Machine Learning Engineer Nanodegree
CAPSTONE PROJECT

Nora Petrova
August 18th, 2017

1. DEFINITION

Project Overview

The laws of nature in our universe impose that certain physical quantities, such as energy and momentum are conserved. Each conservation law is deeply linked to a fundamental symmetry, as shown by Emmy Noether in her theorem [1]. Symmetries in nature are deeply interwoven in the structure of the Standard Model (SM) of particle physics.

The Standard Model (SM) of physics is our most successful theoretical framework to date. It describes the strong, weak and electromagnetic interactions and has been consistently experimentally verified since its formulation in the 70s. Despite its success, it fails to explain some important observed phenomena, such as, matter-antimatter asymmetry in the universe or the structure of generations of elementary particles. Moreover, it does not predict the existence of dark matter or describe quantum gravity. However, the SM does not point to any new unexplored areas which we can look into in order to find answers to these questions. One such way is to find violations of symmetries that are widely believed to be conserved and are intrinsic to the structure of SM. One such symmetry around which this problem revolves is lepton flavour. If significant signal for decays which do not conserve lepton flavour (for charged particles) was found in the dataset, it would challenge some of the assumptions of the SM. The implications for such a a finding would be far-reaching – it could lead to new physics and enrich our understanding on a fundamental level.

This problem was organised in the form of a competition on Kaggle [2]. Physicists from CERN designed the competition and provided any relevant code and datasets. There is a training set and a test set, as well as two datasets relating to verifying the submission. The datasets provided are a mixture of simulated and real data collected by the LHCb detectors observing collisions of accelerated particles with a specific mass range in which $\tau \rightarrow 3\mu$ (tau to three muons) decays cannot happen according to the SM.

Problem Statement

Given a list of collision events and their properties from the LHCb experiment (mixed in with simulated data), predict whether a $\tau \rightarrow 3\mu$ decay happened in the selected collision. Because such events are believed to be rare or non-existent, the goal is to discover this particular decay ($\tau \rightarrow 3\mu$) happening more frequently than expected. The goal is to improve the discriminating power between signal events (where the decay did occur) and background events (where it did not occur). The resulting classifier cannot be too dependent on discrepancies between real and simulated data or on τ mass. I plan to use a supervised learning approach for classification data to solve it.

Metrics

The evaluation metrics are defined in the evaluation page of the competition [3]. The proposed evaluation metric is the Weighed Area Under the ROC curve. The ROC curve is partitioned based on the True Positive Rate (TPR). The predictions must pass two additional checks before they are scored with weighed AUC, namely the agreement test and the correlation test (mentioned above). As this is a well defined problem for a competition, I did not have a choice for the type of metrics used but I agree that AUC metric is reasonable for this problem because this technique allows for the assignment of different weights to different thresholds of signal efficiency. This is important because the sensitivity to a given process is not a linear function of expected background events.

2. ANALYSIS

Data Exploration

Four datasets are provided in this competition:

- **training.csv** is a labelled dataset to use for training the classifier. Background events are composed of real data and simulations.
- **check_agreement.csv** is a labelled dataset with the same features as the training dataset. It is to be used for checking the agreement between simulated and real data.
- **check_correlation.csv** is a labelled dataset (same features as the training dataset) for checking the correlation of the classifier with the tau mass.
- **test.csv** is a non-labelled (signal and background are mixed and indistinguishable) dataset which contains simulated signal events and real background data, simulated events and real data for the control channel.

The training dataset has 50 features, which include metrics, such as: flight distance, properties relating to the tau candidate (mass, momentum, life time, impact parameter and others), variables relating to final states tracks, geometric variables relating to the particles' trajectories through the detector and more.

The dataset contains 67 553 events, out of which, no events contain non-numeric or missing data points. I could not find any irregularities in the data, no outliers or unexpected noise. Each feature contains the full set of 67 553 data points and the whole dataset contains 3 377 650 (67 553 x 50) data points altogether.

