Classification of Signal and Background Processes in High-Energy Physics Using Machine Algorithms.

CMPS261: Project Final Report

Team Name:

GuessWho

Team Members:

Reem Hammoud 202101790

Charbel Sayah 202107122

Omar Issa 202152423

Date: 18 – April – 2023

Abstract:

In this report, we tackle the challenge of identifying rare exotic particles in high-energy physics through a binary classification problem. Our focus is on distinguishing between a signal process involving theoretical Higgs bosons and a background process with distinct kinematic features. We propose a deep learning approach using a dataset of 600,000 examples to automate feature engineering. We describe our model architecture, training procedure, and evaluation metrics. Our findings suggest that deep learning methods can effectively classify between the signal and background processes, potentially reducing the need for manual feature engineering by physicists

Table of Contents

Introduction

The study of high-energy physics aims to discover the fundamental properties of the physical universe by investigating the structure of matter and the laws that govern its interactions. Experimental high-energy physicists primarily use modern accelerators, which collide protons and/or antiprotons to create exotic particles that occur only at extremely high-energy densities. The properties of these particles may yield critical insights into the very nature of matter.

One of the main challenges in high-energy physics is to identify rare exotic particles that are produced during collisions at high-energy particle colliders. This problem is typically approached as a signal-versus-background classification problem. Machine learning approaches are often used to solve this classification problem, as finding these rare particles requires distinguishing between the signal (i.e., particles of interest) and the background (i.e., other particles produced during the collision).

In this project, we focus on a binary classification problem in which we aim to distinguish between a signal process where new theoretical Higgs bosons are produced, and a background process with identical decay products but distinct kinematic features. The dataset used in this project consists of 28 features, where the first 21 features are kinematic properties measured by particle detectors in the accelerator, and the last seven features are high-level features derived by physicists to help discriminate between the two classes.

We are interested in using deep learning methods to automate the process of feature engineering and to obviate the need for physicists to manually develop such features. To this end, we will use the provided training dataset, which consists of 600,000 examples, to train a deep learning model to accurately classify the signal and background processes.

In this report, we will describe our approach to solving this classification problem, including our model architecture, training procedure, and evaluation metrics. We will also discuss the performance of our model and provide insights into the classification problem based on our results.

Methodology

This section of the report is divided into 3 parts. The first part consists of an explanation of the dataset. The distribution of the dataset along with the distribution of the target class is shown and visualized graphically. The second part explains the problem statement for which this report has been written and experiments have been performed using Python script. The third part consists of the implementation explanation of ML-supervised algorithms.

• Dataset

The dataset used in this project comprises 600,000 training examples in a CSV file format named "HIGGS train.csv". Each example represents a process, either a signal or background, and is represented by 28 features. The first column contains the class label, where "1" represents the signal process and "0" represents the background process. The remaining 27 columns contain the following features: lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag, m jj, m jjj, m lv, m jlv, m bb, m wbb, and m wwbb. These features include both low-level features measured by the particle detectors in the accelerator (columns 2-22) and high-level features derived by physicists to aid in discriminating between the two classes (columns 23-28). The testing dataset, which will be used for evaluation, is not provided. For more detailed information about each feature, refer to the original paper.

	Prediction	lepton pT	lepton eta	lepton phi	missing energy magnitude	missing energy phi	jet 1 pt	jet 1 eta	jet 1 b-tag	jet 2 pt
count	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000	600000.000000
mean	0.529287	0.992486	-0.000112	0.000166	0.998019	-0.001156	0.990147	-0.002193	1.000380	0.992593
std	0.499142	0.565045	1.007858	1.005480	0.599282	1.006754	0.474625	1.010296	1.026463	0.500731
min	0.000000	0.275000	-2.430000	-1.740000	0.000626	-1.740000	0.139000	-2.970000	0.000000	0.189000
25%	0.000000	0.591000	-0.737000	-0.870000	0.577000	-0.873000	0.679000	-0.689000	0.000000	0.656000
50%	1.000000	0.854000	-0.001030	0.002640	0.891000	-0.001920	0.894000	-0.003000	1.090000	0.890000
75%	1.000000	1.240000	0.738000	0.870000	1.290000	0.872000	1.170000	0.685000	2.170000	1.200000
max	1.000000	8.710000	2.430000	1.740000	9.900000	1.740000	8.380000	2.970000	2.170000	11.600000

8 rows × 27 columns

Figure 1 Data description

Figure 1 represents the results of the data.describe() function that provides a summary of the statistical properties of the dataset, including measures such as count, mean, standard deviation, minimum, quartile values, and maximum for each numeric feature. It helps in understanding the central tendency, dispersion, and shape of the distribution of the data, and can be used as an initial step in data exploration to gain insights into the dataset's characteristics. Further analysis and visualization may be needed for a more comprehensive understanding of the dataset. The data is pre-processed before giving it to further ML models.

