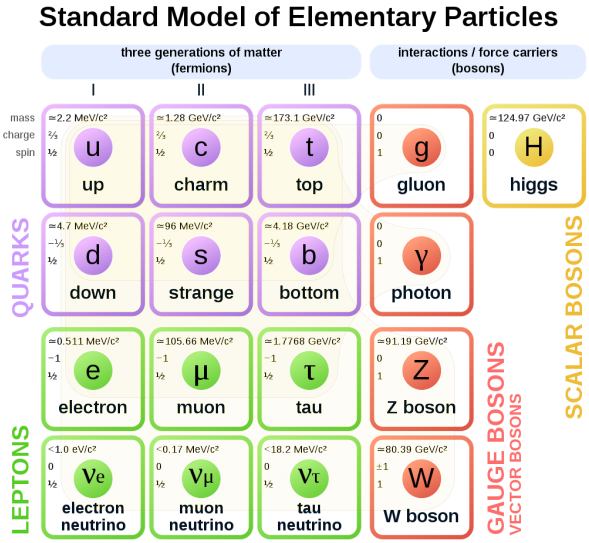
Project Report: Flavors of Physics, The Strange D Meson Decay

Problem Definition

Domain Background

This project is a particle physics problem. Its name is inspired by what physicists call "flavor", the species of an elementary particle. The Standard Model of particle physics is a well-established theory that explains the properties of fundamental particles and their interactions, describing the "flavor" of each particle. As mentioned in Charlotte Louise Mary Wallace CERN Thesis, the Standard Model theory has been tested by multiple experiments, but despite its successes, it is still incomplete, and further research is needed.

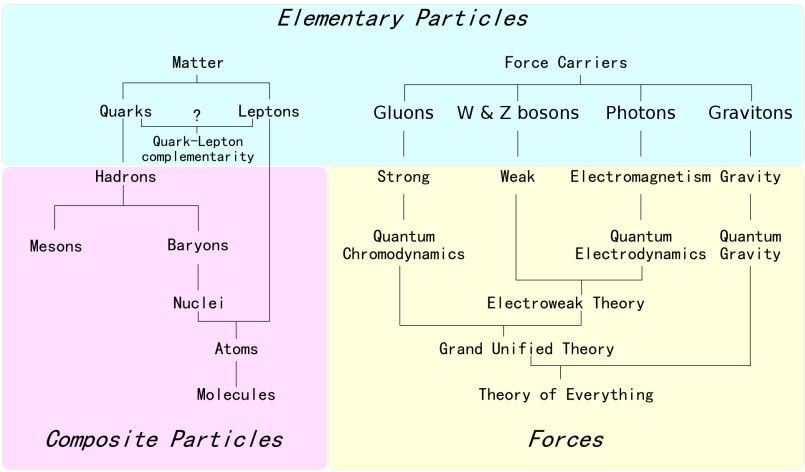
The Standard Model counts six flavors of quarks and six flavors of leptons, as shown below. "Flavor" is essentially a quantum number that characterizes the quantum state of those quarks.



The Ds decay project is influenced by a CERN kaggle competition problem about the flavors of physics. In the initial problem, scientists try to find if it is possible the τ (*tau*) lepton to decay (transform into multiple other particles) to three μ (muon) leptons. The problem I chose, however, concerns the Ds meson or *strange D meson*, a composite particle that consists of one quark or one antiquark, and how often it decays into a ϕ (phi meson) and a π (pi meson or pion) based on multiple factors. The decay is described by the following flow:

$$D_s \rightarrow \varphi\pi$$

You can see where the meson belongs in the subatomic particles map below. The purple part describes the composite particles.



Ander Ryd in his paper argues that the D meson decays have been a challenge, though scientists have been focused on their decays since the particle discovery. As a result, the existing dataset of this project is sufficient and based on well-studied experiment observations.

Problem Statement

The problem falls into the category of binary classification problems. Based on particle collision events (that cause the $D_s \rightarrow \varphi\pi$ decays) and their properties, I am challenged to train a machine learning model that predicts whether the decay we are interested in happens in a collision. The model will be trained in the train set of the existing dataset, and it will be evaluated on the testing set.