A sample row looks like this:

```
id, LifeTime, dira, FlightDistance, FlightDistanceError, IP, IPSig, VertexChi2, pt, DOCAone, DOCAtwo, DOCAthree, IP_p0p2, IP_p1p2, isolationa, isolationb, isolationc, isolationd, isolatione, isolationf, iso, CDF1, CDF2, CDF3, ISO_SumBDT, p0_IsoBDT, p1_IsoBDT, p2_IsoBDT, p0_track_Chi2Dof, p1_track_Chi2Dof, p2_track_Chi2Dof, p0_IP, p1_IP, p2_IP, p0_IPSig, p1_IPSig, p2_IPSig, p0_pt, p1_pt, p2_pt, p0_p, p1_p, p2_p, p0_eta, p1_eta, p2_eta, SPDhits, production, signal, mass, min_ANNmuon
```

```
18453471, 0.001578324, 0.999999344, 14.03333473, 0.681400955, 0.016039478, 0.451886147, 1.900432944, 1482.037476, 0.066666551, 0.060601622, 0.083659522, 0.208855301, 0.074342899, 8, 5, 7, 1, 0, 3, 4, 0.473951876, 0.349446565, 0.329157293, -0.57932359, -0.256309092, -0.215444162, -0.107570335, 1.921700239, 0.866657019, 1.230708122, 0.988053977, 0.60148257, 0.277089804, 16.24318314, 4.580875397, 5.939935684, 353.8197327, 448.3694458, 1393.246826, 3842.096436, 12290.76074, 39264.39844, 3.076006174, 4.003799915, 4.031513691, 458, -99, 0, 1866.300049, 0.277559102
```

More information about the datasets and the meaning of each feature can be found in the data tab of the competition page and more information about the problem, physics background, the design of the competition and evaluation procedures can be found in the research paper provided in the resources section.

