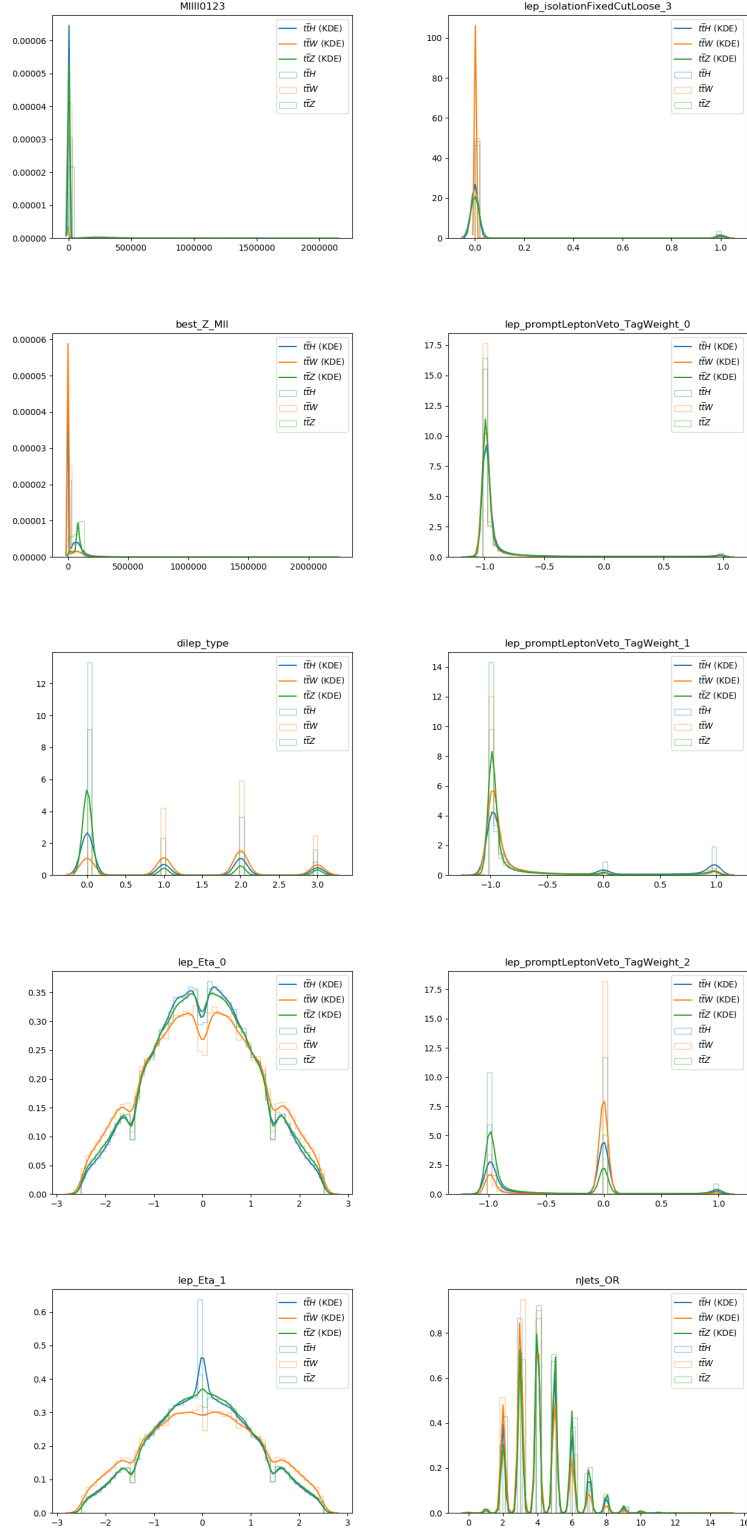


Figure C.1: Full-data histograms (older mc16a set)

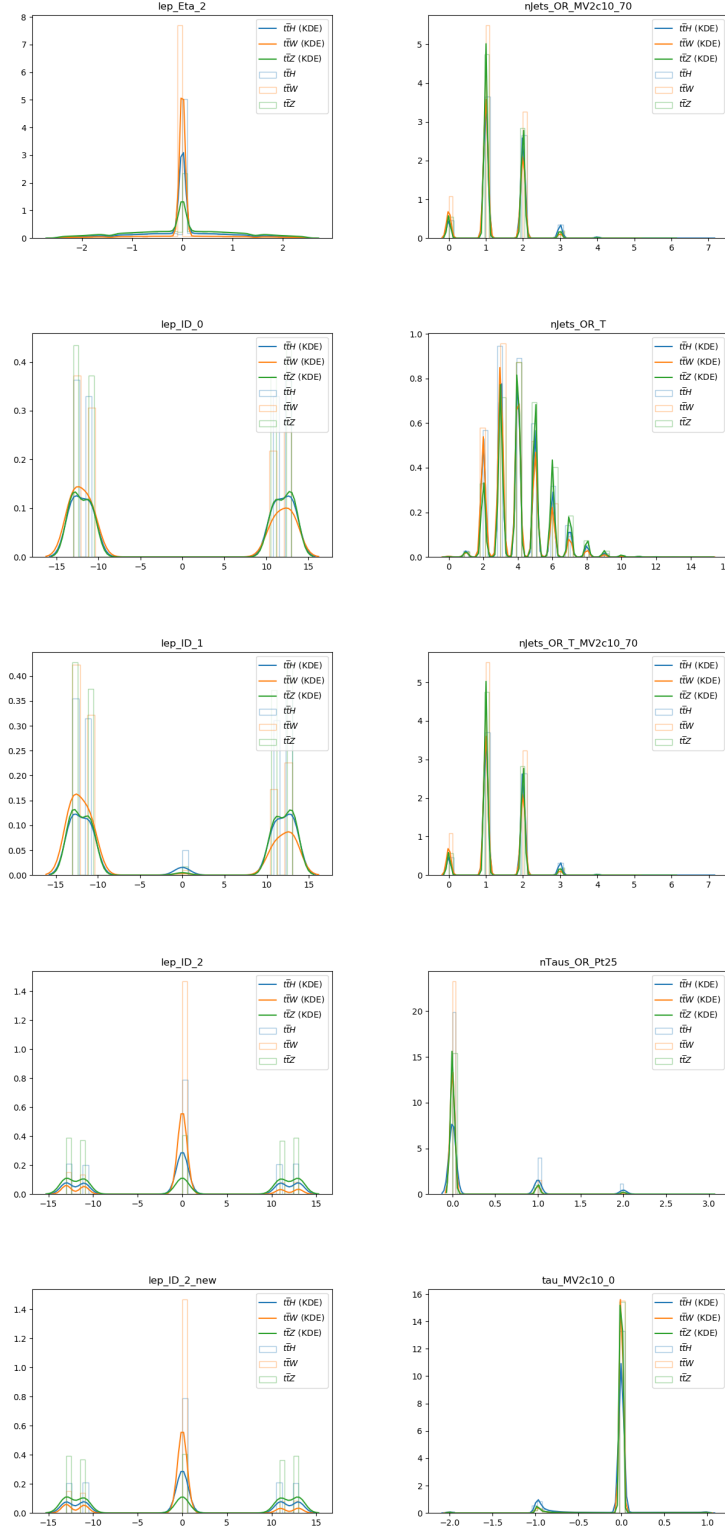


Figure C.1: Full-data histograms (older mc16a set)

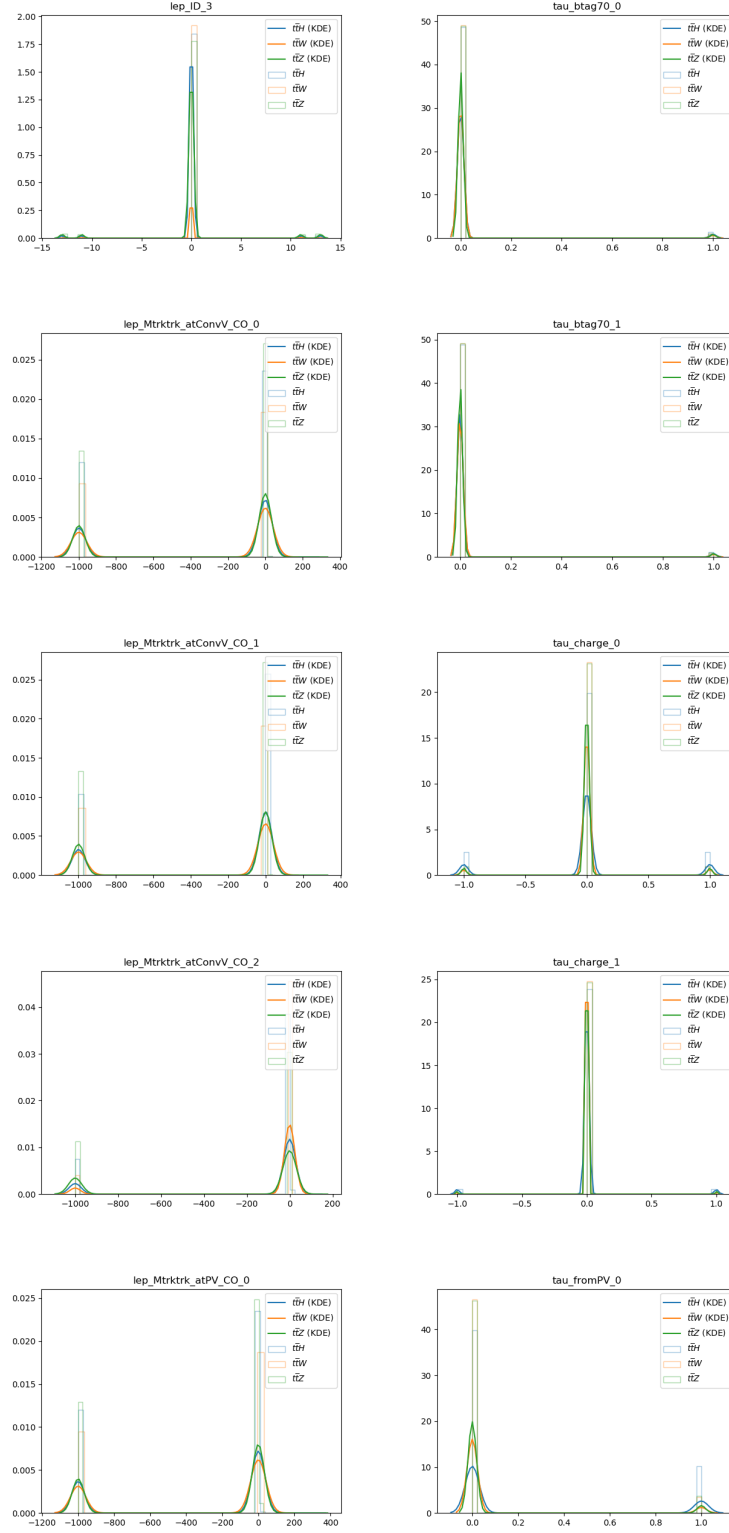


Figure C.1: Full-data histograms (older mc16a set)

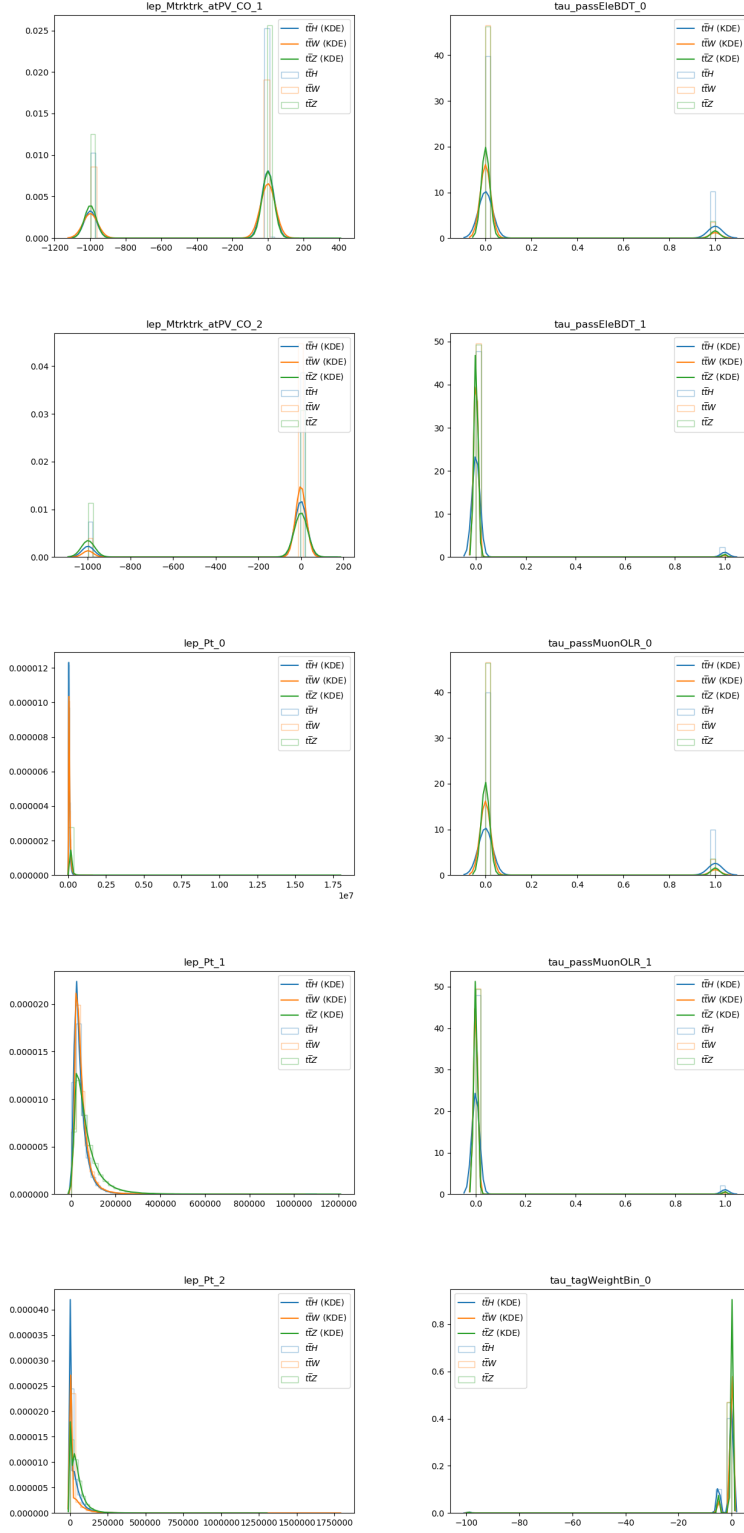


Figure C.1: Full-data histograms (older mc16a set)

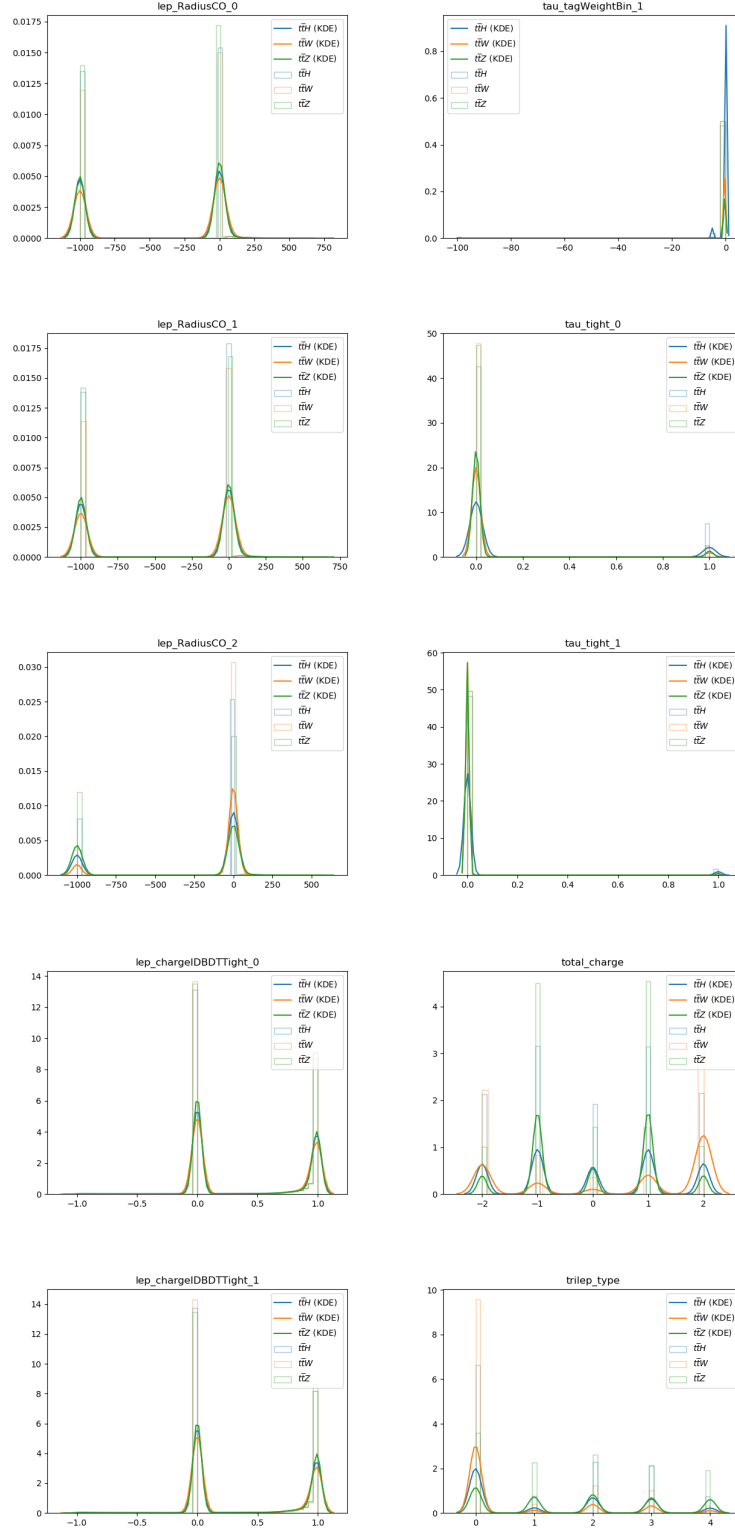


Figure C.1: Full-data histograms (older mc16a set)

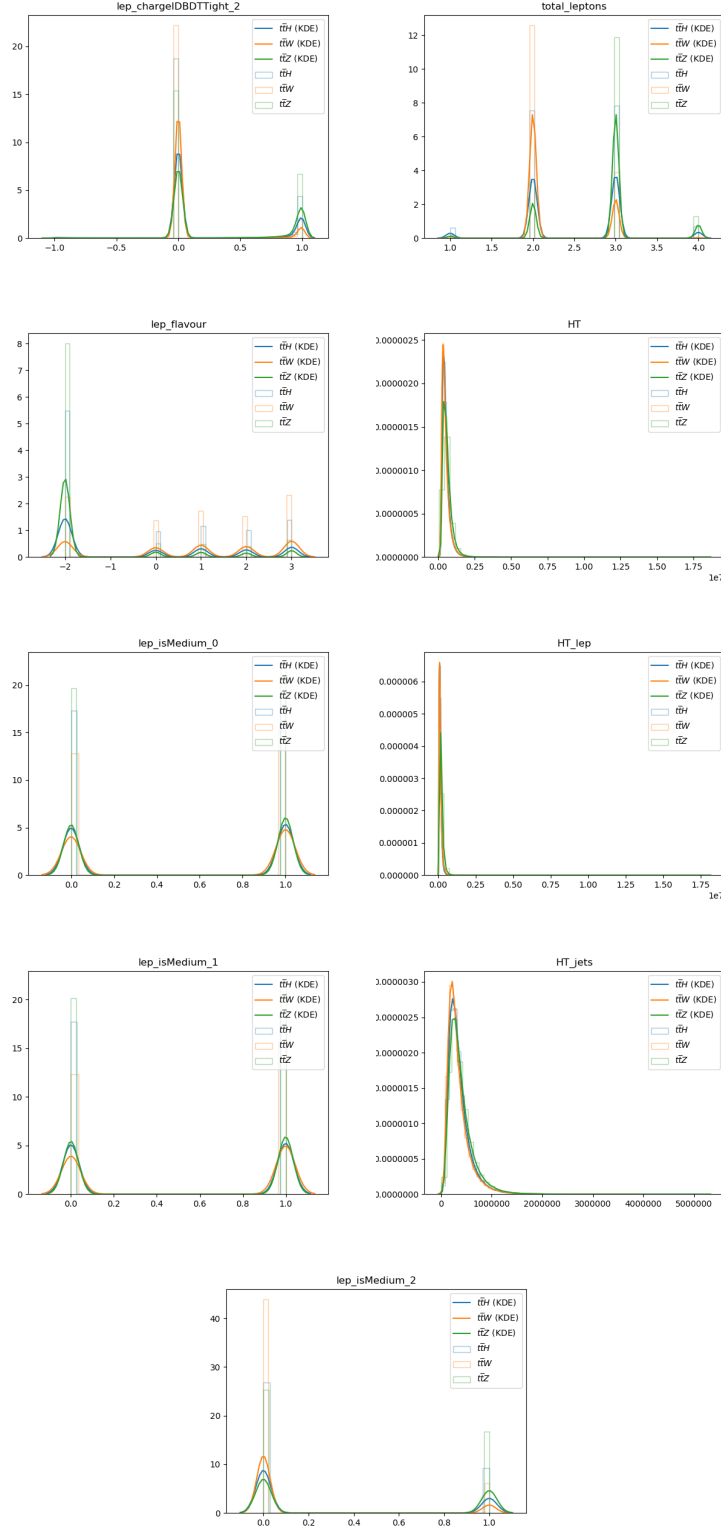


Figure C.1: Full-data histograms (older mc16a set)

Channel-data

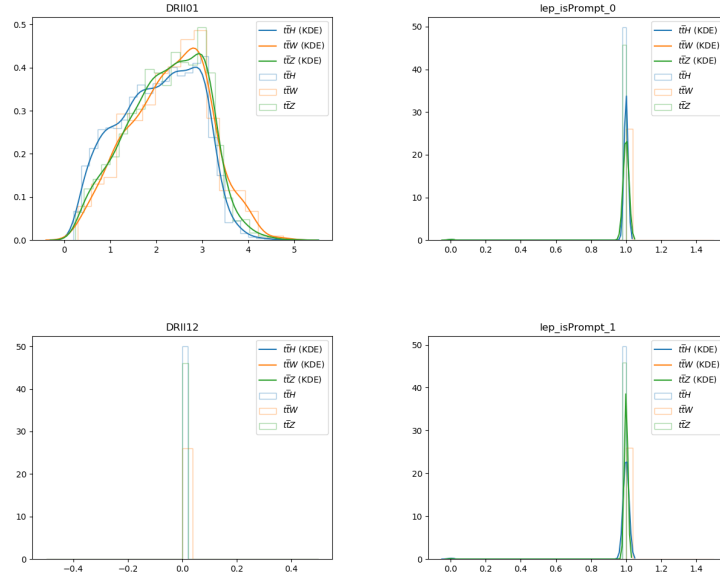


Figure C.2: Channel-data histograms (older mc16a set)

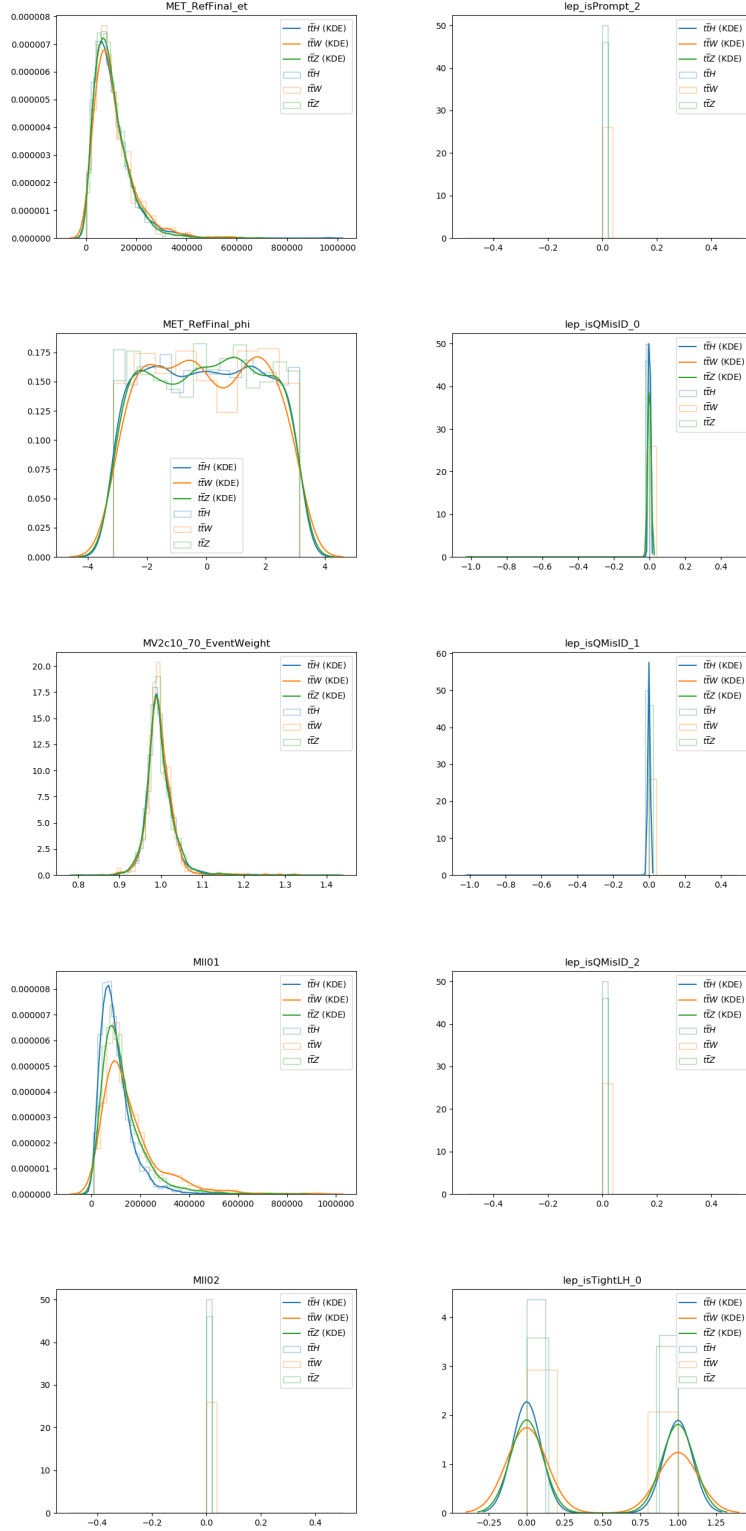


Figure C.2: Channel-data histograms (older mc16a set)

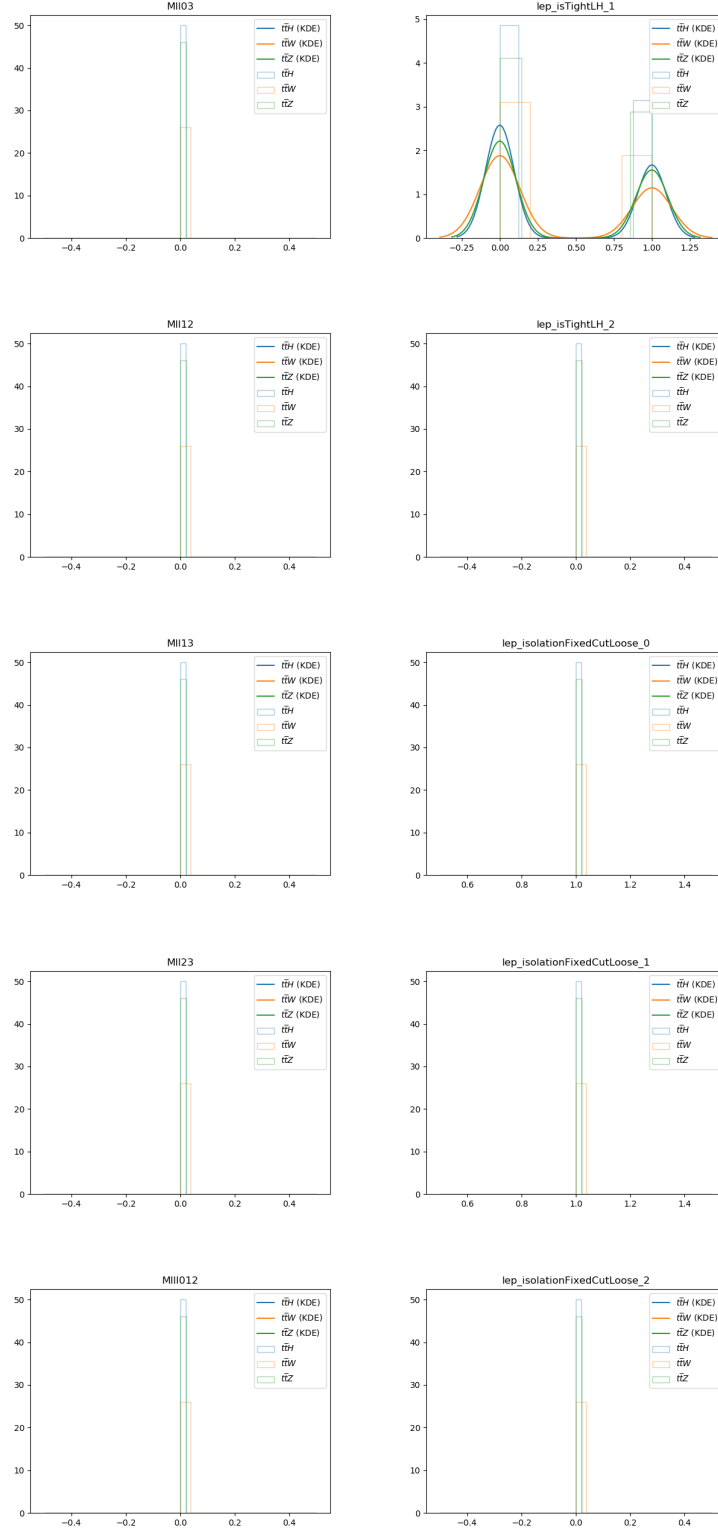


Figure C.2: Channel-data histograms (older mc16a set)

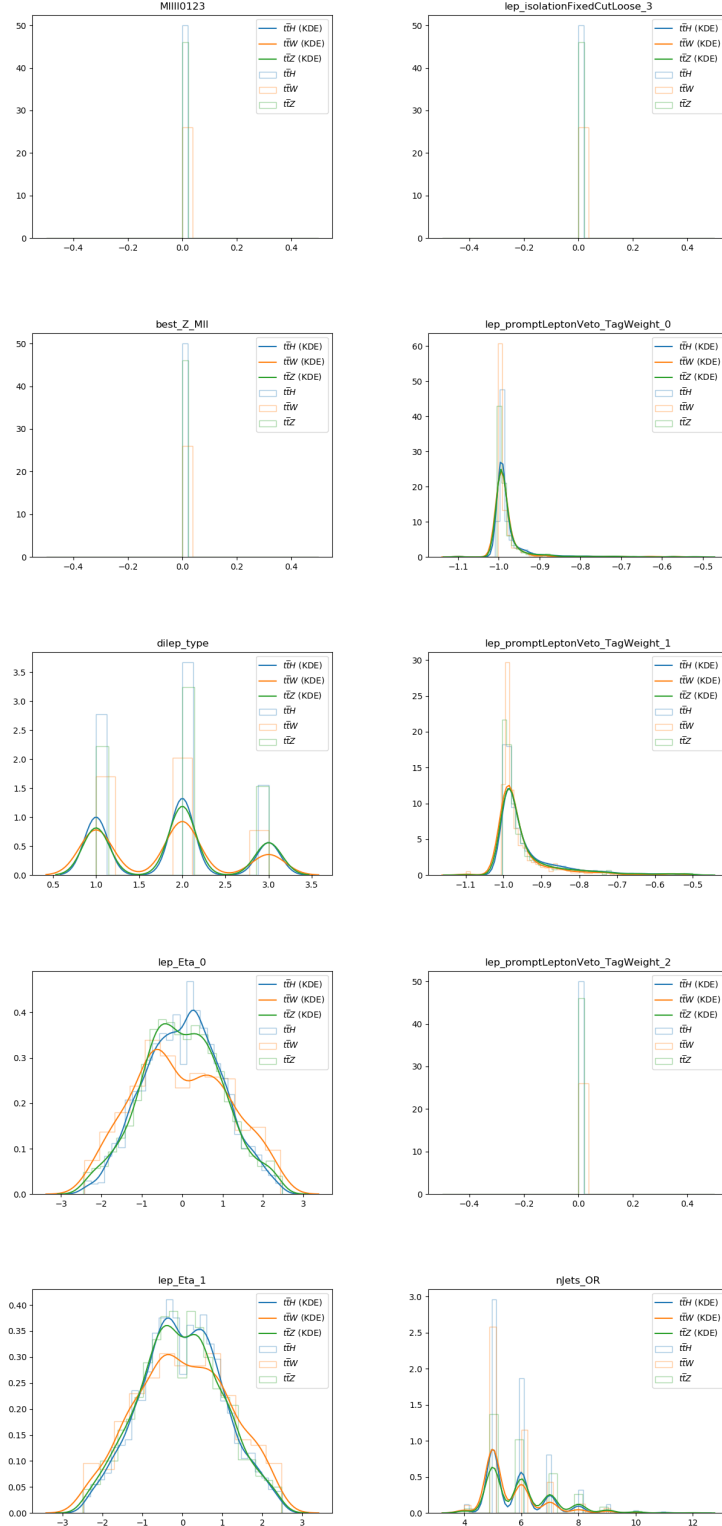


Figure C.2: Channel-data histograms (older mc16a set)

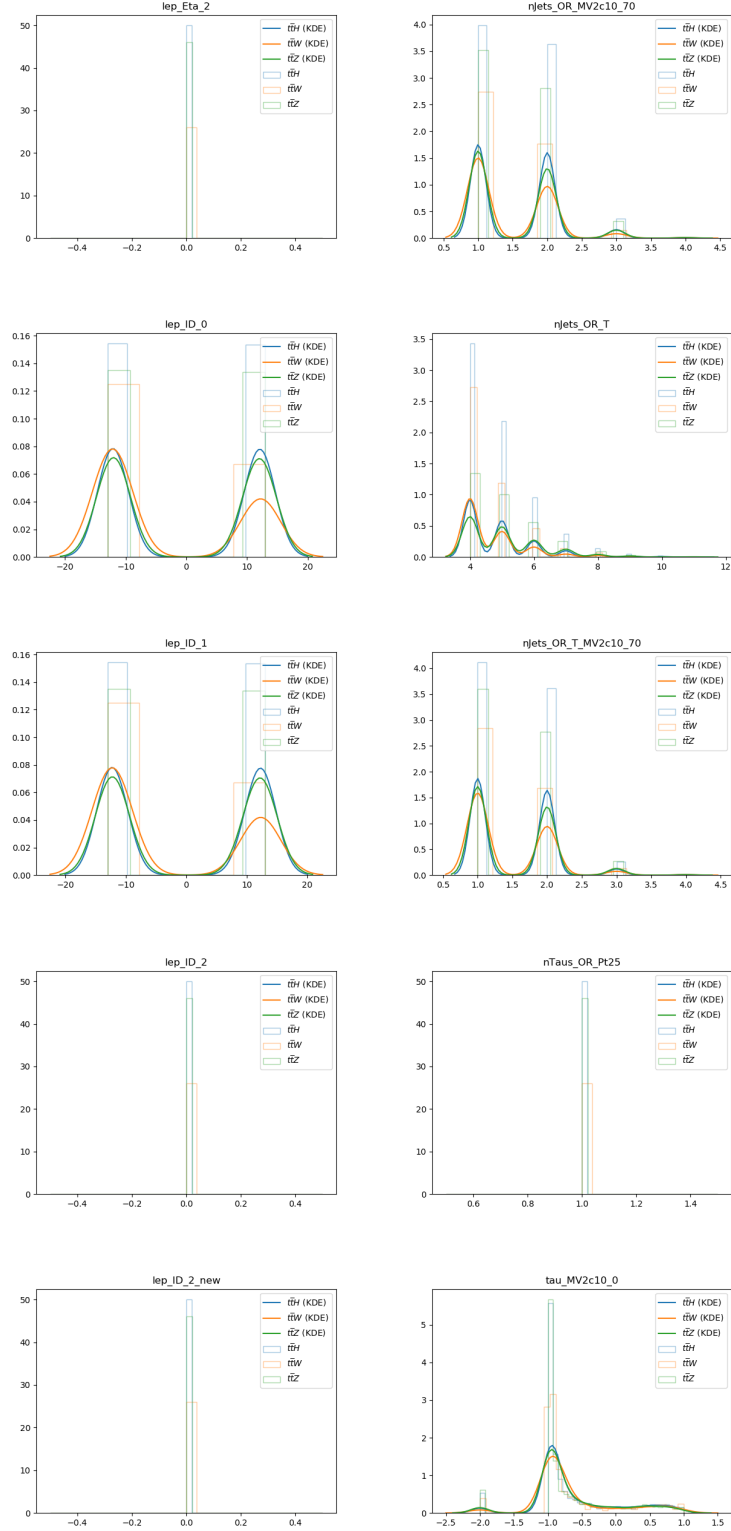


Figure C.2: Channel-data histograms (older mc16a set)

