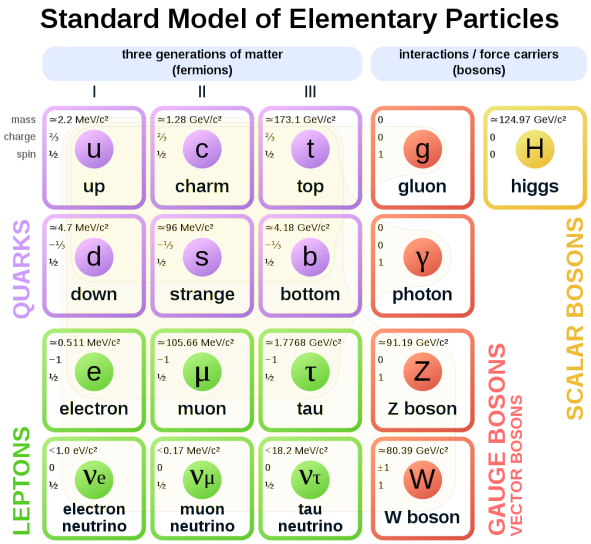
# Project Report: Flavors of Physics, The Strange D Meson Decay

## 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.

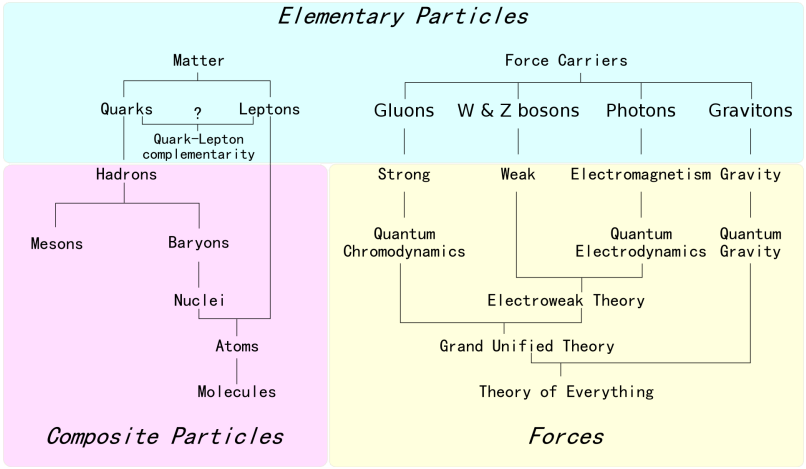
The Standard Model counts six flavors of quarks and six flavors of leptons, as shown below. "Flavor" is essentially a quantum number that characterizes the quantum state of those quarks.




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 $\phi$ (phi meson) and a $\pi$ (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 on the training set, which the 80% of the existing dataset described in the next section, and it will be evaluated on the remaining data.

## 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.

The dataset used for this project can be found here. However, this is a resampled dataset (find more information on that at the end of this section). More information on how to obtain the original dataset can be found in the project proposal.

The dataset used 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.

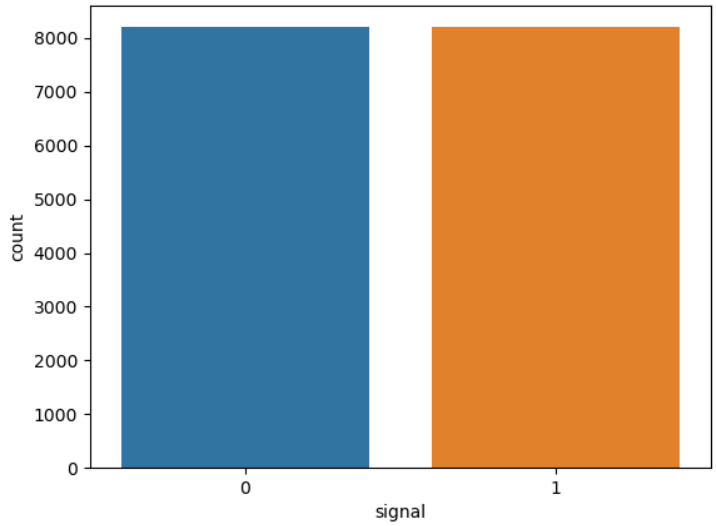
- DOCAthree - Distance of Closest Approach between p0 and p2.
- IP_p0p2 - Impact parameter of the p0 and p2 pair.
- IP_p1p2 - Impact parameter of the p1 and p2 pair.
- isolationa - Track isolation variable.
- isolationb - Track isolation variable.
- isolationc - Track isolation variable.
- isolationd - 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_eta - Pseudorapidity 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.