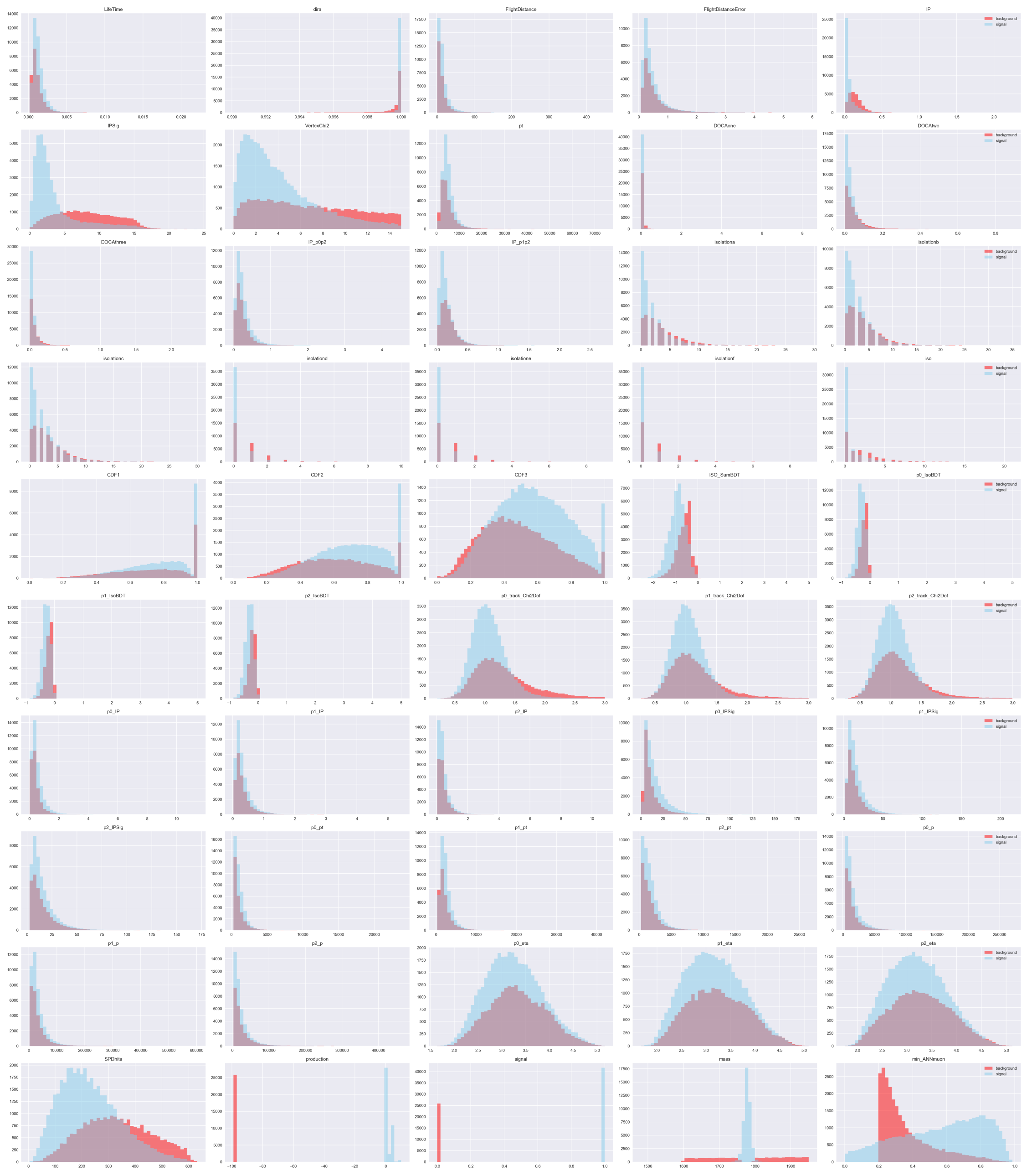
Exploratory Visualisation

In order to help with the main objective of improving the discriminating power between signal events (where the decay did occur) and background events (where it did not), it would be helpful to plot the available features as a function of the signal. The signal is available in the training set and is either 0 (for background events) or 1 (for signal events).

I have decided to plot all features but some of the features won't be used in training: id, mass, signal, production, min_ANNmuon and SPDhits.

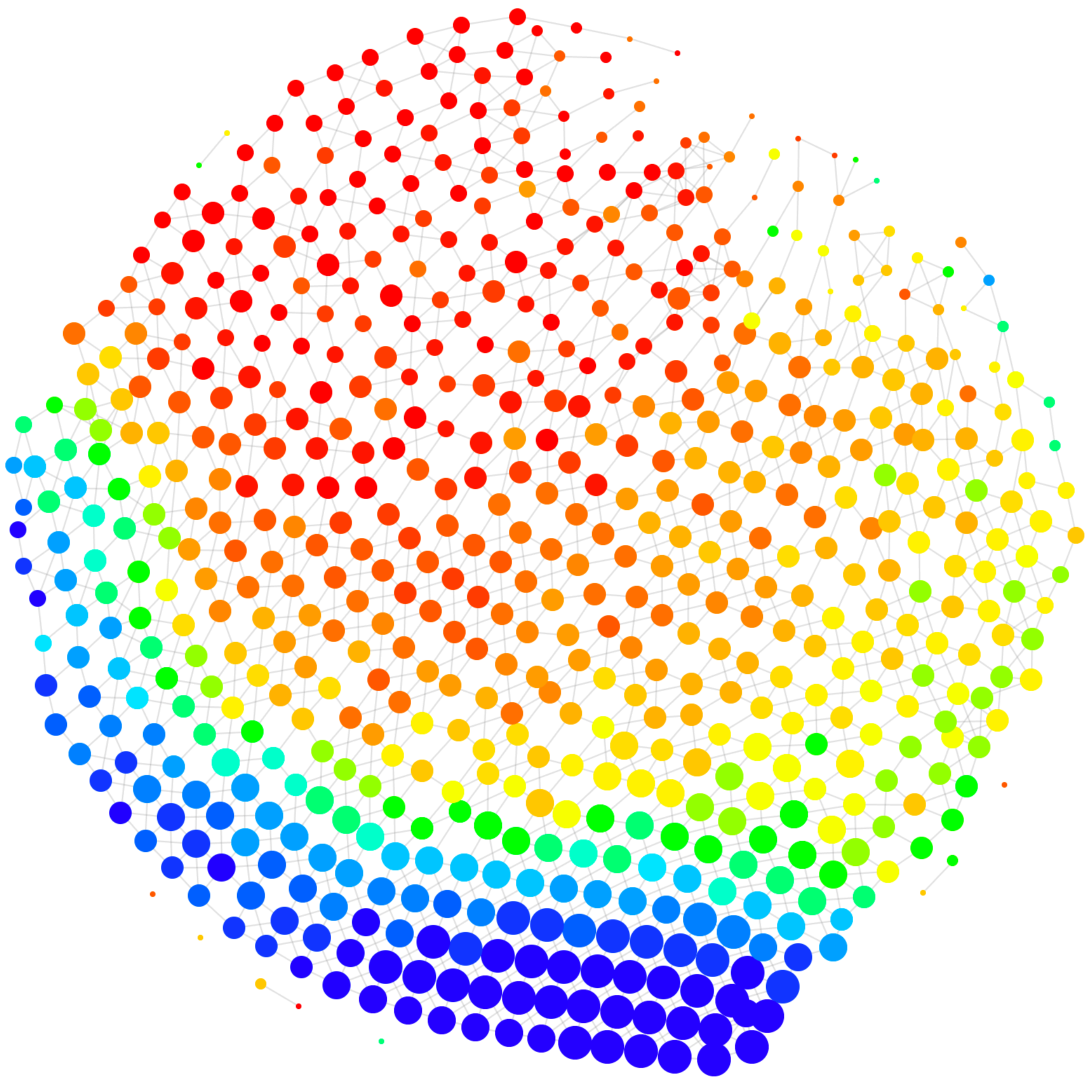
Plotting each of the features in a histogram and using different colours for signal/background events gives a quick way to differentiate between the distribution of the signal across all features.

