

Finding the Right Primary Vertex Using Machine Learning

Kieran McDowall

Supervisor: Gregory Ciezarek

24/08/2022

Abstract

We use a binary classification neural network to find the primary vertex relating to a bottom quark decay using information on the number of tracks measured and certain input variables. Different machine learning packages were explored and models were built using XGboost, Keras and PyTorch. A final model achieved an accuracy of 78.18% on LHCb run 2 data.

1 Introduction

1.1 Motivation

Measurements of decay time and missing momentum are crucial ingredients in measuring many key bottom and charm quark observables at LHCb. The measurement of these quantities relies on correctly identifying which reconstructed proton-proton collision was the true origin of the signal hadron. These reconstructed proton-proton collisions are called primary vertices (PVs). The 'right PV' is the one that produces a Bottom quark and we want to distinguish these PVs from PVs that don't produce a Bottom quark (random background PVs). In this project we are looking specifically at Bottom quarks with $B \rightarrow J/\psi K$ decay.

The current method used to find the right PV is the pointing method (lowest IPCHI2 method). The method works by reconstructing the momentum direction of the hadron. It then chooses the collision position which this points most closely back to (as illustrated fig. 1).

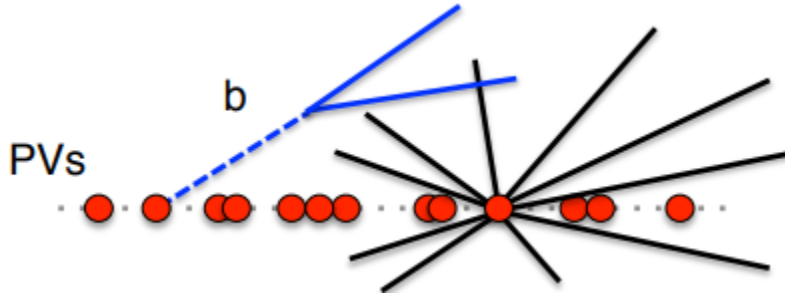


Figure 1: Illustration of the Pointing Method, where we can see a B quark being produced from a PV then undergoing decay. Diagram from M. Williams (2017), LHCb Upgrade II: Adding Precise Timing to the Vertex Locator [1].

This method currently works well and has low misassociation but as luminosity increases, for run 3 and beyond, the number of collisions per event increases. This results in the rate of misassociation (the error in finding the right PV) becoming more of a problem.

1.2 Number of Tracks per Primary Vertex

Instead in this project association is performed based on the properties of the other reconstructed particles produced in the collision using machine learning. PVs are made from tracks, and these tracks

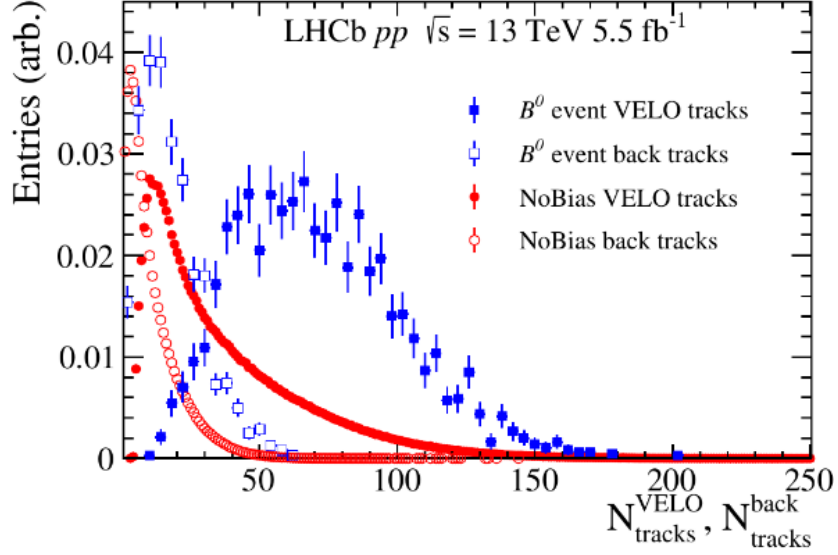


Figure 2: Distribution of the number of VELO tracks and back tracks for NoBias events (red) and B^0 signal events (blue) with a single primary vertex. The vertical scale is arbitrary. Diagram from the LHCb Collaboration [2].

collectively carry information about the physics processes in the collision. From fig. 2 we can see that the number of velo tracks for a B event PV is larger than the number of velo tracks for random background PVs. To have such a clearly defined difference there must be some different kinematic properties associated with the PVs. On average, gluon gluon collisions that produce a B event are harder, more energetic collisions and this gives us a higher number of tracks. These harder collisions also give us more co-linear track angles after the PVs. This is then where machine learning can be implemented, separating the PVs from the kinematic variables and also drawing power from the number of tracks.

