ML ASSIGNMENT 3

OMKAR BHANDARE (22CS30016)

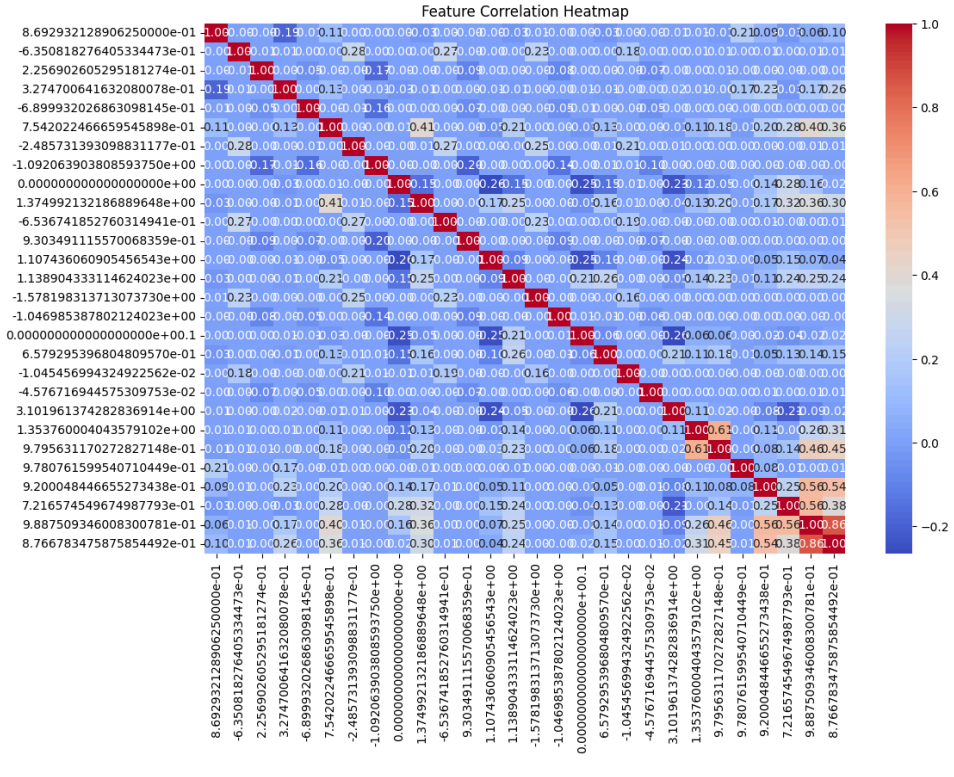
Part A: Support Vector Machines (SVMs) and Kernel Methods - HIGGS Dataset

Introduction

The data has been produced using Monte Carlo simulations. The first 21 features are kinematic properties measured by the particle detectors in the accelerator. The remnant of the 7 features are the functions of the first 21 features; these are high-level features derived by the domain experts to help discriminate between the two classes.
The objective of the assignment is to build a Support Vector Machine (SVM) classifier to predict the classes of the dataset. Since the dataset is large and high dimensional, the assignment expects efficient data handling, advanced feature selection, and model tuning from us.
Note that Accuracy has been used as the primary metric for evaluation throughout.