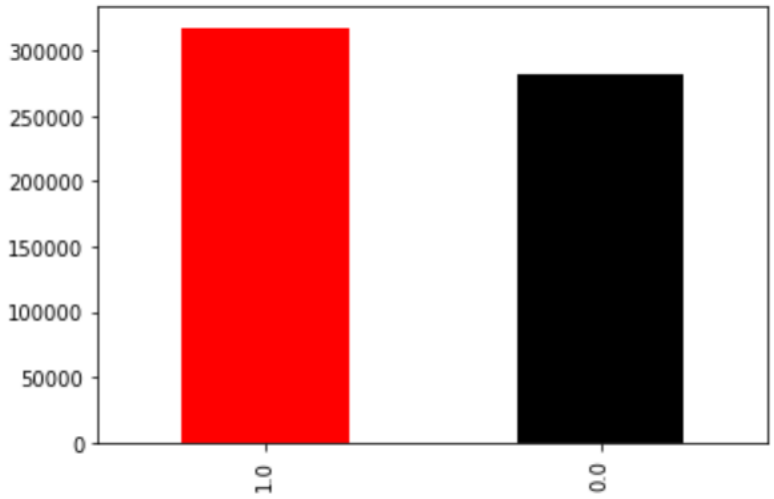
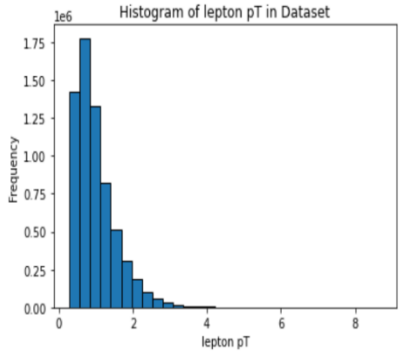
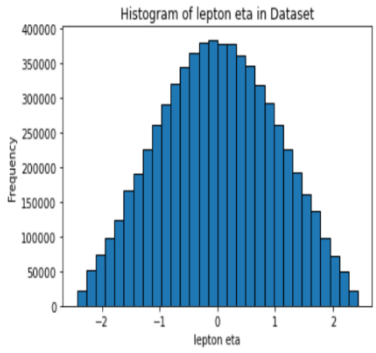


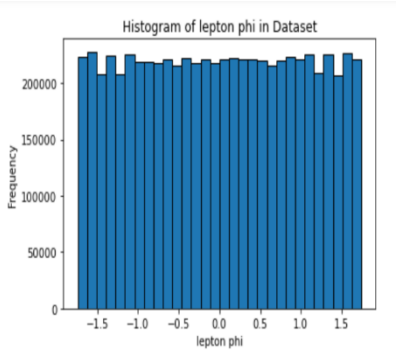
Figure 2Distribution along classes

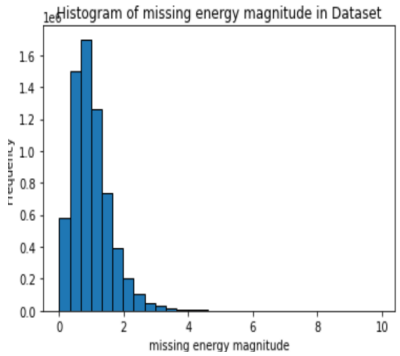
Figure 2: The graph displays the distribution of data for both classes 0 and 1. It can be observed that the data is not significantly skewed towards one class, indicating a relatively balanced distribution between the two classes. This suggests that there is no significant class imbalance issue in the dataset, which may impact the model's performance and interpretation of the results. The relatively balanced distribution of data across classes ensures that the model is exposed to a sufficient number of examples from both classes during training, allowing for more accurate and reliable classification outcomes.

