

**Group Members:**

* Ahmad Konainah (Section 2 with Dr. Mohamed A. Kobeissi) - (202103012)
* Mohammad El Kadri (Section 1 with Dr. Rida Assaf) - (202107920)
* Faisal Minawi (Section 1 with Dr. Rida Assaf) - (202107540)

**Group Name:**

* FAM

**Course Name:**

* CMPS 261: Machine Learning

**Semester and Year:**

* Spring 2022-2023

**Report Title:**

Solving the Higgs Boson Problem using Machine Learning

**Table of Contents**

Group Members, Name, Course & Instructors Information, Report Title ---------------- 1

Table of Contents Page -------------------------------------------------------------------------- 2

Introduction and Accuracy Metric ------------------------------------------------------------- 3

Methodology and Pre-processing steps ------------------------------------------------------- 4

Experiments Performed ------------------------------------------------------------------------- 5

Validation Method & Results and Findings -------------------------------------------------- 6

Recommendation -------------------------------------------------------------------------------- 7

Appendix 1: Logistics Regression Model analysis and results --------------------------- 8

Appendix 2: Decision Trees Model analysis and results ---------------------------------- 14

Appendix 3: Neural Network Model analysis and results -------------------------------- 20

**Introduction:**

**T**he Higgs boson is an elementary particle that plays a crucial role in the Standard Model of particle physics. The HIGGS BOSON problem aims to develop a classification model 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 goal of this problem is to improve the data analysis of particle collisions at high-energy colliders to discover exotic particles that occur only at extremely high-energy densities.

**Accuracy Metric:**

Based on the objectives of the HIGGS BOSON problem, the F1 score metric will be used to evaluate the classification model's accuracy. F1 score is a harmonic mean of precision and recall and is useful when the data is imbalanced. In this problem, the vast majority of particle collisions do not produce exotic particles, making the dataset imbalanced. Therefore, F1 score is a better accuracy metric to evaluate the model performance.

**Methodology:**

The dataset provided for the problem contains 600,000 training examples, each with 28 features. Data preprocessing involves dropping the NaN values and removing rows with inconvenient values. Automated algorithms such as Grid Search and Randomized Search are used to find the best architecture out of each model type. However, these automated algorithms require a lot of time, and lack of advanced hardware can hinder the process. Manual tuning and tweaking of models can overcome these problems, but it requires much analysis.

**Pre-processing Steps:**

The provided dataset contains 28 features, including 21 continuous and 7 categorical features, and 1 binary label (1 for Higgs boson signal, 0 for the background). The dataset also has many missing values and inconvenient rows. Therefore, before training the models, some pre-processing steps were performed on the data to ensure the models' proper training.

***The pre-processing steps include the following:***

* **Removing the NaN values:** The dataset contains many NaN values in different features.
* **Removing rows with inconvenient values:** The dataset also contains rows with -999 values, which indicate that the corresponding feature's measurement was not taken or could not be measured correctly. Also, it contains some cells with strings (‘““0.01””’) and others with the word “error” or the letter “s”.
* **Scaling the features:** The continuous features were scaled using the function from the Scikit-learn library. This function scales the data such that it has a mean of zero and a standard deviation of one.

**Experiments Performed:**

Three classification models, logistic regression, decision trees (single, random forest, and XG\_Boost), and neural network were used for this problem. The experiments performed during the training phase include the following:

* **Logistic Regression:** The logistic regression model was implemented with default hyperparameters. It generated an of .
* **Decision Trees:** Three types of decision tree models were implemented, namely single decision tree, random forest, and XG\_Boost. Grid search and randomized search were used to find the best hyperparameters for each decision tree model. For the random forest and XG\_Boost models, the number of trees was optimized using Grid search.
  + The single decision tree outputted an of .
  + The random forest outputted an of .
  + The XG\_Boost outputted an of .
* **Neural Network:** A neural network model was implemented using the Keras library with two hidden layers. Grid search and randomized search were used to find the best hyperparameters for the neural network model. The Neural Network model generated an of

**Validation Method:**

To validate the models' performance, a 5-fold cross-validation technique was used, where the dataset was divided into five equal parts. The models were trained on four parts of the data and validated on the remaining one part. This process was repeated five times, with each part being used for validation once. The F1 score was used as the evaluation metric for each fold, and the average F1 score was used to compare the models' performance. This method helps to prevent overfitting and provides a more accurate estimate of the model's performance.