Data Exploration

As described in the flavors of physics project, the $D_s \rightarrow \varphi\pi$ decay has a very similar topology as the *tau* decay, and their datasets share almost the same features. In the *tau* decay problem, the Ds decay data is used as part of the CERN evaluation process. This dataset will be used as the main dataset of the $D_s \rightarrow \varphi\pi$ decay problem solution.

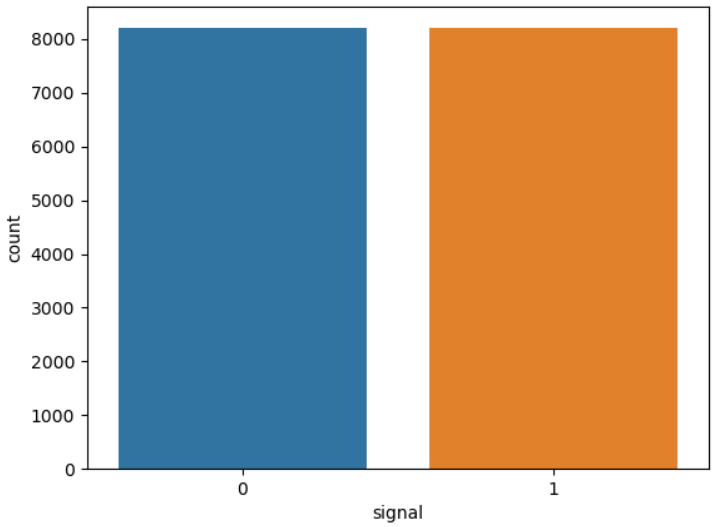
This is a labeled dataset of 16.410 samples and 46 features, which are described below. The *signal* column classifies the samples into *signal events* (decays happening) denoted with *1* and *background events* (decays not happening) denoted with *0*. The features of the dataset are described below:

- FlightDistance - Distance between Ds and PV (primary vertex, the original protons collision point).
- FlightDistanceError - Error on FlightDistance.
- LifeTime - Life time of Ds candidate.
- IP - Impact Parameter of Ds candidate.

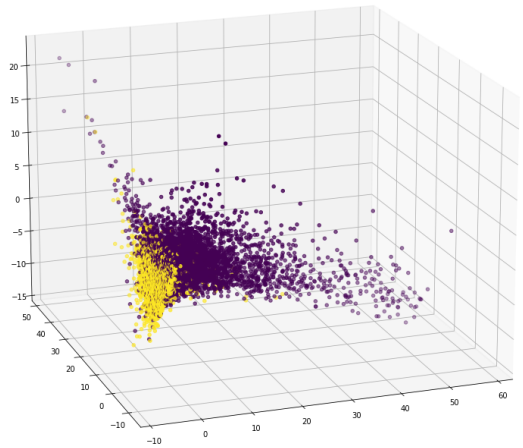
- IPSig - Significance of Impact Parameter.
- VertexChi2 - χ2 of Ds vertex.
- dira - Cosine of the angle between the Ds momentum and line between PV and *Ds* vertex.
- pt - transverse momentum of Ds.
- DOCAone - Distance of Closest Approach between p0 and p1.
- DOCAtwo - Distance of Closest Approach between p1 and p2.
- DOCAthree - Distance of Closest Approach between p0 and p2.
- IP_p0p2 - Impact parameter of the p0 and p2 pair.
- IP_p1p2 - Impact parameter of the p1 and p2 pair.
- isolationa - Track isolation variable.
- isolationb - Track isolation variable.
- isolationc - Track isolation variable.
- isolationd - Track isolation variable.
- isolatione - Track isolation variable.
- isolationf - Track isolation variable.
- iso - Track isolation variable.
- CDF1 - Cone isolation variable.
- CDF2 - Cone isolation variable.
- CDF3 - Cone isolation variable.
- ISO_SumBDT - Track isolation variable.
- p0_IsoBDT - Track isolation variable.
- p1_IsoBDT - Track isolation variable.
- p2_IsoBDT - Track isolation variable.
- p0_track_Chi2Dof - Quality of p0 muon track.
- p1_track_Chi2Dof - Quality of p1 muon track.
- p2_track_Chi2Dof - Quality of p2 muon track.
- p0_pt - Transverse momentum of p0 muon.
- p0_p - Momentum of p0 muon.
- p0_eta - Pseudorapidity of p0 muon.
- p0_IP - Impact parameter of p0 muon.
- p0_IPSig - Impact Parameter Significance of p0 muon.
- p1_pt - Transverse momentum of p1 muon.
- p1_p - Momentum of p1 muon.
- p1_eta - Pseudorapidity of p1 muon.
- p1_IP - Impact parameter of p1 muon.
- p1_IPSig - Impact Parameter Significance of p1 muon.
- p2_pt - Transverse momentum of p2 muon.
- p2_p - Momentum of p2 muon.