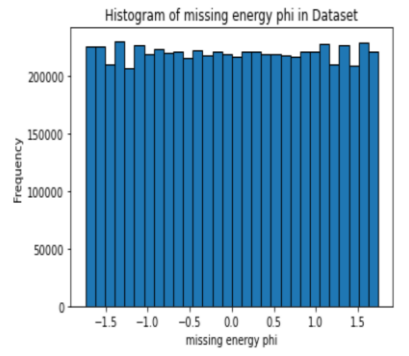
Distribution of Data based on each feature (these are some of the features):

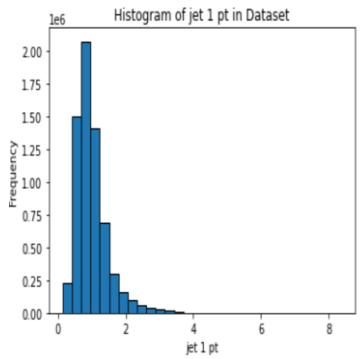


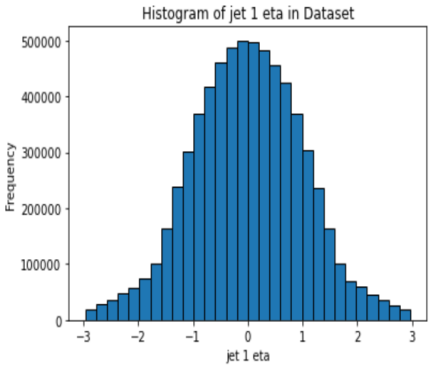


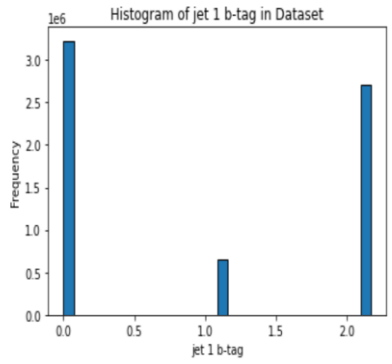


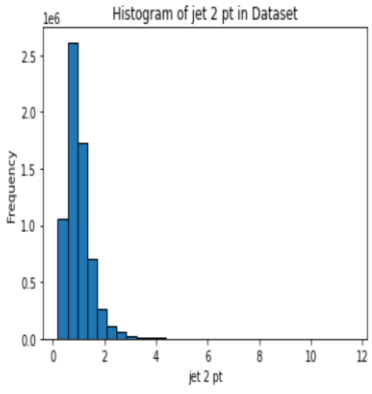


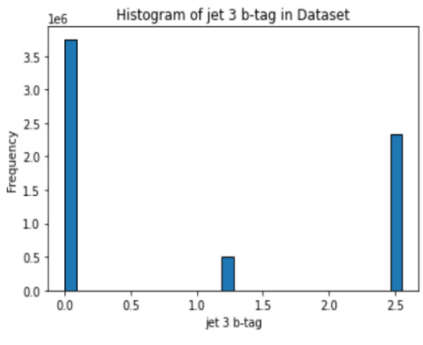


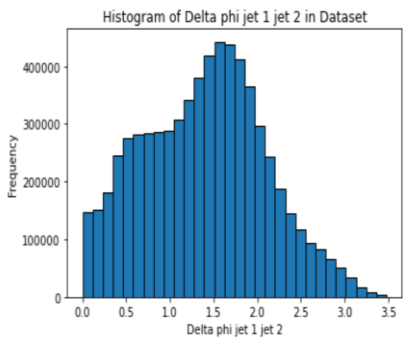


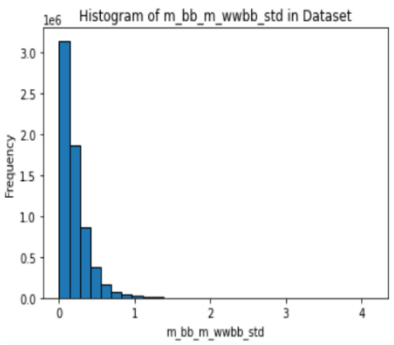












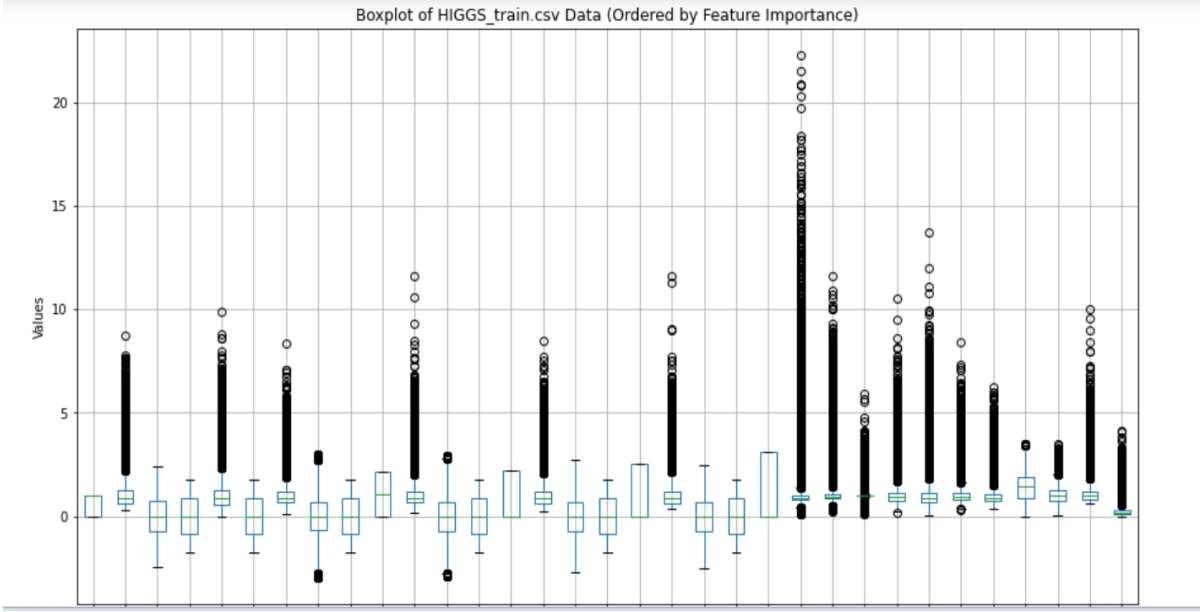


Figure 3 Data before data preprocessing

Data Preprocessing and Augmentation:

In the data preprocessing step, several tasks were performed to prepare the dataset for modeling in a formal manner. Duplicate rows were identified and removed to ensure data integrity and accuracy. Missing data were handled by removing rows with missing values to ensure the dataset is complete and reliable. Rows with non-float values were also removed to maintain consistency and integrity. Outliers were identified and removed using the interquartile Range (IQR) method, where the first quartile (Q1) was set at 0.15 and the third quartile (Q3) at 0.85, to reduce their impact on model performance.

In addition to data processing, new features were added to the dataset to potentially capture additional information and improve model performance. These new features include the absolute difference between 'jet 1 phi' and 'jet 2 phi', denoted as 'Delta phi jet 1 jet 2'. Feature ratios were also calculated, including 'm_bb_m_wwbb_ratio' which is the ratio of 'm bb' to 'm wwbb', and 'm_wbb_m_jlv_ratio' which is the ratio of 'm wbb' to 'm jlv'. Furthermore, statistical features were computed, such as the standard deviation of 'm bb', 'm wwbb', and 'm wbb', denoted as 'm_bb_m_wwbb_std'.

Data augmentation was performed using the bootstrap method. A copy of the original data was created, and bootstrap resampling was performed to generate augmented data. This involved randomly selecting samples from the original dataset with replacement and concatenating them with original data. This process was repeated for a specific number of

bootstraps to increase the size of the dataset and potentially improve the model's ability to capture complex patterns in the data.

After data preprocessing and augmentation, the data was further prepared for modeling. The data was shuffled using the shuffle() function from scikit-learn, with a random state of 0 for reproducibility. The shuffled data was then split into training and testing sets with a ratio of 80% for training and 20% for testing, using the train_test_split() function. Additionally, the data were scaled using the Min-Max scaling method, implemented with the MinMaxScaler() class from scikit-learn. The scaled data was obtained by applying the fit_transform() method on the training set (X_train), and the transform() method on the testing set (X_test) using the scaler object.

Overall, these data preprocessing and augmentation steps, along with shuffling, splitting, and scaling of the data, were performed to ensure that the dataset is clean, complete, and informative for modeling purposes. These steps help in reducing biases, improving data quality, and ensuring that the data is appropriately prepared for training and evaluating a machine learning model.