2 Data

2.1 Pre-processing

In the pre-processing stage ROOT files are converted into a data type that can be read into a neural network. The python package Uproot was used to convert ROOT files into Awkward arrays and then Awkward arrays were converted into Pandas data frames so that data could be saved as a CSV file.

When reading in a ROOT file to a Pandas data frame, multi-index data frames are obtained. One of these indices relates to the event number and the other relates to the number of tracks in this event – this index being of variable length. We call data frames with multi-indices of variable length: jagged data, and dealing with jagged data was the biggest challenge in the pre-processing stage of the project.

Jagged data can be dealt with using a similar approach used in Inclusive Flavour Tagging [5]. In this approach the neural network is able to group the variable number of tracks to its particular event. Another method that can be used to deal with jagged data is to flatten and pad the data frame with a masking value (such that a 5 column data frame with a cut at 30 tracks per event will become a 150 column data frame. This is because each inputs tracks are 'flattened out' so that we obtain many inputs relating to just one track). Missing values in the data frame were then set to the value of -999 (the masking value) and the neural network is then able to infer information about the number of tracks per event from the occurrence of this value. This was the chosen method that was used to deal with jagged data as it was more straightforward to implement and gave good results.

2.2 Track Types

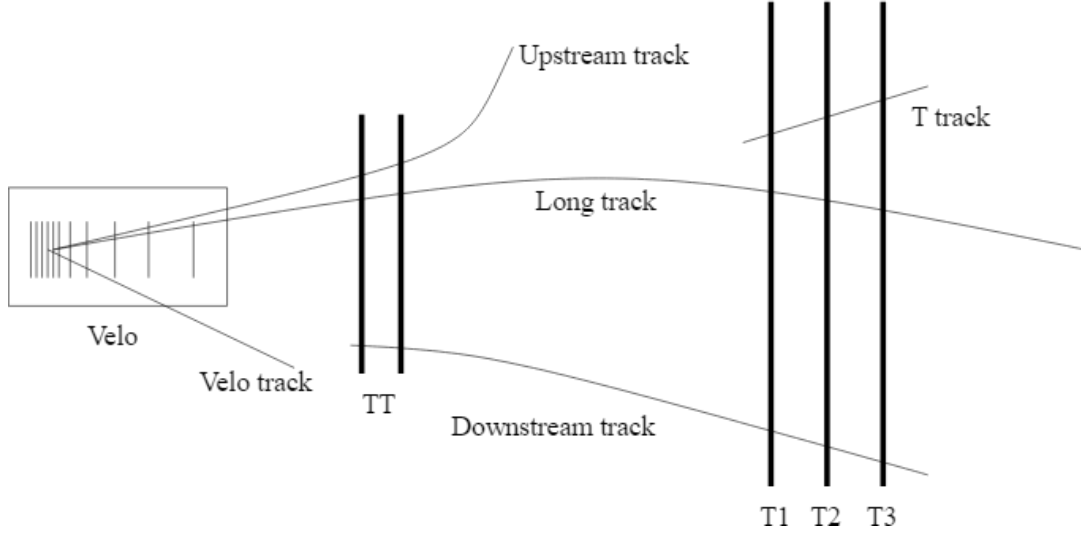


Figure 3: Different track types. In this project velo, upstream and long tracks were considered. Diagram from LHCbTrackingStrategies [6].

There are three different track types that can be input to the neural network, either separately or together. The first being the velo tracks which are reconstructed with just hits in the velo station, they see no magnetic field and therefore have no momentum estimate. The next track type are the upstream tracks which are reconstructed with hits in the velo and t stations (TT). They see some fringe magnetic field (before TT) and are therefore sensitive to bending which gives us a momentum estimate. The final track type are the long tracks which are reconstructed with hits in the velo and t stations (T123). They are sensitive to the bending in the magnetic field (between TT and T1) and therefore have a momentum estimate.

3 Machine Learning Algorithm

3.1 Binary Classification

The machine learning algorithm developed is a binary classification algorithm that categorises PVs producing a B decay: labelled PV = 1, and Random PVs: labelled PV = 0. Track labels are based on the PV that the tracks have closest IPCHI2 value to and we require that the lowest IPCHI2 is less than 4.