- p2_IP - Impact parameter of p2 muon.
- p2_IPSig - Impact Parameter Significance of p2 muon.
- SPDhits - Number of hits in the SPD detector.
- signal - This is the target variable.

The dataset is balanced, with 8.205 signal events and 8.205 background events, with no missing values. The testing set that is used for the calculation of the performance metrics is the 20% of the dataset, 3.282 samples. A simple count plot is below:

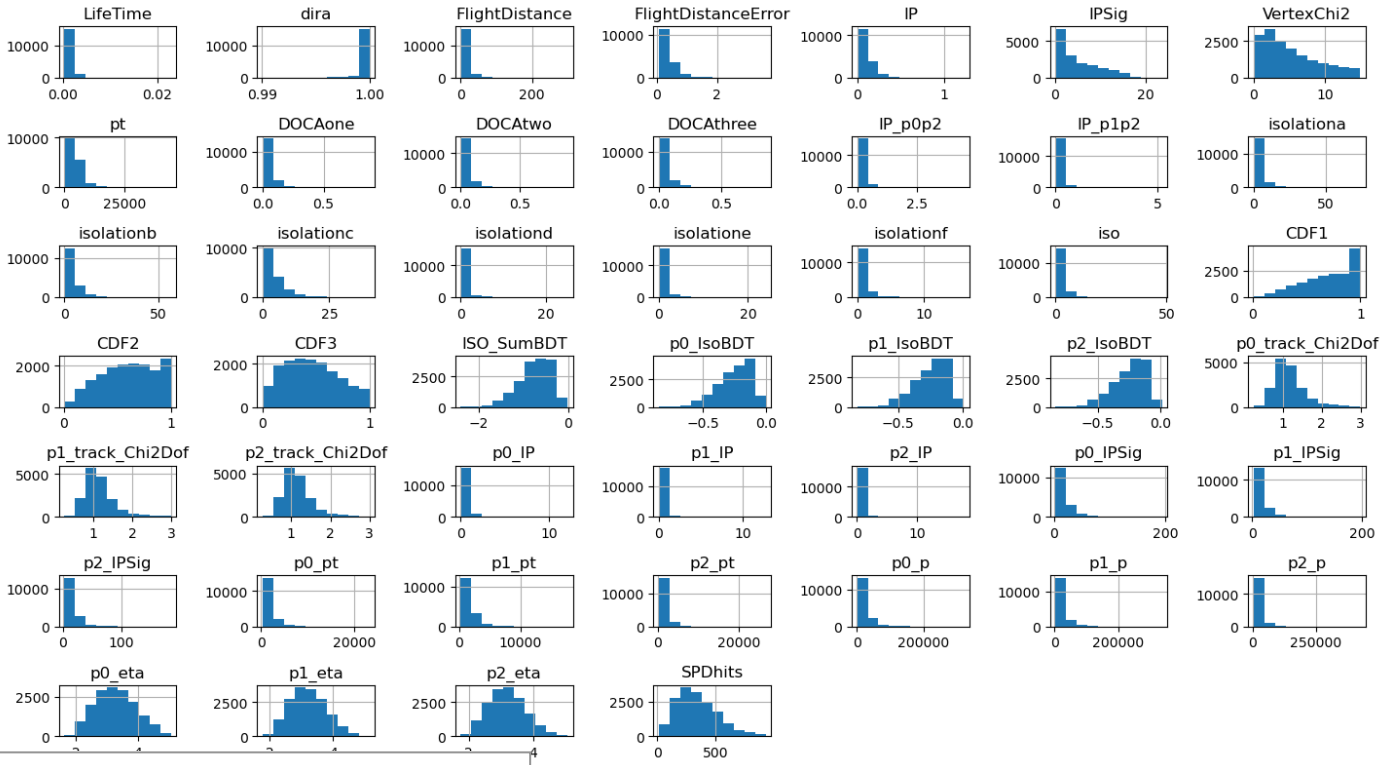


To have a better understanding of the data, I used PCA to reduce the dimentions of the dataset and to be able to visualize them in 3 dimentions. PCA method captures the maximum variance across the features and projects observations