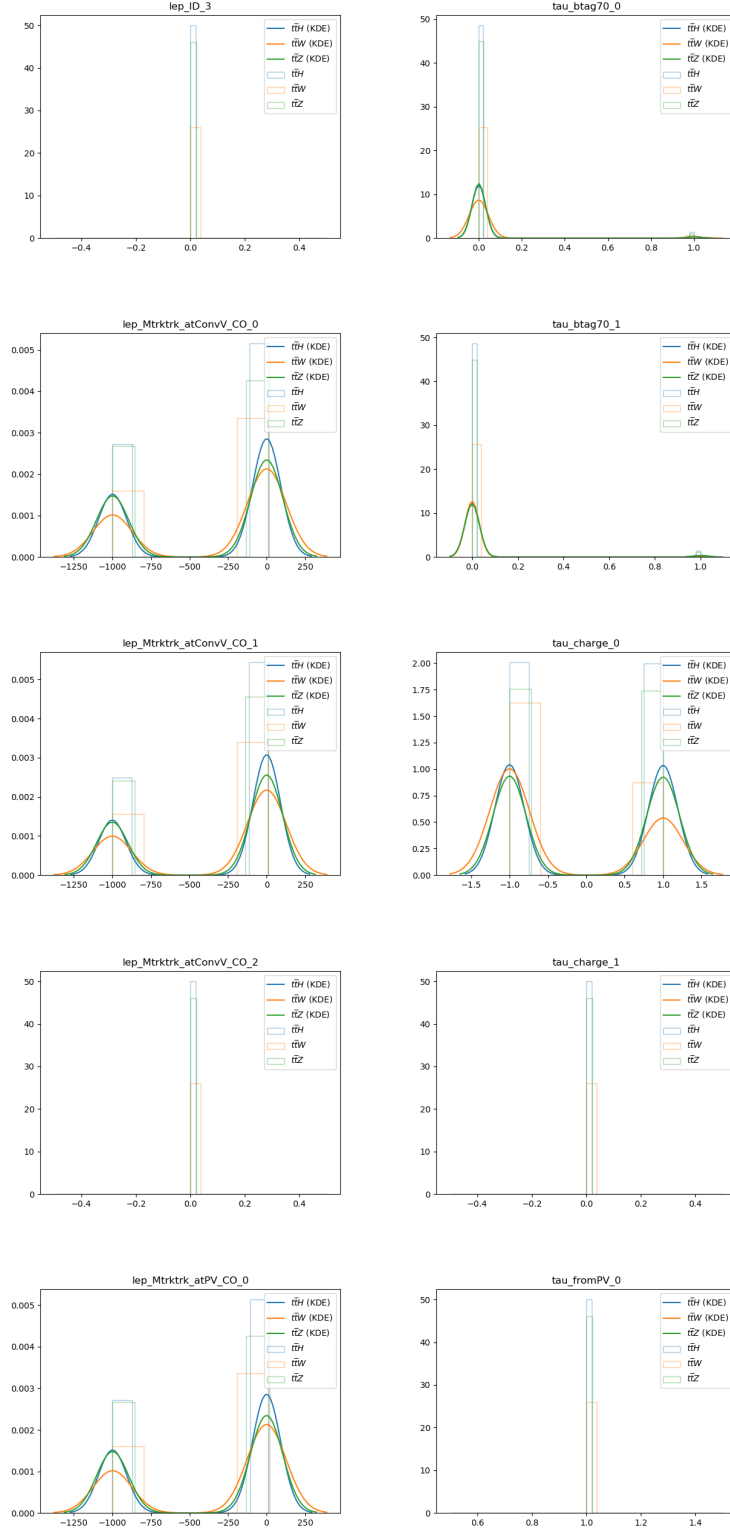


Figure C.2: Channel-data histograms (older mc16a set)

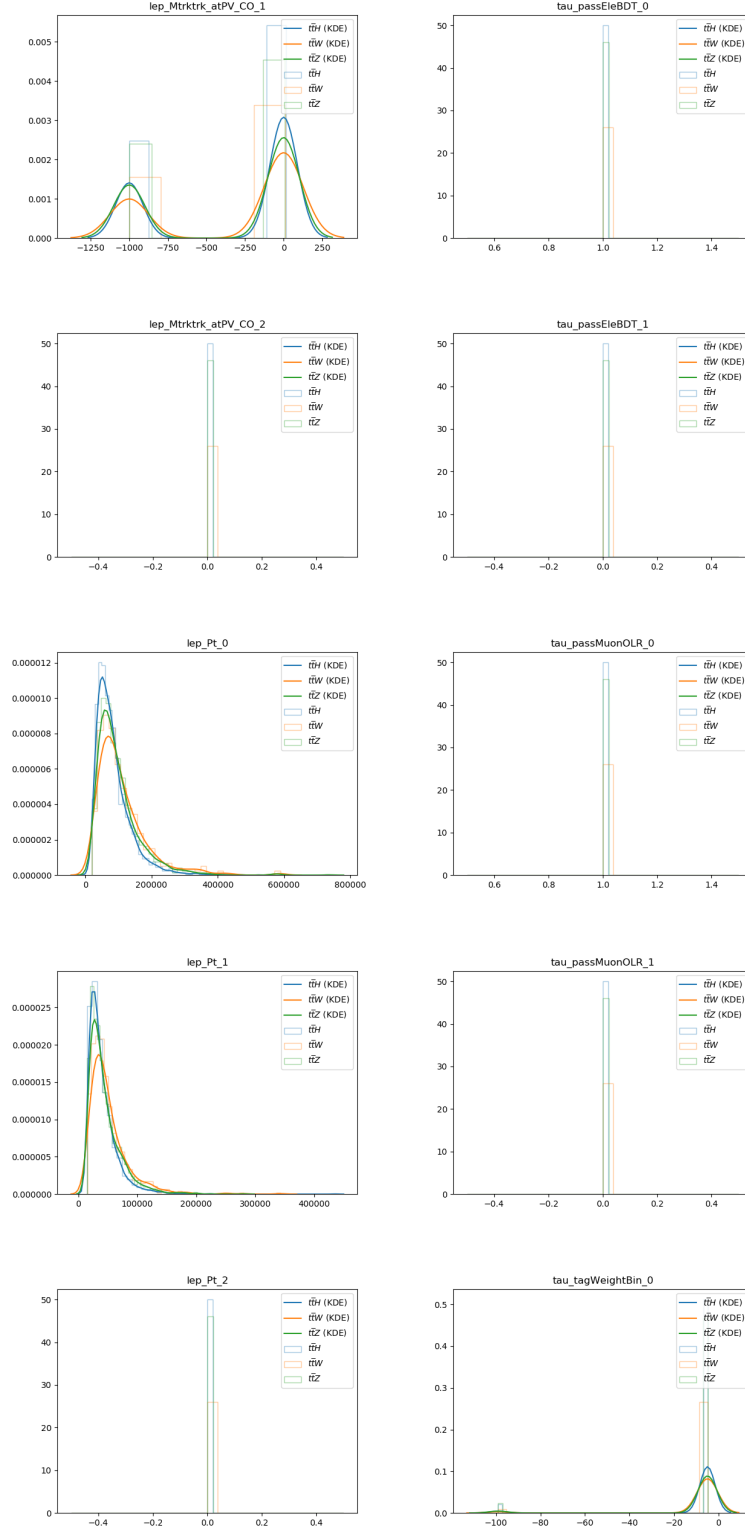


Figure C.2: Channel-data histograms (older mc16a set)

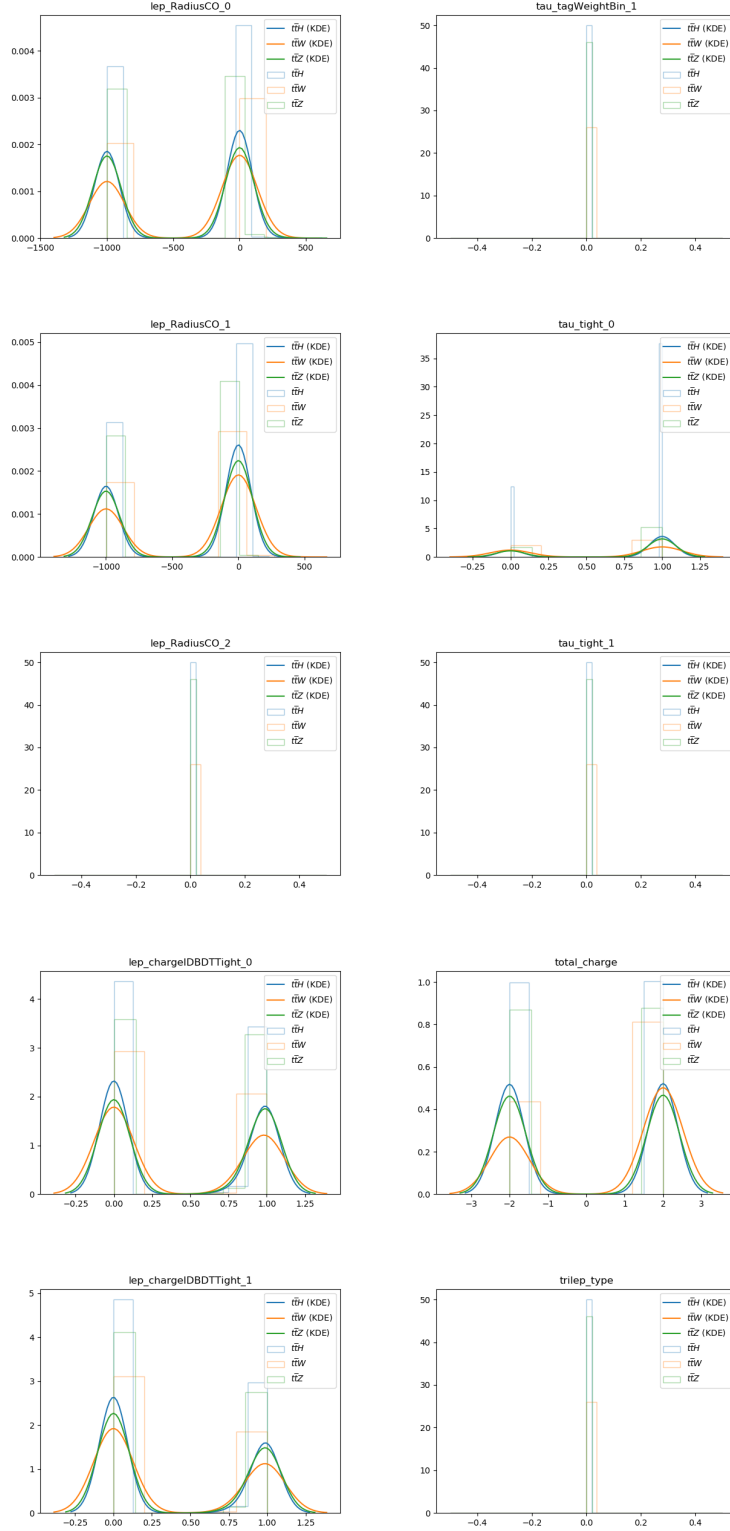


Figure C.2: Channel-data histograms (older mc16a set)

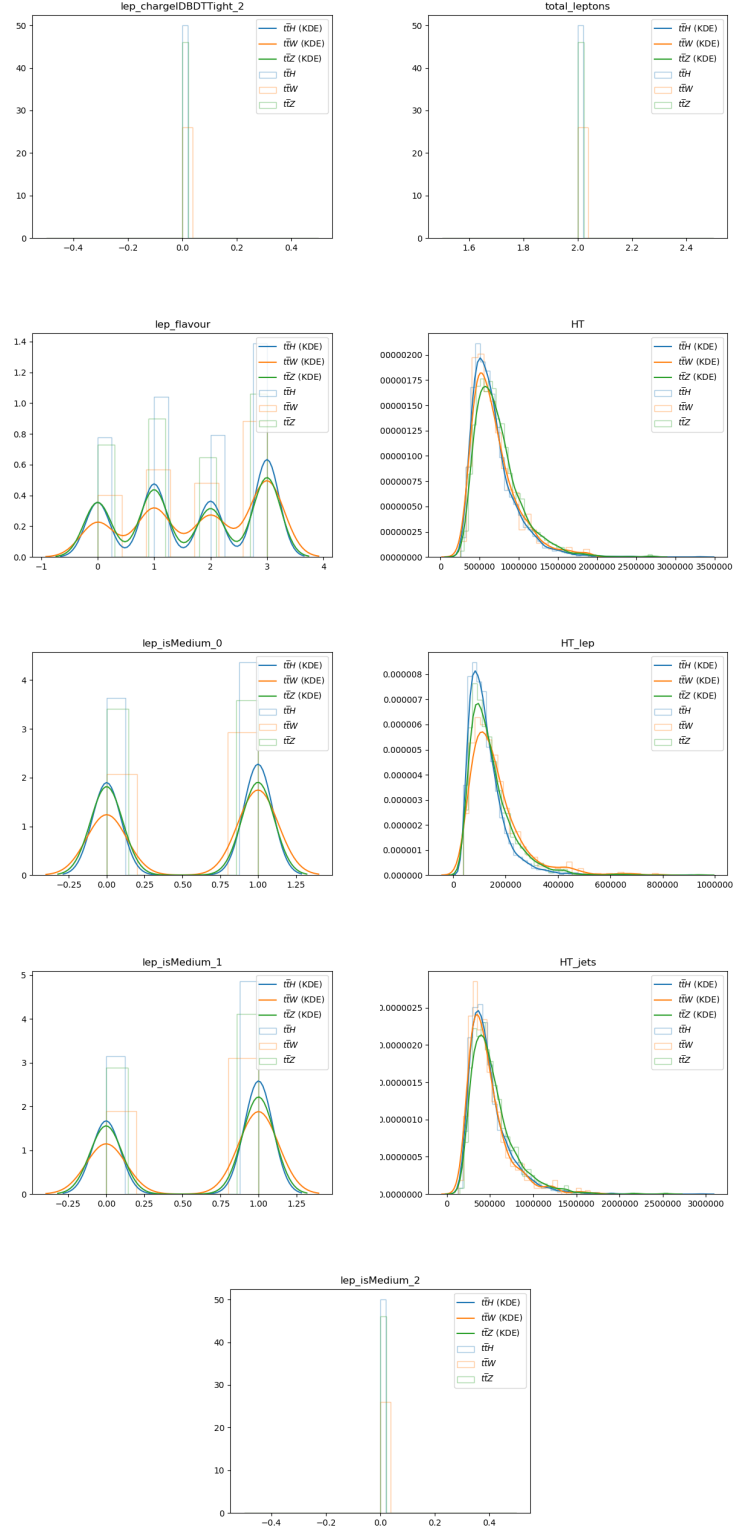


Figure C.2: Channel-data histograms (older mc16a set)

■ C.1.2 Real mc16a & mc16d set

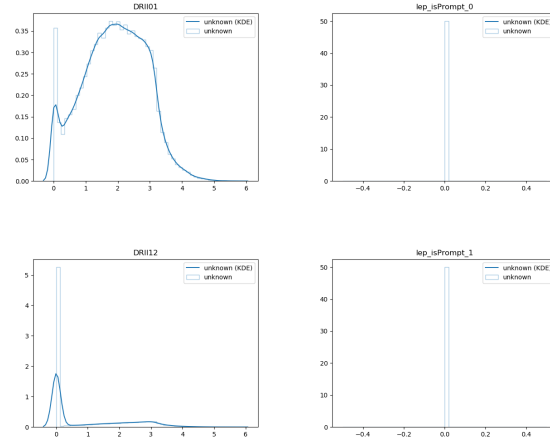


Figure C.3: Real-data histograms (mc16a & mc16d set)

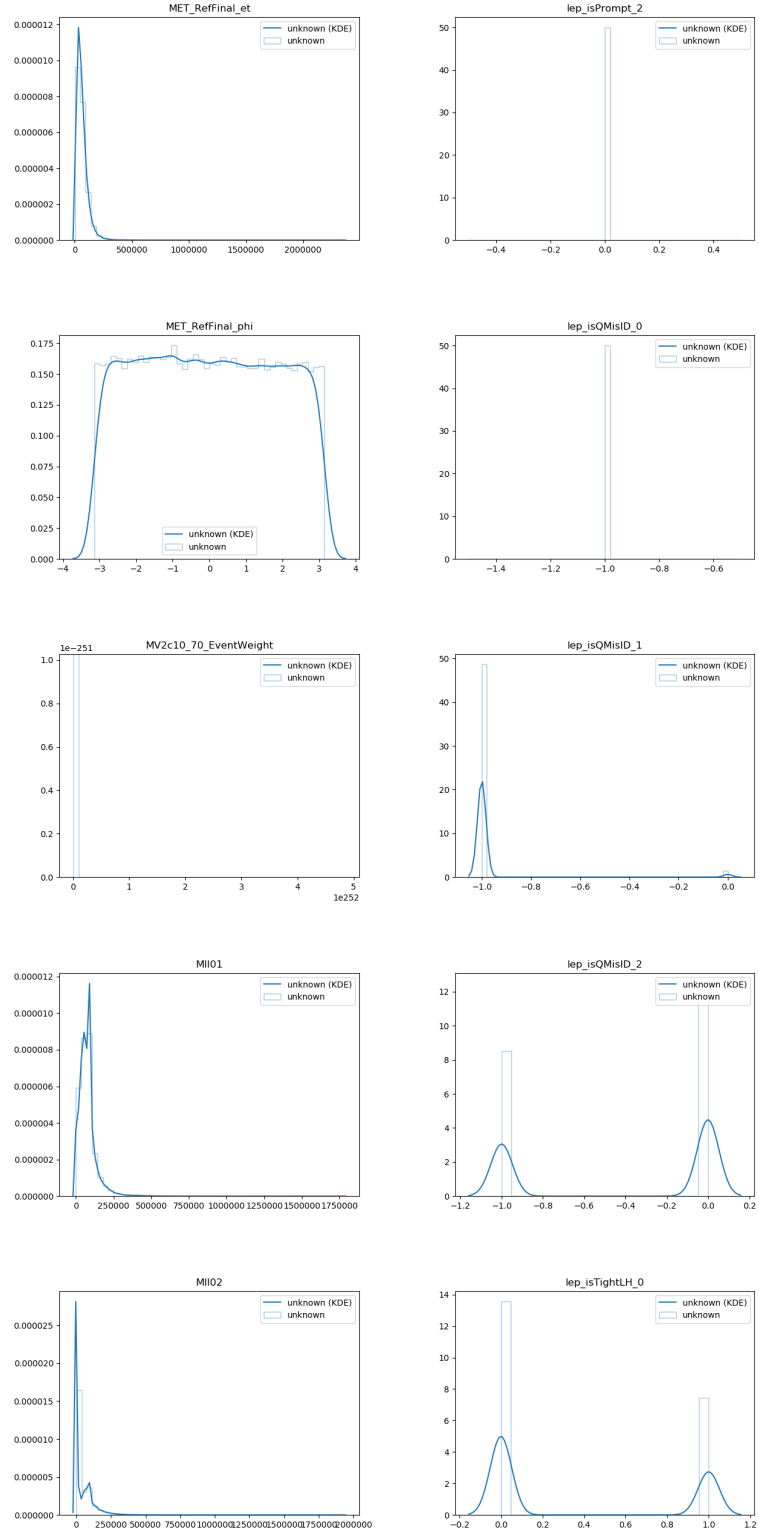


Figure C.3: Real-data histograms (mc16a & mc16d set)

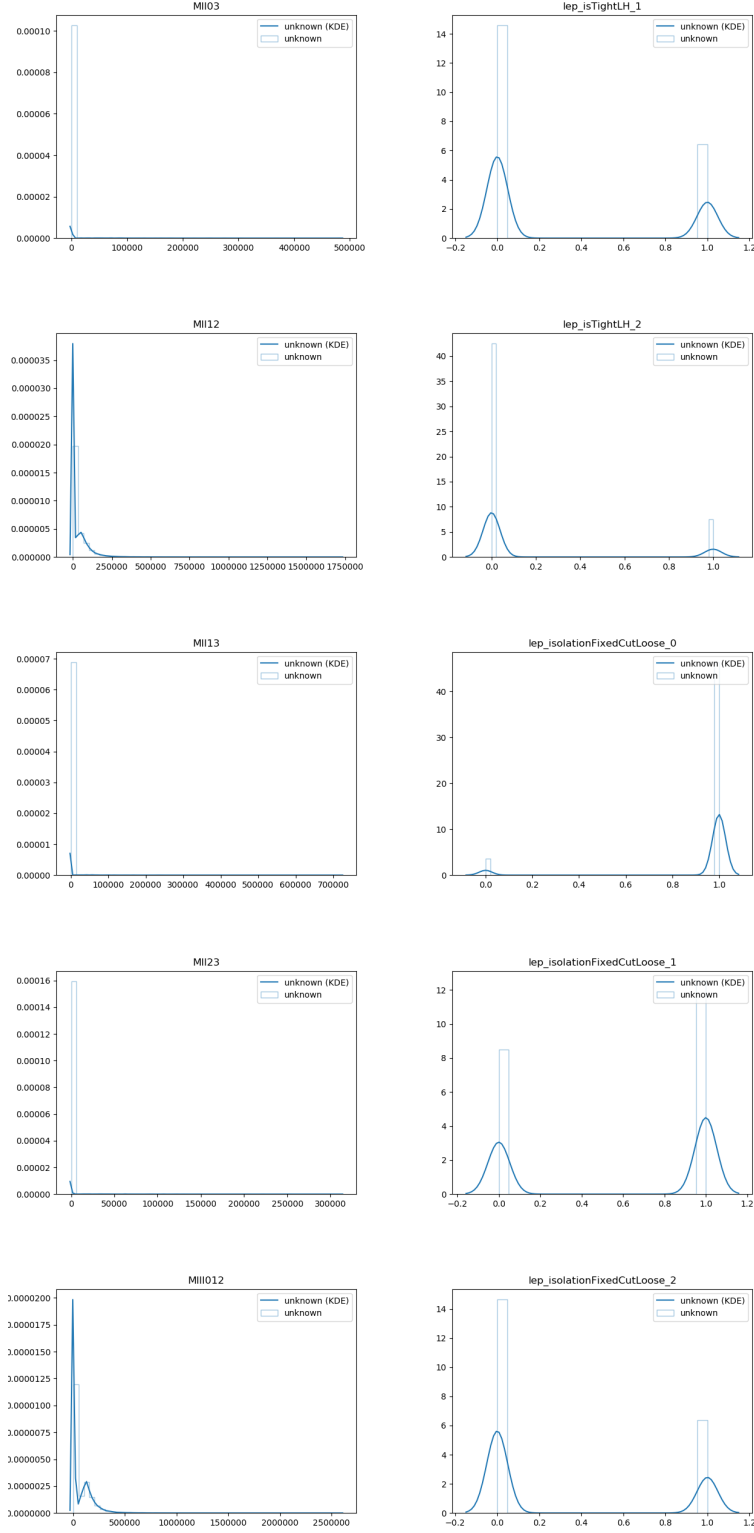


Figure C.3: Real-data histograms (mc16a & mc16d set)

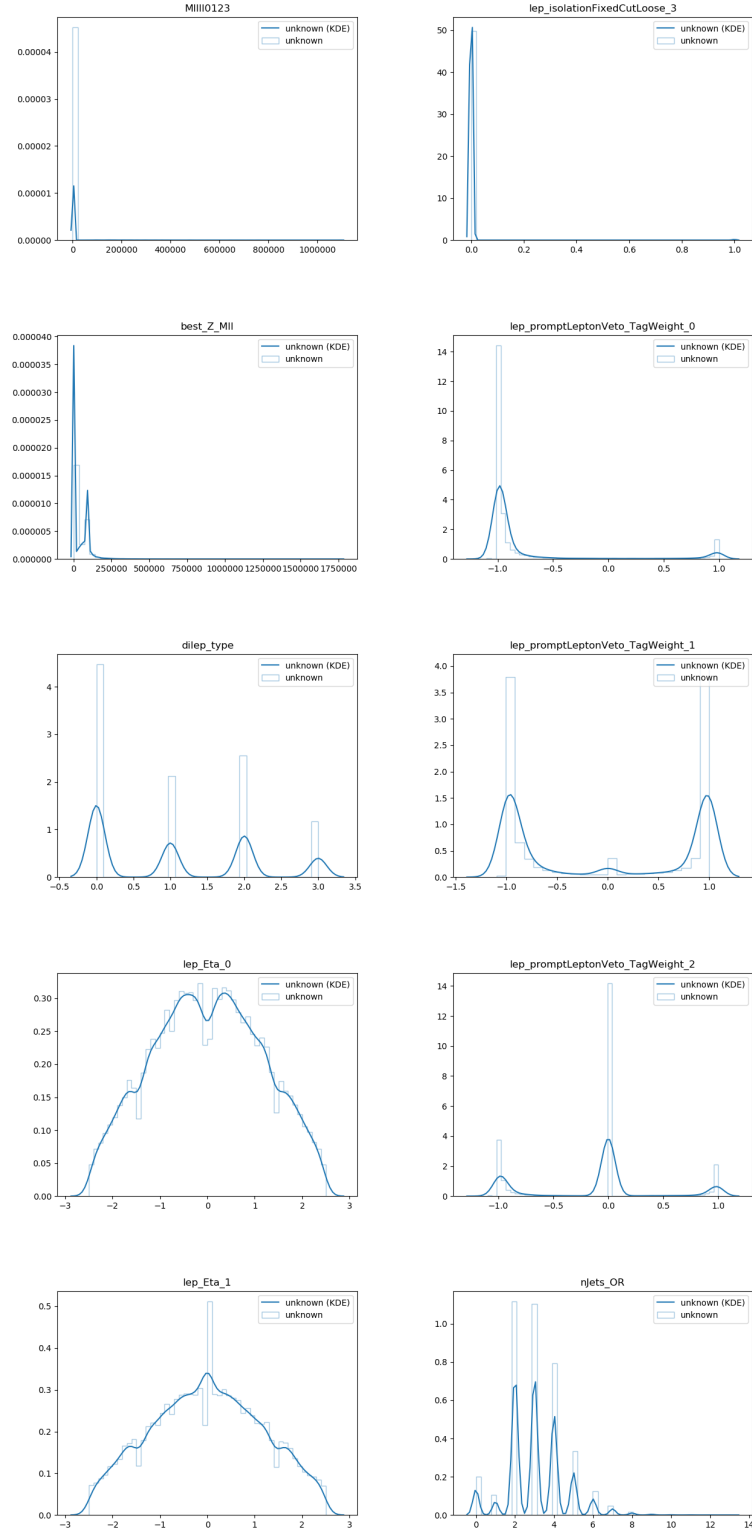


Figure C.3: Real-data histograms (mc16a & mc16d set)

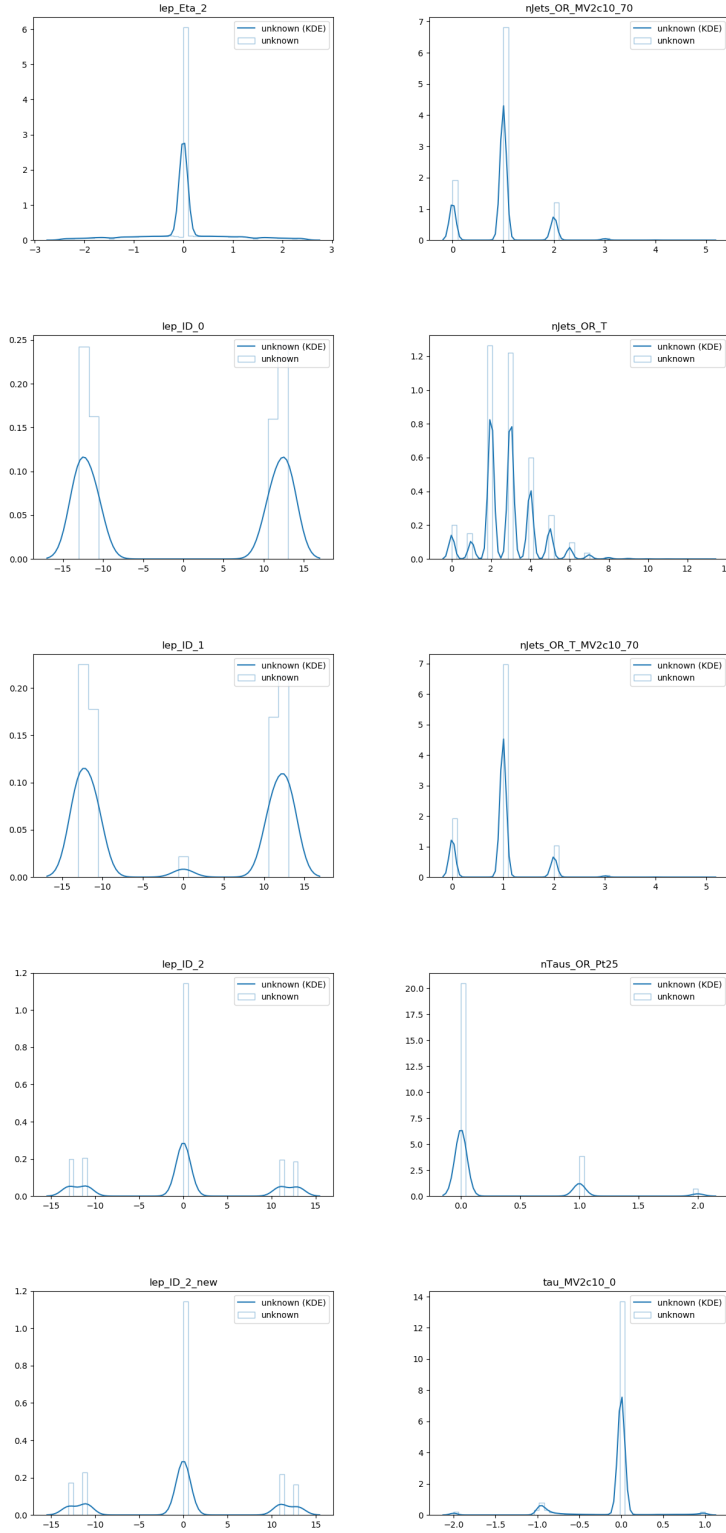


Figure C.3: Real-data histograms (mc16a & mc16d set)

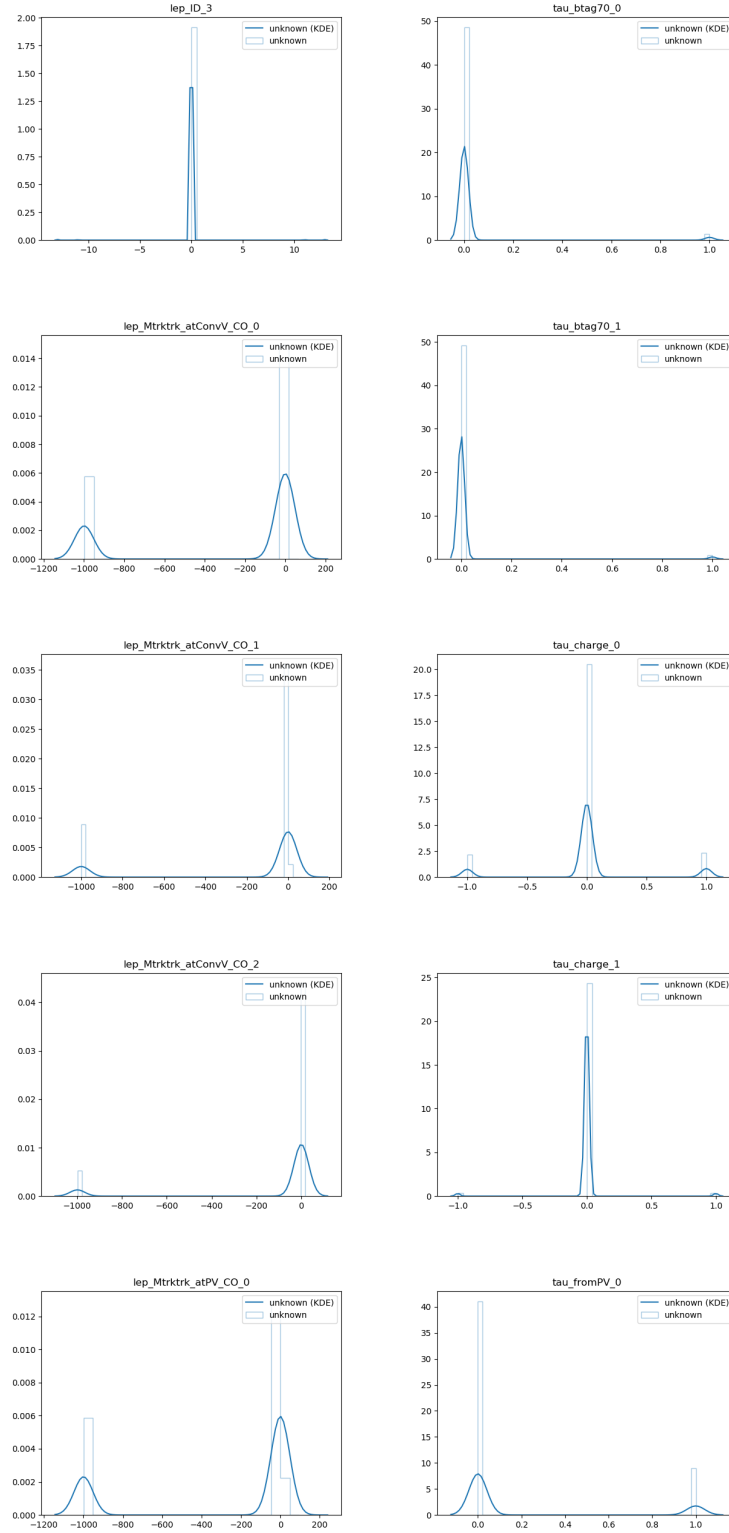


Figure C.3: Real-data histograms (mc16a & mc16d set)

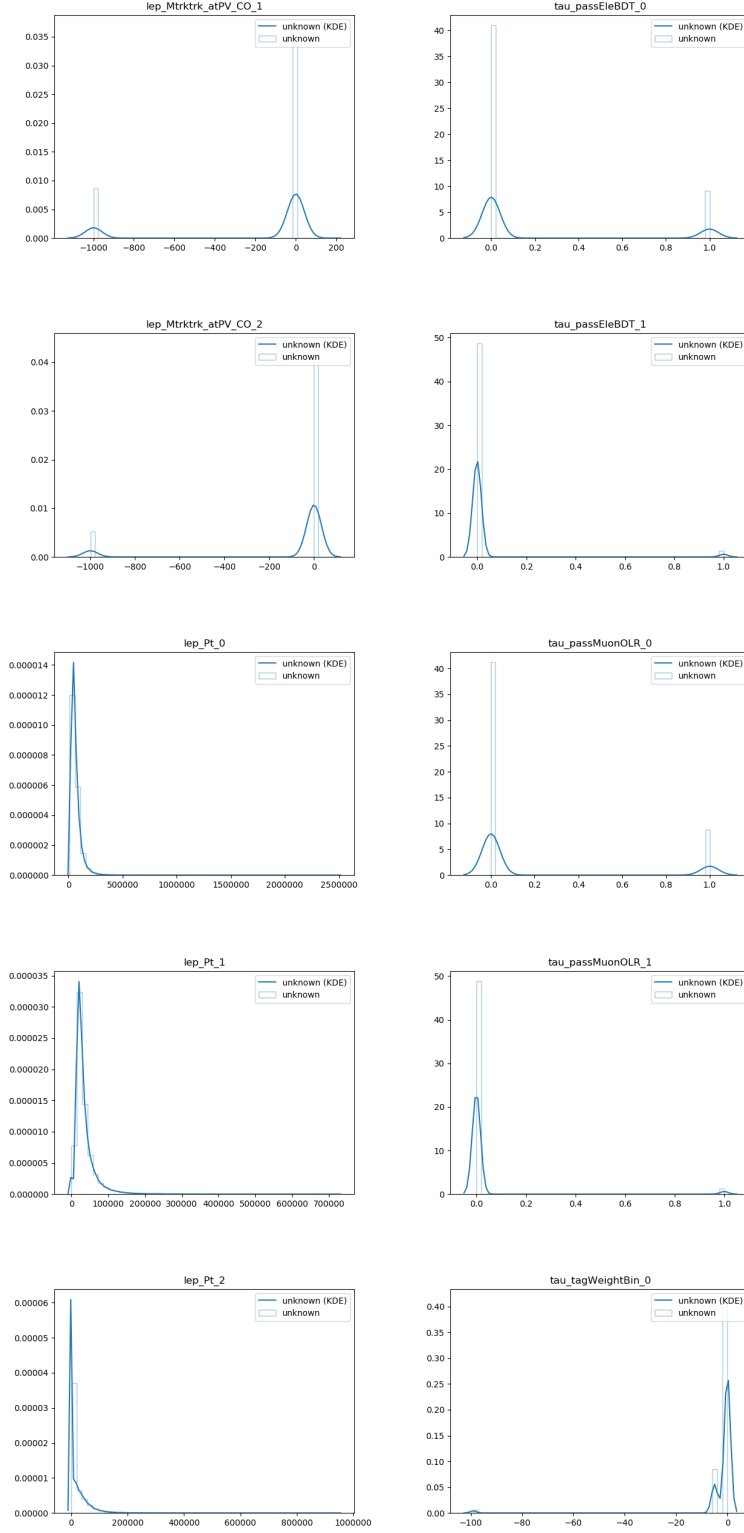


Figure C.3: Real-data histograms (mc16a & mc16d set)