**Balance of Data**

The initial dataset has been resampled as it was heavily imbalanced and with too many samples to be easily downloaded by the reviewers of this project. After resampling, I made sure there were enough data for this analysis. You can refer to the project proposal for more information on how to obtain the original dataset. The resampling technique used is the following:

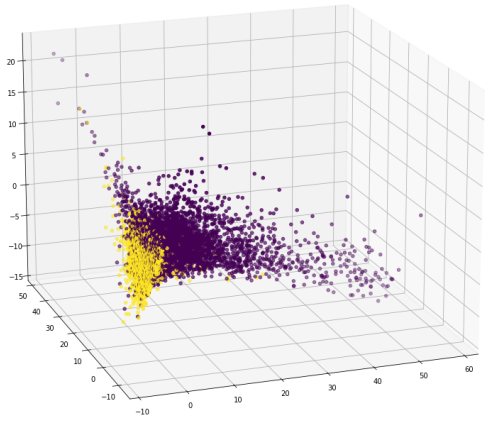
```
from imblearn.under_sampling import RandomUnderSampler
undersampler = RandomUnderSampler(random_state=42)
# X is an array of are the features and y is an 1D array of target
variable
X_resampled, y_resampled = undersampler.fit_resample(X, y)
```

The new dataset produced 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 of the samples per class is presented below:




**Dimentionality Reduction**

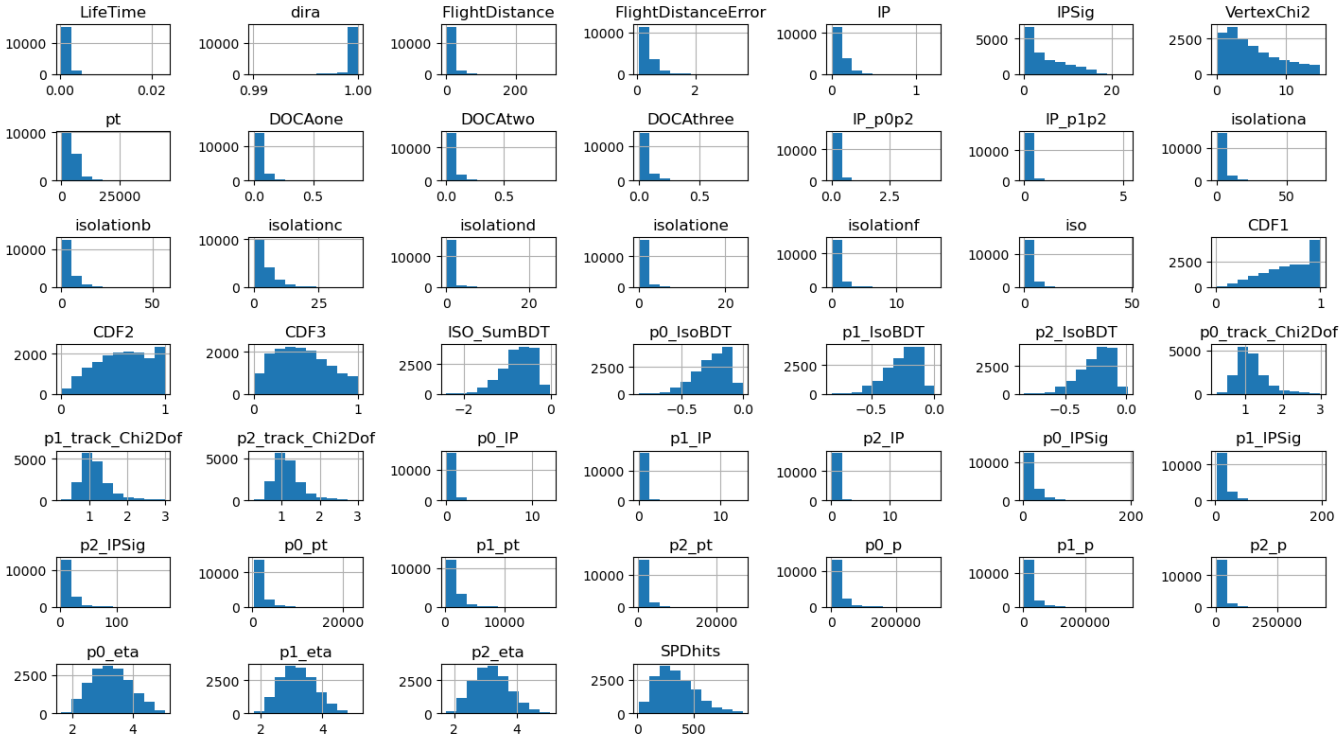
To have a better understanding of the data, I used PCA to reduce the dimentions and to visualize them in the 3 dimentional space. The PCA method captures the maximum variance across the features and projects observations onto mutually uncorrelated vectors (components). Although it is used for dimensionality reduction, it also helps to discover underlying patterns across features. It is a common algorithm used for 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 the `sklearn` 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 from this 3D plot.