3.2 Simple Classifier

As an initial test a simple classifier using a XGboost based decision tree model was built for training and testing. The purpose of this initial test was to understand the level of separation for each input variable (how much can each input increase the models accuracy by). This was executed using only the first track in each event meaning that the classifier doesn't get any predicting power from the number of tracks per event, only power from the input variables relating to the kinematics of the B decay. The input variables that were used are shown in table 1.

Input Variables				
ΔETA	ΔPHI	PHI	ETA	PT

Table 1: Input variables given to the XGboost classifier. Where, ΔETA and ΔPHI are between the tracks and the B candidate. ETA, PHI and PT are purely the kinematic variables for the tracks.

An example of the separation seen by the neural network in its input variables is shown in fig. 5. The simple classifier achieved a maximum accuracy of 57 % using all of the input variables.

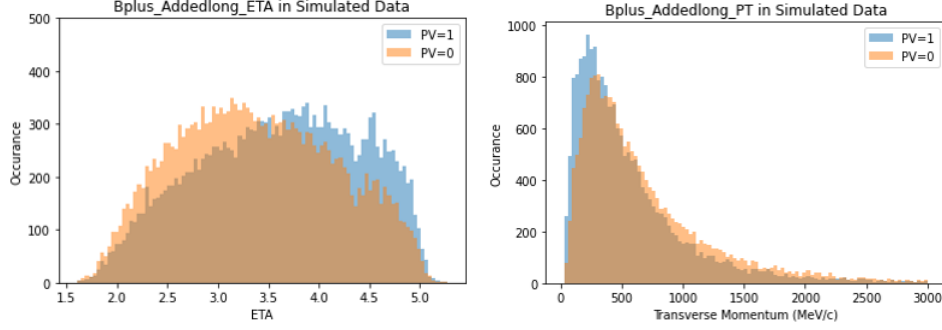


Figure 4: Two input variables given to the XGboost model. The angle ETA on the left and the transverse momentum on the right.

3.3 Neural Networks

To access the predicting power from the number of tracks per event a neural network (NN) was built with PyTorch. The model built was based off models made by N. Nolte and A. Verma [3] [4]. PyTorch was chosen because it is capable of training complex architecture, it is pretty stable when changing model hyperparameters and gives good accuracy results straight away with minimal tuning. It is also one of the only packages that is able to read in jagged data.

Despite the possibility to read in jagged data with PyTorch, the data was flattened and padded meaning other machine learning packages could have been used. A Keras sequential model was built and tested with the same data and returned very similar accuracy results to the PyTorch model. The PyTorch model was explored before the Keras model as it was expected that it would be easier to train. For more information on the specifics of the models see appendix B.

The input variables used for the final model are shown in table 2.

Input Variables							
ΔETA	ΔPHI	PHI	ETA	PT	Added PT	MINIPCHI2	Ghost Prob

Table 2: Input variables given to the neural networks.

3.3.1 Hyperparameter search

The hyperparameters of the final neural network model were to be determined by a hyperparameter search which finds the optimal model parameters for the problem. A full hyperparameter search for the final dataset was not completed due to the time constraints of the project but previous datasets were optimised for, the results of which determined the parameters of the final model. For the PyTorch model, package Ray Tune was used to search through the number of neurons, normalisation (between layers), activation function, lr, loss function, initializer and batch size. Some of the hyperparameters need to be chosen through trial and error due to the limitations of Ray Tune including: the number of layers, epochs and the optimiser.

A hyperparameter search was also built for the Keras model which is much more thorough in its search. The search is able to explore all of the desired model hyperparameters and test random combinations of them. The result of this search on previous datasets returned very similar hyperparameters as chosen for the PyTorch model with a couple of subtle differences. The Keras hyperparameter search results should at least be partially transferable to the PyTorch model.

4 Results

The PyTorch NN was trained and tested on two different data samples: a simulated data sample and a real sample from LHCb run 2 data. The NN achieved an accuracy of 73% for the simulated data using long tracks (and 69% using velo tracks). An accuracy of 76.92% was obtained for the real data using a mixture of velo, upstream and long tracks.

As well as the PyTorch model two other models were trained on the real dataset with mixed tracks. An optimised Keras model achieved an accuracy of 75.70% and a simple classifier using XGboost (this time with information about the number of tracks) scored an accuracy of 78.18% .

All models were trained using a data set with 440,000 events (or entries). To see what impact increasing the training data would have on the accuracy of the models, smaller data sets were constructed to extrapolate a trend in the accuracy increase. The results are shown in fig. 5.