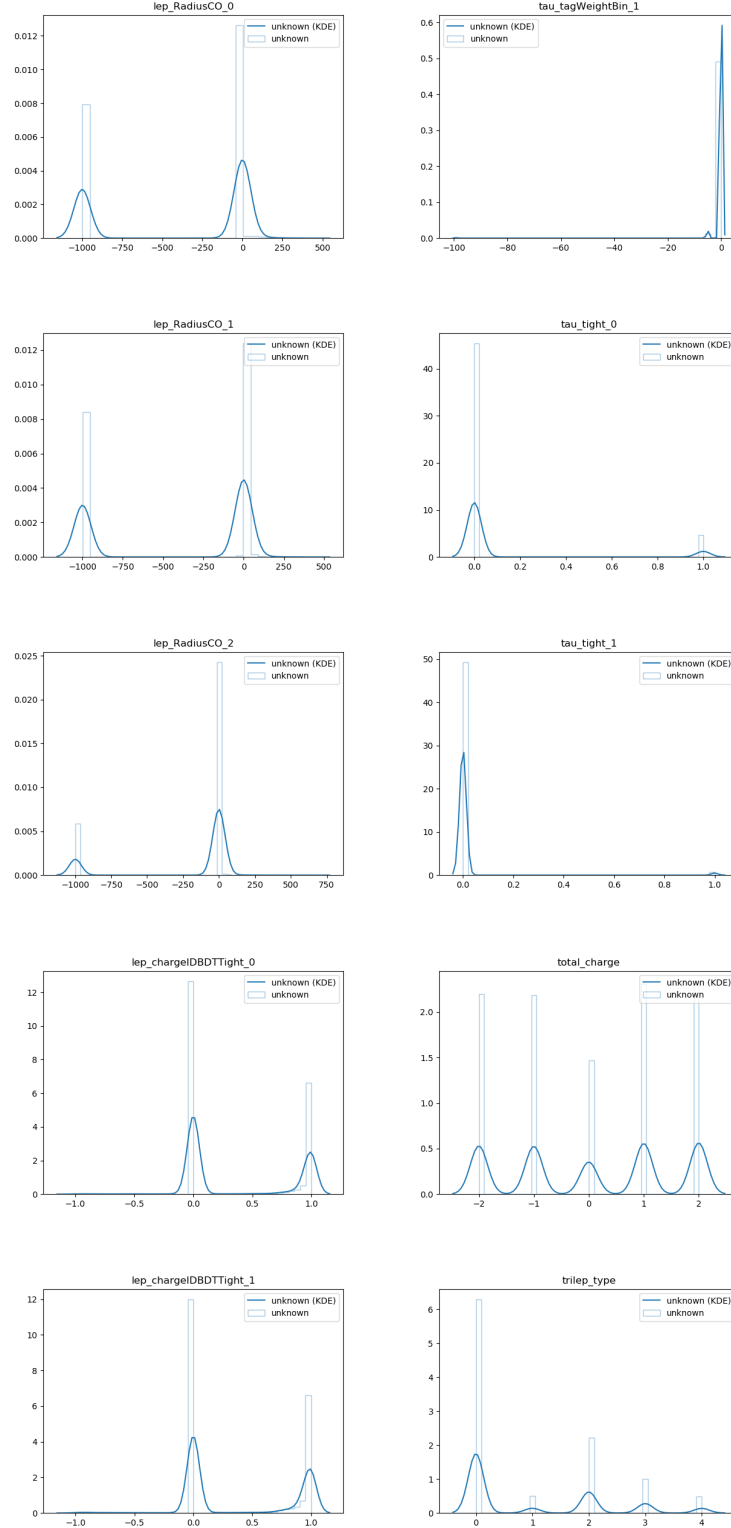


Figure C.3: Real-data histograms (mc16a & mc16d set)

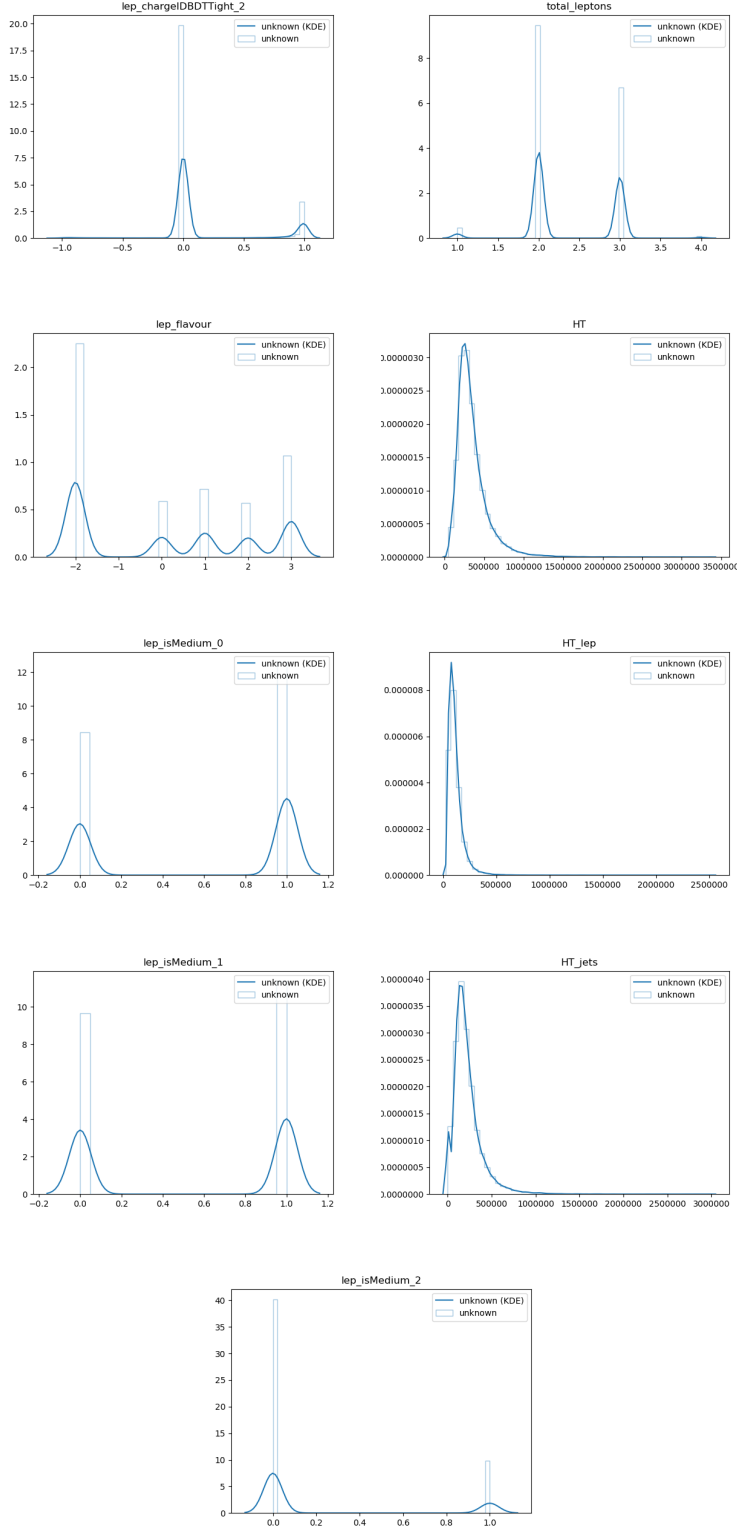
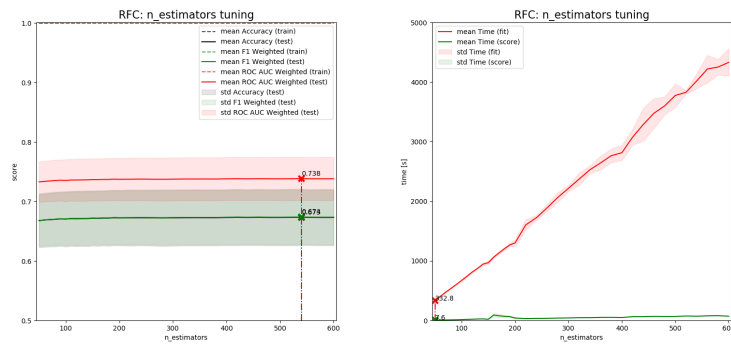


Figure C.3: Real-data histograms (mc16a & mc16d set)

C.2 Tuning

C.2.1 The Random Forest Classifier

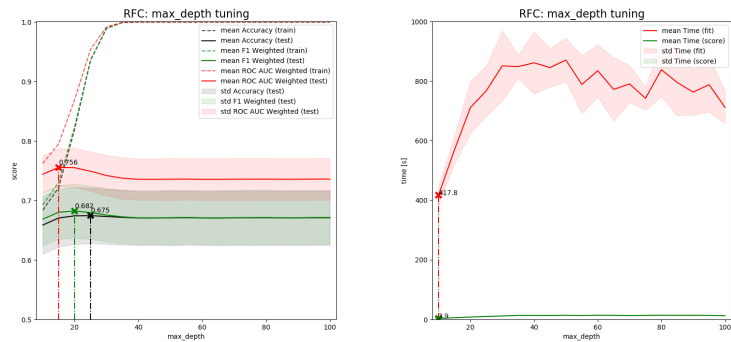
Full-data model



(a) : Score results

(b) : Time results

Figure C.4: n_estimators parameter tuning



(a) : Score results

(b) : Time results

Figure C.5: max_depth parameter tuning

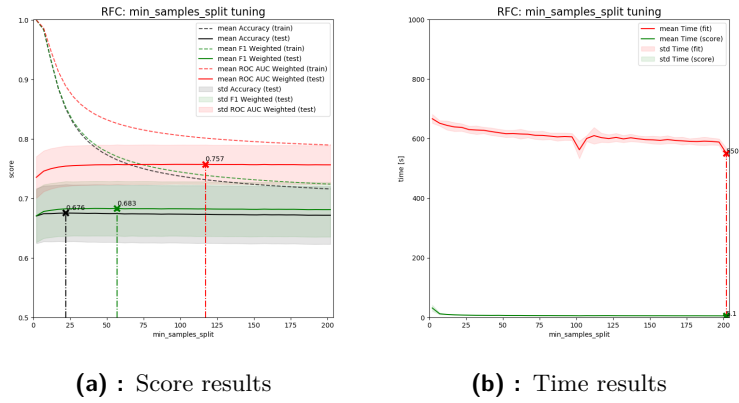


Figure C.6: min_samples_split parameter tuning

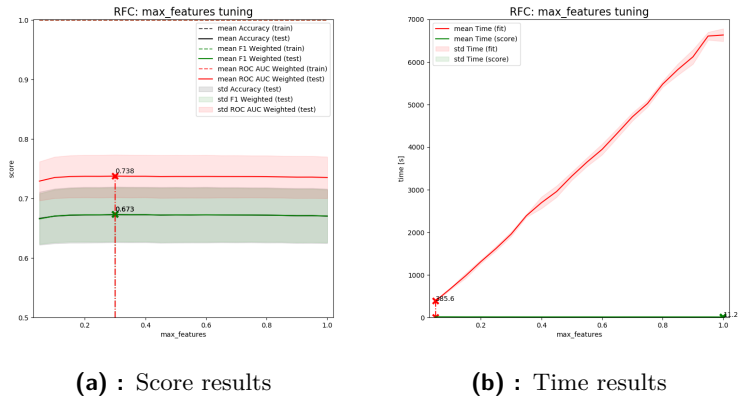


Figure C.7: max_features parameter tuning

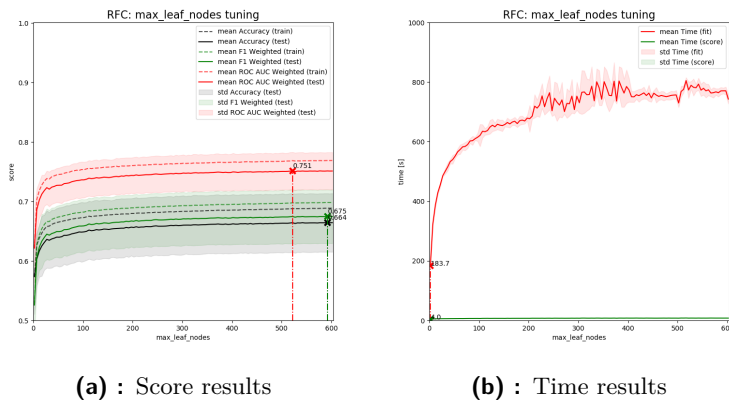


Figure C.8: max_leaf_nodes parameter tuning

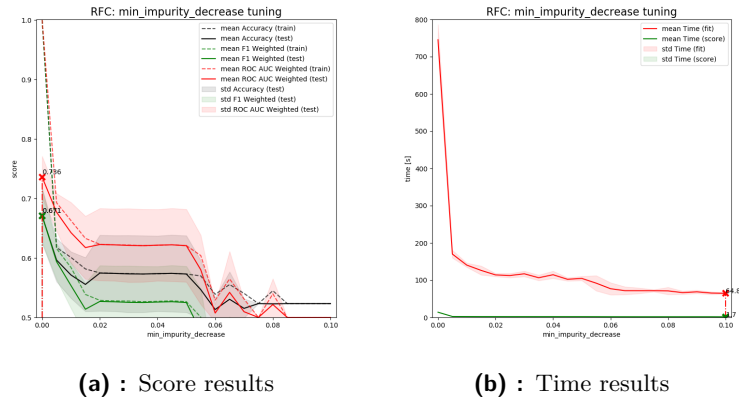


Figure C.9: min_impurity_decrease parameter tuning

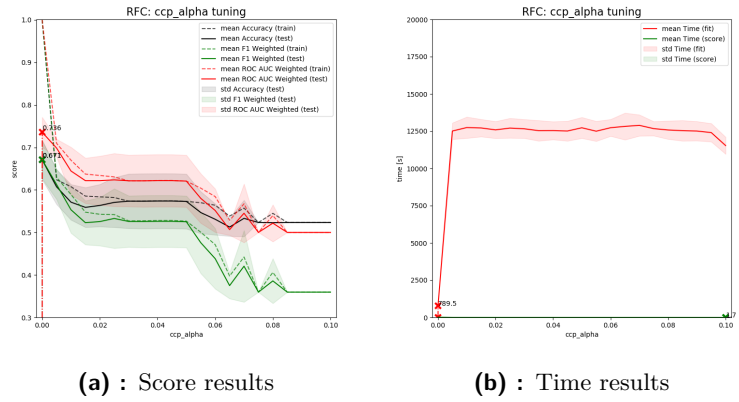


Figure C.10: ccp_alpha parameter tuning

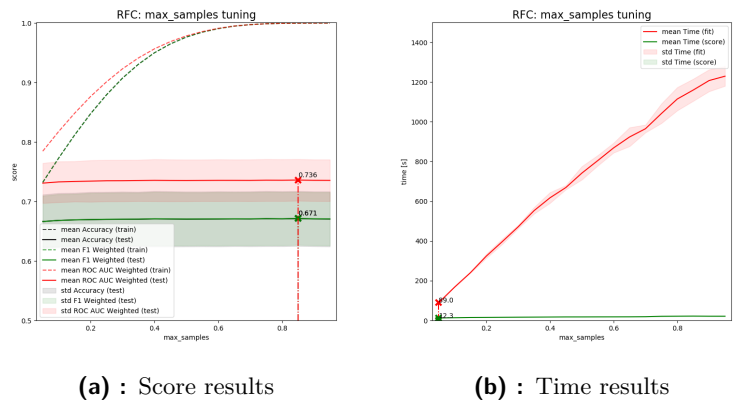
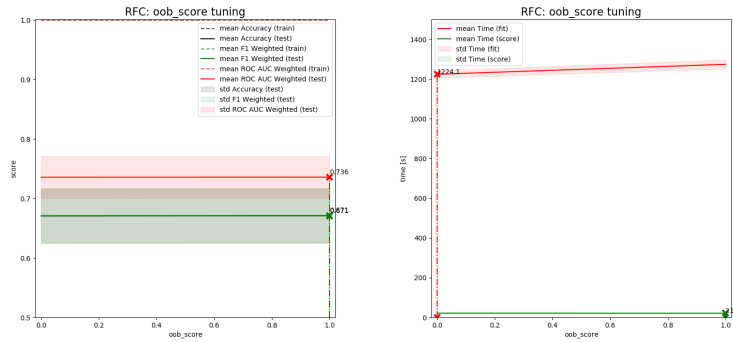
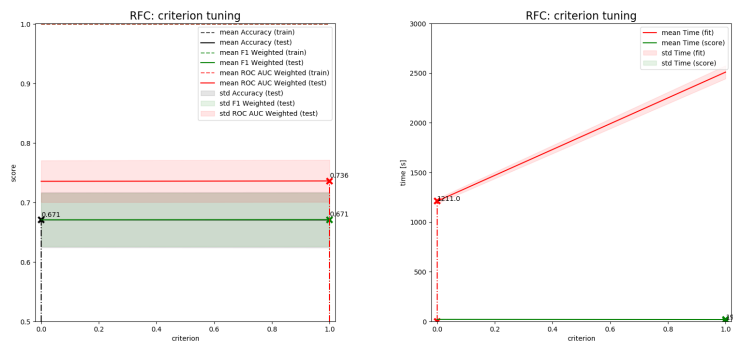


Figure C.11: max_samples parameter tuning



(a) : Score results

(b) : Time results

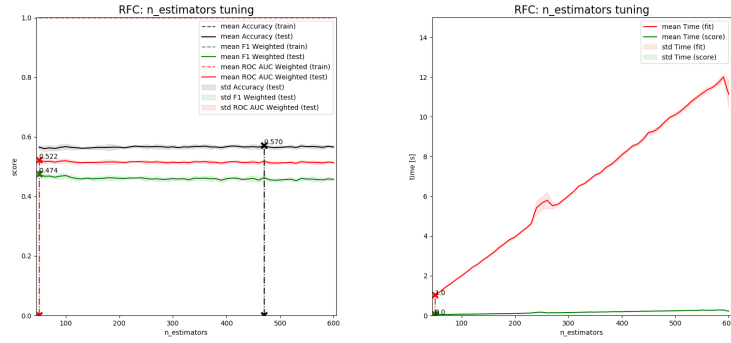
Figure C.12: oob_score parameter tuning (0: False, 1: True)

(a) : Score results

(b) : Time results

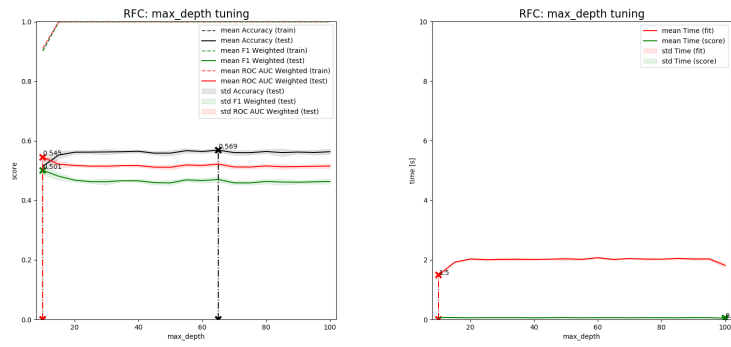
Figure C.13: criterion parameter tuning (0: 'gini', 1: 'entropy')

Channel-data model



(a) : Score results

(b) : Time results

Figure C.14: `n_estimators` parameter tuning

(a) : Score results

(b) : Time results

Figure C.15: `max_depth` parameter tuning

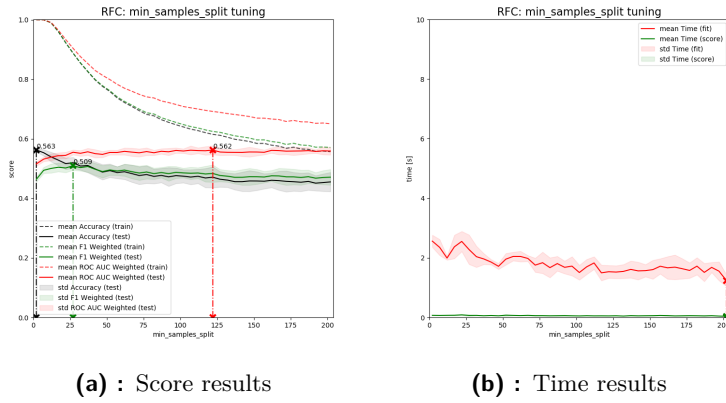


Figure C.16: min_samples_split parameter tuning

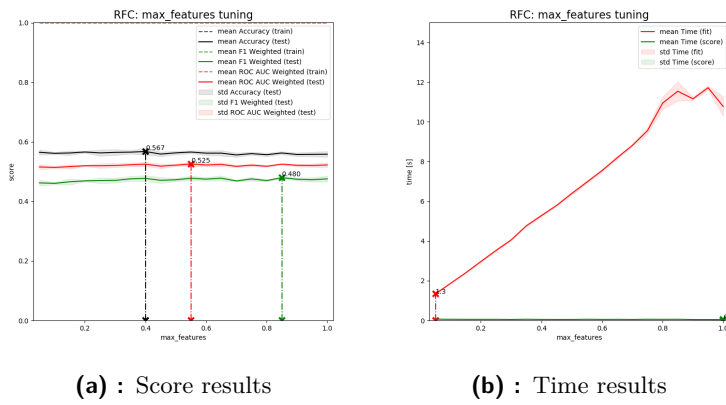


Figure C.17: max_features parameter tuning

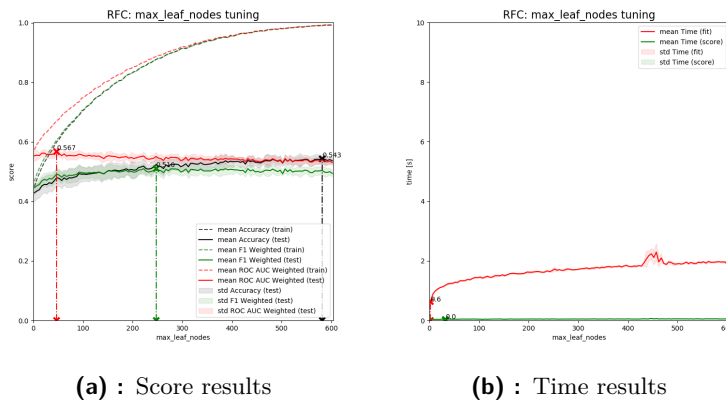


Figure C.18: max_leaf_nodes parameter tuning

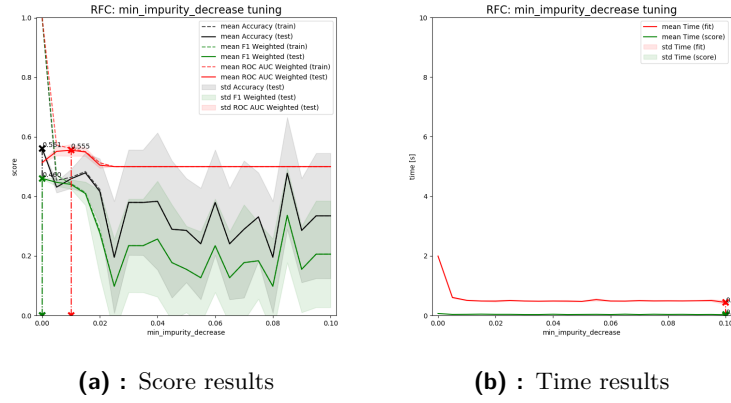


Figure C.19: min_impurity_decrease parameter tuning

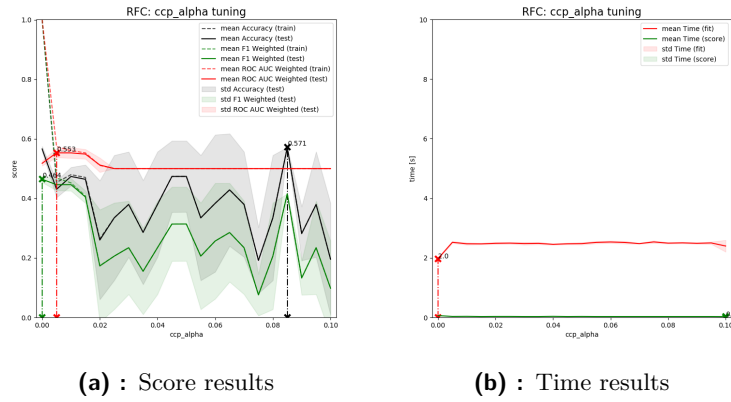


Figure C.20: ccp_alpha parameter tuning

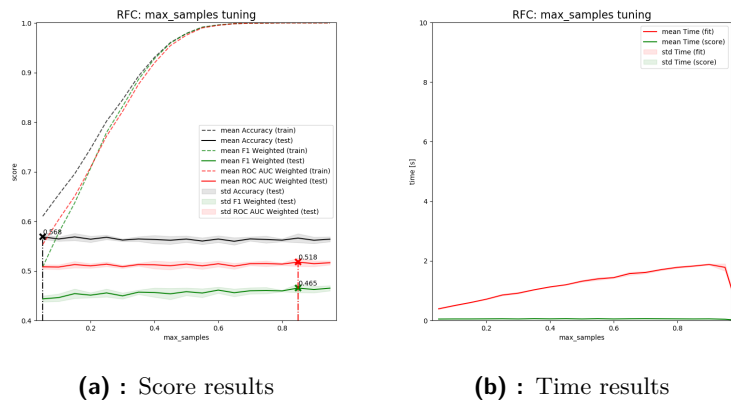
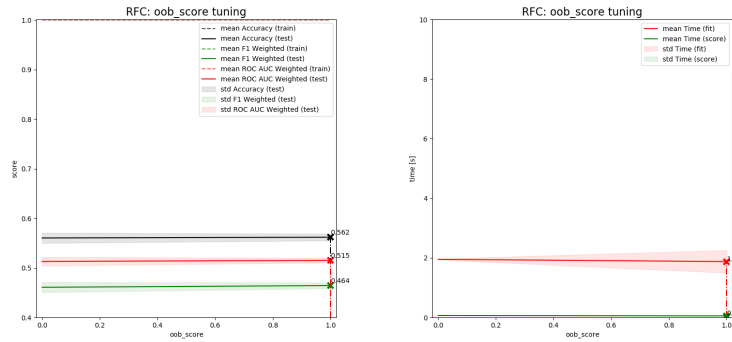
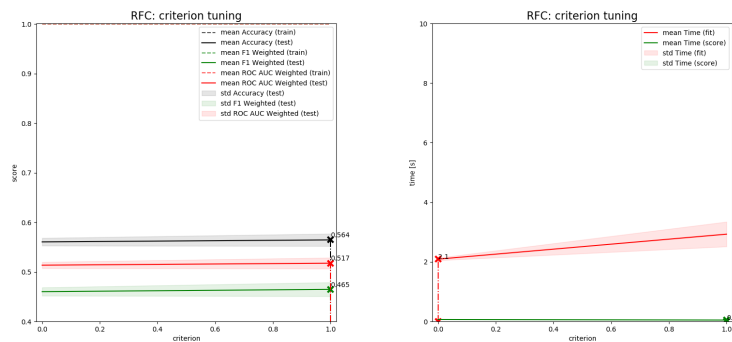


Figure C.21: max_samples parameter tuning



(a) : Score results

(b) : Time results

Figure C.22: `oob_score` parameter tuning (0: False, 1: True)

(a) : Score results

(b) : Time results

Figure C.23: `criterion` parameter tuning (0: 'gini', 1: 'entropy')

C.2.2 The K-Neighbors Classifier

Full-data model

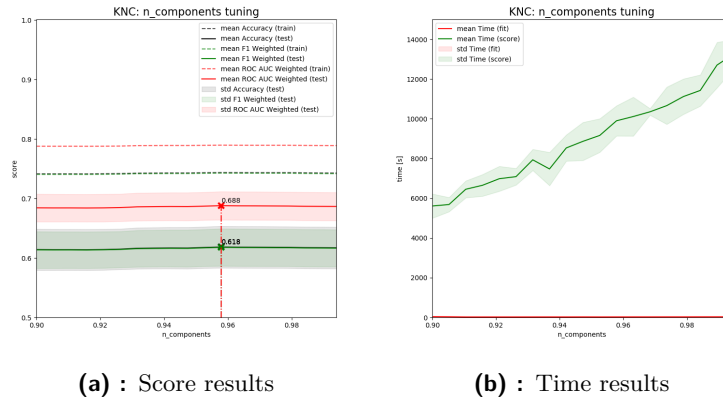


Figure C.24: n_components PCA parameter tuning

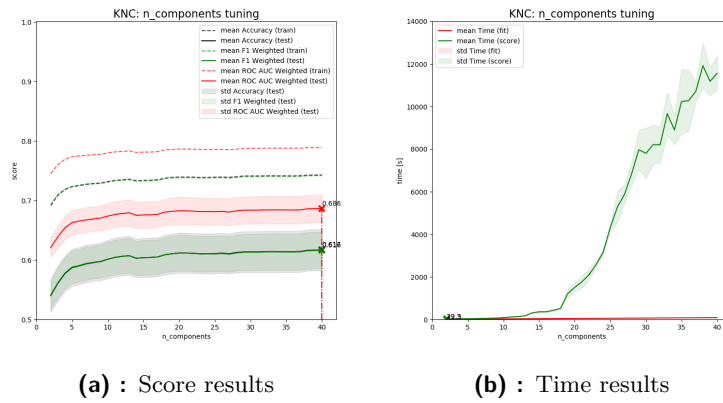


Figure C.25: n_components TSVD parameter tuning

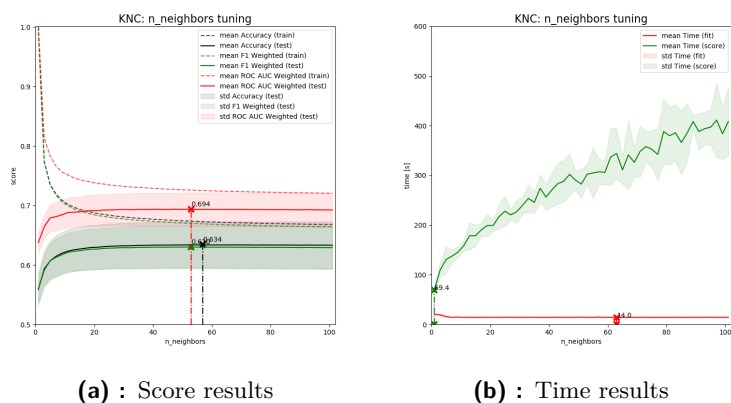


Figure C.26: n_neighbors parameter tuning

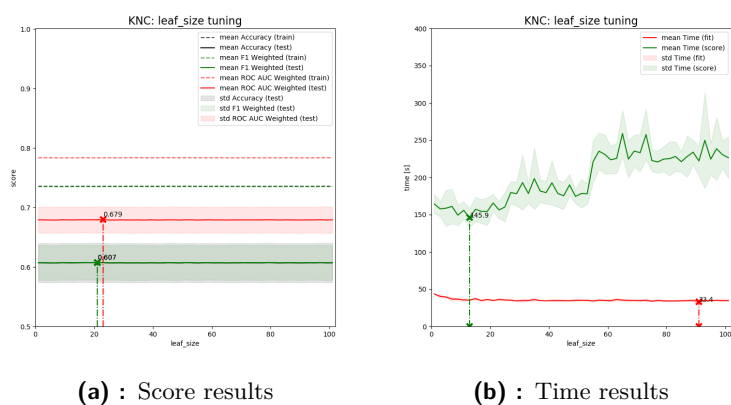


Figure C.27: leaf_size parameter tuning

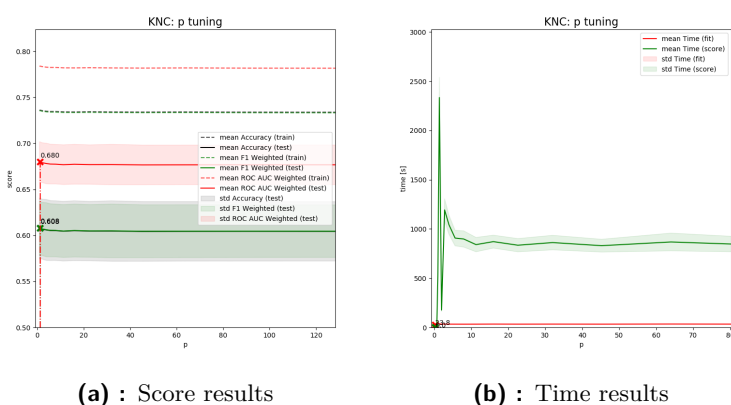


Figure C.28: p parameter tuning

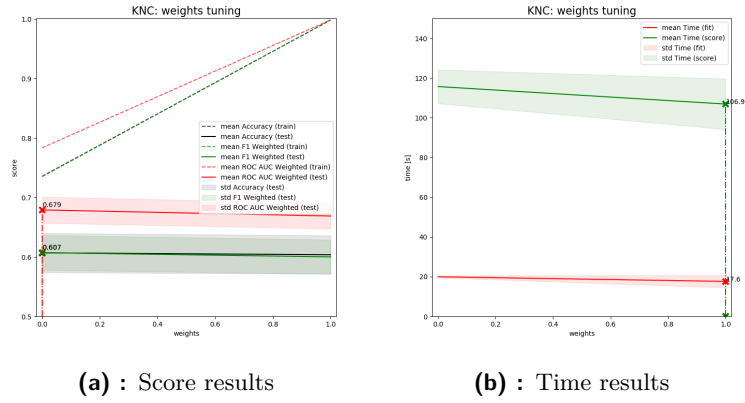


Figure C.29: weights parameter tuning (0: 'uniform', 1: 'distance')

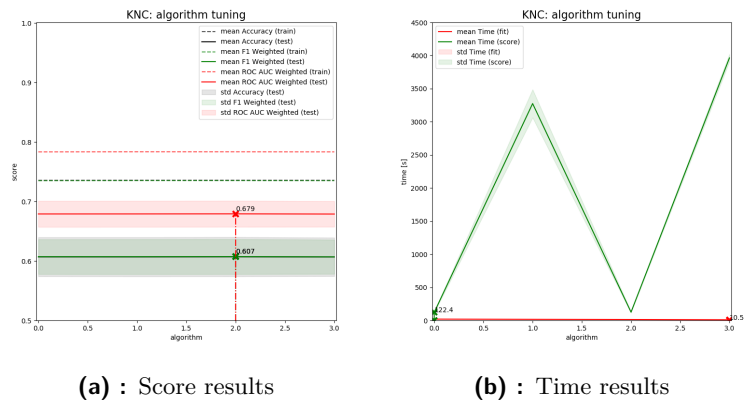
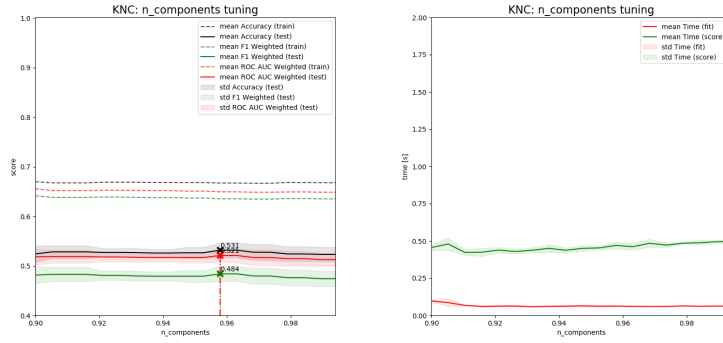


Figure C.30: algorithm parameter tuning (0: 'auto', 1: 'ball_tree', 2: 'kd_tree', 3: 'brute')

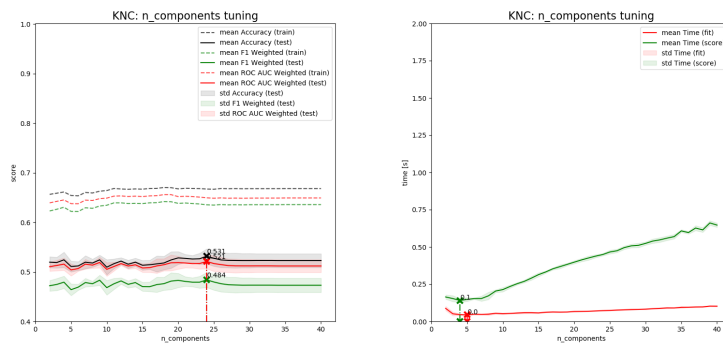
Channel-data model



(a) : Score results

(b) : Time results

Figure C.31: n_components PCA parameter tuning



(a) : Score results

(b) : Time results

Figure C.32: n_components TSVD parameter tuning

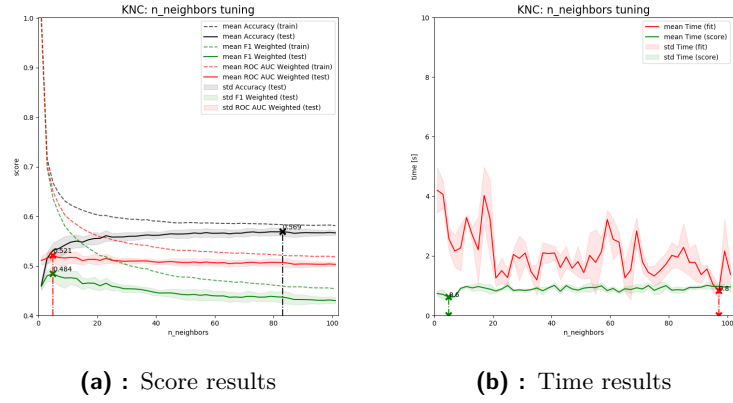


Figure C.33: n_neighbors parameter tuning

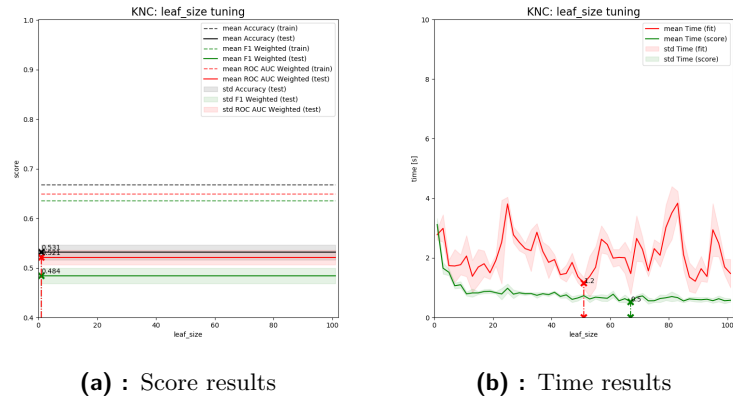


Figure C.34: leaf_size parameter tuning

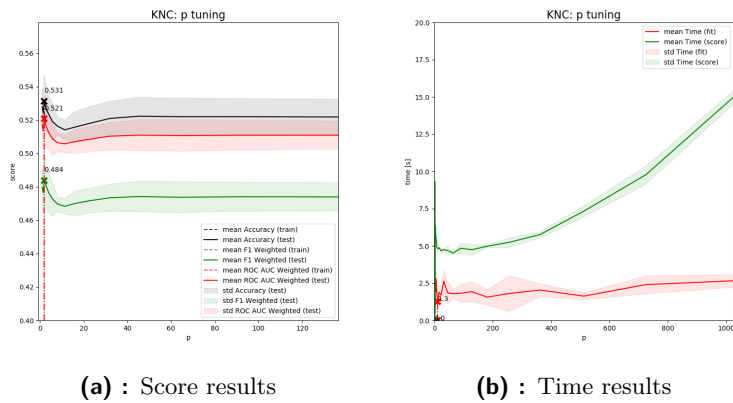


Figure C.35: p parameter tuning

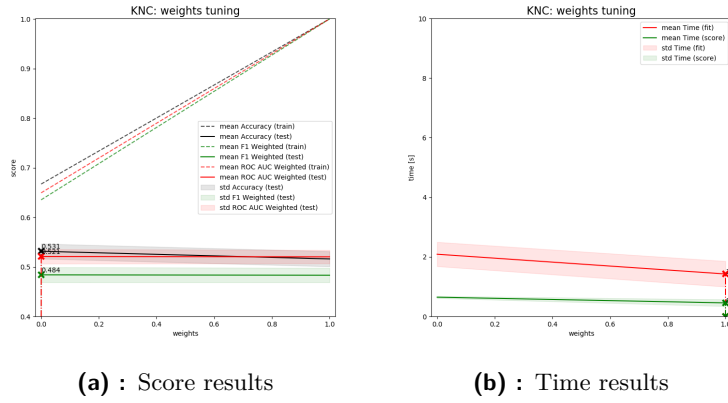


Figure C.36: weights parameter tuning (0: 'uniform', 1: 'distance')