Exploratory Data Analysis

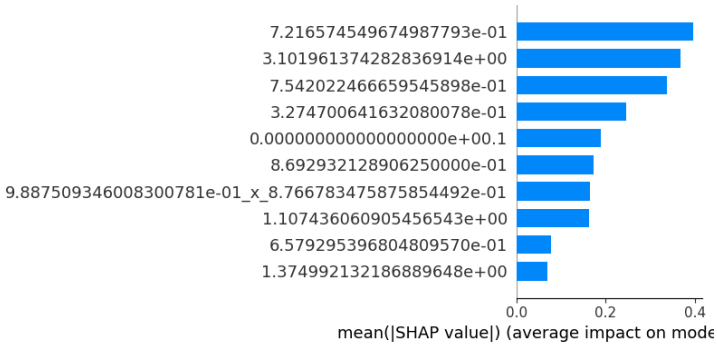
Histogram plots with KDE were used to analyse the distribution of the features (all the graphs can be seen in the submitted Jupyter Notebook).
The NA values in the dataset were observed, but none were found.
The columns were standardised using the StandardScaler of the sklearn. It helped standardise features to have a balanced scale. (As I dug down in depth into the dataset, it was found that in the original paper where the dataset was proposed, the features were already standardised, and the authors had recommended few methods for further normalisation)
Next, outliers were detected using the Z-score method; a higher threshold of 4 was kept to eliminate the “far trouble-causing” outliers. The outliers were detected feature-wise, and then respective data points were removed from the dataset; this method of eliminating the outliers helps in a more nuanced understanding of the dataset, thus helping for a better approach. The summary of outliers feature-wise has been included in the code itself.

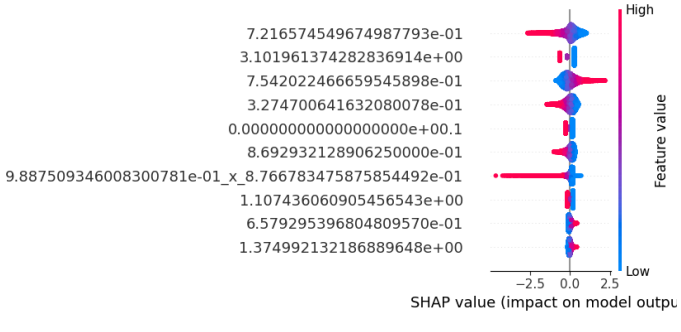


Since I do not carry any domain knowledge, developing exquisite features based on the combination of the available features and their study was not possible. However, some feature engineering techniques which do not require domain knowledge were tried. Out of those, one stood out. The correlation matrix was plotted to observe the closely related features, and a reasonably good threshold was set to consider the correlated pairs. Now, for the correlated pairs, a new feature that contained the product of values of the correlated pairs was developed, and the original features were removed. By doing this, in essence, we eliminated the problem of multicollinearity in the data by removing the correlated features; at the same time, we retained the information that the feature essentially held by taking the product. This method was adopted because after trying out the polynomial feature generation, this yielded better results than those. Since feature engineering is a vast study area, many more techniques can be explored there.
For feature selection, I used the SeleckKBest feature selection method from the sklearn, with the f_classif score function. SelectKBest selects the top K features based on a scoring function; the f_classif function calculates the ANOVA F-value between the feature and the target variable, which is an indicator of correlation between the feature and the target variable (the higher the value, the better the correlation). ANOVA F is a statistical measure to assess the differences between the group means; it is the ratio of two means, namely, between-group variance and within-group variance.

Linear SVM Implementation

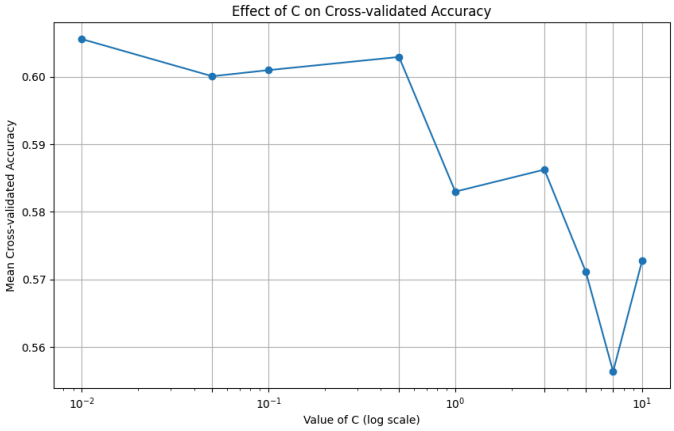
The SVC class of the scikit learn was very computationally inefficient; it took way longer for the model to train since our dataset was large. We needed to optimise the convergence of the SVM loss. One way out was to use Stochastic Gradient Descent or Mini Batch Gradient Descent. After reading a few articles on how to implement SVM with linear kernel efficiently, I got to know SGD class with hinge loss is equivalent to SVM with linear kernel; https://scikit-learn.org/1.5/modules/sgd.html, https://en.wikipedia.org/wiki/Hinge_loss. So, sticking to this fact, I used the SGD. Please note that using the original SVC class to implement the SVM of the desired kernel is always an option. Still, since it is inefficient for a large dataset like the one we are using, it is always good to retort to some efficient implementations. The hyperparameter C was tuned further to get better results. A particular relationship between the alpha of SGD and hyperparameter C has been established according to https://stats.stackexchange.com/questions/216095/how-does-alpha-relate-to-c-in-scikit-learns-sgdclassifier, the relation has been used everywhere in the code without explicitly stating.
Apart from this, SHAP (SHapley Additive exPlanations) analysis was done to explain the model's predictions and assess the importance of the most influential features.