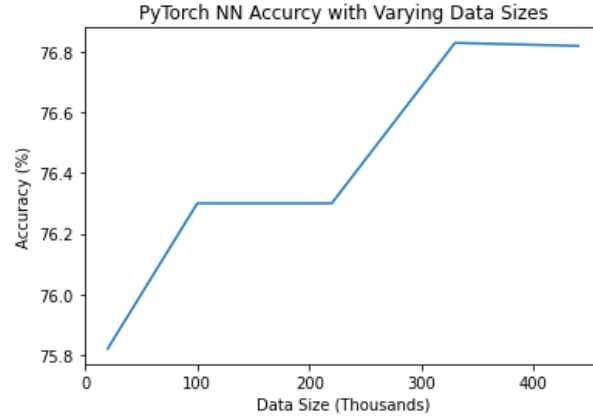


Figure 5: Accuracy scores using different data sizes.

5 Discussion

It is possible to see where the difference in accuracy between the simulated data and real data comes from by looking at the number of tracks per PV for each sample. In fig. 6, for the real data, we can see well defined separation in the number of tracks between the $PV = 0$ and $PV = 1$ labels. This closely matches the separation previously shown in fig 2. Whereas in fig. 7, for the simulated data, we see a messier picture where there is not clearly defined separation between the $PV = 0$ and $PV = 1$ labels.

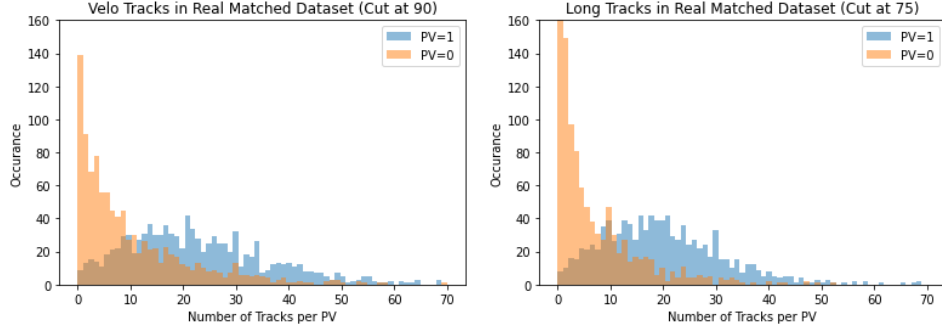


Figure 6: Real data showing the separation in the number of tracks per PV for velo tracks (left) and long tracks (right).

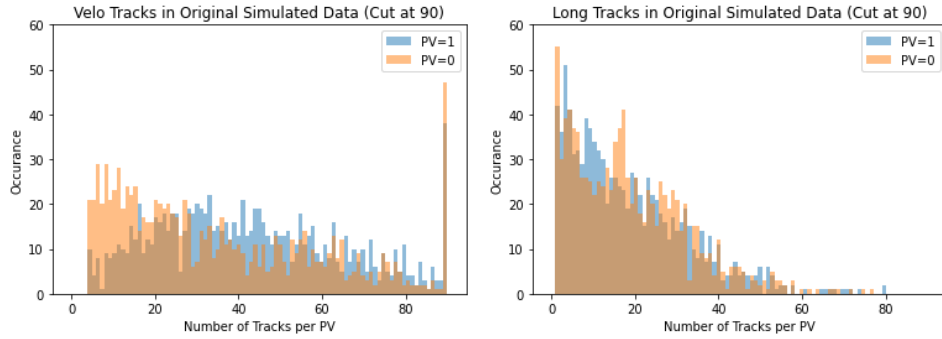


Figure 7: Simulated data showing the separation in the number of tracks per PV for velo tracks (left) and long tracks (right).

If the accuracy for the simulated data isn't coming from the number of tracks per PV it must then be coming from the input variables given to the NN. This confirms that there is definite power to be drawn from the input variables.

Comparing the Keras NN and XGboost BDT results to the PyTorch NN accuracy we can see that they are all similar but that the XGboost model gives the best accuracy. Although, this is without the PyTorch and Keras final models being optimised and after a hyperparameter search we could see both models accuracy scores increasing further. These model results still show that even a simple model can achieve a fairly high accuracy.

From fig. 5 we can see that the accuracy increase with varying data sizes levels off at data sizes of around 330,000. This demonstrates that using a larger data set for training and testing is unlikely to increase the NNs accuracy scores.