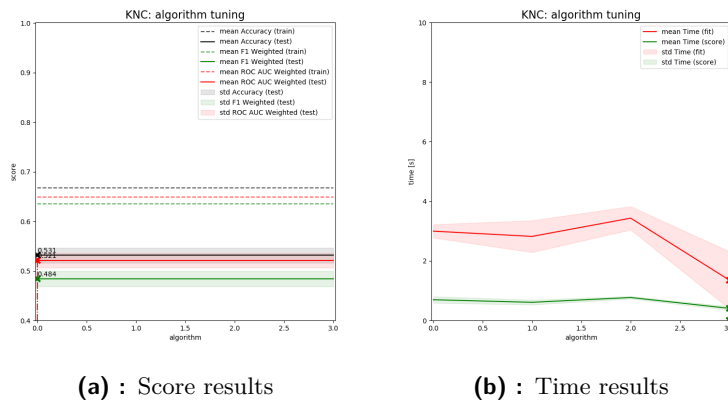


Figure C.37: algorithm parameter tuning (0: 'auto', 1: 'ball_tree', 2: 'kd_tree', 3: 'brute')

C.2.3 Gaussian Naïve Bayes

Full-data model

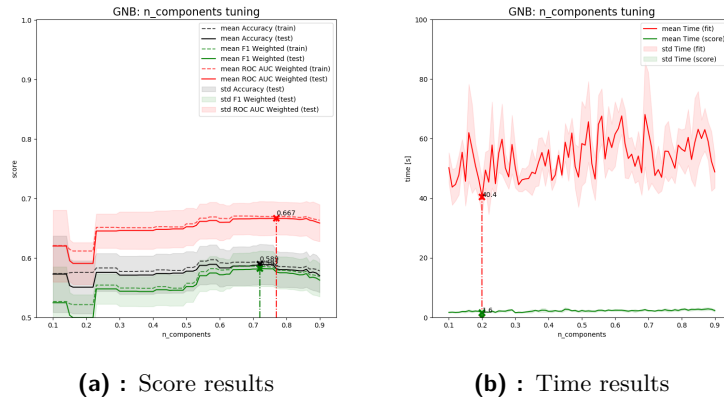


Figure C.38: n_components PCA parameter tuning

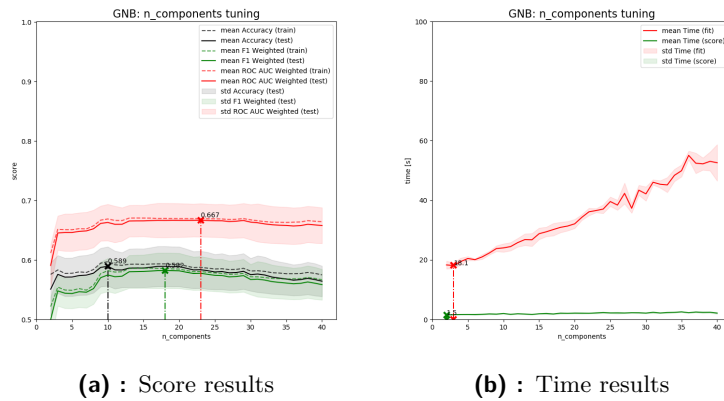
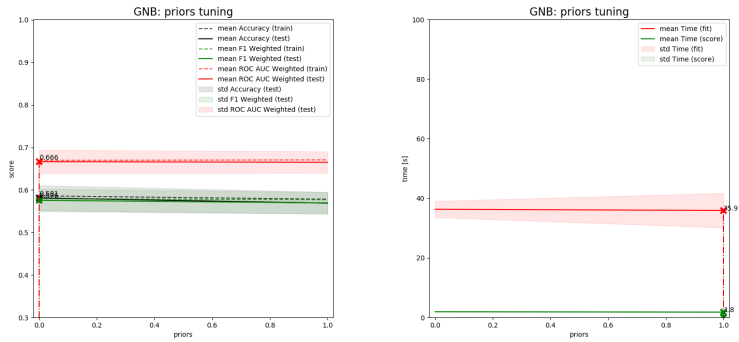


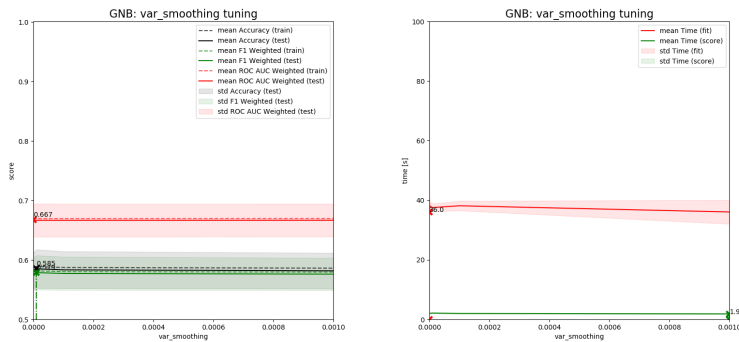
Figure C.39: n_components TSVD parameter tuning



(a) : Score results

(b) : Time results

Figure C.40: priors parameter tuning

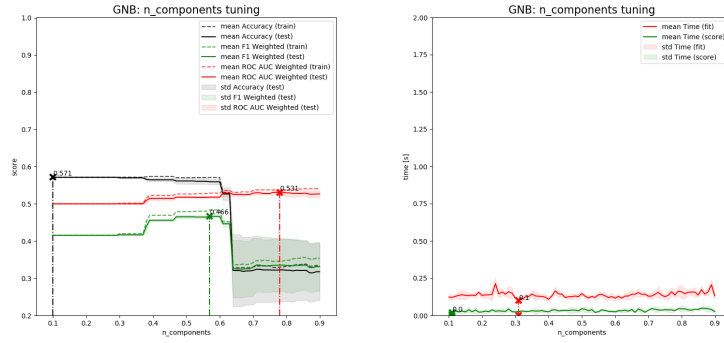


(a) : Score results

(b) : Time results

Figure C.41: var_smoothing parameter tuning

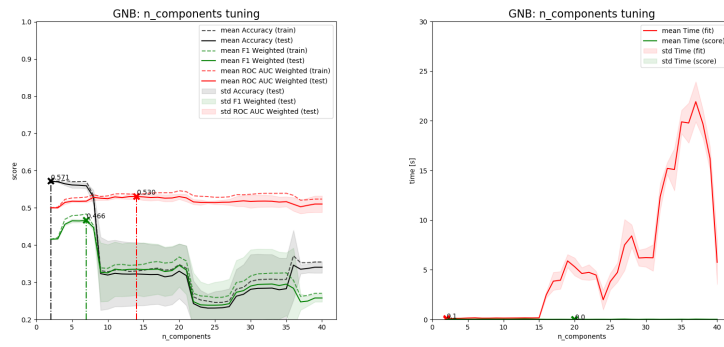
Channel-data model



(a) : Score results

(b) : Time results

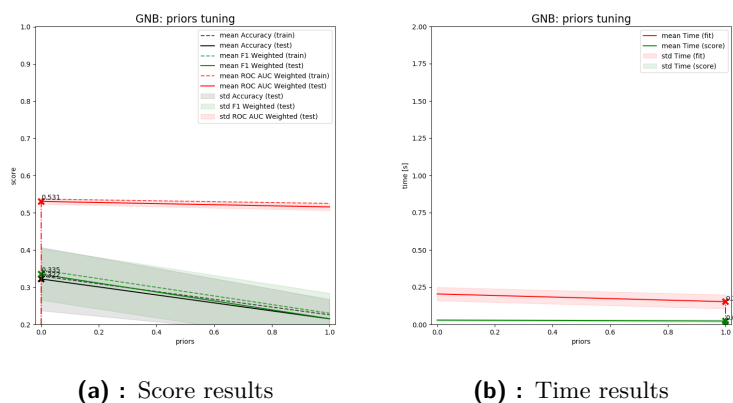
Figure C.42: n_components PCA parameter tuning



(a) : Score results

(b) : Time results

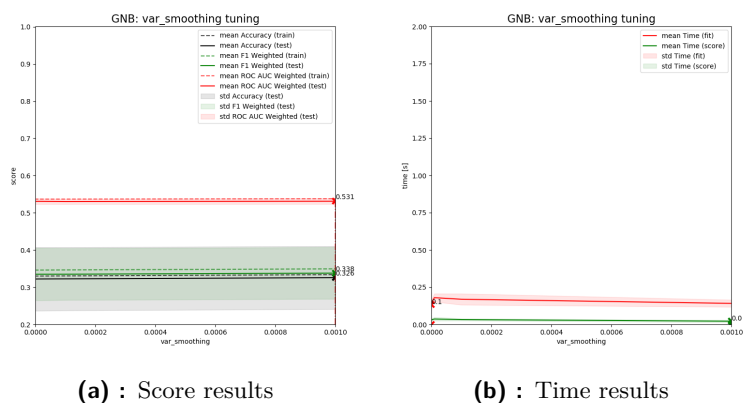
Figure C.43: n_components TSVD parameter tuning



(a) : Score results

(b) : Time results

Figure C.44: priors parameter tuning



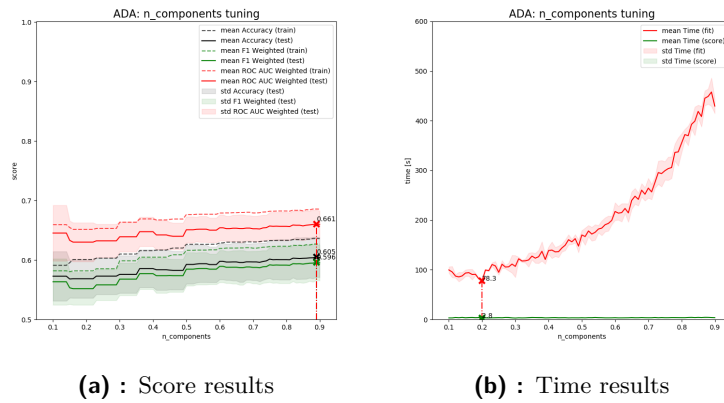
(a) : Score results

(b) : Time results

Figure C.45: var_smoothing parameter tuning

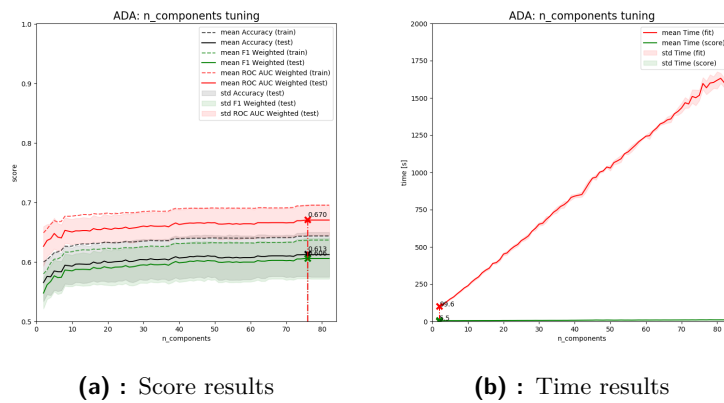
C.2.4 AdaBoost Classifier

Full-data model



(a) : Score results

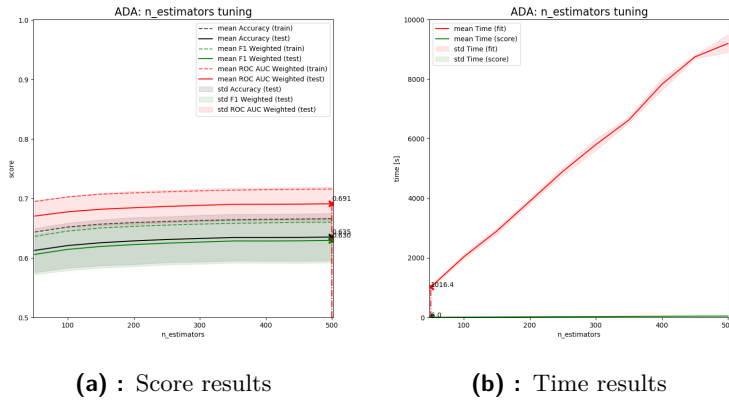
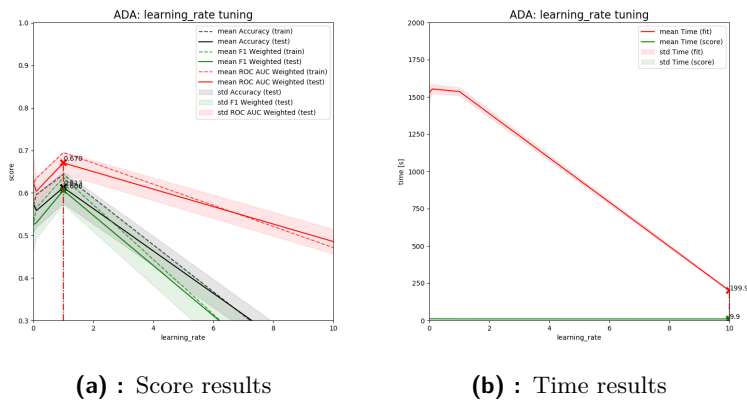
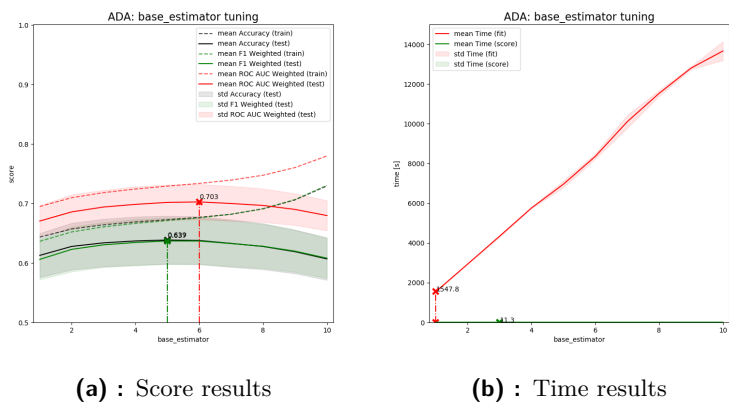
(b) : Time results

Figure C.46: $n_{\text{components}}$ PCA parameter tuning

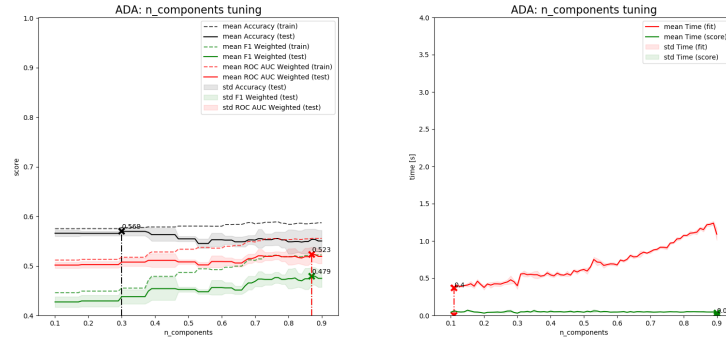
(a) : Score results

(b) : Time results

Figure C.47: $n_{\text{components}}$ TSVD parameter tuning

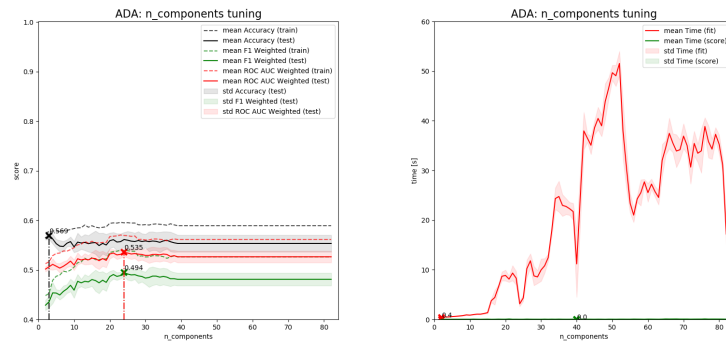
Figure C.48: `n_estimators` parameter tuningFigure C.49: `learning_rate` parameter tuningFigure C.50: `base_estimator` parameter tuning (depth parameter of DT)

Channel-data model



(a) : Score results

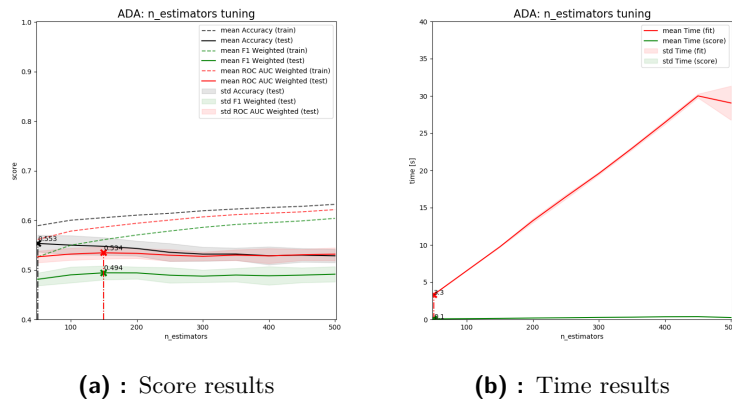
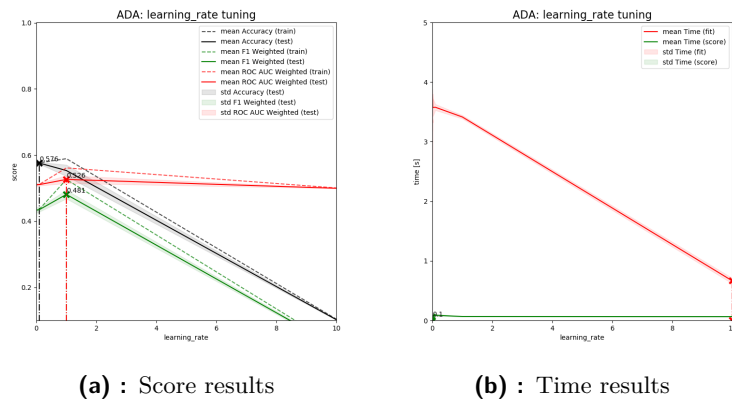
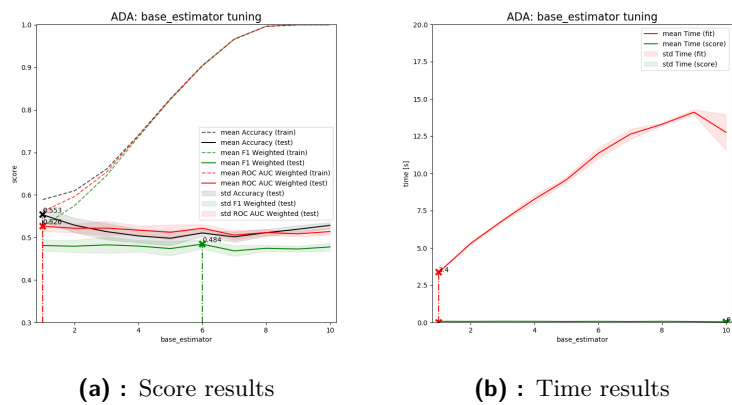
(b) : Time results

Figure C.51: $n_{\text{components}}$ PCA parameter tuning

(a) : Score results

(b) : Time results

Figure C.52: $n_{\text{components}}$ TSVD parameter tuning

Figure C.53: `n_estimators` parameter tuningFigure C.54: `learning_rate` parameter tuningFigure C.55: `base_estimator` parameter tuning (depth parameter of DT)

C.2.5 Gradient Boosting Classifier

Full-data model

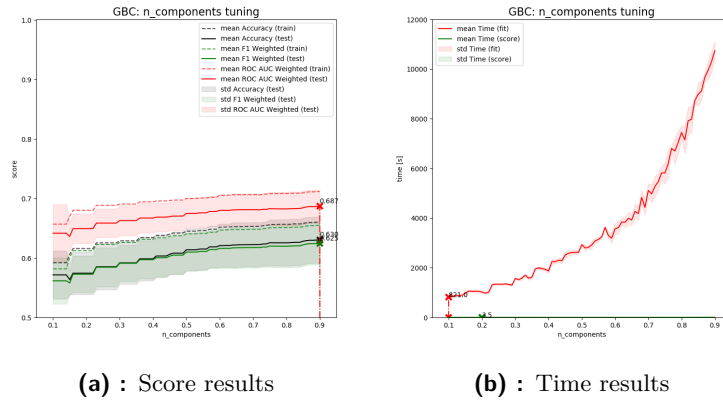


Figure C.56: n_components PCA parameter tuning

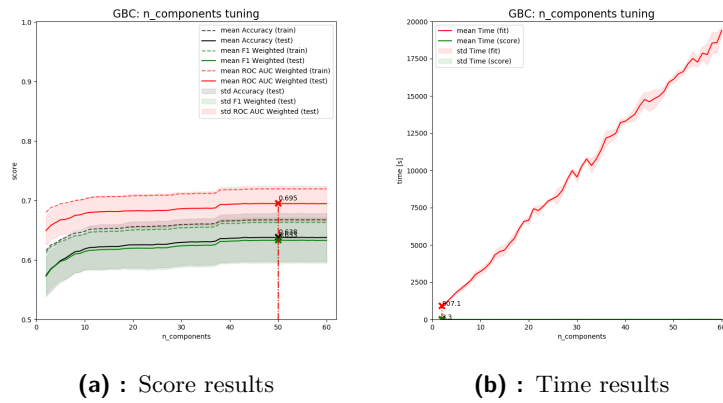
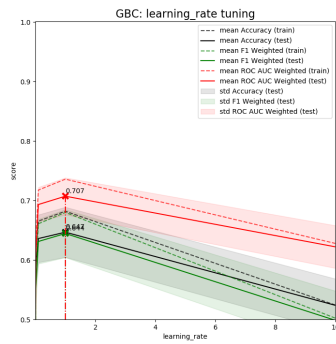
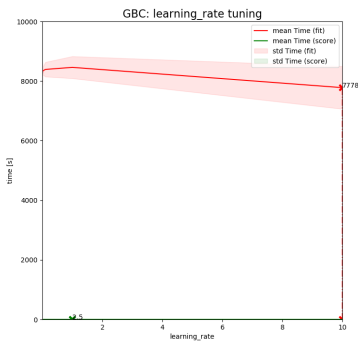


Figure C.57: n_components TSVD parameter tuning

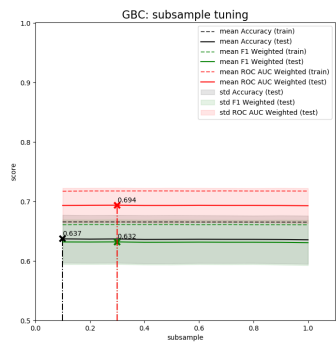


(a) : Score results

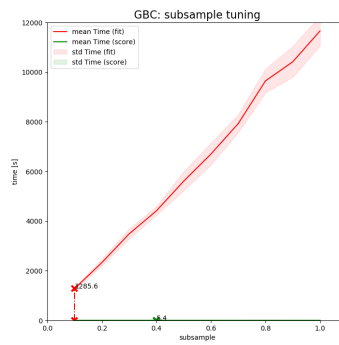


(b) : Time results

Figure C.58: learning_rate parameter tuning

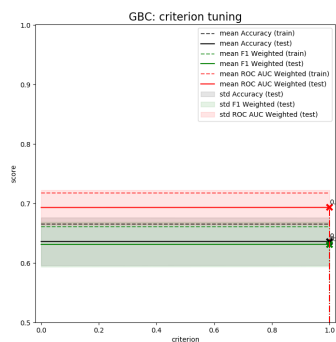


(a) : Score results

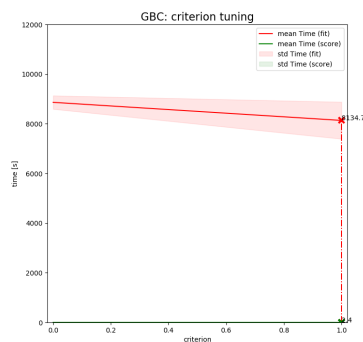


(b) : Time results

Figure C.59: subsample parameter tuning

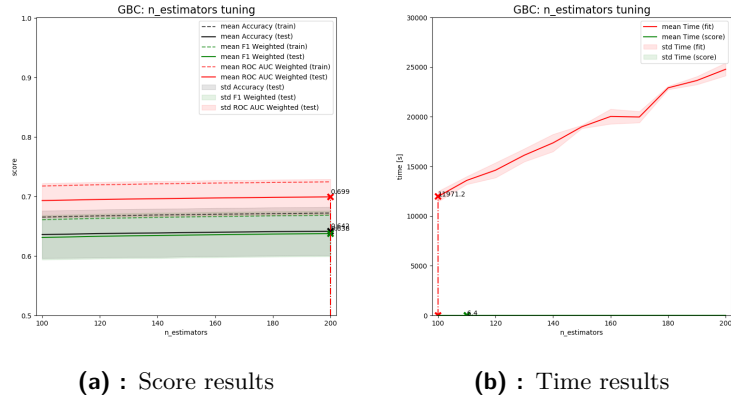
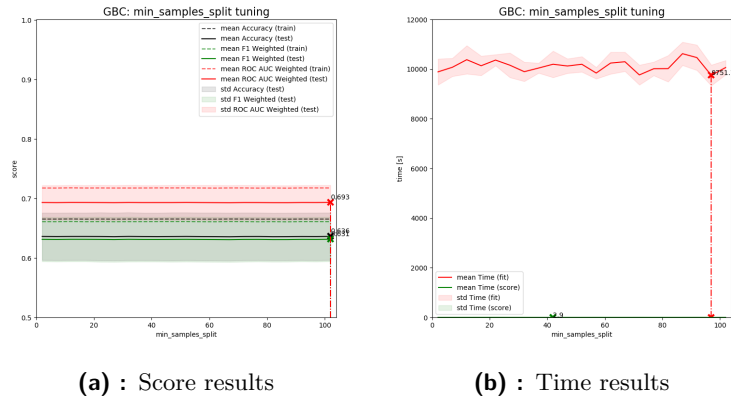
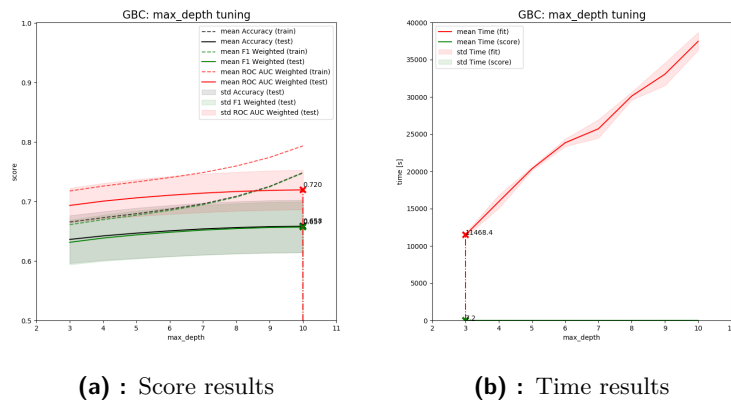


(a) : Score results



(b) : Time results

Figure C.60: criterion parameter tuning (0: 'friedman_mse', 1: 'mse')

Figure C.61: `n_estimators` parameter tuningFigure C.62: `min_samples_split` parameter tuningFigure C.63: `max_depth` parameter tuning

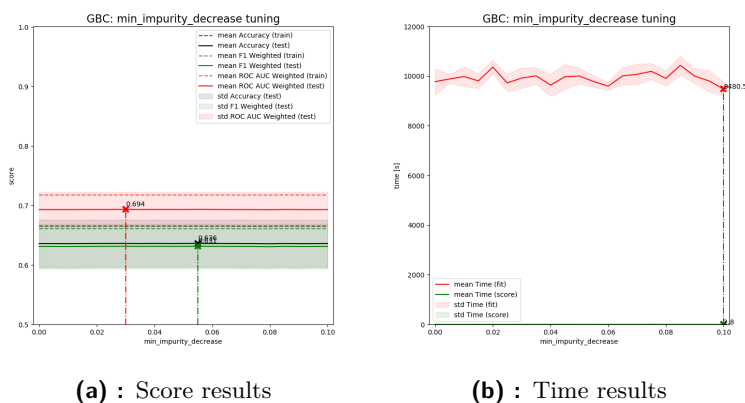


Figure C.64: min_impurity_decrease parameter tuning

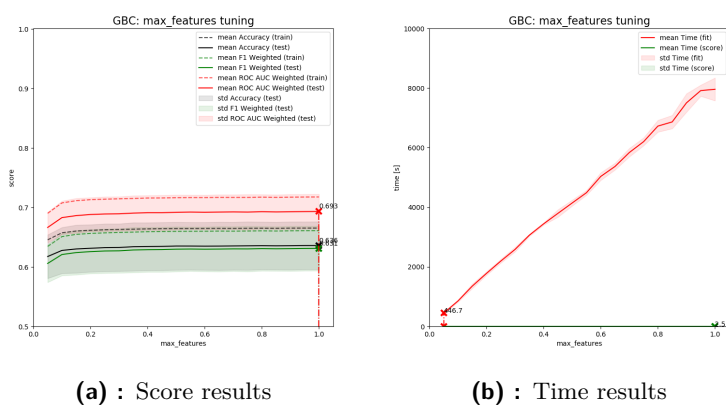


Figure C.65: max_features parameter tuning

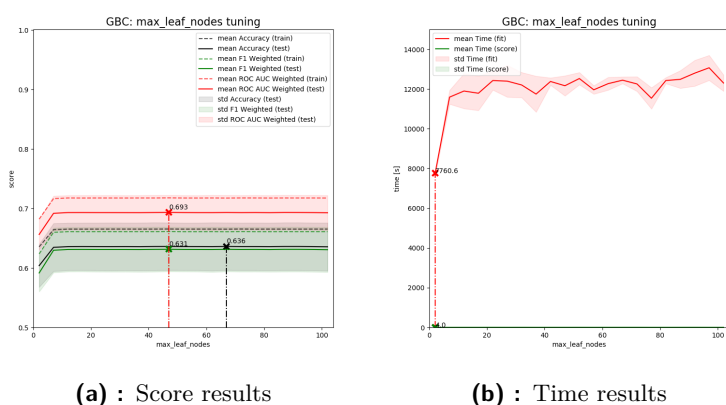
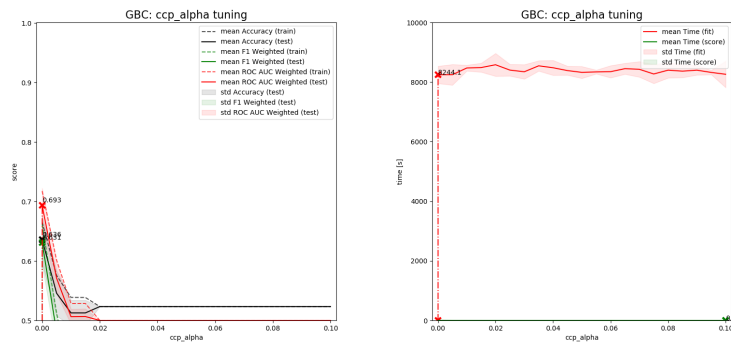


Figure C.66: max_leaf_nodes parameter tuning



(a) : Score results

(b) : Time results

Figure C.67: ccp_alpha parameter tuning

Channel-data model

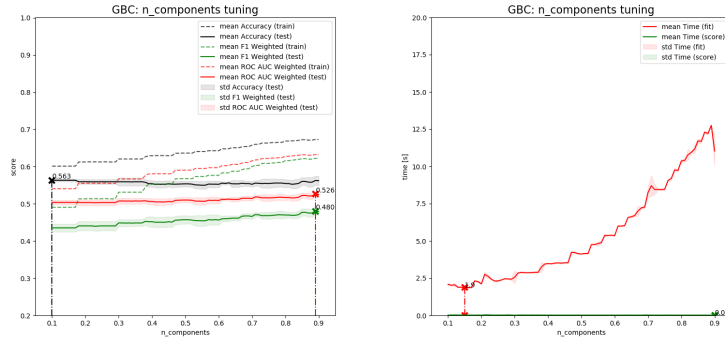


Figure C.68: n_components PCA parameter tuning

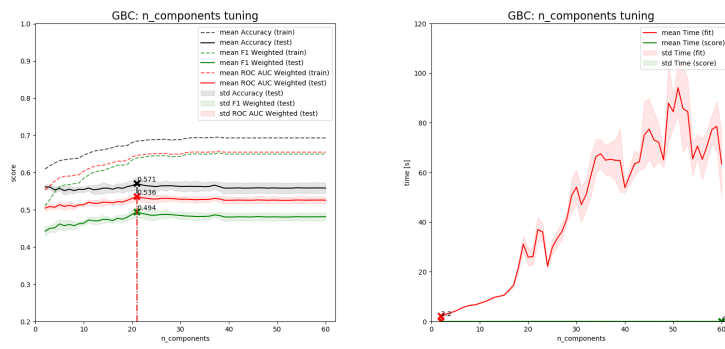


Figure C.69: n_components TSVD parameter tuning

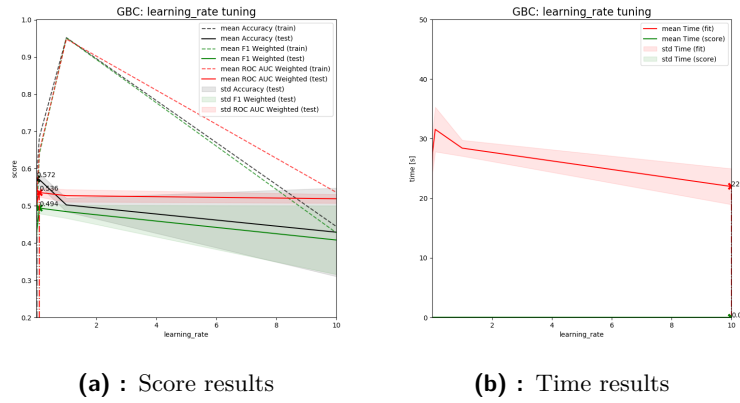


Figure C.70: learning_rate parameter tuning

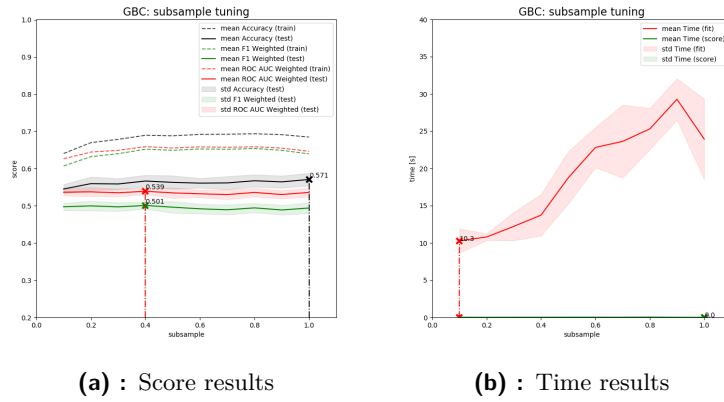


Figure C.71: subsample parameter tuning

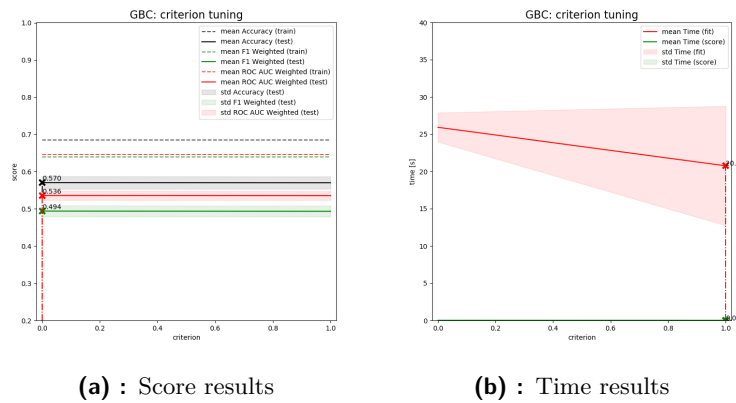
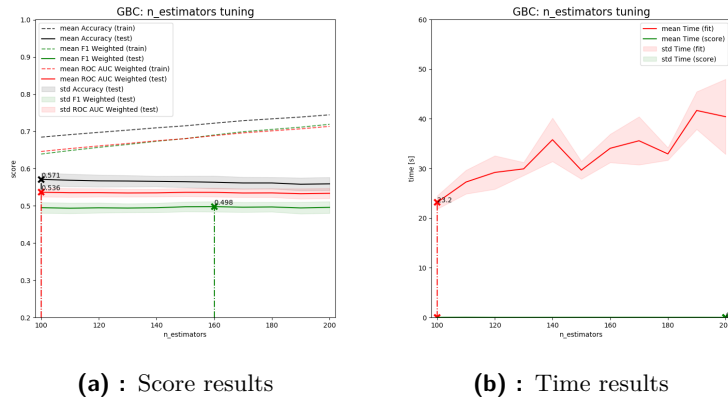
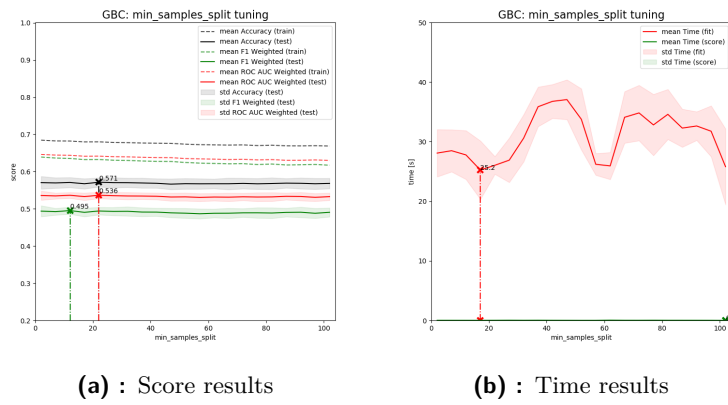
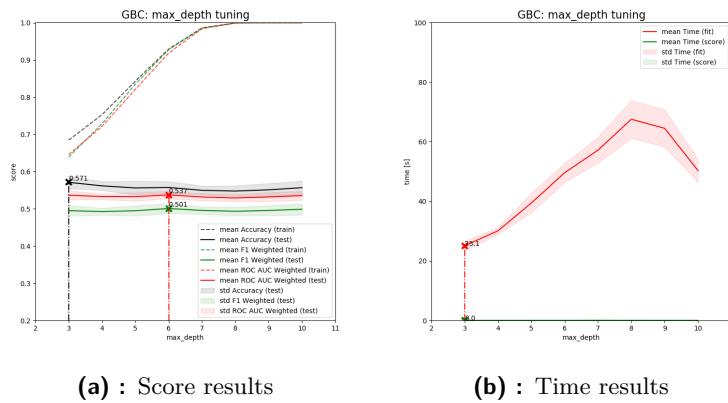