**Features Distribution**

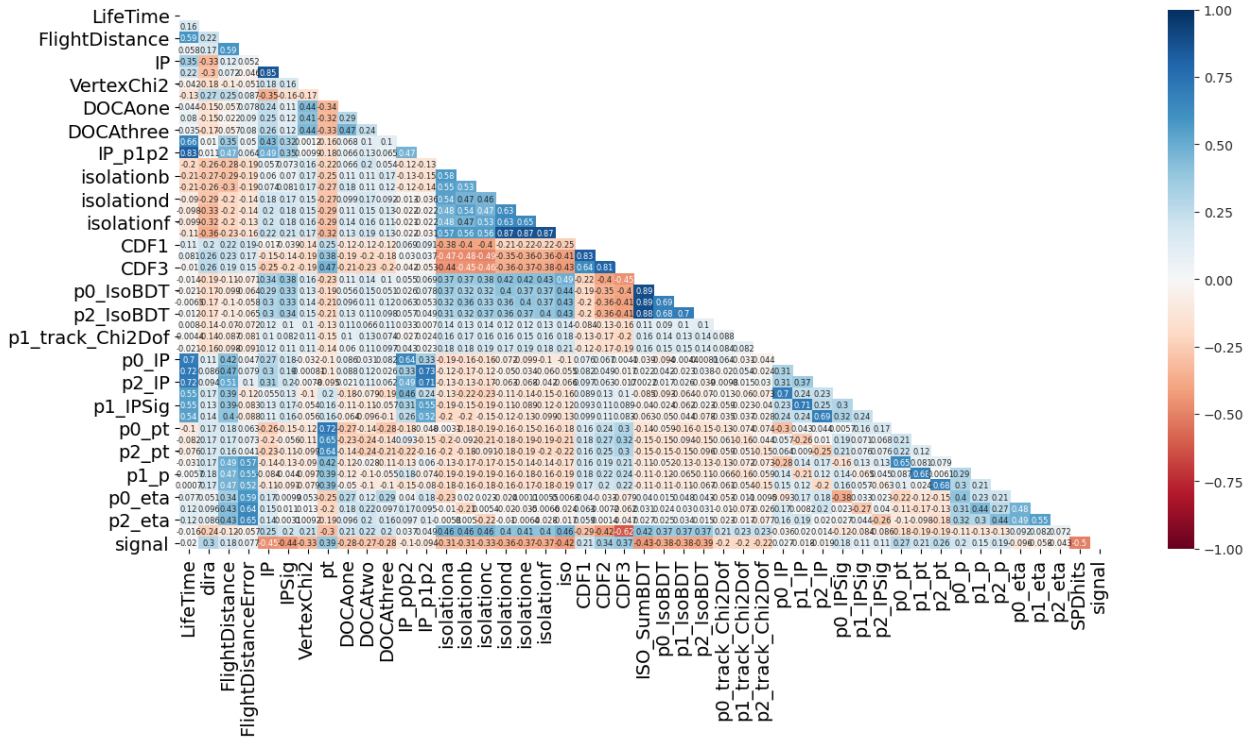
The next step of the data analysis was to take a closer look at the features. A plot of the the distribution of each attribute is presented below.