As is shown in the plot below, the overlap is smallest for the following features: IP, IPSig, VertexChi2, CDF2, CDF3, ISO_SumBDT, p1_IsoBDT, p2_IsoBDT, p0_track_CHi2Dof, p1_track_CHi2Dof, p2_track_CHi2Dof, p0_eta, p1_eta and p2_eta. These features are likely to help in dividing the data.



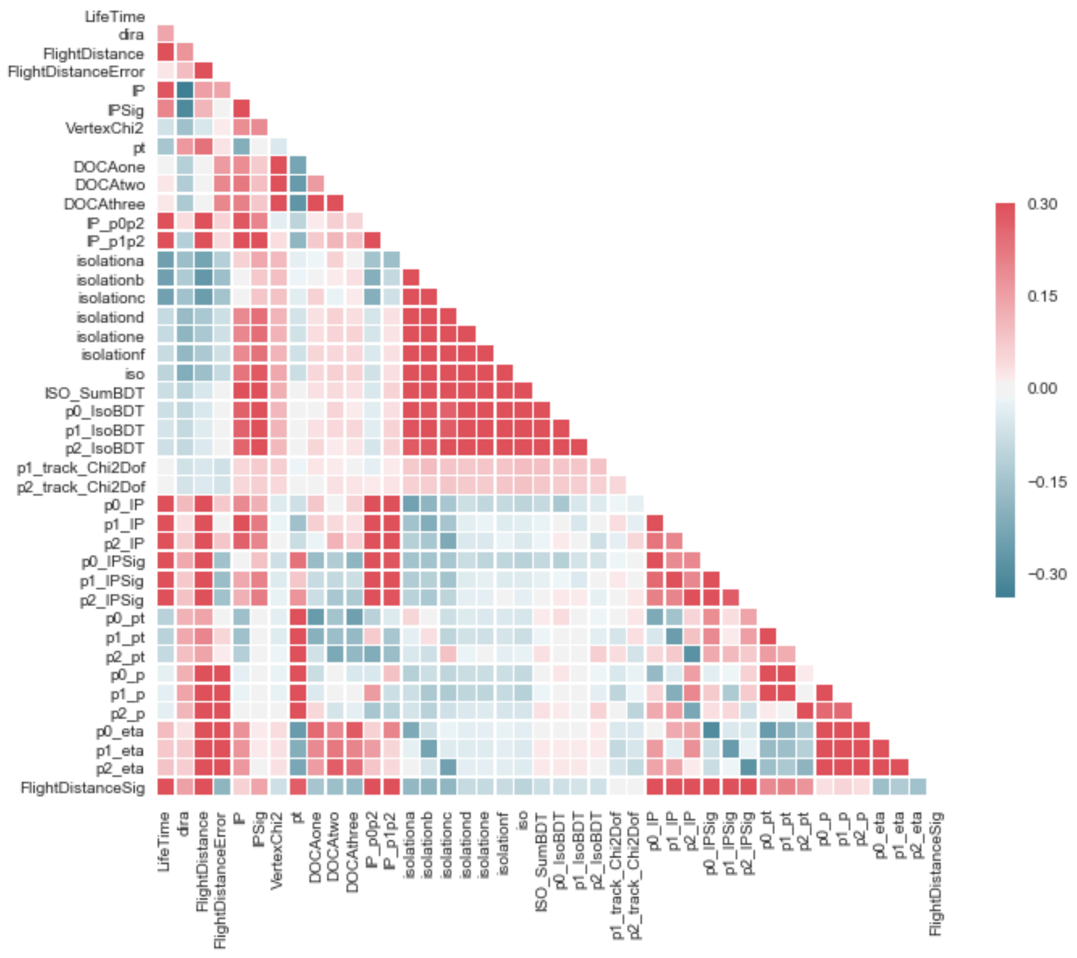
In the official paper that was provided with the competition [4], in section B "Geometric variables in LHCb detector", it is discussed which variables may play significant roles in discriminating between signal and background events. It is suggested that IP (impact parameter), IPSig (significance of impact parameter), Vertex x^2 and isolation variables are relevant.