Figure C.72: criterion parameter tuning (0: 'friedman_mse', 1: 'mse')

Figure C.73: `n_estimators` parameter tuningFigure C.74: `min_samples_split` parameter tuningFigure C.75: `max_depth` parameter tuning

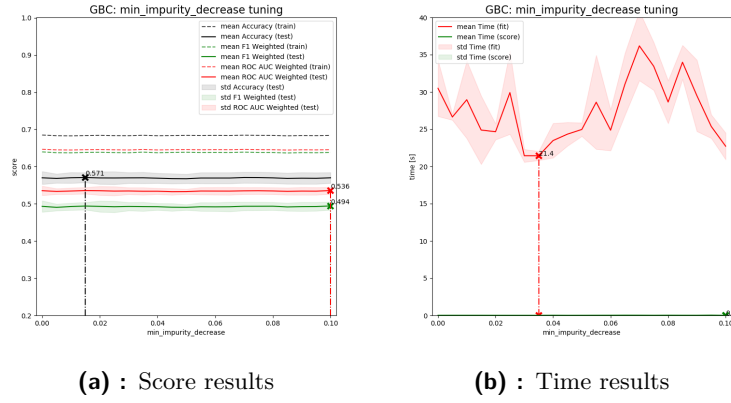


Figure C.76: min_impurity_decrease parameter tuning

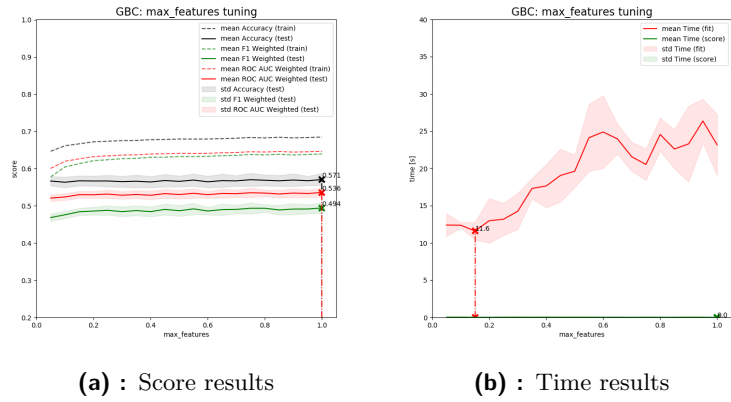


Figure C.77: max_features parameter tuning

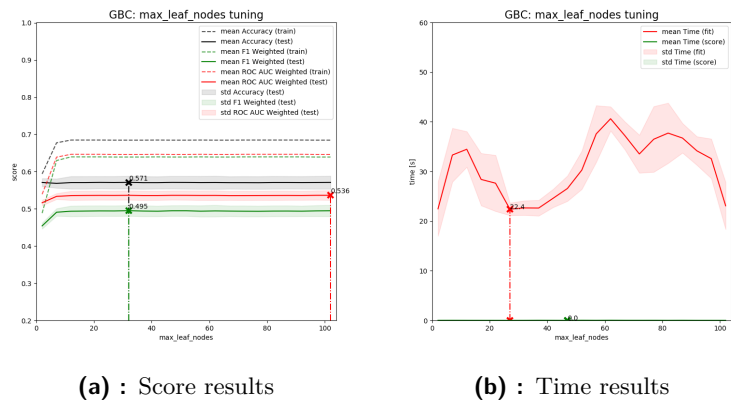
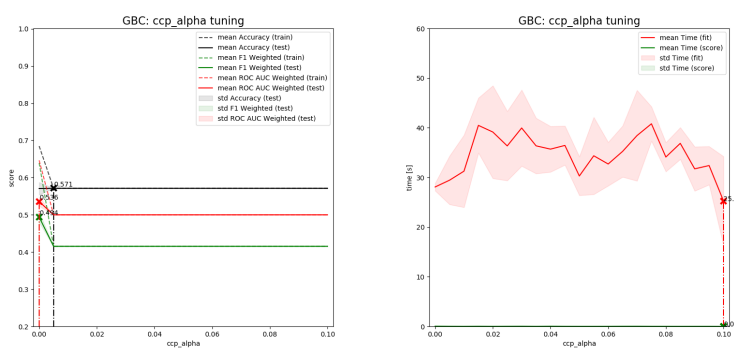


Figure C.78: max_leaf_nodes parameter tuning



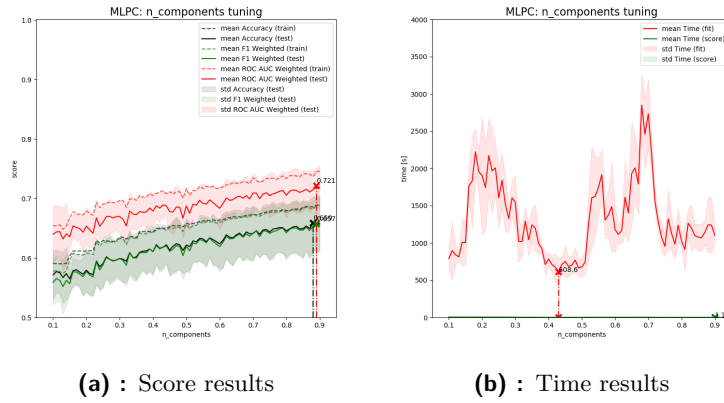
(a) : Score results

(b) : Time results

Figure C.79: ccp_alpha parameter tuning

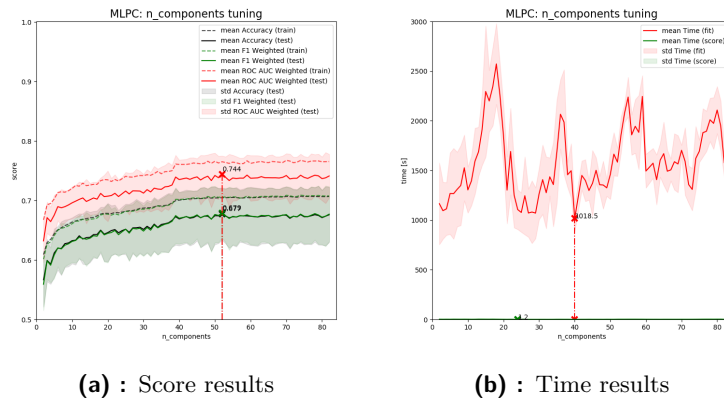
C.2.6 Multi-Layer Perceptron Classifier

Full-data model



(a) : Score results (b) : Time results

Figure C.80: n_components PCA parameter tuning



(a) : Score results (b) : Time results

Figure C.81: n_components TSVD parameter tuning

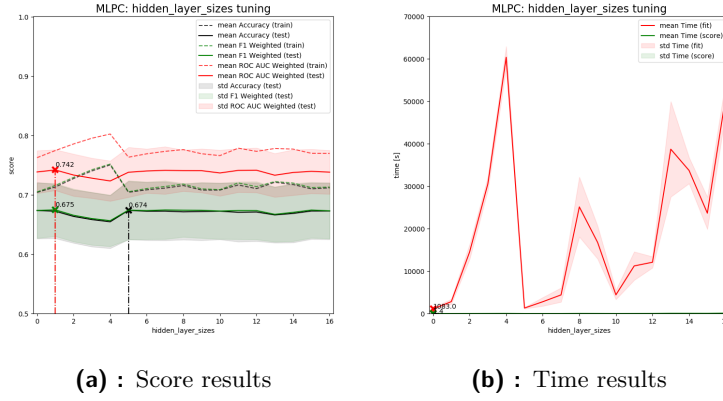


Figure C.82: hidden_layer_sizes parameter tuning (Table 6.13 contains mapping of tuned values to numbers)

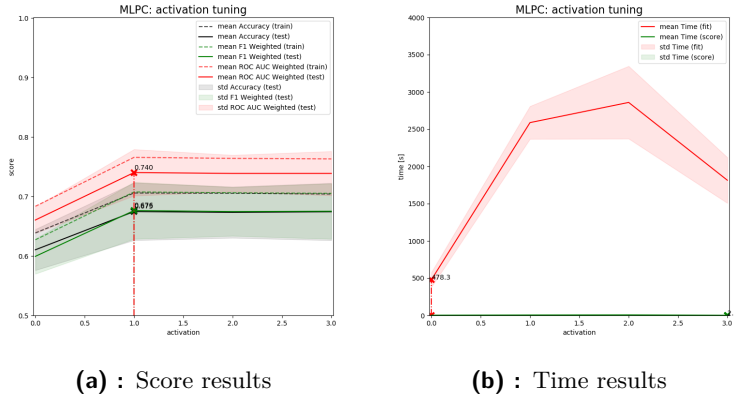


Figure C.83: activation parameter tuning (0: 'identity', 1: 'logistic', 2: 'tanh', 3: 'relu')

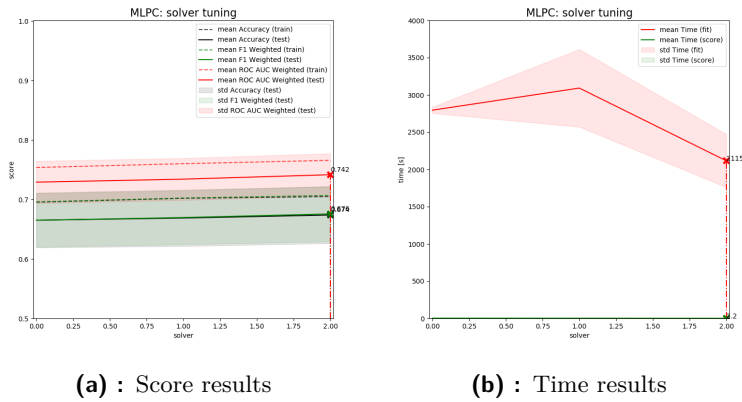


Figure C.84: solver parameter tuning (0: 'lbfgs', 1: 'sgd', 2: 'adam')

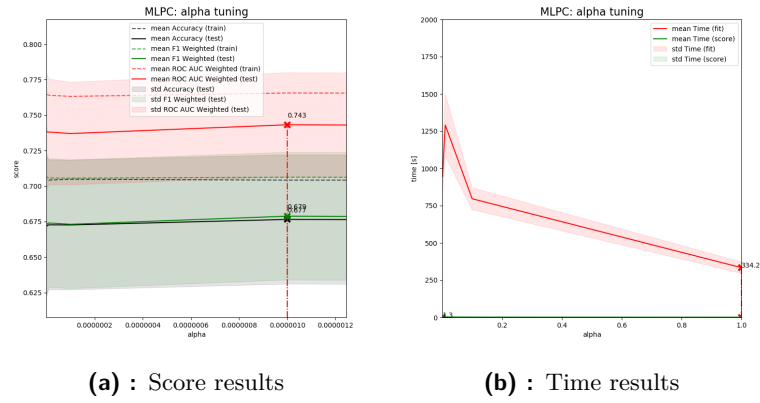


Figure C.85: alpha parameter tuning

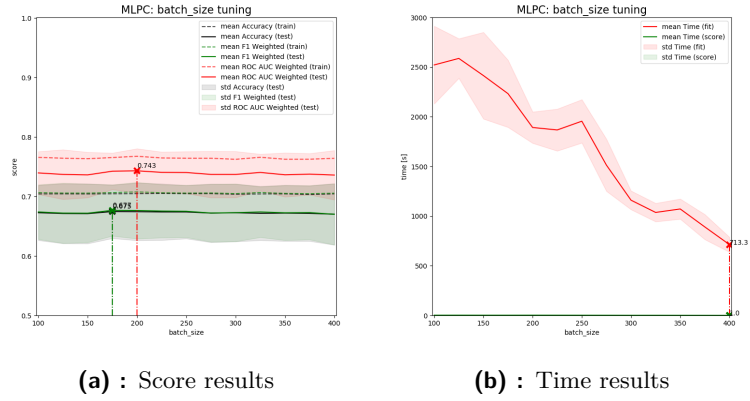


Figure C.86: batch_size parameter tuning

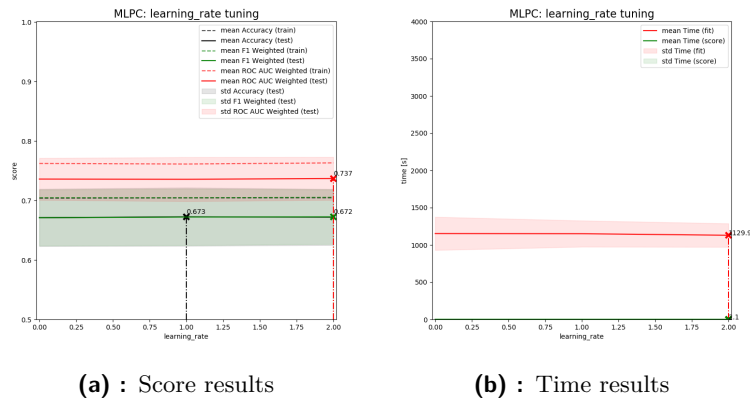
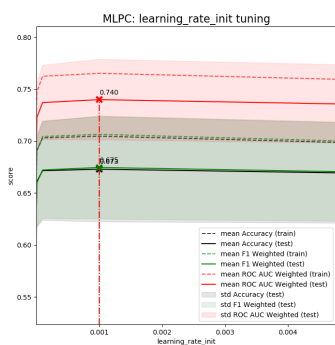
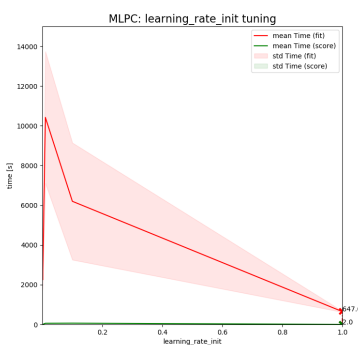


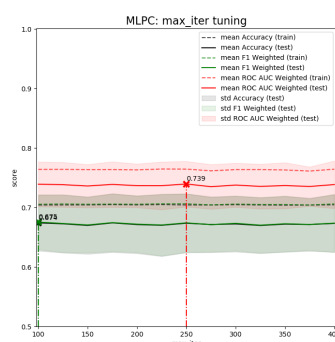
Figure C.87: learning_rate parameter tuning (0: 'constant', 1: 'invscaling', 2: 'adaptive')



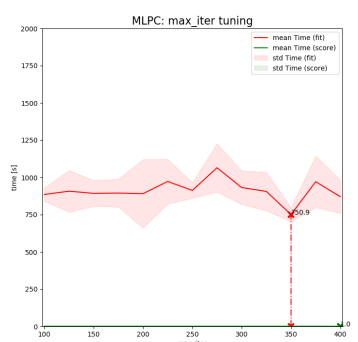
(a) : Score results



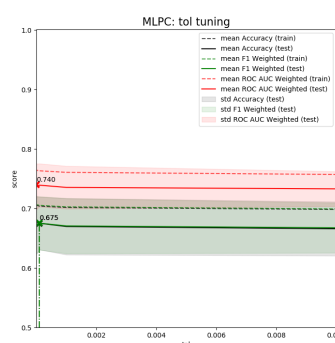
(b) : Time results

Figure C.88: learning-rate-init parameter tuning

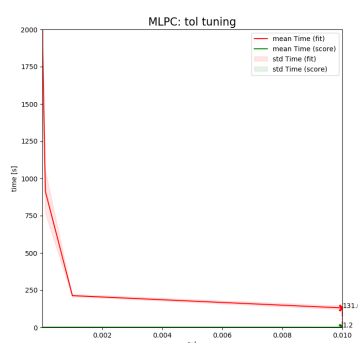
(a) : Score results



(b) : Time results

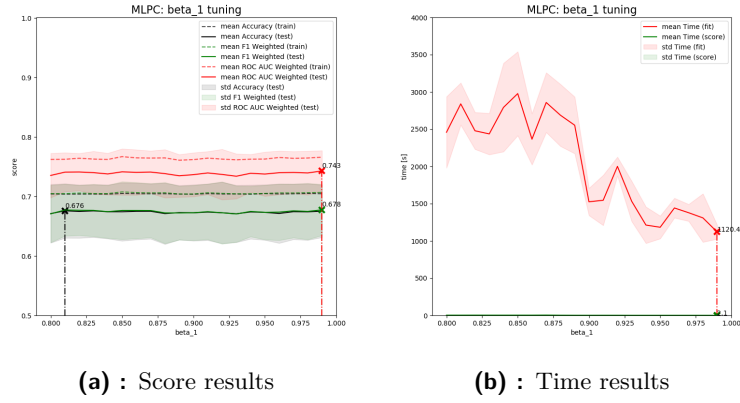
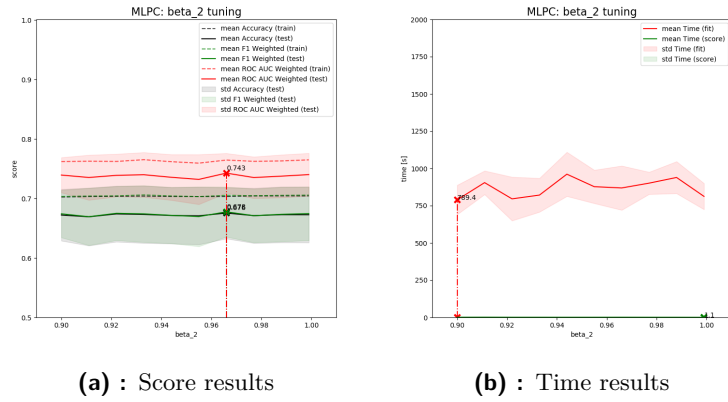
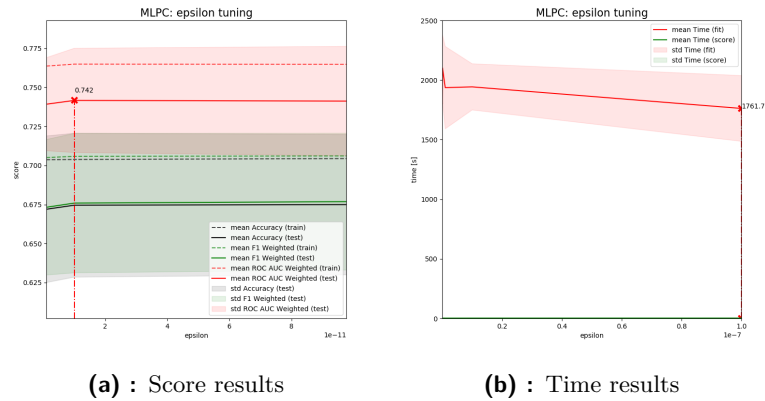
Figure C.89: max_iter parameter tuning

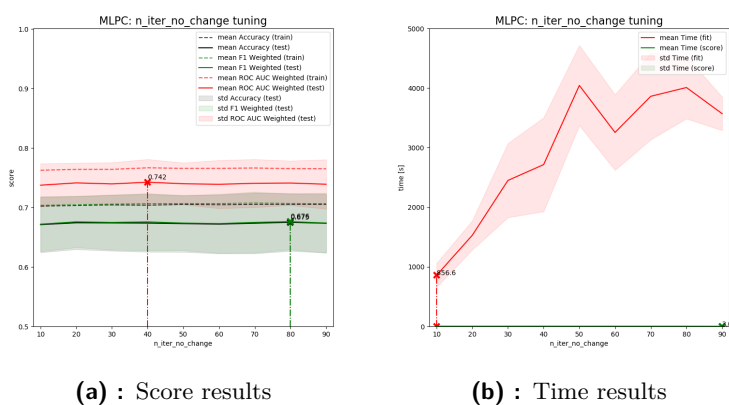
(a) : Score results



(b) : Time results

Figure C.90: tol parameter tuning

Figure C.91: β_1 parameter tuningFigure C.92: β_2 parameter tuningFigure C.93: ϵ parameter tuning

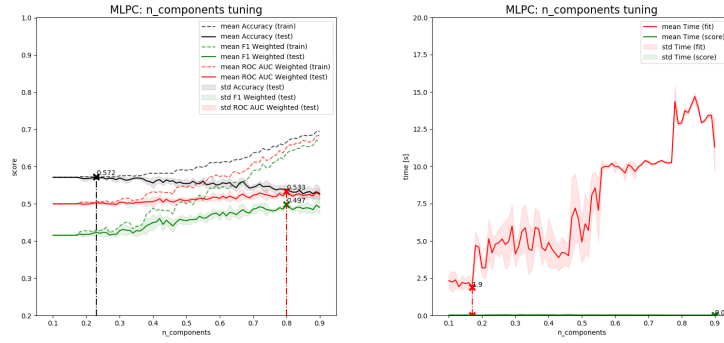


(a) : Score results

(b) : Time results

Figure C.94: `n_iter_no_change` parameter tuning (`early_stopping = 'True'`)

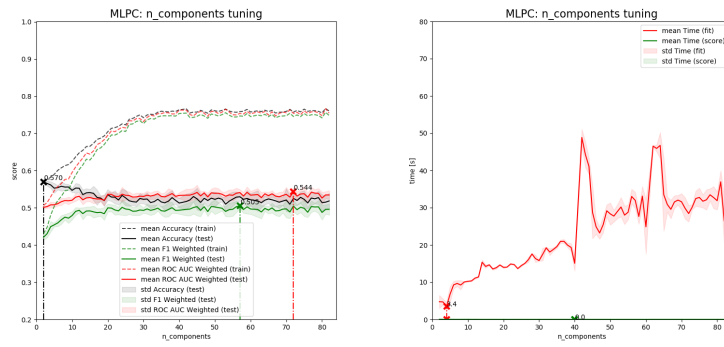
Channel-data model



(a) : Score results

(b) : Time results

Figure C.95: n_components PCA parameter tuning



(a) : Score results

(b) : Time results

Figure C.96: n_components TSVD parameter tuning

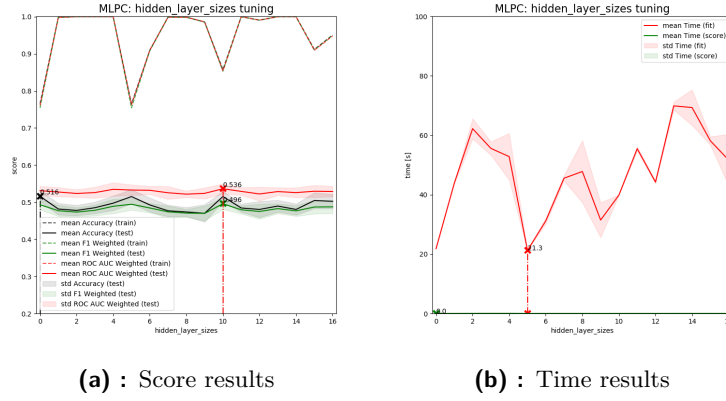


Figure C.97: hidden_layer_sizes parameter tuning (Table 6.13 contains mapping of tuned values to numbers)

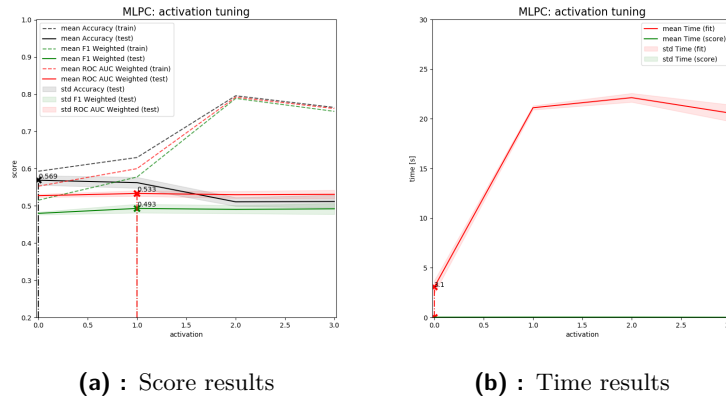


Figure C.98: activation parameter tuning (0: 'identity', 1: 'logistic', 2: 'tanh', 3: 'relu')

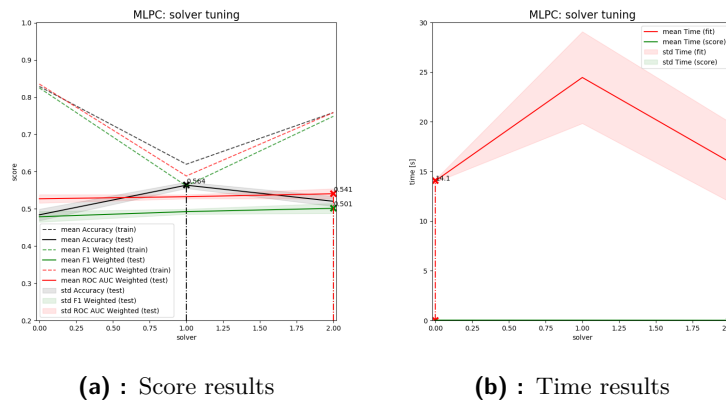


Figure C.99: solver parameter tuning (0: 'lbfgs', 1: 'sgd', 2: 'adam')

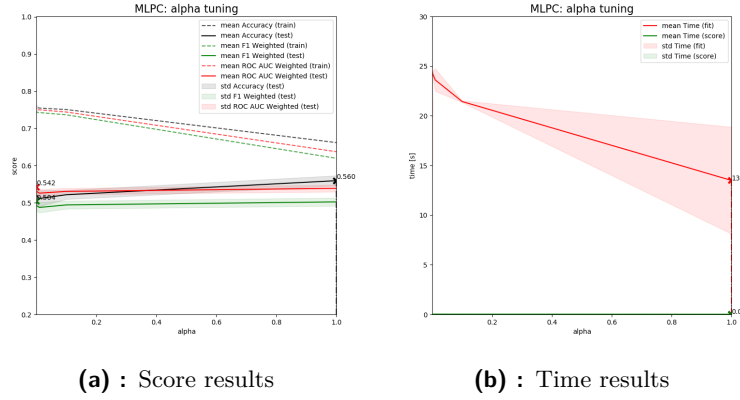


Figure C.100: alpha parameter tuning

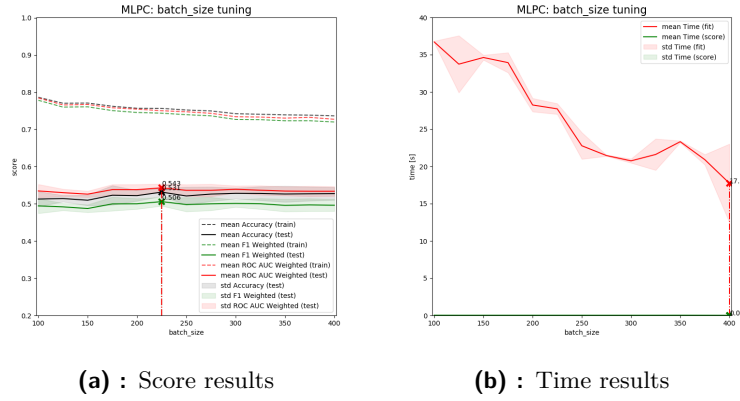


Figure C.101: batch_size parameter tuning

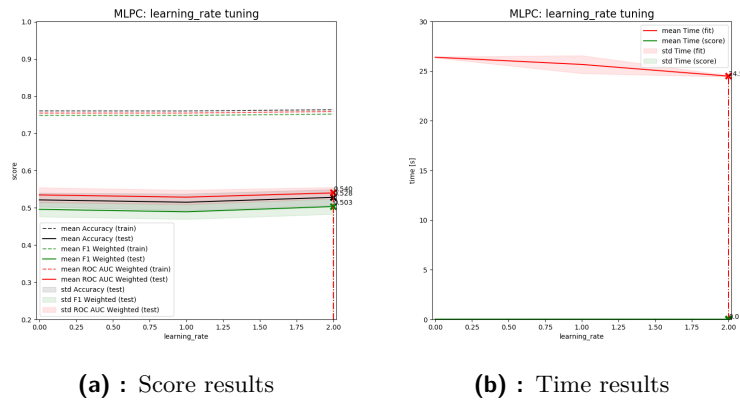
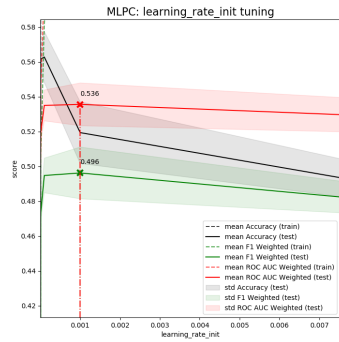
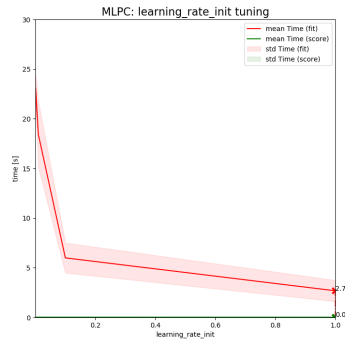


Figure C.102: learning_rate parameter tuning (0: 'constant', 1: 'invscaling', 2: 'adaptive')

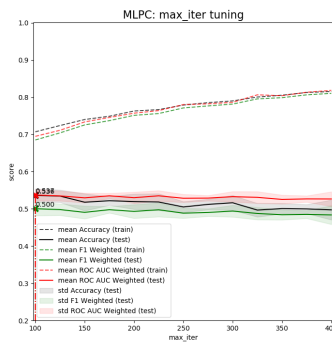


(a) : Score results

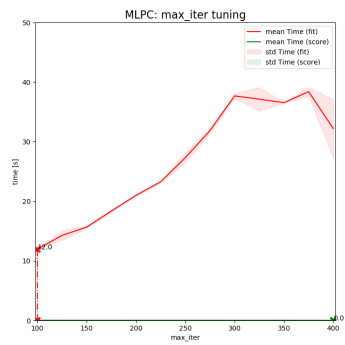


(b) : Time results

Figure C.103: learning-rate-init parameter tuning

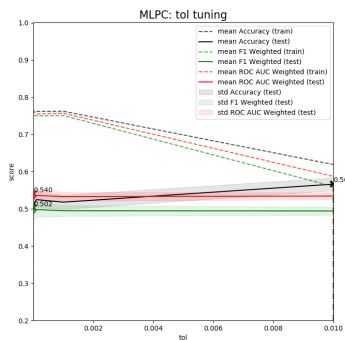


(a) : Score results

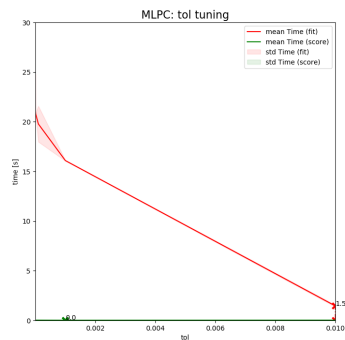


(b) : Time results

Figure C.104: max_iter parameter tuning



(a) : Score results



(b) : Time results

Figure C.105: tol parameter tuning

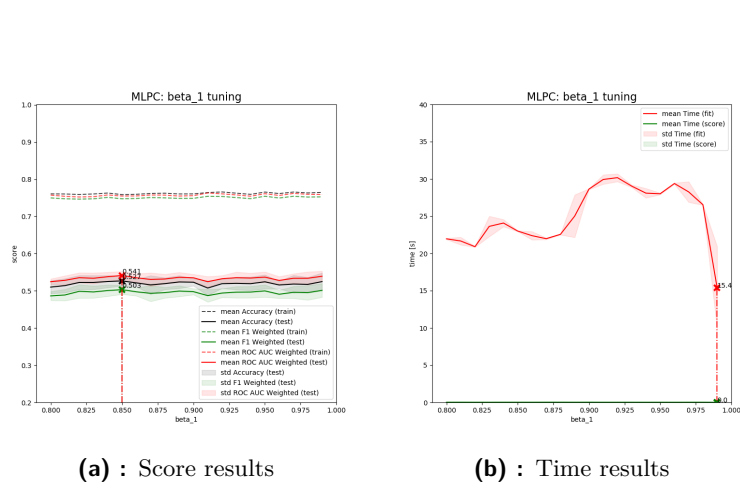


Figure C.106: beta_1 parameter tuning

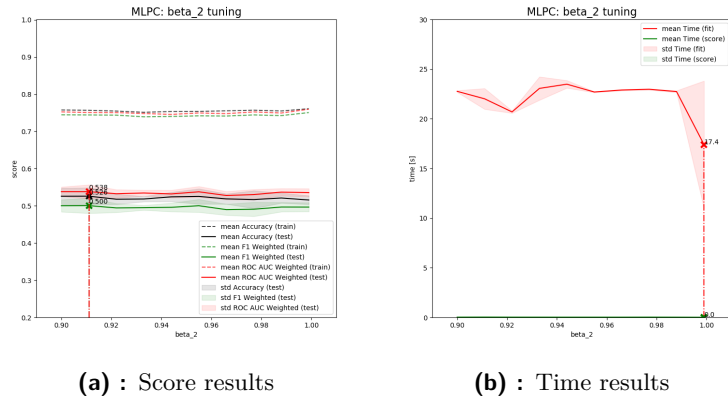


Figure C.107: beta_2 parameter tuning

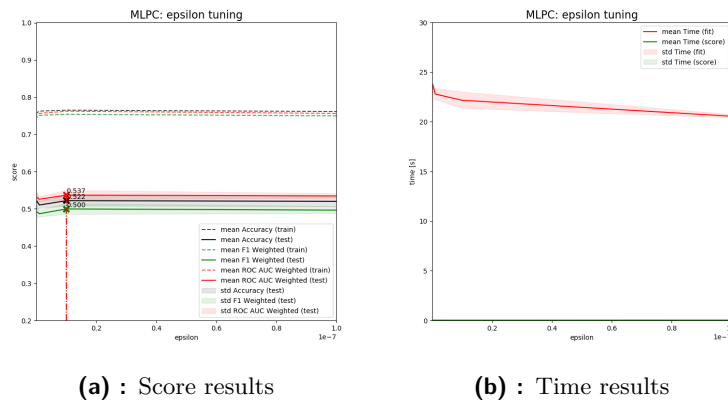


Figure C.108: epsilon parameter tuning

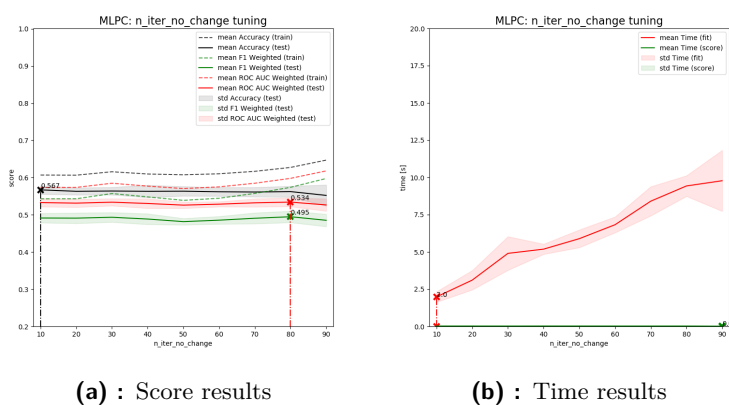


Figure C.109: `n_iter_no_change` parameter tuning (`early_stopping` = 'True')

C.2.7 Support Vector Classifier

Full-data model

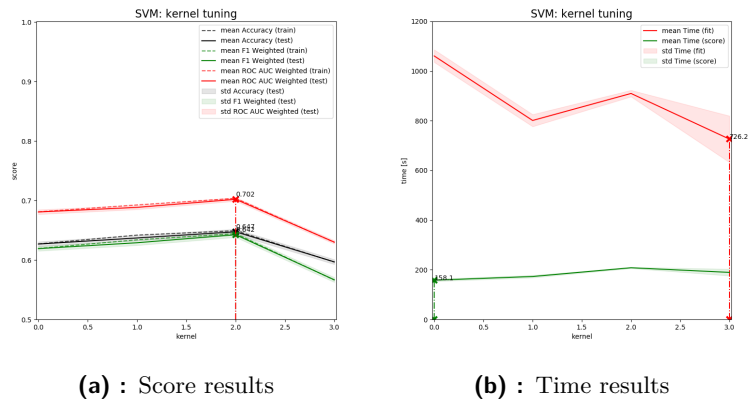


Figure C.110: kernel parameter tuning (0: 'linear', 1: 'poly', 2: 'rbf', 3: 'sigmoid')