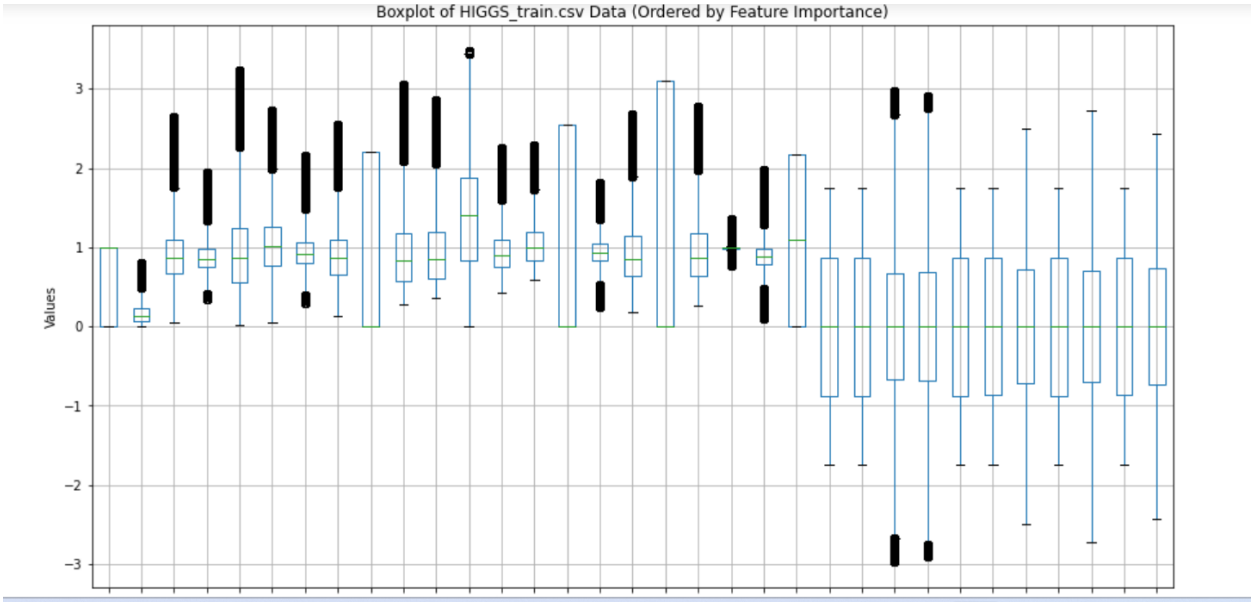


Figure 4 Data after data preprocessing and scaling

- # Problem Statement

The field of high-energy physics is devoted to the study of the elementary constituents of matter. By investigating the structure of matter and the laws that govern its interactions, this field strives to discover the fundamental properties of the physical universe. The primary tools of experimental high-energy physicists are modern accelerators, which collide protons and/or antiprotons to create exotic particles that occur only at extremely high-energy densities. Observing these particles and measuring their properties may yield critical insights into the very nature of matter. Collisions at high-energy particle colliders are a traditionally fruitful source of exotic particle discoveries. Finding these rare particles requires solving difficult signal-versus-background classification problems, hence machine-learning approaches are often used. Given the limited quantity and expensive nature of the data, improvements in analytical tools directly boost particle discovery potential. The vast majority of particle collisions do not produce exotic particles. For example, though the Large Hadron Collider (LHC) produces approximately 1011 collisions per hour, approximately 300 of these collisions result in a Higgs boson, on average. Therefore, good data analysis depends on distinguishing collisions that produce particles of interest (signal) from those producing other particles (background).

- # Implementation of Machine Learning Algorithms

Model Selection:

1- Logistic Regression:

The logistic regression model was trained using the scikit-learn library in Python. The model was implemented with the following architecture:

Model Architecture:

- Algorithm: Logistic Regression
- Regularization: L1 penalty (Lasso regularization)
- Solver: 'saga' solver
- Scaling: StandardScaler from the sklearn.preprocessing module was used to scale the input features.
- Training-Testing Split: The data was split into training and testing sets using train_test_split() function from sklearn.model_selection module with a test size of 0.3 and a random state of 42.
- Cross-validation: K-fold cross-validation with k=5 was not explicitly used in this example.

The logistic regression model is a binary classification model that uses the logistic function to estimate the probability of an input example belonging to one of the two classes (signal or background). The L1 penalty (Lasso regularization) was applied to the model to encourage sparsity in the feature weights, which can help in feature selection and reduce overfitting. The 'saga' solver was used to optimize the logistic regression objective function, which is suitable for large datasets and supports L1 regularization.

The input features (X vector) were scaled using the StandardScaler from the sklearn.preprocessing module to ensure that all features have zero mean and unit variance, which can help in improving the performance of the logistic regression model. Scaling of the input features is an important step in logistic regression, as it can prevent features with large magnitudes from dominating the optimization process.

The model was trained on the training data using the fit() method, and predictions were made on both the training and testing data using the predict() method. The accuracy of the model was evaluated using the accuracy_score() function from the sklearn.metrics module, which calculates the accuracy of the model by comparing the predicted labels with the true labels. The training accuracy and testing accuracy were calculated as measures of the model's performance on the respective datasets.