onto mutually uncorrelated vectors (components). Although it is used for dimensionality reduction, it also helps to discover underlying patterns across features. It is a common algorithm dimentionality reduction before implementing clustering algorithms, but I use it here to get an idea of how the data look like in fewer dimensions. As PCA projects the data to less dimensions, I had to scale the data beforehand with a standard scaler.

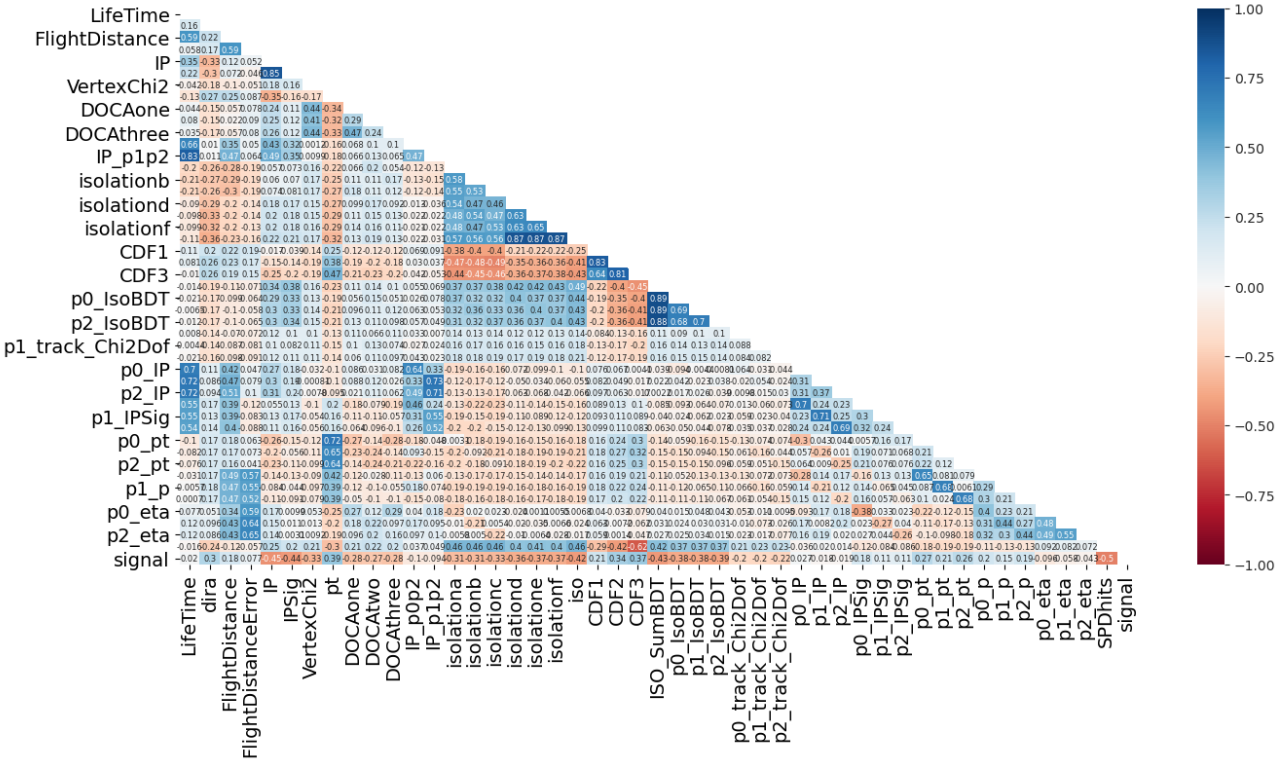


My intention for this plot was to see if I can clearly spot two seperate blobs of data, so a clear seperation between the signal and background events. However, this is something that is visible for this plot 3D plot.