Most variables have a skewed distribution, so the dataset provides a good candidate for scaling the data, for example, with the `sklearn` robust scaler transform to standardize the data in the presence of skewed distributions and outliers.

### Features Correlation

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 the data were scaled with the `sklearn` robust scaler transform before calculating the correlations between the features. The highest coefficients are below 0.9, and over 0.8 for only 4 features, so I decided to include all of them in the training process.




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

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

## Algorithms Implementation

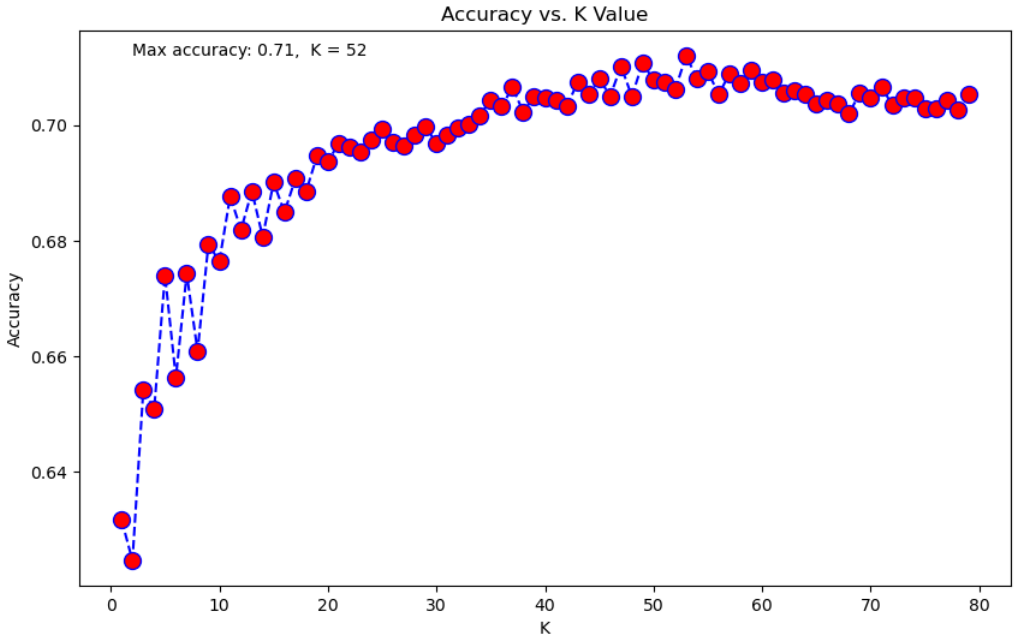
This is a supervised binary classification problem, so the solution will be the output of a binary classifier. There is no constrain in using any classifier in particular for this problem. However, in the evaluation description of the Kaggle competition described so far, it is mentioned that the Kolmogorov–Smirnov (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 for different models, in combination with different data scaling methods. For all the results, a number of performance metrics are gathered in a single table, and the best model is chosen based on the metrics values. More information on the metrics are described in the *Models Performance* section.

### Benchmark Model

As part of the project proposal, I trained the 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 classifier with k=52, and highest accuracy of 71%.




### 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, and a transform that could improve the results, as shown in the data analysis section presented above, I have also scaled the data in three different ways. An example of its importance is a Support Vector Machines model (one of the models trained), 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. The scaling methods used are: `sklearn`'s Standard Scaling, `sklearn`'s Minmax Scaling and `sklearn`'s Robust Scaling.