I have used a python library called kepler-mapper [5] to plot topological graphs of the relevant features with clustering.



The visualisation above includes 30 overlapping (by 10%) intervals for 2 variables: IPSig and Vertex x^2 . When 5 or more points fall within an interval, they are clustered together. When two clusters share one or more points, a link is created between them. The resulting visualisation is a topological network with nodes (clusters) and edges (links). The clusters are coloured differently based on the proportion of signal vs no signal. Yellow nodes represent clusters where the proportion is equal, the nodes within the cluster contain the same amount of signal vs no signal. The red clusters contain a high amount of no-signal events and the blue clusters contain a high amount of signal events. This graph makes it easier to see that these two variables manage to divide the data quite well.

Additionally, it was helpful to see which variables were correlated with each other. I generated a diagonal correlation matrix to examine the correlations. As is shown in the matrix below, there are several groups of strongly correlated features. For example, unsurprisingly, the track isolation variables are highly correlated.



Algorithms and Techniques

As this is a classification task and the tolerance for biases is low, the ideal approach would be an ensemble method where the chance of overfitting is low. Random forests immediately stand out as a good approach, as they are simple to use, fast, can handle many input variables well and produce very good results in a wide variety of cases. Random forests work by combining many “weak learners” to form a “strong learner”. Single decision trees are prone to have high variance and bias and this is mitigated in random forests by averaging to find a good middle ground. On top of that, I plan to use gradient boosting to optimise the learning process. Gradient boosting, as applied to an ensemble method like random forests, builds the model in stages and allows optimisation of a differentiable loss/cost function. It is a framework for iteratively improving any weak learner. Boosted trees have been demonstrated to give state-of-the-art results on many standard classification benchmarks. I will be using the python implementation of XGBoost [6].

To select a suitable set of parameters, I will use grid search (sklearn’s GridSearchCV). Grid search exhaustively generates candidate estimators from a parameter space (specified in a param_grid variable with parameter names and ranges of values against them), fits them and saves the combination of values that produces the best result.

Benchmark

The best possible score that can be achieved for this competition is 1.0 (weighed AUC) in predicting the nature of collision events. I will be using the competition's leaderboard for benchmarking. There have been 3634 submissions from 673 teams to date, with weighted AUC score varying from 0.682 to 1.0, with average of 0.972.

3. METHODOLOGY

Data Preprocessing

The dataset provided for this competition was in very good shape. It did not require cleaning or any pre-processing apart from the exclusion of some input variables provided in the training set. The variables that I excluded were: id, min_ANNmuon (muon identification), production (source of τ), mass (τ candidate invariant mass) and signal because they couldn't be used for training. Additionally, I added a variable named FlightDistanceSig which equals FlightDistance / FlightDistanceError, as it amplifies the signal of low error flight distances. The XGBoost classifier did not require any scaling, so it was as straightforward as instantiating the classifier and fitting it to the data.

Implementation

I used python for the entire solution and used the following libraries: pandas, xgboost, sklearn, seaborn and matplotlib. I also used the provided evaluation.py script from the starter-kit [7]. The implementation steps are listed below.

- Loaded the dataset by using pandas.read_csv to read the datasets (train, test, check_agreement and check_correlation) into data frames
- Added the extra feature to the datasets and filtered out events for which min_ANNmuon was 0.4 or lower, as recommended by the official paper (on page 6)
- Excluded the following features: id, min_ANNmuon, production, mass, signal and SPDhits
- Instantiated the model (xgboost.XGBClassifier), parameters and created a score function (which uses roc_auc_truncated from the provided evaluation.py in the starter kit
- Instantiated GridSearchCV using the xgboost model, the parameters (initially I used the default values for everything), cross validation function (using StratifiedKFold) and the score function
- Used compute_ks provided in starter-kit/evaluation.py file to verify that the KS-value is smaller than 0.09 as required by the agreement test, then repeated for the correlation test (using compute_cvm)
- Calculated the AUC using roc_auc_truncated function with the signal pandas Series as first parameter (train.signal) and the train predictions as second parameter (clf.predict_proba(train[features])[:, 1])