To take a closer look the features, I plotted the distribution of each attribute. Most variables have a skewed distribution, so the dataset provides a good candidate for scaling the data, for example with a robust scaler transform to standardize the data in the presence of skewed distributions and outliers.



The following heatmap shows how linearly correlated the features are. The annotated numbers show the Pearson correlation coefficients. This is a common method intended to reduce the number of input variables to those that are believed to be most useful to a model in order to predict the target variable. Pearson correlation assumes a Gaussian probability distribution to the observations and a linear relationship, so I have scaled the data with a robust scaler transform before calculating the correlations between the features. The highest coefficients are below 0.9, and over 0.8 for 4 features, so I decided to include all featrues in the training dataset.



> *All the plots in this section are generated by executing the data_exploration python file. The dataset described can be found here.*

Note that the dataset has been resampled, as the initial dataset was imbalanced and to the size of the first dataset. You can refer to the project proposal for more information. The resampling technique use is the following:

```
from imblearn.under_sampling import RandomUnderSampler
undersampler = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = undersampler.fit_resample(X, y)
```

From the analysis above it is clear that the data need to be scaled before they are fed to the ML models.

Algorithms Implementation

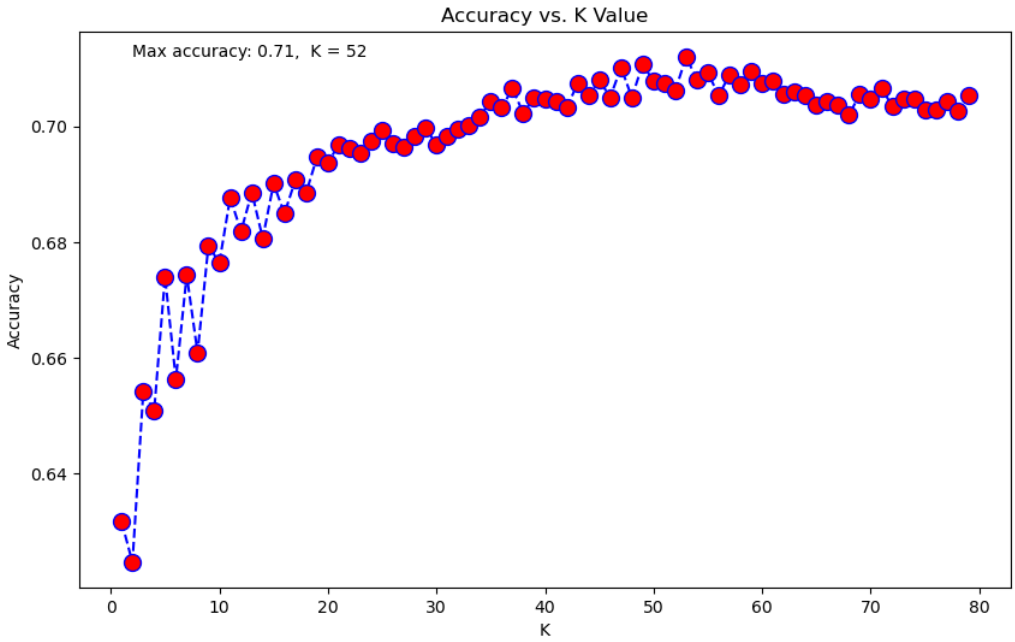
This is a supervised binary classification problem, so the solution will be the output of a binary classifier. There is no constrain in using any classifier in particular for this problem. However, in the evaluation description of the Kaggle competition described so far, it is mentioned that the Kolmogorov–Smirnov (KS) test is used to evaluate the differences between the classifier distribution and the true *signal* values distribution. Also, the KS test should be less than 0.09.