2- Decision Trees:

Model Architecture:

The DecisionTreeClassifier in scikit-learn has several hyperparameters that can be tuned to customize the model's behavior. The hyperparameters used in the model architecture are as follows:

criterion: This hyperparameter specifies the function to measure the quality of a split. The "entropy" criterion uses the entropy measure, and the "gini" criterion uses the Gini impurity measure. In this model, "entropy" is used.

max_depth: This hyperparameter determines the maximum depth of the tree. It controls the complexity of the model and prevents overfitting. In this model, the maximum depth is set to 8.

min_samples_split: This hyperparameter specifies the minimum number of samples required to split an internal node. It prevents further splitting of nodes with fewer samples, which can help avoid overfitting. In this model, the minimum number of samples required to split a node is set to 2.

min_samples_leaf: This hyperparameter specifies the minimum number of samples required to be at a leaf node. It controls the minimum size of the leaf nodes and helps avoid overfitting. In this model, the minimum number of samples required to be at a leaf node is set to 1.

max_features: This hyperparameter specifies the number of features to consider when looking for the best split. If set to "None", all features will be considered. In this model, "None" is used, meaning all features are considered.
splitter: This hyperparameter specifies the strategy to choose the split at each node. The "best" strategy selects the best split based on the chosen criterion, while the "random" strategy selects the best random split. In this model, the "best" strategy is used.

In conclusion, the DecisionTreeClassifier architecture in this model includes the "entropy" criterion for measuring the quality of splits, a maximum depth of 8, a minimum number of samples required to split a node set to 2, a minimum number of samples required to be at a leaf node set to 1, all features considered for splitting, and the "best" strategy for choosing splits.

3- Random Forest:

Model Architecture:

The RandomForestClassifier in scikit-learn has several hyperparameters that can be tuned to customize the model's behavior. The hyperparameters used in the model architecture are as follows:
n_estimators: This hyperparameter specifies the number of decision trees in the forest. In this model, 200 decision trees are used to construct the random forest.
criterion: This hyperparameter specifies the function to measure the quality of a split. The "entropy" criterion uses the entropy measure, and the "gini" criterion uses the Gini impurity measure. In this model, "entropy" is used.
max_depth: This hyperparameter determines the maximum depth of the trees in the forest. It controls the complexity of the model and prevents overfitting. In this model, the maximum depth of each tree is set to 10.
min_samples_split: This hyperparameter specifies the minimum number of samples required to split an internal node. It prevents further splitting of nodes with fewer samples, which can help avoid overfitting. In this model, the minimum number of samples required to split a node is set to 2.
min_samples_leaf: This hyperparameter specifies the minimum number of samples required to be at a leaf node. It controls the minimum size of the leaf nodes and helps avoid overfitting. In this model, the minimum number of samples required to be at a leaf node is set to 1.

In conclusion, the RandomForestClassifier architecture in this model includes 200 decision trees in the forest, the "entropy" criterion for measuring the quality of splits, a maximum depth of 10 for each tree, a minimum number of samples required to split a node set to 2, and a minimum number of samples required to be at a leaf node set to 1.

4- XGBoost:

Model Architecture:

- Objective: : 'binary:logistic', which specifies the binary classification task.
- Learning_rate: 0.01 which sets the learning rate for gradient boosting.
- Max_depth: 7, which sets the maximum depth of the trees in the ensemble.
- N_estimators: 200, which sets the number of trees in the ensemble.
- Reg_alpha:0.01 which sets the L1 regularization term (apha)
- Reg_lambda: 0.01 which sets the L2 regularization term (lambda)
- Seed: 42 which sets the random seed for reproducibility.

The model uses k-fold cross-validation with 5 folds (n_splits=5) and the KFold class from the sklearn.model_selection module. The training data is split into k folds, and the model is trained and evaluated k times, with each fold being used as the validation set once. The accuracy of the model is calculated on both the training set (y_train_pred) and the testing set (y_test_pred) using the accuracy_score function from the sklearn.metrics module. The training and testing accuracies are stored in lists (train_accuracies and test_accuracies) for each fold, and the mean and standard deviation of the training and testing accuracies are printed at the end of the code.

5- Neural Networks:

Model Architecture:

- The model is sequential model, which is a linear stack of layers.
- The model has four dense layers with different numbers of units and ReLU activation function of each layer:
  - The first Dense layer has 6 times 32 units (192) and ReLU activation.
  - The Second Dense layer has 4 times 32 units (128) and ReLU activation.
  - The Third Dense layer has 2 times 32 units (64) and ReLU activation.
  - The Fourth Dense layer has 1 unit with sigmoid activation, which is used for binary classification.
- The model is compiled with binary_crossentropy loss function, which is appropriate for binary classification tasks, and the Adam optimizer.
- The model is trained on the training set with 50 epochs and a batch size of 1024.