- Obtained the predictions for the test data and generated a submission file using the ID and the prediction corresponding to the same ID

After running the validation tests I noticed that the agreement test was failing and after a lot of trial and error and reading threads, I discovered that the features SPDHits (number of hits in the SPD detector), p0_track_Chi2Dof (quality of the p0 muon track), CDF1, CDF2 and CDF3 (cone isolation variables), isolationb, isolationc (track isolation variables) were causing this issue (or contributing to the issue). So, I re-ran everything after adding these features to the exclude list.

Refinement

Initially, I ran grid search with only the default parameters to establish a base benchmark. The private and public scores I obtained from uploading the result file to Kaggle were 0.981241 and 0.978707 respectively. To obtain improved scores, I tried expanding the parameter grid space significantly. I searched through various values for max_depth, learning_rate, n_estimators , objective, min_child_weight, max_delta_step, subsample, colsample_bytree and colsample_bylevel for the XGBClassifier. The best score I obtained was with the following values for each of the parameters (default values for the rest):

Parameter	Value
max_depth	7
learning_rate	0.15
n_estimators	300
objective	binary:logistic
min_child_weight	11
subsample	0.9
colsample_bytree	0.5

The private and public scores I obtained from uploading the final result file to Kaggle were 0.983324 and 0.981796 respectively.

As the XGBClassifier performs superbly even with no optimisation, I did not try any other classifiers and spent time only searching for the optimal set of parameters.

4. RESULTS

Model Evaluation and Validation

Gradient boosting is an improvement on ensemble method approaches, so I decided to start with potentially the best approach for this type of problem after a lot of research on the topic. The final model took a lot of trial and error in terms of picking the best subset of input variables and optimising the parameters fed into the classifier. The model performs extremely well with unseen data (the test dataset and the hidden test dataset on Kaggle).

I explored the following parameters for xgboost.

Parameter name	**Description**	**Tuning**
booster	Type of model, options are gbtree (tree-based models) and gblinear (linear models)	I used gbtree as I wanted a tree based model
n_jobs	Number of parallel threads used to run in xgboost	I used 4.
min_child_weight	Defines the minimum sum of weights of all observations required in a child	I explored values ranging from 2 to 15. The best performing values were 10 and 11.
max_depth	Maximum tree depth for base learners. Used to control over-fitting as higher depth allows model to learn very specific relations.	I searched between the range 3-10. Best performing values were 7 and 8.
learning_rate	Boosting learning rate	Explored values between 0.1-0.5. The value yielding best results was 0.15.
gamma	Minimum loss reduction required to make a further partition on a lead node of the tree	Searched between 0-1, the default value of 0 performed best.
subsample	Represents the fraction of observations to be randomly sampled for each tree. Lower values help the algorithm avoid overfitting.	Explored the values 0.5-1. The value 0.9 performed best.
colsample_bytree	Subsample ratio of columns when constructing each tree	Explored values 0.5-1. Ended up using 0.5 as it gave the highest scores.

Parameter name	Description	Tuning
colsample_bylevel	Subsample ratio of columns for each split, in each level.	Explored values 0.5-1. The default value performed best, so I did not change it.
lambda	L2 regularization term on weights, increase this value will make model more conservative	I explored 0.1-1 for this variable and the default value gave best results.
objective	Defines the loss function to be minimised.	Ran with binary:logistic and multi:softprob. The first one gave best results.