Models Exploration

To solve this problem, I trained a number of binary classifiers, using different hyperparameters tuning methods, in combination with different data scaling methods. For all the different results, a number of performance metrics are gathered in a single table, and the best model is chosen based on the metrics values. The metrics are presented in the *Models Performance* section. As part of the project proposal, I trained the benchmark model presented below.

Benchmark Model

As a benchmark model, I use a simple k-Nearest Neighbor classifier, and grid search for tuning the k hyperparameter. The benchmark model script can be found here. The execution of that script generates the following plot. The plot shows the accuracy of the kNN model for each one of the k values (from 1 to 80). The best model is the kNN model with k=52, and highest accuracy of 71%.



The plot above is generated by executing the benchmark model script

Improving the Benchmark Model

To improve on the benchmark model, I trained a number of different binary classifiers and compared their performance. You can find the script with the modesl exploration here. As scaling of the input data is a requirement for most machine learning estimators in this project, I have also scaled the data in different ways. An example is a Support Vector Machines model (mentioned below), which assumes that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

All scaled methods were used in combination with all the models. Also, different methods of hyperparameter tuning were used for each model.

Scaling methods used

- `sklearn` 's Standard Scaling:

$$z = (x - u)/s$$

where u is the mean of the training samples or zero if `mean=False` , and s is the standard deviation of the training samples or one if `std=False` .

- `sklearn` 's Minmax Scaling:

$$X_{scaled} = X_{std} * (max - min) + min$$

where min, max is the features range

This estimator scales and translates each feature individually such that it is in the given range on the training, zero and one in this case.

- `sklearn` 's Robust Scaling:

Scales features using statistics that are robust to outliers. This Scaler removes the median and scales the data according to the quantile range.

Binary Classifiers trained

The script that implement the following models can be found here, where you can see the ranges of parameters used for hyperparameters tuning in more detail.

The models used are:

- kNN, tuned with the Grid Search Method
  The details of the kNN model are explained in the benchmark model section. For this model, the
  meter tuning was used, as grid search for kNN is not

computationally expensive due to kNN's single parameter. The range of the n nearest neighbors is from 0 to 60.
- Stochastic Gradient Descent, tuned with Random Search for better performance and speed. SGD work well for both small and large datasets and I used it as an optimization technique for different loss functions. A number of loss functions and regularizations are used in the Random Search (refer to the models exploration script for more).
- Support Vector Machines, tuned with Random Search. Different options of *regularization*, *decision function shape* and *kernel* parameters were used for the SVM Random Search (refer to the models exploration script for more).
- XGBoost with Bayes optimization for hyper parameters tuning using skopt Bayes Search, that uses a fixed number of parameters, sampled from a specified distribution. The first optimization maximizes the classifier function (XGBoost in this case) and gives the uses the control of the steps of the bayesian optimization and the steps of the random exploration that can be performed.

The XGBoost model is the one I chose to predict the test set. A seperate script that implements XGBoost and returns the predicted *signal* values can be found here.

Models Performance

TODO: integrate:According to the original kaggle problem described in the problem definition section, the KS test have to be lower than 0.09 for the model to be considered "good enough" to pass. All the models trained are below 0.09, so I had to look for more metrics.

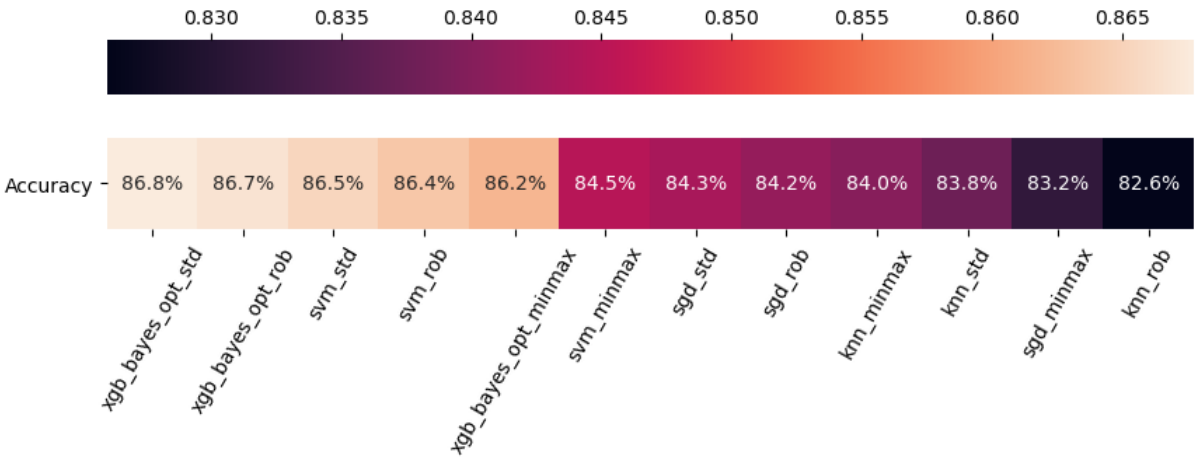
TODO: from kaggle: The control channel dataset is used purely in Kolmogorov–Smirnov test. We require the KS-value of the test to be smaller than 0.09. Only when a submission can pass the agreement test on these ignored data, we can then regard the predictions as valid. Find the pass test instructions here.