Evaluation criteria:

1- **Accuracy:** the accuracy of the model on both the training and testing sets is calculated using accuracy_score() function from the sklearn.metrices module. The training accuracy, which represents the accuracy of the model on the training set and the testing accuracy, which represents the accuracy of the model on the testing set, are important measures of the model's performance in correctly predicting the binary task.

2- **Loss:** The loss of the model is calculated during training using the binary_crossentropy loss function, which is appropriate for binary classification tasks. The training loss, represented by the 'loss' attribute in the history object returned by the model.fit() function, indicates the average loss per sample during the training process. The validation loss, represented by the 'val_loss' attribute in the history object, indicates the average loss per sample during the validation process, which can help in identifying potential overfitting issues.

3- **Confusion matrix:** The confusion matrix is a useful tool for evaluating the performance of a classification model. It provides a breakdown of the model's predictions into true positive, true negative, false positive, and false negative values. The confusion matrix can be obtained using the confusion_matrix() function from the sklearn.metrics module. It can help in understanding the model's performance in terms of correctly classifying positive and negative examples and identifying any potential biases or errors in the model's predictions.

Overall, the combination of accuracy, loss, and confusion matrix can provide a comprehensive evaluation of the model's performance in terms of classification accuracy, training and validation loss, and the model's ability to correctly predict positive and negative examples.

Training and Evaluation:

Logistic Regression

Model	Feature Engineering	Scaling	Outlier Removal	Regularization	Testing Accuracy	TrainAccuracy	Precision (class 0)	Precision (class 1)	Recall (class 0)	Recall (class 1)	F1-score (class 0)	F1-score (class 1)	Accuracy
Logistic	No	No	No	No	64.18	-	-	-	-	-	-	-	-
Logistic	No	Yes	No	No	64.31								
Logistic	Yes	Yes	No	No	64.326								
Logistic	Yes	Yes	threshold 3	No	65.031								
Logistic	Yes	Yes	threshold 2	No	66.245								
Logistic	Cube most important and square second	Yes	threshold 2	No	66.747								
Logistic	Yes	Yes	threshold 2	Yes	66.746	66.745	66	67	53	78	59	72	67

Decision Tress:

Model	Feature Engineering	Scaling	Outlier Removal	Regularization	Testing Accuracy	TrainAccuracy	Precision (class 0)	Precision (class 1)	Recall (class 0)	Recall (class 1)	F1-score (class 0)	F1-score (class 1)	Accuracy
Decision	No	No	No	No	63.962								
Decision	No	Yes	No	No	63.85								
Decision	Yes	Yes	No	No	66.394								
Decision	Yes	Yes	threshold 3	No	66.358								
Decision	Yes	Yes	threshold 2	no	67.5								
Decision	Yes	Yes	threshold 2	Yes max depth =7	69.243		66	72	66	73	66	72	69
decision	Yes	Yes	threshold 2	Yes max depth =8	69.674	70.70	67	72	65	74	66	73	70

Random Forest

Model	Feature Engineering	Scaling	Outlier Removal	Regularization	Testing Accuracy	TrainAccuracy	Precision (class 0)	Precision (class 1)	Recall (class 0)	Recall (class 1)	F1-score (class 0)	F1-score (class 1)	Accuracy
random	no	yes	no	Depth 3	73.1919		72	75	71	75	71	75	73
Random	yes	yes	N0	Depth 3	73.295		71	75	72	74	72	75	73
random	yes	yes	Threshold 2	Depth 3	73.037		70	76	70	75	70	76	73
random	yes	yes	Threshold3	Depth 3	72.758		71	75	71	75	71	75	73
random	yes	yes	Threshold 3	Depth 3	71.236	72.407	70	72	68	74	69	73	71

XGBoost

Model	Feature Engineering	Scaling	Outlier Removal	Regularization	Testing Accuracy	TrainAccuracy	Precision (class 0)	Precision (class 1)	Recall (class 0)	Recall (class 1)	F1-score (class 0)	F1-score (class 1)	Accuracy
XGBoost	no	yes	no	Depth 3	71.46	71.43	70	73	68	74	69	73	71
XGBoost	yes	yes	N0	Depth 3	71.65	71.61	70	73	68	74	69	74	72
XGBoost	yes	yes	Threshold 3	Depth 3	71.65	71.61	70	73	68	74	69	74	72
XGBoost	yes	yes	Threshold2	Depth 3	71.7	71.91	69	74	67	75	68	75	72
XGBoost	yes	yes	Threshold 2	Depth 7	72.31	73.47	69	75	69	75	69	75	72