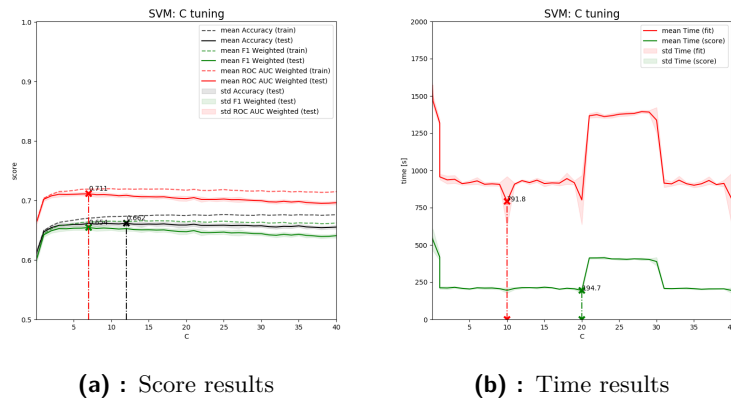


Figure C.111: C parameter tuning

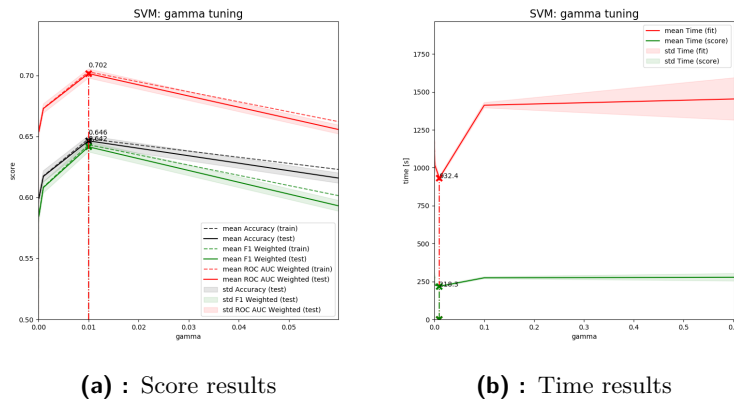


Figure C.112: gamma parameter tuning

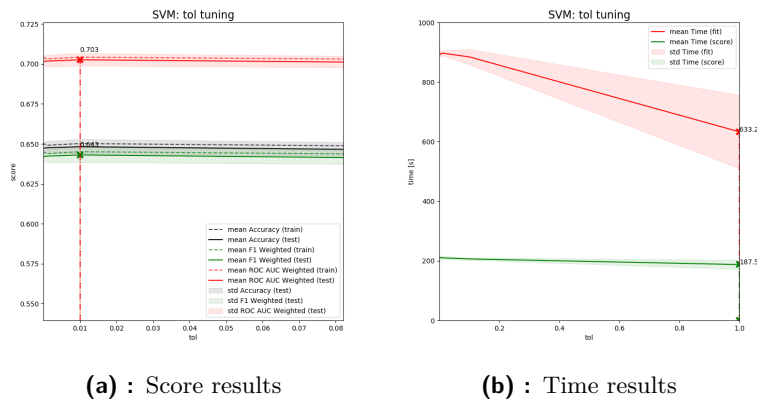


Figure C.113: tol parameter tuning

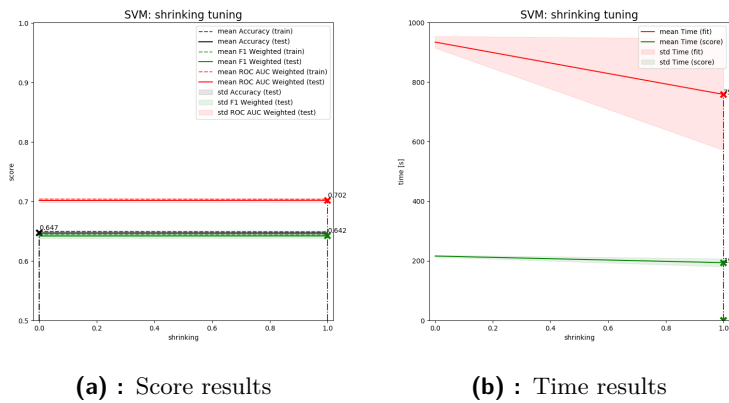
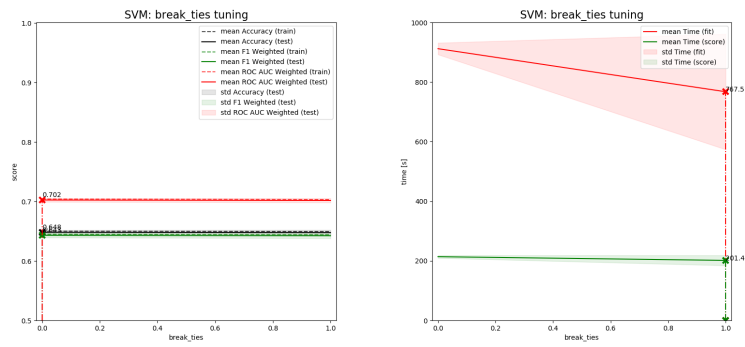


Figure C.114: shrinking parameter tuning (0: False, 1: True)

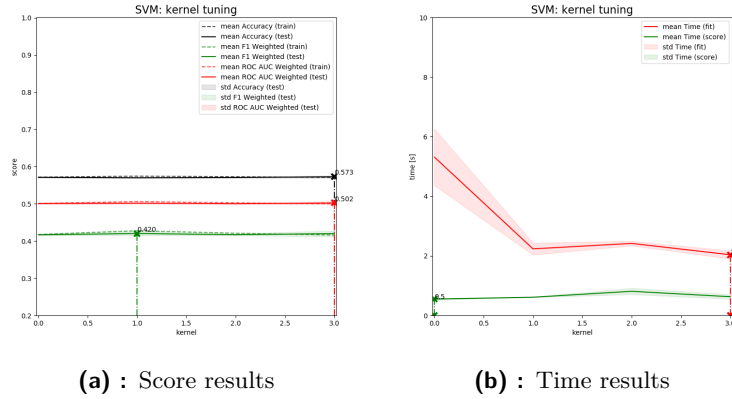


(a) : Score results

(b) : Time results

Figure C.115: break_ties parameter tuning (0: False, 1: True)

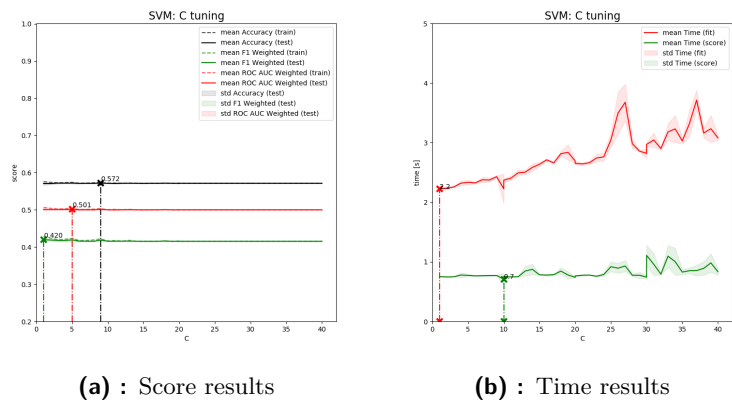
Channel-data model



(a) : Score results

(b) : Time results

Figure C.116: kernel parameter tuning (0: 'linear', 1: 'poly', 2: 'rbf', 3: 'sigmoid')



(a) : Score results

(b) : Time results

Figure C.117: C parameter tuning

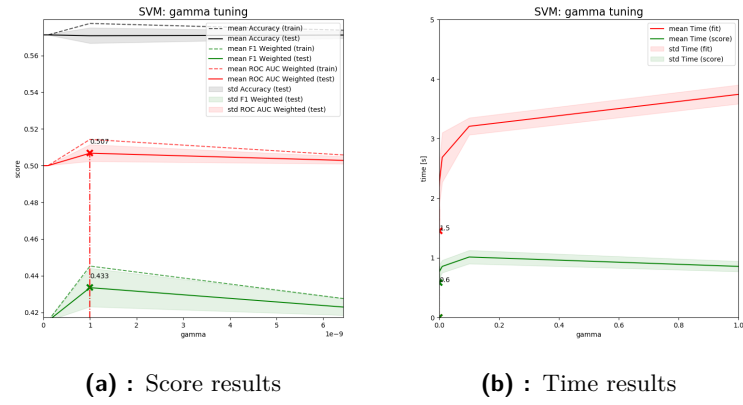


Figure C.118: gamma parameter tuning

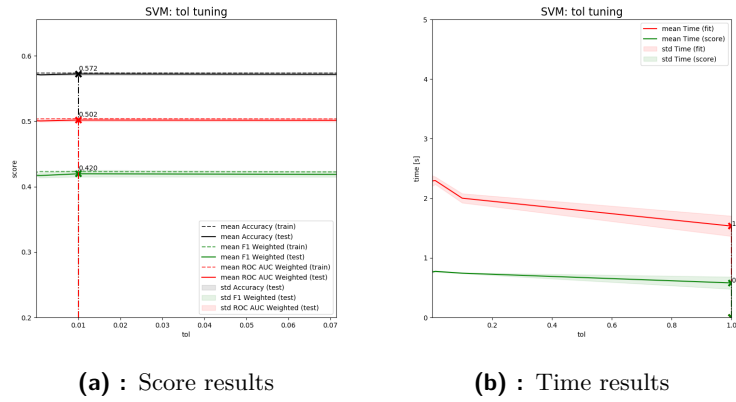


Figure C.119: tol parameter tuning

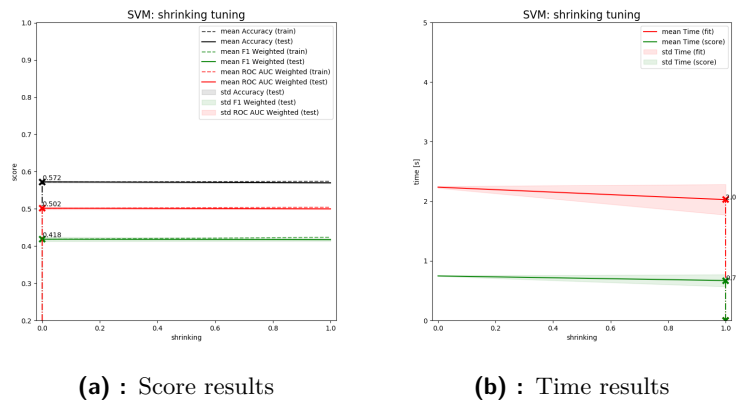
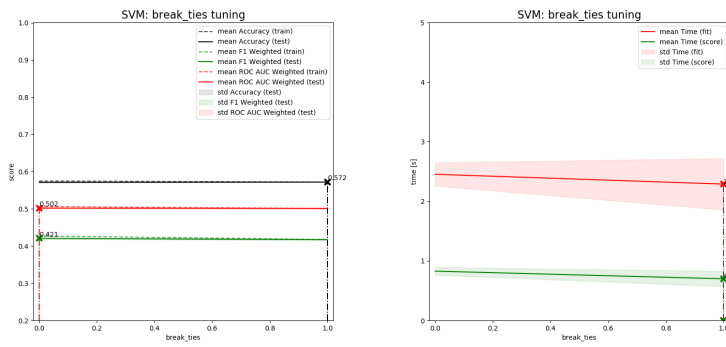


Figure C.120: shrinking parameter tuning (0: False, 1: True)



(a) : Score results

(b) : Time results

Figure C.121: break_ties parameter tuning (0: False, 1: True)

Appendix D

Code

D.1 Utils

D.1.1 converter.py

```
"""
Last modified Apr 13 2020
@author: Jakub Maly
"""

from os import path
import utils.reader as reader

import warnings
import uproot
import re
import pickle
import numpy as np

class Converter:
    """ Class for handling root files.

    This class is used to open root files, use the ROOT Scan() method
    above them,
    convert obtained data to numpy arrays, and save whole result to pkl
    format.

    The data are stored as dictionary. Each feature's data can be
    accessed by calling self.data['feat_name'].

    !Warning!
    For reader.py's purposes, the data dictionary contains extra
    feature called 'criterion012345'. This feature stores
```

```

root criterion used for extracting the data from original tree. As
an end user you should always access the data
from reader.py instance which automatically returns data without
this feature.
"""

def __init__(self, params={}):
    self.verbose = params.get('verbose')
    self.state = 0
    self.file = {}
    self.data = {}
    self.filename = ''
    self.directory = ''
    self.criterion = ''

def open(self, f):
    """ Open specified root file.
    @param self:
    @param f: The root file which is to be open.
    @return:
    """

    if self.verbose is not None:
        print('converter.open():')
        print('\t|-> opening file \'' + f + '\'')

    self.file = uproot.open(f)
    self.filename = f.rsplit('/', 1)[1][:5]
    self.directory = f.rsplit('/', 1)[0]
    self.state = 1

    if self.verbose is not None:
        print('\t----- open done -----')

    @staticmethod
    def to_code(string):
        """ Convert ROOT query to python language.
        @param string: The string which is to be converted to code.
        @return: The converted string, The branches needed for
        successful query.
        """

        expressions = re.split(',|;', string)
        req_branches = []

        operators = []
        for c in string:
            if c == ',' or c == ';':
                operators.append(c)

        for i, expression in enumerate(expressions):
            if expression.find('==') != -1:
                branch = re.split('==', expression)
                req_branches.append(branch[0])
                code = re.sub('[a-zA-Z,0-9,_,]+', lambda m: 'dc[\'%s\']' % m.group(0), branch[0])
                expressions[i] = code + '==' + branch[1]
            elif expression.find('!=') != -1:
                branch = re.split('!=', expression)
                req_branches.append(branch[0])
                code = re.sub('[a-zA-Z,0-9,_,]+', lambda m: 'dc[\'%s\']' % m.group(0), branch[0])
                expressions[i] = code + '!=' + branch[1]
            elif expression.find('<') != -1:
                branch = re.split('<', expression)
                req_branches.append(branch[0])

```

```

        code = re.sub('[a-z,A-Z,0-9,_]+', lambda m: 'dc[\'%s
            \']' % m.group(0), branch[0])
        expressions[i] = code + '<' + branch[1]
    elif expression.find('>') != -1:
        branch = re.split('>', expression)
        req_branches.append(branch[0])
        code = re.sub('[a-z,A-Z,0-9,_]+', lambda m: 'dc[\'%s
            \']' % m.group(0), branch[0])
        expressions[i] = code + '>' + branch[1]
    elif expression.find('<=') != -1:
        branch = re.split('<=', expression)
        req_branches.append(branch[0])
        code = re.sub('[a-z,A-Z,0-9,_]+', lambda m: 'dc[\'%s
            \']' % m.group(0), branch[0])
        expressions[i] = code + '<=' + branch[1]
    elif expression.find('>=') != -1:
        branch = re.split('>=', expression)
        req_branches.append(branch[0])
        code = re.sub('[a-z,A-Z,0-9,_]+', lambda m: 'dc[\'%s
            \']' % m.group(0), branch[0])
        expressions[i] = code + '>=' + branch[1]
    else:
        print('Unsupported comparison!')

code = ''
if len(expressions) > 1:
    for i, expression in enumerate(expressions):
        if i < len(operators):
            code += expression + operators[i]
        else:
            code += expression
else:
    code = expressions[0]

code = '(' + code + ')'
code = code.replace(',', ' ') and '('
code = code.replace('; ', ' or ')

return code, req_branches

def scan(self, tree_name, branches='*', criterion=None):
    """ Scan the file.
    @param self:
    @param tree_name: The name of tree used in query.
    @param branches: Branches on which the request is made.
    @param criterion: Query criterion.
    @return:
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')
        return

    if self.verbose is not None:
        print('converter.scan():')

    dc = {}
    separator = ':'
    self.criterion = criterion # for passing to saved file

    # Handle ROOT notation
    if branches == '*':
        branches = self.file[tree_name].keys()
        branches = list(dict.fromkeys(branches))
    else:

```



```

        branches = re.split(separator, branches)

# Handle existence of file
for i, branch in enumerate(branches):
    if not isinstance(branch, str):
        branches[i] = branch.decode('utf-8') # convert to utf
        -8 (remove 'b' prefix)
if path.exists(self.directory + '/' + self.filename + '.pkl'):
    rd = reader.Reader()
    rd.open(self.directory + '/' + self.filename + ".pkl")

    saved_query = rd.get_query()
    our_query = separator.join(branches)

    if criterion:
        saved_criterion = rd.get_criterion()
        if saved_query == our_query and saved_criterion ==
            criterion:
            raise FileExistsError
    elif saved_query == our_query:
        raise FileExistsError

# Get data of branches
for i, branch in enumerate(branches):
    if self.verbose is not None:
        print('\t|-> reading branch \'' + branches[i] + '\'')
    dc[branches[i]] = self.file[tree_name].array(branches[i])

# Apply criterion if it exists
if criterion:
    criterion, req_branches = self.to_code(criterion)
    if self.verbose is not None:
        print('\t-----')
        print('\t|-> applying criterion \'' + criterion + '\'')
    for branch in req_branches:
        if branch not in dc: # check if branch is missing in
            dictionary
            if self.verbose is not None:
                print('\t\t|-> reading branch \'' + branch +
                    '\'')
            dc[branch] = self.file[tree_name].array(branch)

    result = eval(criterion)
    for key in dc:
        dc[key] = dc[key][result]

# Remove unwanted branches from dc (if they were added due to
    criterion req)
dc = {key: dc[key] for key in branches}

self.data = dc
self.state = 2

if self.verbose is not None:
    print('\t----- scan done -----')

def convert(self):
    """ Convert data to numpy arrays.
    @param self:
    @return:
    """

    if self.state < 2:
        warnings.warn('Warning: File need to be scanned first! Use
            method scan()')
        return

```

```

    if self.verbose is not None:
        print('converter.convert():')
        print('\t|-> converting data to numpy arrays')

    for key in self.data:
        self.data[key] = np.array(self.data[key])

    self.state = 3

    if self.verbose is not None:
        print('\t|----- convert done -----')

def save(self, folder, name):
    """ Save data in pkl format.
    @param self:
    @param folder: Name of the folder
    @param name: Name of the file
    @return:
    """

    if self.state < 3:
        warnings.warn('Warning: File need to be converted first!
            Use method convert()')
        return

    if self.verbose is not None:
        print('converter.save():')
        print('\t|-> saving data to pkl format')

    # Add criterion to dictionary
    self.data['criterion012345'] = self.criterion

    with open(folder + name + '.pkl', 'wb') as f:
        pickle.dump(self.data, f)
        f.close()

    self.state = 4

    if self.verbose is not None:
        print('\t|----- save done -----')

```

■ D.1.2 reader.py

```

"""
Last modified Apr 13 2020
@author: Jakub Maly
"""

import utils.converter as converter

import warnings
import re
import pickle

class Reader:
    """ Class for handling pkl files.

    This class is used to open pkl files, get basic information about
    their structure,
    and obtain their data.

```

Please see converter.py for information about format in which data are saved.

```

"""
def __init__(self, params={}):
    self.verbose = params.get('verbose')
    self.state = 0
    self.file = {}
    self.data = {}
    self.filename = ''
    self.branches = []
    self.query = ''
    self.criterion = ''

def open(self, f):
    """ Open specified pkl file.
    @param self:
    @param f: The pkl file which is to be open.
    @return:
    """

    if self.verbose is not None:
        print('reader.open():')
        print('\t|-> opening file \'' + f + '\'')

    self.file = f
    self.filename = f.rsplit('/', 1)[1][: -4]

    with open(self.file, 'rb') as f:
        self.data = pickle.load(f)

    # Set criterion
    self.criterion = self.data['criterion012345']
    del self.data['criterion012345']

    # Set branches
    temp = []
    for key in self.data:
        temp.append(key)

    self.branches = temp

    # Set query
    temp = ''
    for item in self.branches:
        if temp:
            temp += ':' + item
        else:
            temp += item

    self.query = temp

    self.state = 1

    if self.verbose is not None:
        print('\t----- open done -----')

def print_info(self):
    """ Show basic information about the file.
    @param self:
    @return:
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')

```

```

        return

    if self.verbose is not None:
        print('reader.print_info():')

    if self.criterion:
        print('\t|-> file \'{0}.pkl\'' contains data of \'Scan
              (\'{1}\',\'{2}\')\''.format(self.filename, self.query,

    else:
        print('\t|-> file \'{0}.pkl\'' contains data of \'Scan
              (\'{1}\')\''.format(self.filename, self.query))

def filt_channels(self, features, channels):
    """ Filter data by given features, ignore desired channels.
    @param self:
    @param features: Features used for filtering.
    @param channels: Channels ignored by filtering.
    @return:
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
                      method open()')
        return

    if self.verbose is not None:
        print('reader.filt_channels():')

    # Wildcard handling
    temp_features = list()
    for feature in features:
        if feature[-1] == "*":
            for element in filter((lambda x: re.search(r'~{0}'.
                format(feature[:-1]), x)), self.branches):
                temp_features.append(element)
        else:
            temp_features.append(feature)

    for channel in channels:
        temp_features.append(channel)

    # Missing feature handling
    temp_features_checked = list()
    for feature in temp_features:
        if feature in self.branches:
            temp_features_checked.append(feature)

    self.data = {feature: self.data[feature] for feature in
                  temp_features_checked}

def filt_criterion(self, features, criterion):
    """ Filter data by given features, applies desired criterion
    @param self:
    @param features: Features used for filtering.
    @param criterion: Criterion used for filtering.
    @return:
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use

```

```

self
.
cr
)
)

```

```

        method open()')
    return

if self.verbose is not None:
    print('reader.filt_criterion():')

if criterion is None:

    # Wildcard handling
    temp_features = list()
    for feature in features:
        if feature[-1] == "*":
            for element in filter((lambda x: re.search(r'^{ }'.
                format(feature[:-1]), x)), self.branches):
                temp_features.append(element)
        else:
            temp_features.append(feature)

    # Missing feature handling
    temp_features_checked = list()
    for feature in temp_features:
        if feature in self.branches:
            temp_features_checked.append(feature)

    self.data = {feature: self.data[feature] for feature in
        temp_features_checked}
else:
    cv = converter.Converter()
    criterion, req_branches = cv.to_code(criterion)
    if self.verbose is not None:
        print('\t-----')
        print('\t|-> applying criterion \'' + criterion + '\'')

    dc = self.data
    result = eval(criterion)
    for key in dc:
        dc[key] = dc[key][result]

    self.data = dc
    self.filt_criterion(features, None)

def save(self, f):
    """ Save data in pkl format.
    @param self:
    @param f: Path to file
    @return:
    """

    self.data['criterion012345'] = self.criterion

    with open(f, 'wb') as f:
        pickle.dump(self.data, f)
        f.close()

def get_data(self):
    """ Return file's data.
    @param self:
    @return: Dictionary with data.
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')

    return self.data

```

```

def get_branches(self):
    """ Return file's branches.
    @param self:
    @return: List of branches names.
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')

    return self.branches

def get_query(self):
    """ Return query string.
    @param self:
    @return: Query used for Scan() command.
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')

    return self.query

def get_criterion(self):
    """ Return criterion string.
    @param self:
    @return: Criterion used for Scan() command.
    """

    if self.state < 1:
        warnings.warn('Warning: File need to be opened first! Use
            method open()')

    return self.criterion

```

■ D.1.3 ml_utils.py

```

"""
Last modified May 19 2020
@author: Jakub Maly

Support methods for ML libraries
"""

import utils.reader as reader

import sys

import pickle
import compress_pickle
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import StandardScaler

```



```

    if len(f) is 0:
        f = f_bgr[file_bgr]
    else:
        f = set(f) & set(f_bgr[file_bgr])

# Check features
data_iter = data_sig[next(iter(files_sig))]
for key in f:
    # exclude objects
    if data_iter[key].dtype != object:
        # exclude monte-carlo features
        if 'truth' not in key and 'Truth' not in key:
            f_new.append(key)

# Stack results
print('Stacking signal files...')
for file_sig in files_sig:
    data_iter = data_sig[file_sig]
    for key in f_new:
        # check if X already initialized, *1 - convert boolean to
        # int
        if X_sig_iter is None:
            X_sig_iter = data_iter[key] * 1
        else:
            X_sig_iter = np.column_stack((X_sig_iter, data_iter[key]
                                           ] * 1))

    print('{:}: {}'.format(file_sig, np.shape(X_sig_iter)))
    if X_sig is None:
        X_sig = X_sig_iter
    else:
        X_sig = np.concatenate((X_sig, X_sig_iter))
    X_sig_iter = None
print('-----')
print(np.shape(X_sig))

print('Stacking background files...')
for file_bgr in files_bgr:
    data_iter = data_bgr[file_bgr]
    for key in f_new:
        # check if X already initialized, *1 - convert boolean to
        # int
        if X_bgr_iter is None:
            X_bgr_iter = data_iter[key] * 1
        else:
            X_bgr_iter = np.column_stack((X_bgr_iter, data_iter[key]
                                           ] * 1))

    print('{:}: {}'.format(file_bgr, np.shape(X_bgr_iter)))
    if X_bgr is None:
        X_bgr = X_bgr_iter
    else:
        X_bgr = np.concatenate((X_bgr, X_bgr_iter))
    X_bgr_iter = None
print('-----')
print(np.shape(X_bgr))

print('Concatenating results...')
X = np.concatenate((X_sig, X_bgr))
y_pos = np.ones(len(X_sig))
y_neg = np.zeros(len(X_bgr))
y = np.concatenate((y_pos, y_neg))
f = f_new

return X, y, f

```



```

def get_X_y_f_multiclass(data, params={}):
    """ Loads specified data files.
    @param data: Dictionary with files path
    @param params: Reader params such as verbose
    @return: X, y, f - the data set, labels, feature names

    Warning: For multi-class classification ttH is labeled with 0 and
             any background with 0+i, i>0, i++!
    """

    print('ml_utils.get_X_y_f_multiclass():')

    rd = reader.Reader(params)
    data_classes = {}
    f_classes = {}
    f = list()
    f_new = list()
    X = None
    X_iter = None
    X_iter_inner = None
    y = None
    y_iter = None

    # Store data
    for key in data:
        data_i = {}
        f_i = {}
        for file in data[key]:
            rd.open(file)
            data_i[file] = rd.get_data()
            f_i[file] = rd.get_branches()
        data_classes[key] = data_i
        f_classes[key] = f_i

    # Get common features
    for key in data:
        for file in data[key]:
            if len(f) is 0:
                f = f_classes[key][file]
            else:
                f = set(f) & set(f_classes[key][file])

    # Check features
    for key in data:
        for file in data[key]:
            for feature in f_classes[key][file]:
                if data_classes[key][file][feature].dtype != object:
                    # exclude monte-carlo features
                    if 'truth' not in feature and 'Truth' not in
                        feature:
                        f_new.append(feature)
            break
        break

    # Stack results
    print('Stacking files...')
    for key in data:
        for file in data[key]:
            data_iter = data_classes[key][file]
            for feature in f_new:
                # check if X already initialized, *1 - convert boolean
                # to int
                if X_iter_inner is None:
                    X_iter_inner = data_iter[feature] * 1
                else:

```

```

        X_iter_inner = np.column_stack((X_iter_inner,
                                         data_iter[feature] * 1))

    if X_iter is None:
        X_iter = X_iter_inner
    else:
        X_iter = np.concatenate((X_iter, X_iter_inner))
        X_iter_inner = None

    y_iter = np.ones(len(X_iter)) * key
    if y is None:
        y = y_iter
        X = X_iter
    else:
        y = np.concatenate((y, y_iter))
        X = np.concatenate((X, X_iter))
        X_iter = None

    print('-----')
    print('X dimensions: {}'.format(np.shape(X)))

    return X, y, f_new

def get_X_f_multiclass(data, params={}):
    """ Loads specified data files of unknown class.
    @param data: List with files path
    @param params: Reader params such as verbose
    @return: X, f - the data set, feature names
    """

    print('ml_utils.get_X_f_multiclass():')

    rd = reader.Reader(params)
    data_files = {}
    f_files = {}
    f = list()
    f_new = list()
    X = None
    X_iter = None
    X_iter_inner = None

    # Store data
    for file in data:
        rd.open(file)
        data_files[file] = rd.get_data()
        f_files[file] = rd.get_branches()

    # Get common features
    for file in data:
        if len(f) is 0:
            f = f_files[file]
        else:
            f = set(f) & set(f_files[file])

    # Check features
    for file in data:
        for feature in f_files[file]:
            if data_files[file][feature].dtype != object:
                # exclude monte-carlo features
                if 'truth' not in feature and 'Truth' not in feature:
                    f_new.append(feature)
        break

    # Stack results
    print('Stacking files...')

```

```

for file in data:
    data_iter = data_files[file]
    for feature in f_new:
        # check if X already initialized, *1 - convert boolean to
        # int
        if X_iter_inner is None:
            X_iter_inner = data_iter[feature] * 1
        else:
            X_iter_inner = np.column_stack((X_iter_inner, data_iter
                                             [feature] * 1))

    if X_iter is None:
        X_iter = X_iter_inner
    else:
        X_iter = np.concatenate((X_iter, X_iter_inner))
    X_iter_inner = None

if X is None:
    X = X_iter
else:
    X = np.concatenate((X, X_iter))

print('-----')
print('X dimensions: {}'.format(np.shape(X)))

return X, f_new

def save(f, folder, name):
    """ Saves file.
    @param f: File to be saved
    @param folder: Folder for saving
    @param name: Name of save file
    """

    pickle.dump(f, open(folder + '/' + name + '.pkl', 'wb'))

def save_compress(f, folder, name, method):
    """ Compress and saves file.
    @param f: File to be saved
    @param folder: Folder for saving
    @param name: Name of save file
    @param method: Compression method
    """

    compress_pickle.dump(f, open(folder + '/' + name + '.pkl.' + method
                                   , 'wb'), compression=method)

def load(f):
    """ Loads file.
    @param f: File to be loaded
    @return: Content of file
    """

    return pickle.load(open(f, 'rb'))

def load_compress(f, method):
    """ Loads compressed file.
    @param f: File to be loaded
    @param method: Compression method
    @return: Content of file
    """

```

```

        return compress_pickle.load(open(f, 'rb'), method)

def apply_channel(X_cv, X_test, y_cv, y_test, f, channel):
    """ Performs channel selection
    @param X_cv: The cv set
    @param X_test: The test set
    @param y_cv: The cv labels
    @param y_test: The test labels
    @param f: Feature names
    @param channel: Channel name
    @return: X_cv, X_test, y_cv, y_test, f - the data set, labels,
            feature names
    """

    i = f.index(channel)
    f = np.delete(f, i)

    # cv
    truth_cv = X_cv[:, i]
    X_cv_new = None
    y_cv_new = list()
    X_cv = np.delete(X_cv, i, 1)
    for j, t in enumerate(truth_cv):
        if t == 1.0:
            if X_cv_new is None:
                X_cv_new = X_cv[j, :]
            else:
                X_cv_new = np.row_stack((X_cv_new, X_cv[j, :]))
            y_cv_new.append(y_cv[j])

    y_cv_new = np.array(y_cv_new)

    # test
    truth_test = X_test[:, i]
    X_test_new = None
    y_test_new = list()
    X_test = np.delete(X_test, i, 1)
    for j, t in enumerate(truth_test):
        if t == 1.0:
            if X_test_new is None:
                X_test_new = X_test[j, :]
            else:
                X_test_new = np.row_stack((X_test_new, X_test[j, :]))
            y_test_new.append(y_test[j])

    y_test_new = np.array(y_test_new)

    return X_cv_new, X_test_new, y_cv_new, y_test_new, f

def remove_channel(X_cv, X_test, f, channel):
    """ Performs channel data removal (possible guidance for learning)
    @param X_cv: The cv set
    @param X_test: The test set
    @param f: Feature names
    @param channel: Channel name
    @return: X_cv, X_test, y_cv, y_test, f - the data set, labels,
            feature names
    """

    i = f.index(channel)
    f = np.delete(f, i)

    # cv
    X_cv = np.delete(X_cv, i, 1)

```

```
# test
X_test = np.delete(X_test, i, 1)

return X_cv, X_test, f

def check_feature_order(X, y, f, f_prev):
    """ Checks if features are in desired order
    @param X: Data
    @param y: Ground truth Labels
    @param f: Current order
    @param f_prev: Previous (desired) order
    @return: Reordered data and labels, desired feature order
    """

    X_new = X
    y_new = y

    for feature in f_prev:
        i = f_prev.index(feature)
        j = f.index(feature)

        if i != j:
            print('features columns were exchanged')
            X_new[:, i] = X[:, j]
            y_new[i] = y[j]

    return X_new, y_new, f_prev

def multiclass_roc_auc_score(y_test, y_pred, average="macro",
                             multi_class='ovo'):
    """ Computes ROC/AUC score for multiclass classification
    @param y_test: test labels
    @param y_pred: predicted labels
    @param average: averaging algorithm
    @param multi_class: how to handle multiclass
    @return: ROC/AUC score
    """

    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return roc_auc_score(y_test, y_pred, average=average, multi_class=
                          multi_class)

def multiclass_f1(y_test, y_pred, average="macro", ):
    """ Computes F-1 score for multiclass classification
    @param y_test: test labels
    @param y_pred: predicted labels
    @param average: averaging algorithm
    @return: F-1 score
    """

    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return f1_score(y_test, y_pred, average=average)

def multiclass_accuracy(y_test, y_pred):
    """ Computes accuracy score for multiclass classification
```

```

    @param y_test: test labels
    @param y_pred: predicted labels
    @return: Accuracy score
    """

    lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
    return accuracy_score(y_test, y_pred)

def threshold_probas_multiclass(y_probas, threshold):
    """ Thresholds multiclass probabilities - see thesis (WP tuning)
        for more information
    @param y_probas: Predicted probabilities
    @param threshold: Probability threshold
    @return: Predicted classes
    """

    length = np.shape(y_probas)[0]
    width = np.shape(y_probas)[1]

    y_pred = list()

    for i in range(length):
        bgr_sum = 0
        for j in range(1, width):
            bgr_sum += y_probas[i, j]

        if y_probas[i, 0] >= threshold:
            y_pred.append(0)
        else:
            suggestion = 0
            max_p = 0
            if suggestion == 0:
                for j in range(1, width):
                    if max_p < y_probas[i, j]:
                        max_p = y_probas[i, j]
                        suggestion = j
            y_pred.append(suggestion)

    return y_pred

def threshold_characteristics_multiclass_significance(y_probas, y_test,
    weights, total, other_bgr):
    """ Obtain characteristics for simplified significance maximization
        threshold tuning
    @param y_probas: Predicted probabilities
    @param y_test: Ground truth labels
    @param weights: Weighting factors used for comparison with existing
        results
    @param total: Number of events in real data used for comparison
    @param other_bgr: Number of events in other background in real data
        used for comparison
    @return: Graph data, best threshold
    """

    x_values = list()

    y_eps_0 = list()
    y_eps_1 = list()
    y_eps_2 = list()
    y_S = list()
    y_B = list()

```

```

y_signif = list()
y_signif_simp = list()

max_sig = 0
best_tr = 0

for tr in np.round(np.arange(0.0, 1, 0.01), 2):
    tr = np.round(tr, 3)
    x_values.append(tr)
    print('Threshold: {}'.format(tr))
    y_pred = threshold_probas_multiclass(y_probas, tr)
    response = significance_score_multiclass_compare(y_test, y_pred
        , weights, total, other_bgr, False)
    y_eps_0.append(response['eps'][0])
    y_eps_1.append(response['eps'][1])
    y_eps_2.append(response['eps'][2])
    y_S.append(response['S'])
    y_B.append(response['B'])
    y_signif.append(response['signif'])
    y_signif_simp.append(response['signif_simp'])

    if response['signif_simp'] > max_sig:
        max_sig = response['signif_simp']
        best_tr = tr

return x_values, y_eps_0, y_eps_1, y_eps_2, y_S, y_B, y_signif,
    y_signif_simp, best_tr

def threshold_characteristics_multiclass_sensitivity(y_probas, y_test):
    """ Obtain characteristics for sensitivity maximization threshold
        tuning
    @param y_probas: Predicted probabilities
    @param y_test: Ground truth labels
    @return: Graph data, best threshold
    """

    x_values = list()

    y_fdr = list()
    y_sen = list()

    max_sig = 0
    best_tr = 0

    for tr in np.arange(0, 1.01, 0.01):
        tr = np.round(tr, 3)
        x_values.append(tr)
        print('Threshold: {}'.format(tr))
        y_pred = threshold_probas_multiclass(y_probas, tr)
        fdr = 1 - precision_score(y_test, y_pred, average='weighted')
        sen = recall_score(y_test, y_pred, average='weighted')

        y_fdr.append(fdr)
        y_sen.append(sen)

        if sen > max_sig:
            max_sig = sen
            best_tr = tr

    return x_values, y_fdr, y_sen, best_tr

def significance_score_multiclass_compare(y, y_pred, weights={}, total
    ={}, compensation=0, verbose=True):
    """ Approximate significance score for prediction.