I use `sklearn` accuracy as the main performance metric based on which I compare the models. I am also calculating a number of other metrics in order to have more information about how the models perform, for example in cases like prediction of true positives or negatives. Accuracy is calculated by dividing the number of correct predictions by the total number of samples. As the dataset is balanced with equal class distribution, the accuracy paradox is avoided and the metric does not provide misleading information about the models' performance. Note that I also used *accuracy* as the main evaluation metric to choose the best model from the Random Search or the Grid Search methods I used for hyperparameter tuning. In the case of XGBoost, which is optimized with bayesian optimization, accuracy is not a possible option to choose the best parameters for the model, so AUC

was used to decide which is the best parameters for the model, based on the model's capability to distinguish different classes.

All the evaluation metrics that were calculated are: *Accuracy*, *Kolmogorov–Smirnov test*, *false positive rate, false negative rate, true negative rate, negative predictive value*, *Recall, Precision*, *F1*.

You can refer to the metrics results for all the models' metrics, for all the scaling methods used on the data, before they are fed into the models. The following heatmap only shows the accuracy of all the models.



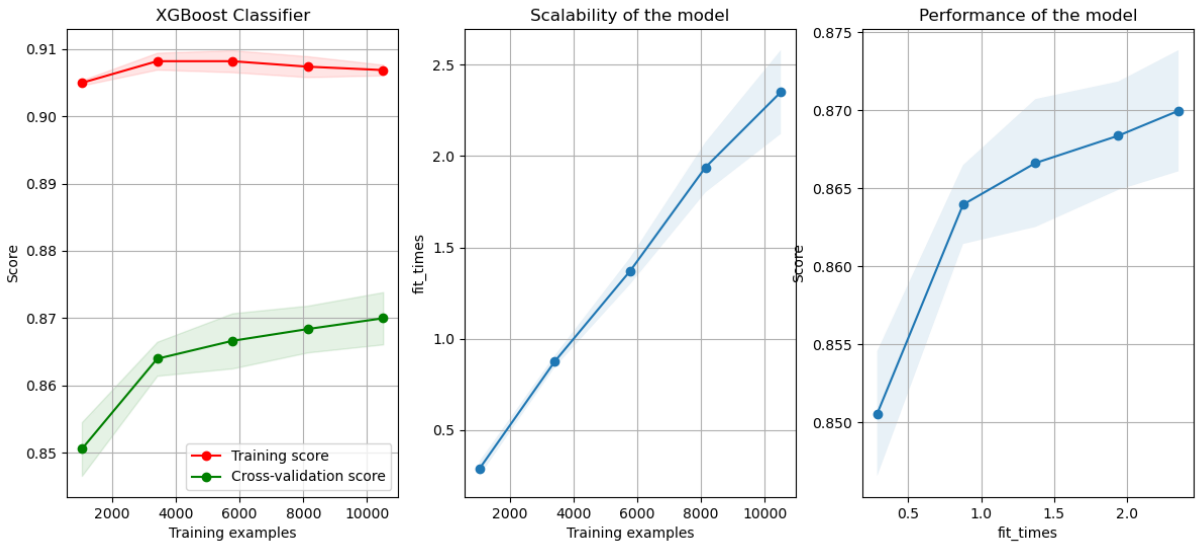
The XGBoost model trained with data scaled with the `sklearn` standard scaler has the highest accuracy (0.87) and the lowest KS (0.004). It also has the highest precision (0.87) and highest true negative rate (0.86) of all the models trained, so it is the best model in terms of how well it can predict both true positive and true negative values.