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

#### Binary Classifiers trained

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

The models used are:

- **kNN tuned with the Grid Search Method**

The data point ot be classified is compared to its nearest points and classified based on which points it is closest and most similar to. For this model, the Grid Search method for hyperparameter 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**

SGD randomly picks data points from the dataset and at each step calculates the derivatives based on which the loss function is minimized. Works 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). The model was trained with Random search for better performance and speed

- **Support Vector Machines tuned with Random Search**

SVM finds a hyperplane in an N-dimensional space that distinctly classifies the data points. 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 optimized with Bayes optimization**

XGBoost is a more complicated algorithm than the ones described so far, and is designed to make the best use of available resources to train the model. It is an ensemble technique where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It uses a gradient descent algorithm to minimize the loss when adding new models. For the hyperparameter tuning part, skopt Bayes Search was used, that utilizes a fixed number of parameters, sampled from a specified distribution. The first optimization maximizes the classifier function (XGBoost in this case) and gives the user the control of the steps of the bayesian optimization and the steps of the random exploration that can be performed. The XGBoost model trained with scaled data using the robust scaler is the one I chose based on the performance metrics analysis presented in the next section. A seperate script that implements XGBoost and returns the predicted *signal* values can be found here. You can also refer to it for the full range of the parameters used for the tuning.

The best parameters of all the models are saved, along with their corresponding performance metrics calculated. You can find them all here.

## Models' Performance

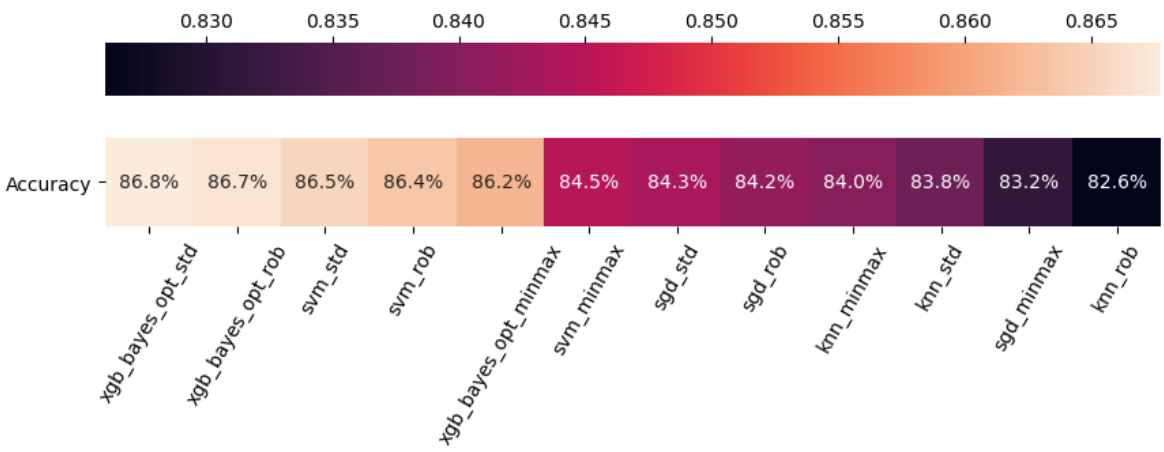
According to the original kaggle problem evaluation instructions described in the problem definition section, the KS test would be the metric based on which the final model would be assessed against. The KS test value has to be lower than 0.09 for the model to be considered "good enough" to pass. All the models trained (except from the benchmark model) are below 0.09, so I considered more performance metrics to rank the models based on their performance.

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 predicting true positives or negatives well.

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 performing model, so AUC was used to decide which is the best parameters, 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* and *F1*.

You can refer to the metrics results for all the models' metrics values. The results include 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 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.