```

```

        Used to compare with known results for given weights, number of
        events, and background compensation constant.
    @param y: Ground truth labels
    @param y_pred: Predicted probabilities
    @param weights: Class weights
    @param total: Number of events in given class
    @param compensation: Other backgrounds constants
    @param verbose: Turn on displaying information about computation
    @return: List containing efficiencies, S, B, and two significances
            (norml & simplified)
    """

    # Selected
    cm = confusion_matrix(y, y_pred)
    cm_len = np.shape(cm)[0]

    # Efficiencies
    sig_total = cm[0, 0]
    epsilon_bgrs = list()
    bgr_sums = list()
    for i in range(1, cm_len):
        sig_total += cm[0, i]
        epsilon_bgrs.append(cm[i, 0])
        bgr_sums.append(sum(cm[i, j] for j in range(cm_len)))
    epsilon_sig = cm[0, 0] / sig_total
    for i in range(len(epsilon_bgrs)):
        epsilon_bgrs[i] = epsilon_bgrs[i] / bgr_sums[i]

    S = epsilon_sig * total[0] * weights[0]
    B = 0
    for i in range(len(epsilon_bgrs)):
        B += epsilon_bgrs[i] * total[i + 1] * weights[i + 1]
    B += compensation

    if verbose:
        print('ml_utils.sig_score():')
        print('\t| -> eps 0: %0.4f' % epsilon_sig)
        for i in range(len(epsilon_bgrs)):
            print('\t| -> eps %d: %0.4f' % (i + 1, epsilon_bgrs[i]))
        print('\t| -> S: %0.4f' % S)
        print('\t| -> B: %0.4f' % B)

    signif = S / np.sqrt(S + B)
    signif_simp = S / np.sqrt(B)
    eps = list()
    eps.append(epsilon_sig)
    for i in range(len(epsilon_bgrs)):
        eps.append(epsilon_bgrs[i])

    ret_dic = {'eps': eps,
               'S': S,
               'B': B,
               'signif': signif,
               'signif_simp': signif_simp}

    return ret_dic

def plot_threshold(x_values, y_values, optimums, title, ylabel, colors,
                  labels, force_threshold_value=None):
    """ Plots graphs for threshold characteristics.
    @param x_values: Values for x-axis (list)
    @param y_values: Values for y-axis (list of lists)
    @param optimums: Whether to search max or min values in y_values
    @param title: Title of plot
    @param colors: Colors for y_values

```



```

@param labels: Labels for y_values
@param force_threshold_value: Value of forced vertical line value (
    if None, vertical line is computed with usage of
    optimums parameter)
"""

plt.figure()
for values, optimum, color, label in zip(y_values, optimums, colors
    , labels):
    plt.plot(x_values, values, color=color, label=label)

    if force_threshold_value is None:
        best_index = 0
        if optimum == 'max':
            best_score = 0
        else:
            best_score = max(values)
        for i, v in enumerate(values):
            if optimum == 'max':
                if best_score < v:
                    best_score = v
                    best_index = i
            else:
                if best_score > v:
                    best_score = v
                    best_index = i
    else:
        best_index = int(force_threshold_value * 100)
        best_score = values[best_index]

    # Plot a dotted vertical line at the best score for that scorer
    marked by x
    plt.plot([x_values[best_index], ] * 2, [0, best_score],
        linestyle='-.', color=color, marker='x',
        markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    plt.annotate("%0.3f" % best_score,
        (x_values[best_index], best_score + 0.005))

plt.xticks(np.arange(0.0, 1.0, step=0.1))
plt.xlabel('Threshold [-]')
plt.ylabel(ylabel)
plt.title(title)
plt.legend(loc="best")

def train_model(X, y, name=None, parameters={}, svd_components=0):
    """ Trains classifier with the data. Also performs basic scaling of
        data.
    @param X: Data
    @param y: Ground truth labels
    @param name: Name of the classifier (None, RFC, KNC, GNB, ADA, GBC)
    @param parameters: Parameters for given model
    @param svd_components: Number of components for Truncated SVD
    @return: Scikit-learn pipeline
    """

    print('ml_utils.train_model():')

    # Scaler
    sc = StandardScaler()

    if name is None or name == 'RFC':
        # Random Forest Classifier - ensemble of decision trees

```

```

print('\t|-> training RandomForestClassifier')
clf = RandomForestClassifier(**parameters)

pipe = Pipeline(steps=[
    ('sc', sc),
    ('clf', clf)])

pipe.fit(X, y)

print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'KNC':
    # K-Neighbors Classifier

    print('\t|-> training KNeighborsClassifier')
    clf = KNeighborsClassifier(**parameters)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

    pipe.fit(X, y)

    print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'GNB':
    # Gaussian Naive Bayes Classifier

    print('\t|-> training GaussianNB')
    clf = GaussianNB(**parameters)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

    pipe.fit(X, y)

    print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'ADA':
    # Gaussian Adaptive Boosting Classifier

    print('\t|-> training AdaBoostClassifier')
    clf = AdaBoostClassifier(**parameters)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

```

```

pipe.fit(X, y)

print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'GBC':
    # Gradient Boosting Classifier

    print('\t|-> training GradientBoostingClassifier')
    clf = GradientBoostingClassifier(**parameters)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

    pipe.fit(X, y)

    print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'MLPC':
    # Multi-Layer Perceptron Classifier

    print('\t|-> training MLPClassifier')
    clf = MLPClassifier(**parameters)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

    pipe.fit(X, y)

    print('\t|-> mean score: %0.2f' % pipe.score(X, y))

elif name == 'SVC':
    # Support Vector Classifier
    N_ESTIMATORS = 10

    print('\t|-> training SVC')
    clf = BaggingClassifier(SVC(**parameters), max_samples=1.0 /
        N_ESTIMATORS, n_estimators=N_ESTIMATORS, n_jobs=-1)

    if svd_components > 0:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('svd', TruncatedSVD(n_components=svd_components)),
            ('clf', clf)])
    else:
        pipe = Pipeline(steps=[
            ('sc', sc),
            ('clf', clf)])

    pipe.fit(X, y)

    print('\t|-> mean score: %0.2f' % pipe.score(X, y))

```

```

    return pipe

def predict(model, X):
    """ Predicts outputs for model (pipeline).
    @param model: Scikit-learn model (pipeline)
    @param X: Inputs for prediction
    @return: Predicted classes
    """

    return model.predict(X)

def predict_proba(model, X):
    """ Predicts outputs for model (pipeline).
    @param model: Scikit-learn model (pipeline)
    @param X: Inputs for prediction
    @return: Predicted class probabilities
    """

    return model.predict_proba(X)

def plot_histograms_multiclass(X, y, classes, feature, bins=None,
                               saving=False, save_folder=None):
    """ Plots histograms with kernel density estimation for multiclass
        data.
    @param X: Data
    @param y: Ground truth labels
    @param classes: Dictionary with classes
    @param feature: Name of feature for which histogram should be
        plotted
    @param bins: Number of bins (None will force automatic selection)
    @param saving: Whether to save plot or no
    @param save_folder: Where to save plot (only when saving is True)
    """

    plt.figure()
    plt.title(feature)
    for key in classes:
        sns.distplot(X[y == key], bins, hist_kws={'histtype': 'step'},
                     kde_kws={'label': '%s (KDE)' % classes[key]},
                     label='%s' % classes[key])

    plt.legend(loc="best")

    if saving:
        plt.savefig(save_folder + feature + '.png')
        plt.close()

def plot_cm(y_pred, y, n, classes, title):
    """ Plots confusion matrix with usage of pretty cm library.
    @param y_pred: Predicted classes
    @param y: Ground truth labels
    @param n: Number of classes (size of cm: nXn)
    @param classes: List with classes names
    @param title: Title of plot
    """

    cm = confusion_matrix(y_pred, y)
    df_cm = pd.DataFrame(cm, range(n), range(n))
    cmpp.pretty_plot_confusion_matrix(df_cm, cmap=make_cmap([(240, 248,
        255), (240, 248, 255)], bit=True), title=title,
                                     classes=classes)

```



```

for key in scores:
    plt.plot([0, 0], [0, 0], color='k', linestyle='--', label='{0:
        {}}'.format(key, np.round(scores[key], 2)))
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--', label='
    Chance')
for i in range(n_classes):
    plt.plot(fprs[i], tprs[i], label='ROC %d - %s (AUC = %0.2f)' %
        (i, classes[i], aucs[i]))

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(title)
plt.legend(loc="lower right")

def plot_pred_multiclass(title, y_preds, classifiers, classes, colors):
    """ Plots predict arrays for multi-class classification.
    @param title: Title of plot
    @param y_preds: List of predictions for classifiers
    @param classifiers: List of classifiers
    @param classes: Dictionary of classes
    @param colors: Colors used for classes
    """

    n_rows = len(y_preds)
    n = len(y_preds[0])
    cmap = make_cmap(colors)

    fig, ax = plt.subplots(nrows=n_rows, figsize=(10, 5))
    plt.subplots_adjust(left=0, bottom=0, right=10, top=1, wspace=0,
        hspace=0)
    ax[0].set_title(title, fontsize=14)

    # Plot bars
    for i, row in enumerate(ax):
        row.imshow(np.array(y_preds[i])[np.newaxis, :], cmap=cmap,
            aspect='auto')
        pos = list(row.get_position().bounds)
        x_text = 0.05
        y_text = pos[1] + 0.011 * (i + 1)
        fig.text(x_text, y_text, classifiers[i], va='center', ha='right',
            , fontsize=10)
        row.set_xlim([-n * 0.1, n + n * 0.1])
        row.set_axis_off()

    # Plot legend
    for key in classes:
        plt.plot([0, 0], [0, 0], color=colors[key], linestyle='--',
            label='{0: {}}'.format(classes[key]))

    plt.legend(loc="lower right")
    plt.tight_layout()

def plot_pred_hist_multiclass(y_preds, classifiers, classes, colors):
    """ Plots predict histograms for multi-class classification.
    @param y_preds: List of predictions for classifiers
    @param classifiers: List of classifiers
    @param classes: Dictionary of classes
    @param colors: Colors used for classes
    """

    for i, y_pred in enumerate(y_preds):
        y_pred = np.array(y_pred)

```



```

        alpha=0.1 if sample == 'test' else 0, color
        =color,
        label="std %s (%s)" % (scorer, sample) if
        sample == 'test' else '')
    ax.plot(X_axis, sample_score_mean, style, color=color,
            alpha=1 if sample == 'test' else 0.7,
            label="mean %s (%s)" % (scorer, sample))

    best_index = 0
    best_score = 0
    for i, r in enumerate(results['mean_test_%s' % scorer]):
        if best_score < r:
            best_score = r
            best_index = i
    # Plot a dotted vertical line at the best score for that scorer
    # marked by x
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x',
            markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax.annotate("%0.3f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")

# Fit time
plt.figure(figsize=(8, 8))
plt.title(name + ': ' + tuned_parameter + ' tuning', fontsize=16)
plt.xlabel(tuned_parameter)
plt.ylabel('time [s]')

ax = plt.gca()
ax.set_xlim(score_x_min, score_x_max)
ax.set_ylim(0, time_y_max)

for sample, color in (('fit', 'r'), ('score', 'g')):
    sample_score_mean = results['mean_%s_time' % sample]
    sample_score_std = results['std_%s_time' % sample]
    ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                    sample_score_mean + sample_score_std,
                    alpha=0.1, color=color, label="std Time (%s)" %
                    sample)
    ax.plot(X_axis, sample_score_mean, '-', color=color,
            alpha=1,
            label="mean Time (%s)" % sample)

    best_index = 0
    best_score = max(results['mean_%s_time' % sample])
    for i, r in enumerate(results['mean_%s_time' % sample]):
        if best_score > r:
            best_score = r
            best_index = i
    ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x',
            markeredgewidth=3, ms=8)

    ax.annotate("%0.1f" % best_score,
                (X_axis[best_index], best_score + 0.005))

plt.legend(loc="best")

def plot_feat_imp(title, model, f, limit, verbose=True):
    """ Plots feature importances for model containing RFC
    classifier.

```



```
@param title: Title of plot
@param model: RFC classifier or pipeline containing one
@param f: List of features
@param limit: Number of features to be plot
@param verbose: Whether lor importances to console
"""

imp = np.round(model['clf'].feature_importances_, 5)
std = np.round(np.std([tree.feature_importances_ for tree in model
    ['clf'].estimators_], axis=0), 5)
ind = np.argsort(imp)[::-1]

imp_plot = list()
std_plot = list()
ind_plot = list()

if verbose:
    print("\nFeatures importance:")
for i in range(limit):
    imp_plot.append(imp[ind[i]])
    std_plot.append(std[ind[i]])
    ind_plot.append(ind[i])
    if verbose:
        print(f[ind_plot[-1]] + ', imp: {0}, std: {0}'.format(
            imp_plot[-1], std_plot[-1]))

plt.figure()
plt.title(title)
plt.bar(range(limit), imp_plot, color="r", label="Importance (with
    std)", yerr=std_plot, align="center")
plt.xticks(range(limit), ind_plot)
plt.xlim([-1, limit])
plt.xlabel('Feature number [-]')
plt.ylabel('Importance [-]')
plt.title(title)
plt.legend(loc='upper right')
```



```
"""
Last modified May 14 2020
@author: Jakub Maly
"""

import utils.converter as converter

def convert_it(data_folder, file_names):
    cv = converter.Converter({'verbose': 10})
    for file_name in file_names:
        cv.open(data_folder + file_name + '.root')
        cv.scan('nominal', '*')
        cv.convert()
        cv.save(data_folder, file_name)

def main():
```

```

""" Older data """
data_folder = '../root_data/mc16a/older/'
file_names = ['345672', '345673', '345674', '410155', '410156',
              '410157', '410218', '410219', '410220']
convert_it(data_folder, file_names)

data_folder = '../root_data/mc16d/older/'
file_names = ['345672', '345673', '345674', '410155', '410156',
              '410157', '410218', '410219', '410220']
convert_it(data_folder, file_names)

""" Newer data """
data_folder = '../root_data/mc16a/newer/'
file_names = ['345873', '345874', '345875', '413008_wCharge']
convert_it(data_folder, file_names)

data_folder = '../root_data/mc16d/newer/'
file_names = ['345873', '345874', '345875', '413008_wCharge']
convert_it(data_folder, file_names)

""" Real data """
data_folder = '../root_data/mc16a/real/'
file_names = ['data15', 'data16']
convert_it(data_folder, file_names)

data_folder = '../root_data/mc16d/real/'
file_names = ['data17']
convert_it(data_folder, file_names)

if __name__ == '__main__':
    """ Script demonstrating root files conversion.

    In this script several files are opened and scanned. Conversion
    with saving is performed in the end.
    """

    main()

```

■ D.2.2 data_filter.py

```

"""
Last modified May 19 2020
@author: Jakub Maly
"""

import utils.reader as reader

# Use new features (77 instead of 83)
# NEW_FEAT = False
NEW_FEAT = True

def filter_it(data_folder, file_names, features, channels):
    rd = reader.Reader({'verbose': 10})
    for file_name in file_names:
        rd.open(data_folder + file_name + '.pkl')
        rd.filt_channels(features, channels)
        if NEW_FEAT:
            rd.save(data_folder + file_name + '_fil_new.pkl')
        else:
            rd.save(data_folder + file_name + '_fil.pkl')

```



```

        'lep_isolationFixedCutLoose_0', '
            lep_isolationFixedCutLoose_1', '
            lep_isolationFixedCutLoose_2',
        'lep_isolationFixedCutLoose_3', '
            lep_promptLeptonVeto_TagWeight_0',
        'lep_promptLeptonVeto_TagWeight_1',
        'lep_promptLeptonVeto_TagWeight_2', 'nJets_OR', '
            nJets_OR_MV2c10_70', 'nJets_OR_T',
        'nJets_OR_T_MV2c10_70', 'nTaus_OR_Pt25', '
            tau_MV2c10_0', 'tau_btag70_0', 'tau_btag70_1',
        'tau_charge_0',
        'tau_charge_1', 'tau_fromPV_0', 'tau_passEleBDT_0',
            'tau_passEleBDT_1', 'tau_passMuonOLR_0',
        'tau_passMuonOLR_1', 'tau_tagWeightBin_0', '
            tau_tagWeightBin_1', 'tau_tight_0', '
            tau_tight_1',
        'top1Mass', 'top2Mass', 'total_charge', '
            trilep_type', 'jet_pT', 'total_leptons', 'HT*']

channels = ['is2LSS1Tau']

""" Older data """
data_folder = '../root_data/mc16a/older/'
file_names = ['345672', '345673', '345674', '410155', '410156',
    '410157', '410218', '410219', '410220']
filter_it(data_folder, file_names, features, channels)

data_folder = '../root_data/mc16d/older/'
file_names = ['345672', '410155', '410156', '410157', '410218',
    '410219', '410220']
filter_it(data_folder, file_names, features, channels)

""" Newer data """
data_folder = '../root_data/mc16a/newer/'
file_names = ['345873', '345874', '345875', '413008_wCharge']
filter_it(data_folder, file_names, features, channels)

data_folder = '../root_data/mc16d/newer/'
file_names = ['345873', '345874', '345875', '413008_wCharge']
filter_it(data_folder, file_names, features, channels)

""" Real data """
data_folder = '../root_data/mc16a/real/'
file_names = ['data15', 'data16']
filter_it(data_folder, file_names, features, channels)

data_folder = '../root_data/mc16d/real/'
file_names = ['data17']
filter_it(data_folder, file_names, features, channels)

if __name__ == '__main__':
    """ Script demonstrating pkl files filtering.

    In this script several files are opened and filtered. Saving is
    performed in the end.
    """

    main()

```

■ D.2.3 data_prepare.py

```

"""
Last modified May 19 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

# Use new features (77 instead of 83)
# NEW_FEAT = False
NEW_FEAT = True

def main():
    """ mc16a/older - mainly used for training """
    if NEW_FEAT:
        data = {0: ['../root_data/mc16a/older/345672_fil_new.pkl', '../root_data/mc16a/older/345673_fil_new.pkl',
                    '../root_data/mc16a/older/345674_fil_new.pkl'],
                1: ['../root_data/mc16a/older/410155_fil_new.pkl'],
                2: ['../root_data/mc16a/older/410156_fil_new.pkl', '../root_data/mc16a/older/410157_fil_new.pkl',
                    '../root_data/mc16a/older/410218_fil_new.pkl', '../root_data/mc16a/older/410219_fil_new.pkl',
                    '../root_data/mc16a/older/410220_fil_new.pkl']}
    else:
        data = {0: ['../root_data/mc16a/older/345672_fil.pkl', '../root_data/mc16a/older/345673_fil.pkl',
                    '../root_data/mc16a/older/345674_fil.pkl'],
                1: ['../root_data/mc16a/older/410155_fil.pkl'],
                2: ['../root_data/mc16a/older/410156_fil.pkl', '../root_data/mc16a/older/410157_fil.pkl',
                    '../root_data/mc16a/older/410218_fil.pkl', '../root_data/mc16a/older/410219_fil.pkl',
                    '../root_data/mc16a/older/410220_fil.pkl']}

    X, y, f = ml_utils.get_X_y_f_multiclass(data, {'verbose': 10})

    if NEW_FEAT:
        ml_utils.save_compress(X, 'data/older', 'X_fil_new_com_mc16a', 'gzip')
        ml_utils.save(y, 'data/older', 'y_fil_new_mc16a')
        ml_utils.save(f, 'data/older', 'f_fil_new_mc16a')
    else:
        ml_utils.save_compress(X, 'data/older', 'X_fil_com_mc16a', 'gzip')
        ml_utils.save(y, 'data/older', 'y_fil_mc16a')
        ml_utils.save(f, 'data/older', 'f_fil_mc16a')

    """ mc16d/older - mainly used for testing """
    if NEW_FEAT:
        data = {0: ['../root_data/mc16d/older/345672_fil_new.pkl'],
                1: ['../root_data/mc16d/older/410155_fil_new.pkl'],
                2: ['../root_data/mc16d/older/410156_fil_new.pkl', '../root_data/mc16d/older/410157_fil_new.pkl',
                    '../root_data/mc16d/older/410218_fil_new.pkl', '../root_data/mc16d/older/410219_fil_new.pkl',
                    '../root_data/mc16d/older/410220_fil_new.pkl']}
    else:
        data = {0: ['../root_data/mc16d/older/345672_fil.pkl'],
                1: ['../root_data/mc16d/older/410155_fil.pkl'],

```

```

        2: ['../root_data/mc16d/older/410156_fil.pkl', '../
            root_data/mc16d/older/410157_fil.pkl',
            '../root_data/mc16d/older/410218_fil.pkl', '../
            root_data/mc16d/older/410219_fil.pkl',
            '../root_data/mc16d/older/410220_fil.pkl']
    }

X, y, f = ml_utils.get_X_y_f_multiclass(data, {'verbose': 10})

if NEW_FEAT:
    ml_utils.save_compress(X, 'data/older', 'X_fil_new_com_mc16d',
                           'gzip')
    ml_utils.save(y, 'data/older', 'y_fil_new_mc16d')
    ml_utils.save(f, 'data/older', 'f_fil_new_mc16d')
else:
    ml_utils.save_compress(X, 'data/older', 'X_fil_com_mc16d', '
                           gzip')
    ml_utils.save(y, 'data/older', 'y_fil_mc16d')
    ml_utils.save(f, 'data/older', 'f_fil_mc16d')

""" newer/mc16a&mc16d - mainly used for testing and comparison with
    internal note """
if NEW_FEAT:
    data = {
        0: ['../root_data/mc16a/newer/345873_fil_new.pkl', '../
            root_data/mc16a/newer/345874_fil_new.pkl',
            '../root_data/mc16a/newer/345875_fil_new.pkl', '../
            root_data/mc16d/newer/345873_fil_new.pkl',
            '../root_data/mc16d/newer/345874_fil_new.pkl', '../
            root_data/mc16d/newer/345875_fil_new.pkl'],
        1: ['../root_data/mc16a/newer/413008_wCharge_fil_new.pkl',
            '../root_data/mc16d/newer/413008_wCharge_fil_new.pkl'],
        2: ['../root_data/mc16a/newer/410156_fil_new.pkl', '../
            root_data/mc16a/newer/410157_fil_new.pkl',
            '../root_data/mc16a/newer/410218_fil_new.pkl', '../
            root_data/mc16a/newer/410219_fil_new.pkl',
            '../root_data/mc16a/newer/410220_fil_new.pkl', '../
            root_data/mc16d/newer/410156_fil_new.pkl',
            '../root_data/mc16d/newer/410157_fil_new.pkl',
            '../root_data/mc16d/newer/410218_fil_new.pkl',
            '../root_data/mc16d/newer/410219_fil_new.pkl',
            '../root_data/mc16d/newer/410220_fil_new.pkl']
    }
else:
    data = {0: ['../root_data/mc16a/newer/345873_fil.pkl', '../
        root_data/mc16a/newer/345874_fil.pkl',
            '../root_data/mc16a/newer/345875_fil.pkl', '../
            root_data/mc16d/newer/345873_fil.pkl',
            '../root_data/mc16d/newer/345874_fil.pkl', '../
            root_data/mc16d/newer/345875_fil.pkl'],
        1: ['../root_data/mc16a/newer/413008_wCharge_fil.pkl',
            '../root_data/mc16d/newer/413008_wCharge_fil.pkl'],
        2: ['../root_data/mc16a/newer/410156_fil.pkl', '../
            root_data/mc16a/newer/410157_fil.pkl',
            '../root_data/mc16a/newer/410218_fil.pkl', '../
            root_data/mc16a/newer/410219_fil.pkl',
            '../root_data/mc16a/newer/410220_fil.pkl', '../
            root_data/mc16d/newer/410156_fil.pkl',
            '../root_data/mc16d/newer/410157_fil.pkl',
            '../root_data/mc16d/newer/410218_fil.pkl',
            '../root_data/mc16d/newer/410219_fil.pkl',
            '../root_data/mc16d/newer/410220_fil.pkl']
    }

X, y, f = ml_utils.get_X_y_f_multiclass(data, {'verbose': 10})

```



```

train_split = 80
test_split = 100 - train_split

""" older/mc16a - train and test splits """
if NEW_FEAT:
    X = ml_utils.load_compress('data/older/X_fil_new_com_mc16a.pkl.
                               gzip', 'gzip')
    y = ml_utils.load('data/older/y_fil_new_mc16a.pkl')
    f = ml_utils.load('data/older/f_fil_new_mc16a.pkl')
else:
    X = ml_utils.load_compress('data/older/X_fil_com_mc16a.pkl.gzip
                               ', 'gzip')
    y = ml_utils.load('data/older/y_fil_mc16a.pkl')
    f = ml_utils.load('data/older/f_fil_mc16a.pkl')

# Shuffle and slit data
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=random_state, train_size=train_split / 100)
# Returns data valid for channel, also removes channel from
# features
X_ch, _, y_ch, _, _ = ml_utils.apply_channel(X, X, y, y, f, '
is2LSS1Tau')
X_train_ch, X_test_ch, y_train_ch, y_test_ch, _ = ml_utils.
    apply_channel(X_train, X_test, y_train, y_test, f,
,
is2LSS1Tau
')

# Removes channel from features
X, _, f_new = ml_utils.remove_channel(X, X, f, 'is2LSS1Tau')
X_train, X_test, _ = ml_utils.remove_channel(X_train, X_test, f, '
is2LSS1Tau')

# Save
if NEW_FEAT:
    ml_utils.save_compress(X, 'splits/older', 'X_new_mc16a', 'gzip
    ')
    ml_utils.save(y, 'splits/older', 'y_new_mc16a')
    ml_utils.save_compress(X_ch, 'splits/older', 'X_new_ch_mc16a',
    'gzip')
    ml_utils.save(y_ch, 'splits/older', 'y_new_ch_mc16a')
    ml_utils.save_compress(X_train, 'splits/older', 'X_new_train_{}
    _mc16a'.format(train_split), 'gzip')
    ml_utils.save(y_train, 'splits/older', 'y_new_train_{}_mc16a'.
    format(train_split))
    ml_utils.save_compress(X_test, 'splits/older', 'X_new_test_{}
    _mc16a'.format(test_split), 'gzip')
    ml_utils.save(y_test, 'splits/older', 'y_new_test_{}_mc16a'.
    format(test_split))
    ml_utils.save_compress(X_train_ch, 'splits/older', '
    X_new_train_{}_ch_mc16a'.format(train_split), 'gzip')
    ml_utils.save(y_train_ch, 'splits/older', 'y_new_train_{}
    _ch_mc16a'.format(train_split))
    ml_utils.save_compress(X_test_ch, 'splits/older', 'X_new_test_
    {}_ch_mc16a'.format(test_split), 'gzip')
    ml_utils.save(y_test_ch, 'splits/older', 'y_new_test_{}
    _ch_mc16a'.format(test_split))
    ml_utils.save(f_new, 'splits/older', 'f_new_mc16a')
else:
    ml_utils.save_compress(X, 'splits/older', 'X_mc16a', 'gzip')
    ml_utils.save(y, 'splits/older', 'y_mc16a')
    ml_utils.save_compress(X_ch, 'splits/older', 'X_ch_mc16a', '
    gzip')
    ml_utils.save(y_ch, 'splits/older', 'y_ch_mc16a')
    ml_utils.save_compress(X_train, 'splits/older', 'X_train_{}
    _mc16a'.format(train_split), 'gzip')

```



```

ml_utils.save(y_train, 'splits/older', 'y_train_{}_mc16a'.
              format(train_split))
ml_utils.save_compress(X_test, 'splits/older', 'X_test_{}_mc16a'
                      '.format(test_split), 'gzip')
ml_utils.save(y_test, 'splits/older', 'y_test_{}_mc16a'.format(
    test_split))
ml_utils.save_compress(X_train_ch, 'splits/older', 'X_train_{}_
    _ch_mc16a'.format(train_split), 'gzip')
ml_utils.save(y_train_ch, 'splits/older', 'y_train_{}_ch_mc16a'
              '.format(train_split))
ml_utils.save_compress(X_test_ch, 'splits/older', 'X_test_{}_
    _ch_mc16a'.format(test_split), 'gzip')
ml_utils.save(y_test_ch, 'splits/older', 'y_test_{}_ch_mc16a'.
              format(test_split))
ml_utils.save(f_new, 'splits/older', 'f_mc16a')

""" older/mc16d - test split """
f_prev = f

if NEW_FEAT:
    X = ml_utils.load_compress('data/older/X_fil_new_com_mc16d.pkl.
        gzip', 'gzip')
    y = ml_utils.load('data/older/y_fil_new_mc16d.pkl')
    f = ml_utils.load('data/older/f_fil_new_mc16d.pkl')
else:
    X = ml_utils.load_compress('data/older/X_fil_com_mc16d.pkl.gzip',
        'gzip')
    y = ml_utils.load('data/older/y_fil_mc16d.pkl')
    f = ml_utils.load('data/older/f_fil_mc16d.pkl')

# Check if new data have same feature order (can get shuffled
    during conversion)
X, y, f = ml_utils.check_feature_order(X, y, f, f_prev)

X_ch, _, y_ch, _, _ = ml_utils.apply_channel(X, X, y, y, f, '
    is2LSS1Tau')
X, _, f_new = ml_utils.remove_channel(X, X, f, 'is2LSS1Tau')

if NEW_FEAT:
    ml_utils.save_compress(X, 'splits/older', 'X_new_mc16d', 'gzip'
        ')
    ml_utils.save(y, 'splits/older', 'y_new_mc16d')
    ml_utils.save_compress(X_ch, 'splits/older', 'X_new_ch_mc16d',
        'gzip')
    ml_utils.save(y_ch, 'splits/older', 'y_new_ch_mc16d')
    ml_utils.save(f_new, 'splits/older', 'f_new_mc16d')
else:
    ml_utils.save_compress(X, 'splits/older', 'X_mc16d', 'gzip')
    ml_utils.save(y, 'splits/older', 'y_mc16d')
    ml_utils.save_compress(X_ch, 'splits/older', 'X_ch_mc16d', '
        gzip')
    ml_utils.save(y_ch, 'splits/older', 'y_ch_mc16d')
    ml_utils.save(f_new, 'splits/older', 'f_mc16d')

""" newer/mc16a&mc16d - train and test splits """
if NEW_FEAT:
    X = ml_utils.load_compress('data/newer/
        X_fil_new_com_mc16a_mc16d.pkl.gzip', 'gzip')
    y = ml_utils.load('data/newer/y_fil_new_mc16a_mc16d.pkl')
    f = ml_utils.load('data/newer/f_fil_new_mc16a_mc16d.pkl')
else:
    X = ml_utils.load_compress('data/newer/X_fil_com_mc16a_mc16d.
        pkl.gzip', 'gzip')
    y = ml_utils.load('data/newer/y_fil_mc16a_mc16d.pkl')
    f = ml_utils.load('data/newer/f_fil_mc16a_mc16d.pkl')

```

```

X, y, f = ml_utils.check_feature_order(X, y, f, f_prev)

# Shuffle and split data
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    random_state=random_state, train_size=train_split / 100)

X_ch, _, y_ch, _, _ = ml_utils.apply_channel(X, X, y, y, f, '
is2LSS1Tau')
X_train_ch, X_test_ch, y_train_ch, y_test_ch, _ = ml_utils.
    apply_channel(X_train, X_test, y_train, y_test, f,
,
is2LSS1Tau
')

X, _, f_new = ml_utils.remove_channel(X, X, f, 'is2LSS1Tau')
X_train, X_test, _ = ml_utils.remove_channel(X_train, X_test, f, '
is2LSS1Tau')

if NEW_FEAT:
    ml_utils.save_compress(X, 'splits/newer', 'X_new_mc16a_mc16d',
        'gzip')
    ml_utils.save(y, 'splits/newer', 'y_new_mc16a_mc16d')
    ml_utils.save_compress(X_train, 'splits/newer', 'X_new_train_{}
        _mc16a_mc16d'.format(train_split), 'gzip')
    ml_utils.save(y_train, 'splits/newer', 'y_new_train_{}
        _mc16a_mc16d'.format(train_split))
    ml_utils.save_compress(X_test, 'splits/newer', 'X_new_test_{}
        _mc16a_mc16d'.format(test_split), 'gzip')
    ml_utils.save(y_test, 'splits/newer', 'y_new_test_{}
        _mc16a_mc16d'.format(test_split))
    ml_utils.save_compress(X_train_ch, 'splits/newer', '
        X_new_train_{}_ch_mc16a_mc16d'.format(train_split), 'gzip')
    ml_utils.save(y_train_ch, 'splits/newer', 'y_new_train_{}
        _ch_mc16a_mc16d'.format(train_split))
    ml_utils.save_compress(X_test_ch, 'splits/newer', 'X_new_test_
        {}_ch_mc16a_mc16d'.format(test_split), 'gzip')
    ml_utils.save(y_test_ch, 'splits/newer', 'y_new_test_{}
        _ch_mc16a_mc16d'.format(test_split))
    ml_utils.save_compress(X_ch, 'splits/newer', '
        X_new_ch_mc16a_mc16d', 'gzip')
    ml_utils.save(y_ch, 'splits/newer', 'y_new_ch_mc16a_mc16d')
    ml_utils.save(f_new, 'splits/newer', 'f_new_mc16a_mc16d')
else:
    ml_utils.save_compress(X, 'splits/newer', 'X_mc16a_mc16d', '
        gzip')
    ml_utils.save(y, 'splits/newer', 'y_mc16a_mc16d')
    ml_utils.save_compress(X_train, 'splits/newer', 'X_train_{}
        _mc16a_mc16d'.format(train_split), 'gzip')
    ml_utils.save(y_train, 'splits/newer', 'y_train_{}_mc16a_mc16d
        '.format(train_split))
    ml_utils.save_compress(X_test, 'splits/newer', 'X_test_{}
        _mc16a_mc16d'.format(test_split), 'gzip')
    ml_utils.save(y_test, 'splits/newer', 'y_test_{}_mc16a_mc16d'.
        format(test_split))
    ml_utils.save_compress(X_train_ch, 'splits/newer', 'X_train_{}
        _ch_mc16a_mc16d'.format(train_split), 'gzip')
    ml_utils.save(y_train_ch, 'splits/newer', 'y_train_{}
        _ch_mc16a_mc16d'.format(train_split))
    ml_utils.save_compress(X_test_ch, 'splits/newer', 'X_test_{}
        _ch_mc16a_mc16d'.format(test_split), 'gzip')
    ml_utils.save(y_test_ch, 'splits/newer', 'y_test_{}
        _ch_mc16a_mc16d'.format(test_split))
    ml_utils.save_compress(X_ch, 'splits/newer', 'X_ch_mc16a_mc16d
        ', 'gzip')
    ml_utils.save(y_ch, 'splits/newer', 'y_ch_mc16a_mc16d')
    ml_utils.save(f_new, 'splits/newer', 'f_mc16a_mc16d')

```

```

""" SVC train split: 10% of original train split - newer data are
    used for NEW_FEAT"""
if NEW_FEAT:
    X = ml_utils.load_compress('splits/newer/X_new_mc16a_mc16d.pkl.
        gzip', 'gzip')
    y = ml_utils.load('splits/newer/y_new_mc16a_mc16d.pkl')
    X_train, X_test, y_train, y_test = train_test_split(X, y,
        train_size=0.1)
    ml_utils.save_compress(X_train, 'splits/newer', '
        X_new_mc16a_mc16d_svc', 'gzip')
    ml_utils.save(y_train, 'splits/newer', 'y_new_mc16a_mc16d_svc')
else:
    X = ml_utils.load_compress('splits/older/X_mc16a.pkl.gzip', '
        gzip')
    y = ml_utils.load('splits/older/y_mc16a.pkl')
    X_train, X_test, y_train, y_test = train_test_split(X, y,
        train_size=0.1)
    ml_utils.save_compress(X_train, 'splits/older', 'X_mc16a_svc',
        'gzip')
    ml_utils.save(y_train, 'splits/older', 'y_mc16a_svc')

""" real/mc16a&mc16d - test split """
if NEW_FEAT:
    X = ml_utils.load_compress('data/real/X_fil_new_com_mc16a_mc16d
        .pkl.gzip', 'gzip')
    y = np.zeros(X.shape[0])
    f = ml_utils.load('data/real/f_fil_new_mc16a_mc16d.pkl')
else:
    X = ml_utils.load_compress('data/real/X_fil_com_mc16a_mc16d.pkl
        .gzip', 'gzip')
    y = np.zeros(X.shape[0])
    f = ml_utils.load('data/real/f_fil_mc16a_mc16d.pkl')

X, y, f = ml_utils.check_feature_order(X, y, f, f_prev)

X_ch, _, y_ch, _, _ = ml_utils.apply_channel(X, X, y, y, f, '
    is2LSS1Tau')
X, _, f_new = ml_utils.remove_channel(X, X, f, 'is2LSS1Tau')

if NEW_FEAT:
    ml_utils.save_compress(X, 'splits/real', 'X_new_mc16a_mc16d', '
        gzip')
    ml_utils.save_compress(X_ch, 'splits/real', '
        X_new_ch_mc16a_mc16d', 'gzip')
    ml_utils.save(f_new, 'splits/real', 'f_new_mc16a_mc16d')
else:
    ml_utils.save_compress(X, 'splits/real', 'X_mc16a_mc16d', 'gzip
        ')
    ml_utils.save_compress(X_ch, 'splits/real', 'X_ch_mc16a_mc16d',
        'gzip')
    ml_utils.save(f_new, 'splits/real', 'f_mc16a_mc16d')

if __name__ == '__main__':
    """ Script demonstrating data splitting.

    In this script data are opened and split. Channel is obtained from
    the splits and channel feature is removed.
    """

    main()

```

■ D.2.5 data_view.py

```

"""
Last modified May 18 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

import numpy as np

# Use new features (77 instead of 83)
NEW_FEAT = False
# NEW_FEAT = True

# Print features' names
FEATURES = False
# FEATURES = True

# Plot and save histograms - for training and real data
# input files and output folder need to be changed manually in the code
# if histograms for different data are desired
HISTOGRAMS = False
# HISTOGRAMS = True

def main():
    """ older/mc16a """
    if NEW_FEAT:
        X = ml_utils.load_compress('splits/older/X_new_mc16a.pkl.gz',
                                   'gzip')
        y = ml_utils.load('splits/older/y_new_mc16a.pkl')
        X_ch = ml_utils.load_compress('splits/older/X_new_ch_mc16a.pkl.
                                     gzip', 'gzip')
        y_ch = ml_utils.load('splits/older/y_new_ch_mc16a.pkl')
        f = ml_utils.load('splits/older/f_new_mc16a.pkl')
    else:
        X = ml_utils.load_compress('splits/older/X_mc16a.pkl.gz', '
                                   gzip')
        y = ml_utils.load('splits/older/y_mc16a.pkl')
        X_ch = ml_utils.load_compress('splits/older/X_ch_mc16a.pkl.gz',
                                     'gzip')
        y_ch = ml_utils.load('splits/older/y_ch_mc16a.pkl')
        f = ml_utils.load('splits/older/f_mc16a.pkl')

    n_feat = len(f)

    print('\nolder mc16a ({} features) data:'.format(n_feat))
    print('features in X: {}'.format(X.shape[1]))
    print('total data in X: {}'.format(X.shape[0]))
    print('{} of ttH'.format(sum(y == 0)))
    print('{} of ttW'.format(sum(y == 1)))
    print('{} of ttZ'.format(sum(y == 2)))

    print('\nolder mc16a ({} features) data - channel:'.format(n_feat))
    print('features in X_ch: {}'.format(X_ch.shape[1]))
    print('total data in X_ch: {}'.format(X_ch.shape[0]))
    print('{} of ttH'.format(sum(y_ch == 0)))
    print('{} of ttW'.format(sum(y_ch == 1)))
    print('{} of ttZ'.format(sum(y_ch == 2)))

    """ older/mc16d """
    if NEW_FEAT:
        X = ml_utils.load_compress('splits/older/X_new_mc16d.pkl.gz',
                                   'gzip')

```

```

y = ml_utils.load('splits/older/y_new_mc16d.pkl')
X_ch = ml_utils.load_compress('splits/older/X_new_ch_mc16d.pkl.
    gzip', 'gzip')
y_ch = ml_utils.load('splits/older/y_new_ch_mc16d.pkl')
f = ml_utils.load('splits/older/f_new_mc16d.pkl')
else:
    X = ml_utils.load_compress('splits/older/X_mc16d.pkl.gzip', '
        gzip')
    y = ml_utils.load('splits/older/y_mc16d.pkl')
    X_ch = ml_utils.load_compress('splits/older/X_ch_mc16d.pkl.gzip
        ', 'gzip')
    y_ch = ml_utils.load('splits/older/y_ch_mc16d.pkl')
    f = ml_utils.load('splits/older/f_mc16d.pkl')

n_feat = len(f)

print('\nolder mc16d ({} features) data:'.format(n_feat))
print('features in X: {}'.format(X.shape[1]))
print('total data in X: {}'.format(X.shape[0]))
print('{} of ttH'.format(sum(y == 0)))
print('{} of ttW'.format(sum(y == 1)))
print('{} of ttZ'.format(sum(y == 2)))

print('\nolder mc16d ({} features) data - channel:'.format(n_feat))
print('features in X_ch: {}'.format(X_ch.shape[1]))
print('total data in X_ch: {}'.format(X_ch.shape[0]))
print('{} of ttH'.format(sum(y_ch == 0)))
print('{} of ttW'.format(sum(y_ch == 1)))
print('{} of ttZ'.format(sum(y_ch == 2)))

""" newer/mc16a&mc16d """
if NEW_FEAT:
    X = ml_utils.load_compress('splits/newer/X_new_mc16a_mc16d.pkl.
        gzip', 'gzip')
    y = ml_utils.load('splits/newer/y_new_mc16a_mc16d.pkl')
    X_ch = ml_utils.load_compress('splits/newer/
        X_new_ch_mc16a_mc16d.pkl.gzip', 'gzip')
    y_ch = ml_utils.load('splits/newer/y_new_ch_mc16a_mc16d.pkl')
    f = ml_utils.load('splits/newer/f_new_mc16a_mc16d.pkl')
else:
    X = ml_utils.load_compress('splits/newer/X_mc16a_mc16d.pkl.gzip
        ', 'gzip')
    y = ml_utils.load('splits/newer/y_mc16a_mc16d.pkl')
    X_ch = ml_utils.load_compress('splits/newer/X_ch_mc16a_mc16d.
        pkl.gzip', 'gzip')
    y_ch = ml_utils.load('splits/newer/y_ch_mc16a_mc16d.pkl')
    f = ml_utils.load('splits/newer/f_mc16a_mc16d.pkl')

print('\nnewer mc16a&mc16d ({} features) data:'.format(n_feat))
print('features in X: {}'.format(X.shape[1]))
print('total data in X: {}'.format(X.shape[0]))
print('{} of ttH'.format(sum(y == 0)))
print('{} of ttW'.format(sum(y == 1)))
print('{} of ttZ'.format(sum(y == 2)))

print('\nnewer mc16a&mc16d ({} features) data - channel:'.format(
    n_feat))
print('features in X_ch: {}'.format(X_ch.shape[1]))
print('total data in X_ch: {}'.format(X_ch.shape[0]))
print('{} of ttH'.format(sum(y_ch == 0)))
print('{} of ttW'.format(sum(y_ch == 1)))
print('{} of ttZ'.format(sum(y_ch == 2)))

""" real/mc16a&mc16d """
if NEW_FEAT:
    X = ml_utils.load_compress('splits/real/X_new_mc16a_mc16d.pkl.