SHAP (SHapley Additive exPlanations) is a game-theoretic approach to explain the output of any machine learning model. It assigns importance to each feature, indicating its contribution to the model's

prediction. It helps us understand the model's decision-making process, identify biases, and build more robust models.

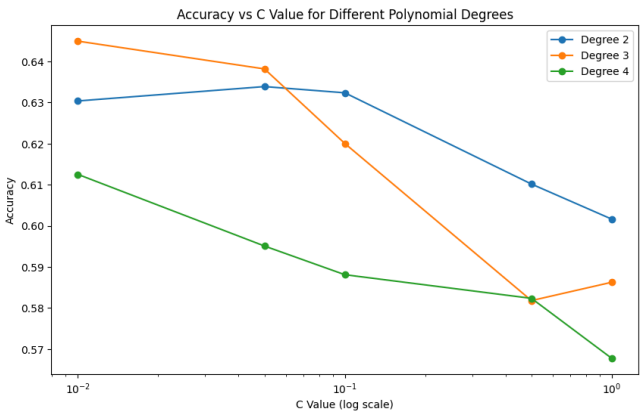


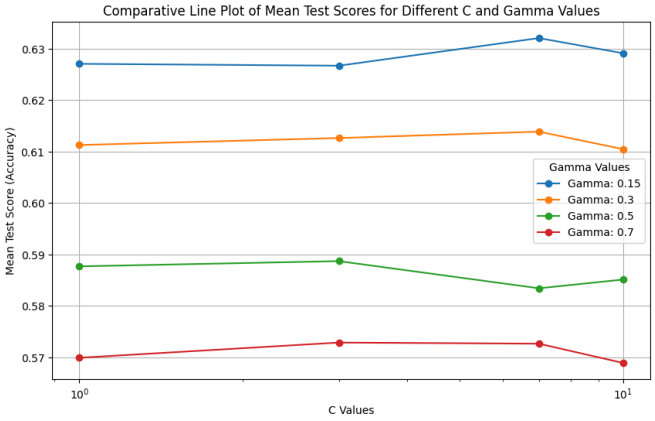
SVM with Kernels

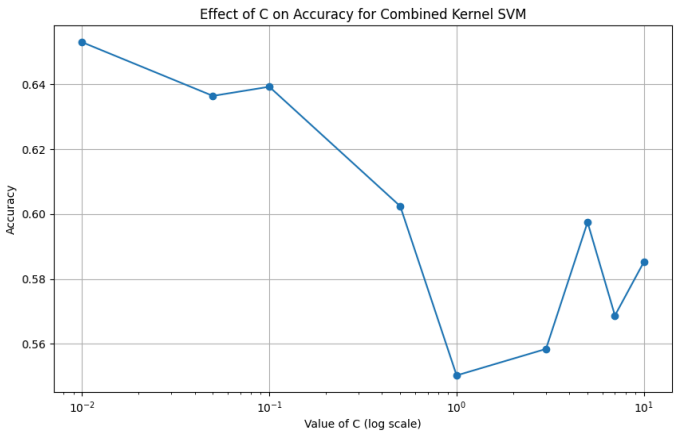
Polynomial, RBF and custom kernels were implemented to explore SVMs with kernels. In each case, the original SVC class from the scikit learn library was inefficient from the implementation point of view. Since it took a lot of time to train and test the models, it was needed to retort to the SGD class; in each case, polynomial features, RBFSampler, and a customised pipeline for feature transformation were used to approximate the kernel behaviour. Note that, since the SVC class was highly inefficient for the given dataset, and we also had limited computational resources, I used sampling and then SGD for approximating the kernels. We can use the SVC class if we want the exact kernel behaviour, but it will consume time.