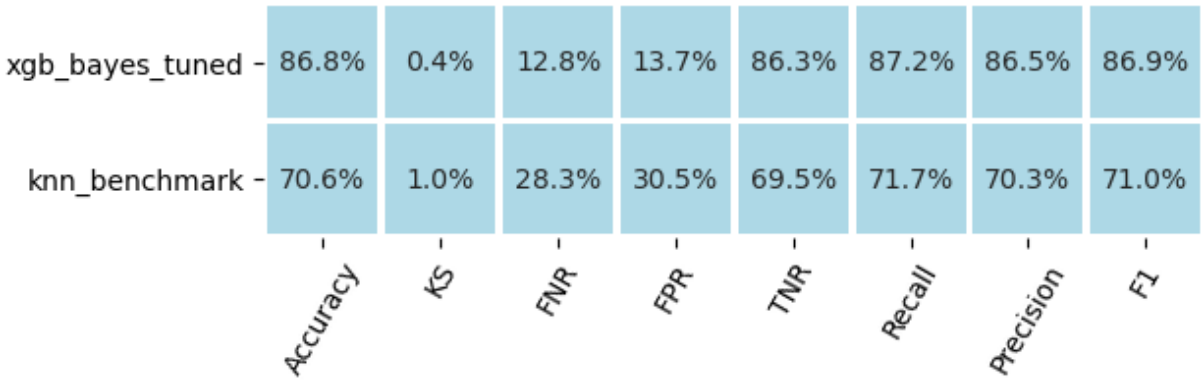
## Conclusion

The XGBoost model is the model I choose to solve the $D_s \rightarrow \varphi\pi$ problem. The KS test is below 0.09, so according to the kaggle competition description, it passes the test for approval.

Since the dataset is balanced, it was safe enough to use a simple metrics as accuracy based on which I would pick the right model. I also made sure I consider metrics that asses true or false positive or nagetive values. The accuracy of the XGBoost model chosen is 87%, which I believe is good enough for solving this problem, and the model can also predict true positive and negative values well.

**Comparison of models**

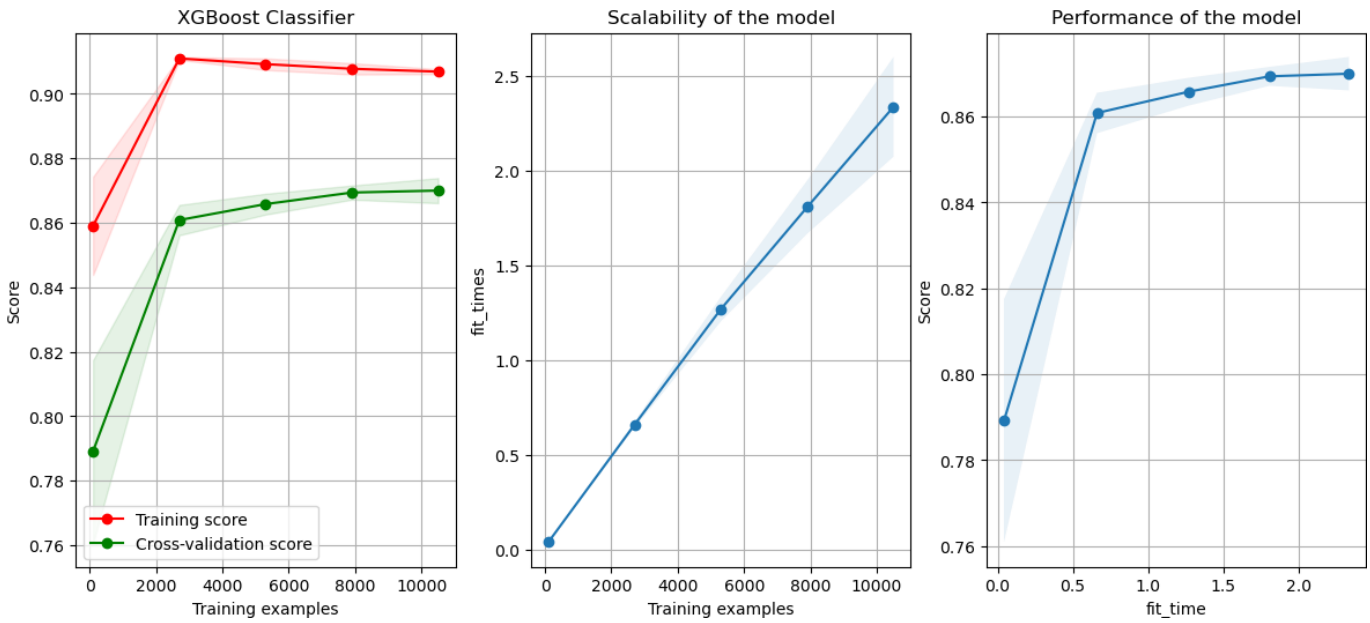
A comparison of the XGBoost model with the benchmark model follows.