Future steps in the progress of the project could involve: a final hyperparameter optimisation (as previously mentioned). Combining the machine learning approach with the pointing method to see

how big the difference in decay times and missing momentum resolutions are. Seeing how the NN extends to run 3 conditions, whether it needs new training, and how big a difference that then makes.

6 Conclusion

We already knew B event PVs have a higher number of tracks compared to background PVs. It has been proved that neural networks can access definite separation power from this information.

It has also been confirmed that there is more power accessible to NNs from particular input variables relating to the kinematics of the B decay. A final model achieved an accuracy of 78.18% on real LHCb run 2 data using a simple classifier built with XGboost. This accuracy was higher than the accuracy for both Keras and PyTorch models although after hyperparameter optimisation these accuracy scores could increase. We also conclude that larger datasets are unlikely to greatly increase the accuracy of these models.

Using this machine learning approach to find the right primary vertex could cut down on the mis-association seen in the pointing method and this could prove important for run 3 and for future runs with higher luminosity.

A Data Selection

Data sample candidates taken from:

$BetaSBu2.JpsiK.DetachedLine, BplusMINIPCHI2 < 6, mu, KMINIPCHI2 > 6, BplusFDCHI2 > 50, BplusENDVERTX_CHI2 < 18, nPVs = 2, BplusMINIPCHI2NEXTBEST - MINIPCHI2 > 10, 5220 < M(JpsiK) < 5320$

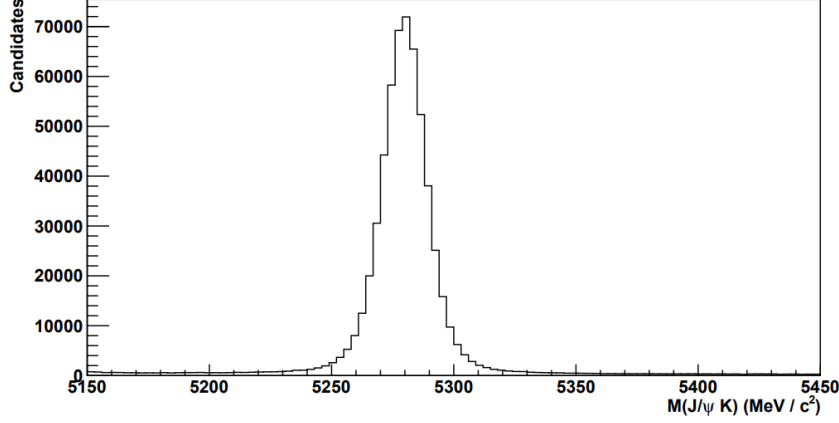


Figure 8:

B Model Architectures

B.1 PyTorch Model

The PyTorch model is a feed forward neural network.

- Configuration:
 - 4 hidden layers
 - 500 neurons in each layer: [500,500,500,500]
- Hyperparameters:
 - Normalisation: BatchNorm1d, Dropout: LazyBatchNorm1d, Activation function: ReLu, Learning rate: 1e-3, Loss function: Binary Crossentropy (BCE) with logits, Initializer: Linear, Epochs: 40, Batch size: 150,000, Optimiser: AdamW.

Run times (not optimised for): 10.5 minutes for 40 epochs and a prediction (prediction about 8s). On 16 Intel Core Processors (Broadwell, IBRS). Only slight performance gain from 20 epochs to 40 epochs so there is a possibility to half this run time. GPUs, instead of CPUS, could be used in future.

B.2 Keras Model

The Keras model is a sequential NN.

- Configuration:
 - 4 hidden layers
 - Neurons arranged as: [300,300,600,1200]
- Hyperparameters:
 - Activation function: Sigmoid, Final activation function: Sigmoid, Loss function: BCE, Epochs: 50, Batch size: 175,000, Optimiser: Adam.

Run times (not optimised for): 8 minutes for 50 epochs and a prediction. On 16 Intel Core Processors (Broadwell, IBRS). GPUs could be used in future.

References

- [1] M. Williams (2017), LHCb Upgrade II: Adding Precise Timing to the Vertex Locator.
- [2] LHCb Collaboration (2022), Evidence for modification of b quark hadronization in high-multiplicity pp collisions at $s = \sqrt{13}$ TeV. Phys. Rev. Lett.
- [3] N. Nolte (2022), GitLab, DeepSets for Flavour Tagging.
- [4] A. Verma (2020), Towards Data Science, PyTorch [Tabular] - Binary Classification.
- [5] D. O’Hanlon (2020), GitLab, InclusiveFlavourTagging .
- [6] P. Sevfert (2012), Twiki, LHCbTrackingStrategies.