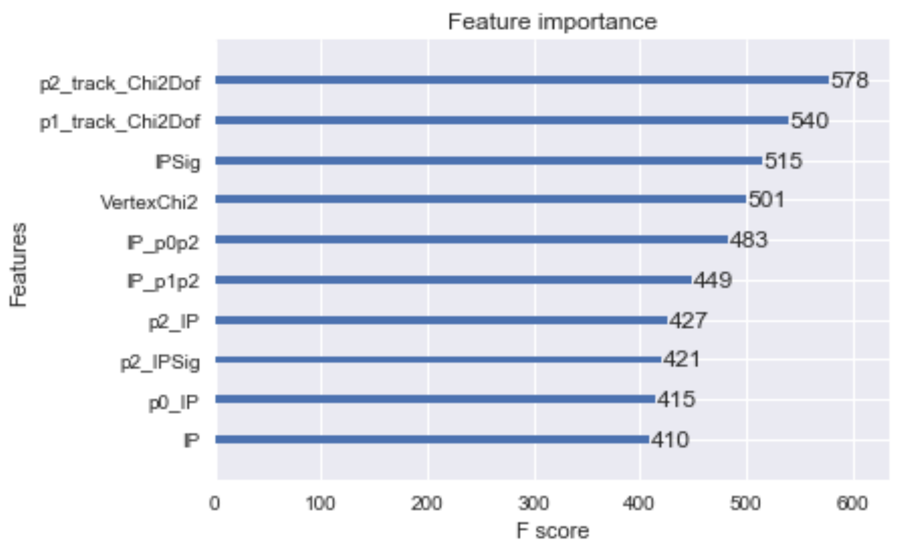
Justification

The benchmark I reported earlier in the report was the average of all the submissions for this problem on Kaggle, which was 97.2%. My best solution scored 98.2%, which is slightly better than average (by approximately 1/10th standard deviation). My score falls close to the 2nd quartile which is ~98.4%. While my solution doesn't shine as compared to all other submissions, the competition level on Kaggle is usually quite high and the result itself is also reasonably good, so I am happy with my result.

5. CONCLUSION

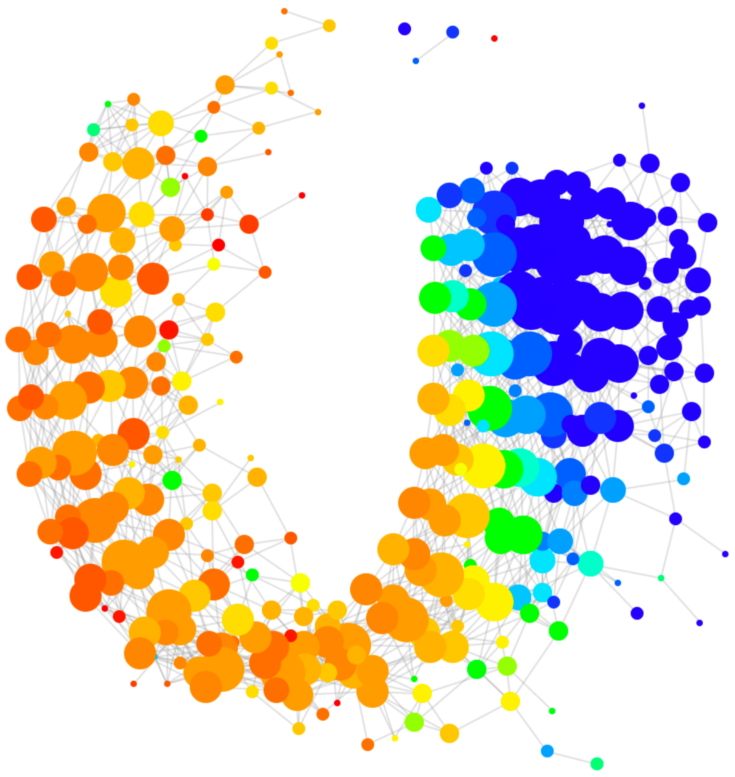
Free-Form Visualisation

After fitting the xgboost classifier with the optimal set of parameters, I could obtain the most important features and plot them using the plot_importance function.