All the required parameters were tuned for each kernel using the GridSearchCV (except for the custom kernel). Limited values of hyperparameters were tested because of computational restrictions(The laptop hung when too many parameters were tested; also, sometimes it reported the 'cannot allocate memory' error, so I limited myself to only a few values of hyperparameters).

SHAP plots are also reported at the end of the implementation of each kernel.







Custom Kernel: I used a combination of RBF kernel and polynomial kernel of degree 3 for the custom kernel. These two had the highest accuracies in the previous results, so I decided to combine them to get better results. For this, I used the RBFSampler in combination with the PolynomialFeatures to get the data to scale up to the custom kernel, and then I used the SGDClassifier to apply the SVM.

Time Complexity Analysis

For linear SVM using SGD classifier, the time complexity can be approximated to O(T.n.d), where T = number of iterations, n = number of samples, d = number of features

For a polynomial kernel, the complexity can be approximated to O(T.n.d(poly)), where T = number of iterations, n = number of samples, and d(poly) represents the dimensionality after polynomial feature expansion, for a degree x polynomial, the d(x) can be approximated to (d+x, x) {(n, r) stands for n choose x}.

The RBF kernel's complexity can be approximated to O(T.n.d(rbf)), where; T = number of iterations, n = number of samples, d(rbf) = number of components used in RBFSampler

For the custom kernel used in the code, the complexity of the kernel can be approximated to O(T.n.(d(poly) + d(rbf))), since it uses a linear combination of the RBF and polynomial kernel of some degree.

Summary

Kernel	**Max. Accuracy Achieved (after tuning)**
Linear	61.00 %
Polynomial	65.03 %
RBF	63.67 %
Custom (RBF + Polynomial)	65.29 %

As we can see, the custom kernel gives a greater accuracy than the rest. Thus, a linear combination of RBF and polynomial kernel is more robust than the given dataset. There is a scope for improving this custom kernel by performing exhaustive parameter tuning. Currently, only C has been tuned against the polynomial of degree 3 and gamma = 0.15; further, it can be extended for varied degrees, gamma and values of C. (couldn't be done due to computational limitations on the local setup).

Terminologies and their Meanings

Accuracy: Accuracy is the ratio of correctly predicted observations to total observations. It is a straightforward measure that indicates the overall correctness of the model.
Accuracy = Number of correct predictions / Total number of predictions

Precision: Precision (also called Positive Predictive Value) measures the proportion of correctly predicted positive observations to the total predicted positive observations.
Precision = (TP) / (TP + FP)

Recall: Also known as Sensitivity or True Positive Rate, Recall measures the proportion of correctly predicted positive observations to all observations that are actually positive.
Recall = (TP) / (TP + FN)

F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between the two metrics and is especially useful when there is an uneven class distribution.
F1-Score = 2 x ((Precision x Recall) / (Precision + Recall))

ROC-AUC: AUC represents the degree or measure of separability and tells how well the model can distinguish between classes. It is the area under the Receiver Operating Characteristic (ROC) curve, which plots the True Positive Rate (Recall) against the False Positive Rate (1 - Specificity). More information on ROC can be found here. A higher AUC value indicates a better-performing model that effectively distinguishes between positive and negative classes.

TP (True Positive): The number of cases where the model correctly predicted the positive class.

FP (False Positive): The number of cases where the model incorrectly predicted the positive class when it was negative.

TN (True Negative): The number of cases where the model correctly predicted the negative class.

FN (False Negative): The number of cases where the model incorrectly predicted the negative class when it was positive.