Conclusion

You will collect results about the performance of the models used, visualize significant quantities, and validate/justify these values. Finally, you will construct conclusions about your results, and discuss whether your implementation adequately solves the problem.

Learning curves

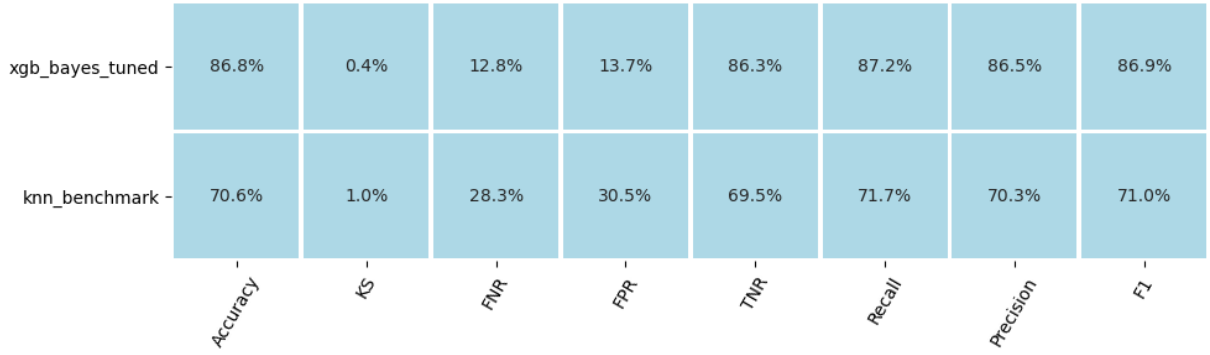
Learning curve graphs:



TODO: add learning curve plot understanding

"""The first plot is the learning curve
The plots in the second row show the times required by the models to train with various sizes of training dataset. The plots in the third row show how much time was required to train the models for each training sizes."""

Comparison with the benchmark model



TODO: Add a structure of the project in README

TODO Further improvements:

- Improve feature selection

- Use model stacking

Project Proposal

You can find here the submission of the project proposal.

Sources:

- https://en.wikipedia.org/wiki/Flavour_(particle_physics)
- https://cds.cern.ch/record/2713513?ln=en
- https://en.wikipedia.org/wiki/Standard_Model
- https://cds.cern.ch/record/2196092/files/CERN-THESIS-2016-064.pdf
- https://en.wikipedia.org/wiki/Particle_decay
- https://en.wikipedia.org/wiki/D_meson
- https://en.wikipedia.org/wiki/Phi_meson
- https://en.wikipedia.org/wiki/Pion
- https://wiki.classe.cornell.edu/pub/People/AndersRyd/DHadRMP.pdf
- https://www.kaggle.com/c/flavours-of-physics/overview
- https://en.wikipedia.org/wiki/Kolmogorov–Smirnov_test
- https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/lifecycle
- http://cds.cern.ch/record/2668282/files/BPH-17-004-pas.pdf
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/
- https://en.wikipedia.org/wiki/Accuracy_paradox
- http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/
- https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python
- https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html
- https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb
- https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/
- https://github.com/Punchyou/blog-binary-classification-metrics
- https://www.youtube.com/watch?v=aXpsCyXXMJE
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
- https://towardsdatascience.com/a-guide-to-svm-parameter-tuning-8bfe6b8a452c
- m/types-of-classification-in-machine-learning/

- https://www.kaggle.com/nanomathias/bayesian-optimization-of-xgboost-lb-0-9769
- https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/
- https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
- https://machinelearningmastery.com/quick-and-dirty-data-analysis-with-pandas/
- https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/
- https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf

Books:
Data Science from Scratch
Machine learning with python
Feature Egnineering