```

```

        gzip', 'gzip')
    X_ch = ml_utils.load_compress('splits/real/X_new_ch_mc16a_mc16d
        .pkl.gzip', 'gzip')
    f = ml_utils.load('splits/real/f_new_mc16a_mc16d.pkl')
else:
    X = ml_utils.load_compress('splits/real/X_mc16a_mc16d.pkl.gzip
        ', 'gzip')
    X_ch = ml_utils.load_compress('splits/real/X_ch_mc16a_mc16d.pkl
        .gzip', 'gzip')
    f = ml_utils.load('splits/real/f_mc16a_mc16d.pkl')

print('\nreal mc16a&mc16d ({ features) data:'.format(n_feat))
print('features in X: {}'.format(X.shape[1]))
print('total data in X: {}'.format(X.shape[0]))

print('\nreal mc16a&mc16d ({ features) data - channel:'.format(
    n_feat))
print('features in X_ch: {}'.format(X_ch.shape[1]))
print('total data in X_ch: {}'.format(X_ch.shape[0]))

if FEATURES:
    print('\n Features:')
    for feature in f:
        print(feature)

if HISTOGRAMS:
    """ older/mc16a, 83 features """
    X = ml_utils.load_compress('splits/older/X_mc16a.pkl.gzip', '
        gzip')
    y = ml_utils.load('splits/older/y_mc16a.pkl')
    X_ch = ml_utils.load_compress('splits/older/X_ch_mc16a.pkl.gzip
        ', 'gzip')
    y_ch = ml_utils.load('splits/older/y_ch_mc16a.pkl')
    f = ml_utils.load('splits/older/f_mc16a.pkl')

    classes = {0: '$t\overline{t}H$',
                1: '$t\overline{t}W$',
                2: '$t\overline{t}Z$'}

    for i, feature in enumerate(f):
        ml_utils.plot_histograms_multiclass(X[:, i], y, classes,
            feature, saving=True,
            save_folder='data/older
                /histograms_mc16a
                /')
        ml_utils.plot_histograms_multiclass(X_ch[:, i], y_ch,
            classes, feature, saving=True,
            save_folder='data/older
                /
                histograms_ch_mc16a
                /')

    """ real/mc16a&mc16d, 83 features """
    X = ml_utils.load_compress('splits/real/X_mc16a_mc16d.pkl.gzip
        ', 'gzip')
    y = np.zeros(X.shape[0])
    X_ch = ml_utils.load_compress('splits/real/X_ch_mc16a_mc16d.pkl
        .gzip', 'gzip')
    y_ch = np.zeros(X_ch.shape[0])
    f = ml_utils.load('splits/real/f_mc16a_mc16d.pkl')

    classes = {0: 'unknown'}

    # in case of 'ValueError: array must not contain infs or NaNs'
    # navigate to the 'scipy\lib\util.py', line 239
    # and change it to 'a = np.nan_to_num(a)'

```

```

# this is probably caused by kernel estimation function
# resulting in nan matrix, even when data have real values
for i, feature in enumerate(f):
    ml_utils.plot_histograms_multiclass(X[:, i], y, classes,
                                        feature, saving=True,
                                        save_folder='data/real/
                                        histograms_mc16a_mc16d
                                        /')
    ml_utils.plot_histograms_multiclass(X_ch[:, i], y_ch,
                                        classes, feature, saving=True,
                                        save_folder='data/real/
                                        histograms_ch_mc16a_mc16d
                                        /')

if __name__ == '__main__':
    """ Script demonstrating how to get information about final data.

    In this script several files are opened and information is printed
    about them. Code sections can be uncommented to
    obtain list of features or histograms.
    """

    main()

```

■ D.2.6 tuning scripts (only RFC listed)

```

"""
Last modified May 9 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import make_scorer

import numpy as np

import datetime
import sys

# If use non-binary classification (default 1)
MULTICLASS = 1
# Use channel data
CHANNEL = 1

# Number of cores for computations
N_JOBS = 8

# Additional counter in case of multiple files for one day
counter = 1

# 2020-03-24
# min_samples_split = range(2, 103, 5)
# min_samples_split = range(107, 203, 5)
# # min_samples_split = range(2, 203, 5)

# 2020-03-25
# n_estimators = range(50, 151, 10)

```

```

# n_estimators = range(160, 201, 10)
# n_estimators = range(220, 401, 20)
# n_estimators = range(420, 601, 20)
# # n_estimators = range(50, 601, 10)

# 2020-03-26
# max_depth = range(10, 101, 5)

# 2020-03-27
# max_features = np.linspace(0.05, 1, 20)

# 2020-03-28
# max_leaf_nodes = range(2, 103, 5)
# max_leaf_nodes = range(107, 203, 5)
# max_leaf_nodes = range(207, 303, 5)
# max_leaf_nodes = range(307, 403, 5)
# max_leaf_nodes = range(407, 503, 5)
# max_leaf_nodes = range(507, 603, 5)
# # max_leaf_nodes = range(2, 603, 5)

# 2020-03-29
# min_impurity_decrease = np.linspace(0, 0.1, 21)

# 2020-03-30
# ccp_alpha = np.linspace(0, 0.1, 21)

# 2020-03-31
# max_samples = np.linspace(0.05, 1, 20)

# 2020-04-01
# oob_score = [False, True]

# 2020-04-02
criterion = ['gini', 'entropy']

def main():
    if not CHANNEL:
        X = ml_utils.load_compress('splits/older/X_mc16a.pkl.gz', '
            gzip')
        y = ml_utils.load('splits/older/y_mc16a.pkl')
    else:
        X = ml_utils.load_compress('splits/older/X_ch_mc16a.pkl.gz', '
            gzip')
        y = ml_utils.load('splits/older/y_ch_mc16a.pkl')

    # Convert to single background problem
    if not MULTICLASS:
        y = [1.0 if e1 == 2.0 else e1 for e1 in y]
        y = np.array(y)

        # Class weights
        # [ATL-COM-PHYS-2018-410]
        w_ttH = 5.5650
        w_ttW = 4.9338 + 3.9677
        w_sum = w_ttH + w_ttW

        class_weights = {0: w_ttH / w_sum,
            1: w_ttW / w_sum}

        parameters = {'class_weight': class_weights}
    else:
        # Class weights
        # [ATL-COM-PHYS-2018-410]
        w_ttH = 5.5650
        w_ttW = 4.9338

```



```

w_ttZ = 3.9677
w_sum = w_ttH + w_ttW + w_ttZ

class_weights = {0: w_ttH / w_sum,
                  1: w_ttW / w_sum,
                  2: w_ttZ / w_sum}

parameters = {'class_weight': class_weights}

sc = StandardScaler()
clf = RandomForestClassifier(**parameters)

pipe = Pipeline(steps=[
    ('sc', sc),
    ('clf', clf)])

grid_dict = { # 'clf__min_samples_split': min_samples_split,
              # 'clf__n_estimators': n_estimators,
              # 'clf__max_depth': max_depth,
              # 'clf__max_features': max_features,
              # 'clf__max_leaf_nodes': max_leaf_nodes,
              # 'clf__min_impurity_decrease': min_impurity_decrease,
              # 'clf__ccp_alpha': ccp_alpha,
              # 'clf__max_samples': max_samples,
              # 'clf__oob_score': oob_score,
              'clf__criterion': criterion
            }

sc_dict = {'Accuracy': make_scorer(ml_utils.multiclass_accuracy),
           'F1 Weighted': make_scorer(ml_utils.multiclass_f1,
                                       average='weighted'),
           'ROC AUC Weighted': make_scorer(ml_utils.
                                           multiclass_roc_auc_score, average='weighted',
                                           multi_class='ovr')}

dt = datetime.datetime.now()
dt = dt.date()
if not CHANNEL:
    dt = str(dt) + '-' + str(counter)
else:
    dt = str(dt) + '-' + str(counter) + '-ch'

old_stdout = sys.stdout
log_file = open('logs/tuning_rfc_{}.log'.format(dt), 'w')
sys.stdout = log_file
print('Output: {}')

model = GridSearchCV(estimator=pipe, param_grid=grid_dict, scoring=
    sc_dict, refit='ROC AUC Weighted',
                    return_train_score=True, verbose=10, n_jobs=
                        N_JOBS)

model = model.fit(X, y)

print('}')
print('Best score: {} + str(model.best_score_) + '}')
print('Best params: {}'.format(model.best_params_))
print('Results: {}'.format(model.cv_results_))

ml_utils.save(model.cv_results_, 'tuning', 'tuning_rfc_{}'.format(
    dt))

sys.stdout = old_stdout
log_file.close()

```

```

if __name__ == '__main__':
    """ Script demonstrating RFC tuning process.

    In this script RFC parameters are tuned. Also three metrics to
    measure performance are introduced. Uncommenting
    desired parameter will make it being tuned. Only one can be tuned
    at the time. Results are store in log and tuning
    folders.
    """

    main()

```

■ D.2.7 tuning results view scripts (only RFC listed)

```

"""
Last modified May 11 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

from sklearn.metrics import make_scorer
import matplotlib.pyplot as plt
import pandas as pd
from tabulate import tabulate

# Use channel results
CHANNEL = 1

def main():
    # Scorers
    scorer_dict = {'Accuracy': make_scorer(ml_utils.multiclass_accuracy
    ),
                  'F1 Weighted': make_scorer(ml_utils.multiclass_f1,
                  average='weighted'),
                  'ROC AUC Weighted': make_scorer(ml_utils.
                  multiclass_roc_auc_score, average='weighted',
                  multi_class='ovr')}

    """ min_samples_split """
    # parameter = 'min_samples_split'
    # if not CHANNEL:
    #     results = [ml_utils.load('tuning/tuning_rfc_2020-03-24-1.pkl
    #     '), ml_utils.load('tuning/tuning_rfc_2020-03-24-2.pkl')]
    #     score_y_min = 0.5
    #     time_y_max = 1000
    # else:
    #     results = [ml_utils.load('tuning/tuning_rfc_2020-03-24-1-ch.
    #     pkl')]
    #     score_y_min = 0
    #     time_y_max = 10
    #
    # # graph boundaries
    # score_x_min = 0
    # score_x_max = 204

    """ n_estimators """
    # parameter = 'n_estimators'
    # if not CHANNEL:

```

```

#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-25-1.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-25-2.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-25-3.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-25-4.pkl
#         ')]
#     score_y_min = 0.5
#     time_y_max = 5000
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-25-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 15
#
# score_x_min = 45
# score_x_max = 605

""" max_depth """
# parameter = 'max_depth'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-26-1.pkl
#         ')]
#     score_y_min = 0.5
#     time_y_max = 1000
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-26-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 10
#
# score_x_min = 8
# score_x_max = 102

""" max_features """
# parameter = 'max_features'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-27-1.pkl
#         ')]
#     score_y_min = 0.5
#     time_y_max = 7000
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-27-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 15
#
# score_x_min = 0.03
# score_x_max = 1.02

""" max_leaf_nodes """
# parameter = 'max_leaf_nodes'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-28-1.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-28-2.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-28-3.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-28-4.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-28-5.pkl
#         '),
#                 ml_utils.load('tuning/tuning_rfc_2020-03-28-6.pkl
#         ')]

```

```

#     score_y_min = 0.5
#     time_y_max = 1000
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-28-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 10
#
# score_x_min = 0
# score_x_max = 604

""" min_impurity_decrease """
# parameter = 'min_impurity_decrease'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-29-1.pkl
#         '')]
#     score_y_min = 0.5
#     time_y_max = 800
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-29-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 10
#
# score_x_min = -0.002
# score_x_max = 0.102

""" ccp_alpha """
# parameter = 'ccp_alpha'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-30-1.pkl
#         '')]
#     score_y_min = 0.3
#     time_y_max = 20000
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-30-1-ch.
#         pkl'')]
#     score_y_min = 0
#     time_y_max = 10
#
# score_x_min = -0.002
# score_x_max = 0.102

""" max_samples """
# parameter = 'max_samples'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-31-1.pkl
#         '')]
#     score_y_min = 0.5
#     time_y_max = 1500
# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-03-31-1-ch.
#         pkl'')]
#     score_y_min = 0.4
#     time_y_max = 10
#
# score_x_min = 0.03
# score_x_max = 0.97

""" oob_score """
# parameter = 'oob_score'
# if not CHANNEL:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-04-01-1.pkl
#         '')]
#     score_y_min = 0.5
#     time_y_max = 1500

```

```

# else:
#     results = [ml_utils.load('tuning/tuning_rfc_2020-04-01-1-ch.
#                             pkl')]
#     score_y_min = 0.4
#     time_y_max = 10
#
# score_x_min = -0.02
# score_x_max = 1.02

""" criterion """
parameter = 'criterion'
if not CHANNEL:
    results = [ml_utils.load('tuning/tuning_rfc_2020-04-02-1.pkl')]
    score_y_min = 0.5
    time_y_max = 3000
else:
    results = [ml_utils.load('tuning/tuning_rfc_2020-04-02-1-ch.pkl
                              ')]
    score_y_min = 0.4
    time_y_max = 10

score_x_min = -0.02
score_x_max = 1.02

# Prepare data in Pandas
df = None
for result in results:
    if df is None:
        df = pd.DataFrame(result)
    else:
        df = pd.concat([df, pd.DataFrame(result)], ignore_index=
                        True)

# Print pretty df table
# print(tabulate(df, headers='keys', tablefmt='psql'))

# Substitute values for strings
if parameter == 'criterion':
    df = df.replace('gini', 0)
    df = df.replace('entropy', 1)

# Plot data
ml_utils.plot_search_results('RFC', df, scorer_dict, parameter,
                             score_x_min, score_x_max, score_y_min, time_y_max)
plt.show()

if __name__ == '__main__':
    """ Script demonstrating processing of RFC tuning results.

    In this script results from tuning are loaded and plotted. Both
    score and time are observed. Results for parameter
    can be obtained by uncommenting given section.
    """

    main()

```

■ D.2.8 train scripts (only RFC listed)

```

"""
Last modified May 18 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

import matplotlib.pyplot as plt

# Use new features (77) and newer data
# NEW = False
NEW = True

# Train new classifier
TRAIN = 0
# Use default parameters
DEFAULT = 0
# Use channel
CHANNEL = 0
# Test channel-data on full-data trained model (channel needs to be 1)
CROSSEVAL = 0
# Test on older mc16d data
OLDER_MC16D = 0
# Test on newer mc16a and mc16d data
NEWER = 0

# Plot features importance
FEAT = 1

# Number of cores available for training
N_JOBS = 8

def main():
    if TRAIN:
        if not CHANNEL:
            if NEW:
                X_train = ml_utils.load_compress('splits/newer/
                    X_new_train_80_mc16a_mc16d.pkl.gz', 'gzip')
                y_train = ml_utils.load('splits/newer/
                    y_new_train_80_mc16a_mc16d.pkl')
            else:
                X_train = ml_utils.load_compress('splits/older/
                    X_train_80_mc16a.pkl.gz', 'gzip')
                y_train = ml_utils.load('splits/older/y_train_80_mc16a.
                    pkl')
        else:
            if NEW:
                X_train = ml_utils.load_compress('splits/newer/
                    X_train_80_ch_mc16a_mc16d.pkl.gz', 'gzip')
                y_train = ml_utils.load('splits/newer/
                    y_train_80_ch_mc16a_mc16d.pkl')
            else:
                X_train = ml_utils.load_compress('splits/older/
                    X_train_80_ch_mc16a.pkl.gz', 'gzip')
                y_train = ml_utils.load('splits/older/
                    y_train_80_ch_mc16a.pkl')

    # Class weights
    # [ATL-COM-PHYS-2018-410]
    w_ttH = 5.5650
    w_ttW = 4.9338
    w_ttZ = 3.9677

```

```

w_sum = w_ttH + w_ttW + w_ttZ

class_weights = {0: w_ttH / w_sum,
                  1: w_ttW / w_sum,
                  2: w_ttZ / w_sum}

if DEFAULT:
    parameters = {'class_weight': class_weights,
                  'n_jobs': N_JOBS}
elif not CHANNEL:
    parameters = {'n_estimators': 540,
                  'max_depth': 15,
                  'min_samples_split': 117,
                  'max_features': 0.3,
                  'max_leaf_nodes': 522,
                  'min_impurity_decrease': 0.0,
                  'ccp_alpha': 0.0,
                  'max_samples': 0.85,
                  'oob_score': True,
                  'criterion': 'gini',
                  'class_weight': class_weights,
                  'n_jobs': N_JOBS}
else:
    parameters = {'n_estimators': 50,
                  'max_depth': 10,
                  'min_samples_split': 122,
                  'max_features': 0.55,
                  'max_leaf_nodes': 47,
                  'min_impurity_decrease': 0.0,
                  'ccp_alpha': 0.0,
                  'max_samples': 0.85,
                  'oob_score': True,
                  'criterion': 'gini',
                  'class_weight': class_weights,
                  'n_jobs': N_JOBS}

model = ml_utils.train_model(X_train, y_train, 'RFC',
                             parameters)

if not CHANNEL:
    if DEFAULT:
        ml_utils.save(model, 'models', 'RFC_default%s' % ('_new'
                                                           ' if NEW else ''))
    else:
        ml_utils.save(model, 'models', 'RFC_tuned%s' % ('_new'
                                                         ' if NEW else ''))
else:
    if DEFAULT:
        ml_utils.save(model, 'models', 'RFC_default_ch%s' % ('_new'
                                                              ' if NEW else ''))
    else:
        ml_utils.save(model, 'models', 'RFC_tuned_ch%s' % ('_new'
                                                            ' if NEW else ''))

else:
    if not CHANNEL:
        if OLDER_MC16D:
            X = ml_utils.load_compress('splits/older/X_mc16d.pkl.
                                       gzip', 'gzip')
            y = ml_utils.load('splits/older/y_mc16d.pkl')
        elif NEWER:
            X = ml_utils.load_compress('splits/newer/X_mc16a_mc16d.
                                       pkl.gzip', 'gzip')
            y = ml_utils.load('splits/newer/y_mc16a_mc16d.pkl')
        else:
            if NEW:

```

```

        X_train = ml_utils.load_compress('splits/newer/
            X_new_train_80_mc16a_mc16d.pkl.gz', 'gzip')
        y_train = ml_utils.load('splits/newer/
            y_new_train_80_mc16a_mc16d.pkl')
        X_test = ml_utils.load_compress('splits/newer/
            X_new_test_20_mc16a_mc16d.pkl.gz', 'gzip')
        y_test = ml_utils.load('splits/newer/
            y_new_test_20_mc16a_mc16d.pkl')
    else:
        X_train = ml_utils.load_compress('splits/older/
            X_train_80_mc16a.pkl.gz', 'gzip')
        y_train = ml_utils.load('splits/older/
            y_train_80_mc16a.pkl')
        X_test = ml_utils.load_compress('splits/older/
            X_test_20_mc16a.pkl.gz', 'gzip')
        y_test = ml_utils.load('splits/older/
            y_test_20_mc16a.pkl')

    if DEFAULT:
        model = ml_utils.load('models/RFC_default%s.pkl' % (
            '_new' if NEW else ''))
    else:
        model = ml_utils.load('models/RFC_tuned%s.pkl' % ('_new'
            if NEW else ''))

else:
    if OLDER_MC16D:
        X = ml_utils.load_compress('splits/older/X_ch_mc16d.pkl
            .gz', 'gzip')
        y = ml_utils.load('splits/older/y_ch_mc16d.pkl')
    elif NEWER:
        X = ml_utils.load_compress('splits/newer/
            X_ch_mc16a_mc16d.pkl.gz', 'gzip')
        y = ml_utils.load('splits/newer/y_ch_mc16a_mc16d.pkl')
    else:
        if NEW:
            X_train = ml_utils.load_compress('splits/newer/
                X_new_train_80_ch_mc16a_mc16d.pkl.gz', 'gzip'
            )
            y_train = ml_utils.load('splits/newer/
                y_new_train_80_ch_mc16a_mc16d.pkl')
            X_test = ml_utils.load_compress('splits/newer/
                X_new_test_20_ch_mc16a_mc16d.pkl.gz', 'gzip')
            y_test = ml_utils.load('splits/newer/
                y_new_test_20_ch_mc16a_mc16d.pkl')
        else:
            X_train = ml_utils.load_compress('splits/older/
                X_train_80_ch_mc16a.pkl.gz', 'gzip')
            y_train = ml_utils.load('splits/older/
                y_train_80_ch_mc16a.pkl')
            X_test = ml_utils.load_compress('splits/older/
                X_test_20_ch_mc16a.pkl.gz', 'gzip')
            y_test = ml_utils.load('splits/older/
                y_test_20_ch_mc16a.pkl')

    if DEFAULT:
        model = ml_utils.load('models/RFC_default_ch%s.pkl' %
            ('_new' if NEW else ''))
    elif CROSSEVAL:
        model = ml_utils.load('models/RFC_tuned%s.pkl' % ('_new'
            if NEW else ''))
    else:
        model = ml_utils.load('models/RFC_tuned_ch%s.pkl' % (
            '_new' if NEW else ''))

classes = {0: 'ttH',

```



```

        1: 'ttW',
        2: 'ttZ'}

s = str
if DEFAULT:
    if not CHANNEL:
        s = 'default'
    else:
        s = 'default_ch'
else:
    if not CHANNEL or CROSSEVAL:
        s = 'tuned'
    else:
        s = 'tuned_ch'

if OLDER_MC16D or NEWER:
    y_pred = ml_utils.predict(model, X)
    y_pred_proba = ml_utils.predict_proba(model, X)

    acc = ml_utils.multiclass_accuracy(y, y_pred)
    f1 = ml_utils.multiclass_f1(y, y_pred, average='weighted')
    auc = ml_utils.multiclass_roc_auc_score(y, y_pred, average
        ='weighted', multi_class='ovr')

    ml_utils.plot_roc_multiclass(
        'RFC_%s test %s' % (s, '(100% of newer mc16a & mc16d
            set)' if NEWER else '(100% of older mc16d set)'),
        y,
        y_pred_proba, classes,
        {'Accuracy': acc, 'F1 Weighted': f1,
         'ROC AUC Weighted': auc})
else:
    y_train_pred = ml_utils.predict(model, X_train)
    y_train_pred_proba = ml_utils.predict_proba(model, X_train)
    y_test_pred = ml_utils.predict(model, X_test)
    y_test_pred_proba = ml_utils.predict_proba(model, X_test)

    acc_train = ml_utils.multiclass_accuracy(y_train,
        y_train_pred)
    f1_train = ml_utils.multiclass_f1(y_train, y_train_pred,
        average='weighted')
    auc_train = ml_utils.multiclass_roc_auc_score(y_train,
        y_train_pred, average='weighted', multi_class='ovr')
    acc_test = ml_utils.multiclass_accuracy(y_test, y_test_pred
    )
    f1_test = ml_utils.multiclass_f1(y_test, y_test_pred,
        average='weighted')
    auc_test = ml_utils.multiclass_roc_auc_score(y_test,
        y_test_pred, average='weighted', multi_class='ovr')

    ml_utils.plot_roc_multiclass(
        'RFC_%s train (80%% of %s)' % (s, 'newer mc16a & mc16d
            set' if NEW else 'older mc16a set'), y_train,
        y_train_pred_proba, classes,
        {'Accuracy': acc_train, 'F1 Weighted': f1_train,
         'ROC AUC Weighted': auc_train})
    ml_utils.plot_roc_multiclass(
        'RFC_%s test (20%% of %s)' % (s, 'newer mc16a & mc16d
            set' if NEW else 'older mc16a set'), y_test,
        y_test_pred_proba,
        classes,
        {'Accuracy': acc_test, 'F1 Weighted': f1_test, 'ROC AUC
            Weighted': auc_test})

if FEAT:
    if NEW:

```

```

        f = ml_utils.load('splits/newer/f_mc16a_mc16d.pkl')
    else:
        f = ml_utils.load('splits/older/f_mc16a.pkl')

    ml_utils.plot_feat_imp('RFC_%s feature importances' % s,
                           model, f, 15)

plt.show()

if __name__ == '__main__':
    """ Script demonstrating training and test process for RFC.

    In this script RFC is trained/tested for default/tuned parameters.
    In total four variables can be manipulated to
    achieve results for tran/test, default/tuned, full-data/channel-
    data, and normal/crosseval.

    Also test on January and March data can be obtained.
    """

    main()

```

■ D.2.9 tuning_wp.py

```

"""
Last modified May 19 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

import matplotlib.pyplot as plt

# Use new features (77) and newer data traine models
# NEW = False
NEW = True

# Choose CERN approach (significance) maximization or ML approach (
# sensitivity maximization)
CERN = 1
# CERN = 0

# For which classifier should be WP tuned
# CLASS = 'RFC'
# CLASS = 'KNC'
# CLASS = 'GNB'
# CLASS = 'ADA'
# CLASS = 'GBC'
CLASS = 'MLPC'
# CLASS = 'SVC'

def main():
    if NEW:
        X_test = ml_utils.load_compress('splits/newer/
            X_new_test_20_mc16a_mc16d.pkl.gz', 'gzip')
        y_test = ml_utils.load('splits/newer/y_new_test_20_mc16a_mc16d.
            pkl')
        model = ml_utils.load('models/{}_tuned_new.pkl'.format(CLASS))
    else:
        X_test = ml_utils.load_compress('splits/newer/X_mc16a_mc16d.pkl
            .gzip', 'gzip')

```



```

        ['green', 'sienna'], ['S', 'B'],
        best_tr)
    ml_utils.plot_threshold(x_values, [y_signif, y_signif_simp], ['
        max', 'max'],
        'Significance Approximations to
        Threshold characteristics',
        'Significance Approximation [-]', ['
        darkred', 'r'], ['S/sqrt(S+B)', 'S/
        sqrt(B)'])
    print('Best Threshold: {}'.format(best_tr))

    ml_utils.plot_cm(y_pred, y_test, 3, classes, 'CM - test (
        default threshold)')
    ml_utils.plot_cm(ml_utils.threshold_probas_multiclass(y_probas,
        best_tr), y_test, 3, classes,
        'CM - test ({} threshold)'.format(best_tr))
    plt.show()

else:
    x_values, y_fdr, y_sen, best_tr = ml_utils.
        threshold_characteristics_multiclass_sensitivity(y_probas,
        y_test)

    ml_utils.plot_threshold(x_values, [y_fdr, y_sen], ['min', 'max
        '],
        'FDR & Sensitivity to Threshold
        characteristics',
        '[-]', ['darkblue', 'orange'],
        ['FDR (1 - precision)', 'Sensitivity (
        recall)'])
    print('Best Threshold: {}'.format(best_tr))

    ml_utils.plot_cm(ml_utils.threshold_probas_multiclass(y_probas,
        best_tr), y_test, 3, classes,
        'CM - test ({} threshold)'.format(best_tr))
    plt.show()

if __name__ == '__main__':
    """ Script demonstrating how to tune WP.

        In this script two WP tuning approaches are listed.
        """

    main()

```

■ D.2.10 eval_real.py

```

"""
Last modified May 20 2020
@author: Jakub Maly
"""

import utils.ml_utils as ml_utils

import matplotlib.pyplot as plt

import numpy as np

# Use new features (77) and newer data trained models
# NEW = False
NEW = True

```

```
# Save predictions for future
SAVE = False
# SAVE = True

# Load saved predictions for instant plotting
# LOAD = False
LOAD = True

def main():
    if NEW:
        X_test = ml_utils.load_compress('splits/real/X_new_mc16a_mc16d.
           .pkl.gzip', 'gzip')
    else:
        X_test = ml_utils.load_compress('splits/real/X_mc16a_mc16d.pkl.
            gzip', 'gzip')

    # Windows int64 not supported
    X_test = np.array(X_test, np.int32)

    classifiers = ['RFC', 'KNC', 'GNB', 'ADA', 'GBC', 'MLPC', 'SVC']

    thresholds = {'RFC': 0.26,
                  'KNC': 0.08,
                  'GNB': 0,
                  'ADA': 0.33,
                  'GBC': 0.03,
                  'MLPC': 0.16,
                  'SVC': 0.14}

    classes = {0: 't$\overline{t}$H',
               1: 't$\overline{t}$W',
               2: 't$\overline{t}$Z'}

    colors = [(0.12156862745098039, 0.4666666666666667,
               0.7058823529411765),
              (1.0, 0.4980392156862745, 0.054901960784313725),
              (0.17254901960784313, 0.6274509803921569,
               0.17254901960784313)]

    y_preds = list()
    for classifier in classifiers:
        if LOAD:
            y_pred = ml_utils.load(
                'predictions/%s_pred_real_mc16a_mc16d.pkl' % (
                    classifier + '_y_new' if NEW else classifier + '_y'
                ))
        else:
            model = ml_utils.load('models/%s_tuned%s.pkl' % (classifier
                , '_new' if NEW else ''))
            y_pred = ml_utils.threshold_probab_multiclass(ml_utils.
                predict_proba(model, X_test), thresholds[classifier])
            if SAVE:
                ml_utils.save(y_pred, 'predictions',
                    '%s_pred_real_mc16a_mc16d' % (classifier
                        + '_y_new' if NEW else classifier + '_y'))

            y_preds.append(y_pred)

    ml_utils.plot_pred_multiclass('1D heat maps of classifiers
        predictions', y_preds, classifiers, classes, colors)
    ml_utils.plot_pred_hist_multiclass(y_preds, classifiers, classes,
        colors)

    plt.show()
```

```
if __name__ == '__main__':  
    """ Script demonstrating real data evaluation.  
  
    In this script all previously tuned classifiers are tested on real  
        data.  
    """  
  
    main()
```


Appendix E

Bibliography

- [Amo13] J. Amos, *Higgs: Five decades of noble endeavour bbc news science and environment*.
- [BBC12] BBC, *Higgs boson-like particle discovery claimed at lhc*.
- [Buc07] A. Buckley, *The problem with root (a.k.a. the root of all evil)*, InsectNation (2007).
- [Byl16] O. B. Bylund, *Measurement of ttz and ttw production at atlas in 13 tev data, using trilepton and same charge dimuon final states*, ATL-PHYS-PROC-2016-117 (2016).
- [CERa] CERN, *Calorimetry*, ATLAS Technical Proposal.
- [CERb] ———, *Inner detector*, ATLAS Technical Proposal.
- [CERc] ———, *Magnet system*, ATLAS Technical Proposal.
- [CERd] ———, *Overall detector concept*, ATLAS Technical Proposal.
- [CER08] ———, *Cern particle detector: Atlas completes world's largest jigsaw puzzle*, ScienceDaily (2008).
- [CER12] ———, *Lhc to run at 4 tev per beam in 2012*, Media and Press Relations (Press release) (2012).
- [Cha06] O. Chapelle, *Training a support vector machine in the primal*.
- [cita] *Aerial view of lhc*, <https://www.pinterest.com/pin/508414245409340997/>.
- [citb] *Atlas detector*, <https://atlas.cern/discover/detector>.

- [citc] *Atlas inner detector*, <https://atlas.cern/discover/detector/inner-detector>.
- [citd] *Higgs boson production with a pair of top quarks*, http://gkantoni.github.io/feynman/auto_examples/Particle_Physics/plot_ttH.html.
- [Col18a] The ATLAS Collaboration, *Observation of higgs boson production in association with a top quark pair at the lhc with the atlas detector*.
- [Col18b] ———, *Observation of higgs boson production in association with a top quark pair at the lhc with the atlas detector*, Physics Letters B **784** (2018), 173–191.
- [Col19a] ———, *Analysis of tth and ttw production in multilepton final states with the atlas detector*, ATLAS-CONF-2019-045 (2019).
- [Col19b] ———, *Search for the associated production of a higgs boson and a top quark pair in multilepton final states in 80 fb⁻¹ pp collisions at $\sqrt{s} = 13$ TeV with the atlas detector*, ATL-COM-PHYS-2018-410 (2019).
- [Cor10] Oracle Corporation, *Oracle grid engine*.
- [CT] H. H. Hoos K. Leyton-Brown C. Thornton, F. Hutter, *Autoweika: Combined selection and hyperparameter optimization of classification algorithms*.
- [ERN12] ERN, *Latest results from atlas higgs search*, ATLAS News (2012).
- [Joy] J. Joyce, *Bayes' theorem*, The Stanford Encyclopedia of Philosophy.
- [Kor08] A. Korytov, *Introduction to elementary particle physics*.
- [M⁺] Wes McKinney et al., *Data structures for statistical computing in python*, Proceedings of the 9th Python in Science Conference.
- [MF] K. Eggenberger J. Springenberg M. Blum F. Hutter M. Feurer, A. Klein, *Efficient and robust automated machine learning*, Advances in Neural Information Processing Systems 28 (NIPS 2015).
- [Nav18a] A. Navlani, *Adaboost classifier in python*, Datacamp (2018).
- [Nav18b] Avinash Navlani, *Knn classification using scikit-learn*, Datacamp (2018).
- [Oli] Travis E Oliphant, *A guide to numpy*.
- [Paz] L. Paz, *Official compresspickle repository*.

- [PB04] D. Whiteson P. Baldi, P. Sadowski, *Searching for exotic particles in high-energy physics with deep learning*, Annu. Rev. Nucl. Part. Sci. 2018 **68** (2004), 1–22.
- [PB14] ———, *Searching for exotic particles in high-energy physics with deep learning*.
- [Piv] J. Pivarski, *Official uproot repository*.
- [Pos] P. Posik, *Artificial intelligence, decision tasks, learning*, CTU in Prague, Faculty of Electrical Engineering, Dept. of Cybernetics.
- [PVG⁺] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research.
- [QR19] Shuai Han Qiubing Ren, Mingchao Li, *Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives*.
- [Roo19] M. Roos, *Deep learning neurons versus biological neurons*, Towards Data Science (2019).
- [Sap19] G. Saporito, *How does a neural network make predictions?*, Towards Data Science (2019).
- [Sca10] D.A. Scannicchio, *Atlas trigger and data acquisition: Capabilities and commissioning*, Nuclear Instruments and Methods in Physics Research Section A **617** (2010), 306–309.
- [Str12] M. Strassler, *The higgs faq 2.0*, ProfMattStrassler.com (2012).
- [Tay12] L. Taylor, *Observation of a new particle with a mass of 125 gev*, CMS Public Website (2012).
- [TUoEA12] School of Physics The University of Edinburgh and Astronomy, *Peter higgs: Curriculum vitae*.
- [Was19] M. Waseem, *A quick guide to learn support vector machine in python*, Edureka (2019).
- [Web15] J. Webb, *Lhc smashes energy record with test collisions*.
- [Yiu19] T. Yiu, *Understanding random forest*, Towards Data Science (2019).