The accuracy of the benchmark model is low, and the model does not pass the acceptance tests, as its KS test is below 9%. Considering all the models used, I believe the greatest improvement in performance came from scaling the data, as the majority of the features were unevenly distributed. Even the kNN models trained with scaled data, produced results with improvement in accurasy of at least 12% (data scaled with the robust scaler) in comparison to the kNN benchmark model.

**Future improvements on the model**

The following suggestions concern future improvements on the chosen XGBoost model. To have a deeper understanding of how well the model can learn, I used a learning curve shown below.



The first plot is the learning curve. It shows the relationship between the training examples size and the training and validation sets' accuracy score. In this plot, the training accuracy score is higher as expected, and both sets' accuracy are improving to a point of stability. The gap between the two lines is also low, so the model is generally a good fit. An higher accuracy score on the validation set could maybe be achieved by increasing the number of training samples, so this is something to consider in future improvements of this model.

The second plot shows the times required by the models to train with various sizes of training datasets. The third plot shows how much time was required to train the models for each training size.

As mentioned above, a higher number of training samples might improve the performance of the validation set, so a future improvement could be to use a bigger part of the original dataset, and transform the dataset into a less imbalanced one. An alternative to accuracy performance metrics could be essential to choose the model with the best fit.

Other areas to explore:

- Try adding/creating more features.
- Improve feature selection using different selection methods to get a smaller, but better combination of features.
- Use model stacking. A combination of weaker learners could result a high performing model.

## Project Proposal

You can find here the submission of the project proposal.

## Sources:


- http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/
- http://cds.cern.ch/record/2668282/files/BPH-17-004-pas.pdf
- https://cds.cern.ch/record/2196092/files/CERN-THESIS-2016-064.pdf
- https://cds.cern.ch/record/2713513?ln=en
- https://en.wikipedia.org/wiki/Accuracy_paradox
- https://en.wikipedia.org/wiki/D_meson
- https://en.wikipedia.org/wiki/Flavour_(particle_physics)
- https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test
- https://en.wikipedia.org/wiki/Particle_decay
- https://en.wikipedia.org/wiki/Phi_meson
- https://en.wikipedia.org/wiki/Pion
- https://en.wikipedia.org/wiki/Precision_and_recall
- https://en.wikipedia.org/wiki/Standard_Model

- https://github.com/Punchyou/blog-binary-classification-metrics
- https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html
- https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/
- https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/
- https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/
- https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/
- https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/
- https://machinelearningmastery.com/quick-and-dirty-data-analysis-with-pandas/
- https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/
- https://machinelearningmastery.com/types-of-classification-in-machine-learning/
- https://medium.com/analytics-vidhya/feature-selection-techniques-2614b3b7efcd
- https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26
- https://neptune.ai/blog/evaluation-metrics-binary-classification
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- https://scikit-learn.org/stable/modules/sgd.html
- https://scikit-learn.org/stable/modules/svm.html
- https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html
- https://towardsdatascience.com/a-guide-to-svm-parameter-tuning-8bfe6b8a452c

- https://towardsdatascience.com/feature-selection-correlation-and-p-value-da8921bfb3cf
- https://towardsdatascience.com/how-to-find-the-optimal-value-of-k-in-knn-35d936e554eb
- https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
- https://towardsdatascience.com/understand-your-data-with-principle-component-analysis-pca-and-discover-underlying-patterns-d6cadb020939
- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://wiki.classe.cornell.edu/pub/People/AndersRyd/DHadRMP.pdf
- https://www.cs.cmu.edu/~kdeng/thesis/feature.pdf
- https://www.dataquest.io/blog/learning-curves-machine-learning/
- https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python
- https://www.kaggle.com/c/flavours-of-physics/overview
- https://www.kaggle.com/nanomathias/bayesian-optimization-of-xgboost-lb-0-9769
- https://www.youtube.com/watch?v=aXpsCyXXMJE
- https://xgboost.readthedocs.io/en/latest/index.html

Books:

- https://www.oreilly.com/library/view/data-science-from/9781491901410/
- https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/
- https://www.oreilly.com/library/view/hands-on-machine-learning/9781491962282/