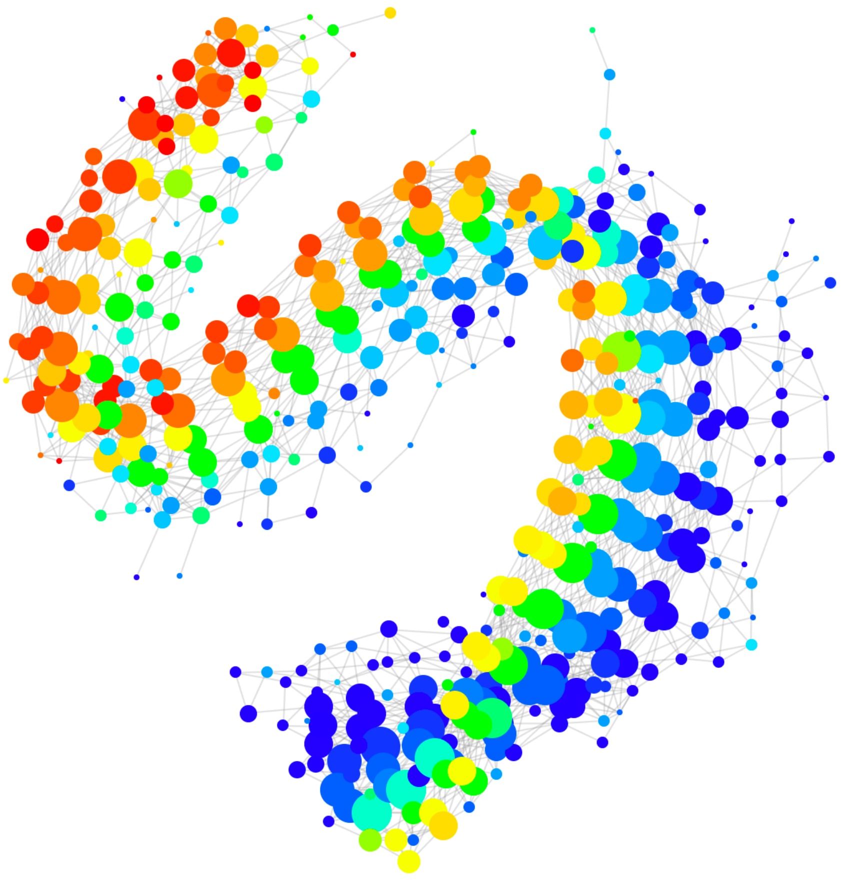


As the plot shows p2_track_Chi2Dof and p1_track_Chi2Dof are the two most important features. They represent the quality of the p1 and the p2 muon tracks. Then, the significance of the impact parameter (IPSig), followed by VertexChi2, IP_p0p2 (impact of the p0 and p2 pair) and IP_p1p2 (impact parameter of the p1 and p2 pair).

Using this information I was able to generate another topological graph using the 3 most important features.



The graph shows that these features divide the data reasonably well too. I also wanted to plot the feature VertexChi2 against the two muon track variables. This yielded the following visualisation which also clearly divides the data.



Both of the visualisations use the same parameters as the first one – 30 intervals with overlap of 10%. The boosted trees managed to find these differences and use them in order to discriminate against signal and background events and yield very good results.

Reflection

This problem presupposed a good grasp of physics and a lot of research and reading of papers in order to obtain a good understanding of the challenge. I spent a lot of time trying to understand what the variables meant and visualising them, so I could understand the relationships between them better. I started by reading the three papers provided on the competition page, they helped by providing more context regarding why the problem is important to solve and also why it is challenging. The post [8] about the ML approach in High Energy Physics was really helpful. I spent a large chunk of the research process reading about what approach yields good results for this type of problem and concluded that random forests are probably the most sensible approach. Then, I discovered that using gradient boosting yields improved results and also it was a popular approach amongst the other competitors. I read about how gradient boosting works and why it is as effective as it is. It was difficult to have any kind of intuition about which features were important, so I relied heavily on the algorithm itself to be discerning. I found the problem very interesting, so it was curious to read more about the background and also refresh my knowledge of the standard model. It makes sense that gradient boosting performs well for this type of problem and also in general as it generalises well and allows the usage of any loss function.

Improvement

A possible improvement on my solution would be to engineer new features out of the existing ones. Because all the input features are numeric and have inter-relationships, it would be helpful to try to distill some of those relationships into separate variables or to come up with new ones that help the model perform better. Also, expanding the grid search parameter space could have produced more optimised parameters but that requires a lot of time and it does tend to yield diminishing returns after a while. Moreover, I did not explore all of the parameters that were available for the XGBClassifier, so, potentially some of the default values for them could be improved.

References

1. https://en.wikipedia.org/wiki/Noether%27s_theorem
2. https://www.kaggle.com/c/flavours-of-physics
3. https://www.kaggle.com/c/flavours-of-physics#evaluation
4. https://kaggle2.blob.core.windows.net/competitions/kaggle/4488/media/lhcb_description_official.pdf
5. https://github.com/MLWave/kepler-mapper
6. http://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf
7. https://github.com/yandexdataschool/flavours-of-physics-start
8. https://www.kaggle.com/c/flavours-of-physics/discussion/15974