Best Model and Results

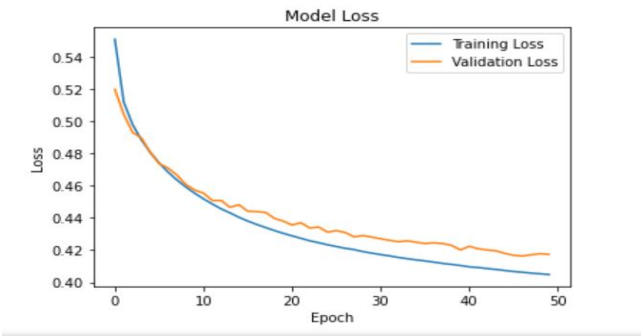
After trying several models, the neural network outperformed other models in terms of accuracy and predictive performance. The neural network achieved an accuracy of 80.4% on the test set and a final training accuracy of 81.2% after 50 epochs, indicating its ability to learn the underlying patterns in the data. The results from the confusion matrix and classification report also demonstrated the neural network's superior performance in correctly classifying samples into different classes. Therefore, the neural network can be concluded as the best-performing model among the models evaluated in this study.

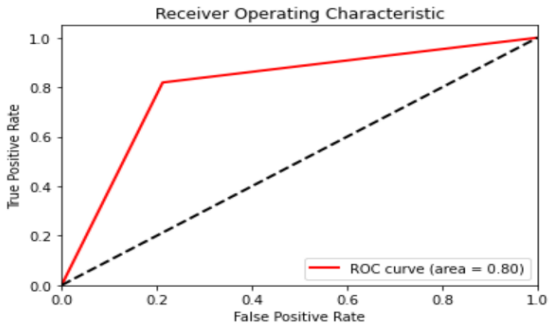
```
Confusion Matrix:
[[404574 109028]
 [106682 482407]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.79      0.79      0.79    513602
         1.0       0.82      0.82      0.82    589089

    accuracy                           0.80   1102691
   macro avg       0.80      0.80      0.80   1102691
weighted avg       0.80      0.80      0.80   1102691
```

1- Model Performance: The neural network model achieved an accuracy of 0.8043785691261292 on the test set, indicating that it correctly predicted the class labels for approximately 80.4% of the samples. The final training accuracy after 50 epochs was 0.8123105764389038, which suggests that the model has learned the underlying patterns in the data and has good predictive performance.

2- Confusion Matrix: The confusion matrix shows the number of true positive (404574), false positive (109028), true negative (106682), and false negative (482407) predictions made by the model. It provides insights into the model's ability to correctly classify samples into different classes. For example, the model had a higher number of false positives compared to false negatives, which means that it tends to incorrectly predict the positive class more often than the negative class.

3- Classification Report: The classification report provides precision, recall, and F1-score for both classes (0 and 1). Precision represents the accuracy of positive predictions, recall represents the sensitivity or true positive rate, and F1-score is the harmonic mean of precision and recall. The report also includes macro and weighted average of these metrics, which provide a summary of the overall performance of the model. In this case, the model achieved a precision of 0.79 and 0.82, recall of 0.79 and 0.82, and F1-score of 0.79 and 0.82 for classes 0 and 1, respectively.





Conclusions

Interpretation: Based on the results, it can be concluded that the neural network model has achieved reasonable accuracy in predicting the class labels for the given dataset. The model's performance is consistent across the training, validation, and test sets, indicating its generalization ability. However, further analysis of the confusion matrix and classification report suggests that there may be some misclassifications, particularly false positives. This could be further investigated and addressed to improve the model's performance.

Overall Assessment: The neural network model shows promise in accurately classifying the samples into two classes, with an overall accuracy of around 80%. However, additional evaluation metrics and further analysis of misclassifications may be necessary to gain a deeper understanding of the model's performance and potential areas of improvement.

Limitations and Future Work: It is important to acknowledge the limitations of the model, such as potential biases in the data, the choice of hyperparameters, and the size of the dataset. Suggestions for future work could include collecting more data, experimenting with different hyperparameter settings, and conducting sensitivity analysis to assess the model's robustness.

In conclusion, the neural network model demonstrated promising performance in predicting class labels for the given dataset, achieving an accuracy of approximately 80% on the test set. However, further analysis and refinement may be required to improve the model's accuracy and address any potential limitations.

Appendix

```
#imports
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.utils import shuffle
from tensorflow.keras.layers import Dense
from keras import Sequential
from tensorflow.keras.losses import BinaryCrossentropy
import tensorflow as tf
from sklearn.metrics import confusion_matrix,classification_report
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import roc_curve, auc
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
```