**Results and Findings:**

After implementing the three classification models, we evaluated their performance using the 5-fold cross-validation technique, where the F1 score was used as the evaluation metric. The results showed that the neural network model outperformed the other models with an of . The logistic regression model and decision tree models performed relatively poorly compared to the neural network model.

The neural network model's superior performance can be attributed to its ability to learn complex, non-linear relationships between the input features and the output label. Additionally, the neural network model's ability to automatically learn feature representations without requiring extensive feature engineering is advantageous when working with high-dimensional data such as particle physics data.

Overall, the results of this study demonstrate the effectiveness of using machine learning models, particularly neural network models, to solve signal-versus-background classification problems in high-energy physics data analysis. Further exploration and analysis of the neural network model can be done to improve the model's performance.

**Recommendation:**

Based on the results, it is recommended to use a neural network model to solve signal-versus-background classification problems in high-energy physics data analysis. Further exploration and analysis of the neural network model can be done to improve the model's performance. In addition, the use of more advanced hardware and techniques such as transfer learning can potentially enhance the performance of the model.

**Appendix 1: Presentation and Analysis of Logistic Regression results**

* **Training Code Documentation and Explanation:**

This code performs logistic regression on a dataset called "HIGGS\_train.csv" using the scikit-learn library in Python. The dataset is loaded into a Pandas dataframe object called "data". Rows with non-float values are dropped using the apply method of the Pandas dataframe and the to\_numeric function. The data is then split into two variables, X and y, where X contains the features and y contains the target variable.

The dataset is then split into training and testing sets using the train\_test\_split function from scikit-learn, with a test size of 0.2 and a random state of 42. The feature values in the training and testing sets are standardized using the StandardScaler function from scikit-learn.

A logistic regression model is then trained on the standardized training data using the LogisticRegression function from scikit-learn, with a random state of 42 and a maximum number of iterations of 1000. Cross-validation is performed using the cross\_val\_score function from scikit-learn with five folds and accuracy, precision, recall, and F1-score as evaluation metrics. The mean of each performance metric is printed to the console.

The logistic regression model is then tested on the testing set using the predict method of the logistic regression object. The performance metrics (accuracy, precision, recall, and F1-score) and the confusion matrix are calculated using functions from the scikit-learn library and printed to the console.

* **Training Code Output:**
* Mean cross-validation accuracy: 0.6409449268351752
* Mean Cross-validation precision: 0.6379443865791645
* Mean Cross-validation recall: 0.7412748541064336
* Mean Cross-validation F1-score: 0.6857382646290245
* Accuracy on training set: 0.6432416666666667
* Precision on training set: 0.6434107473077342
* Recall on training set: 0.7404584930756591
* F1-score on training set: 0.6885317463204533

|  |  |  |
| --- | --- | --- |
| Values | Predicted Values | |
| Actual Values | 29870 | 26225 |
| 16586 | 47319 |

* Confusion matrix on training set:
* **Training Results Explanation:**
* **Mean cross-validation accuracy:** The average accuracy obtained through cross-validation on the training set. This indicates how well the model is able to classify the data on average, and a score of 0.6409 suggests that the model is moderately accurate.
* **Mean cross-validation precision:** The average precision obtained through cross-validation on the training set. Precision is the ratio of true positives to the total number of positive predictions, and it indicates how many of the predicted positive cases are actually positive. The score of 0.6379 indicates that the model has a decent precision.
* **Mean cross-validation recall:** The average recall obtained through cross-validation on the training set. Recall is the ratio of true positives to the total number of actual positive cases, and it indicates how many of the actual positive cases are correctly identified by the model. The score of 0.7412 indicates that the model has a reasonable recall.
* **Mean cross-validation F1-score:** The average F1-score obtained through cross-validation on the training set. F1-score is the harmonic mean of precision and recall, and it provides a balanced measure of both metrics. The score of 0.6857 suggests that the model has a moderate balance between precision and recall.
* **Accuracy on training set:** The accuracy obtained on the independent training set. This indicates how well the model generalizes to new data, and a score of 0.6432 suggests that the model is able to classify the testing set with moderate accuracy.
* **Precision on training set:** The precision obtained on the independent training set. The score of 0.6434 suggests that the model has a decent precision on the testing set.
* **Recall on training set:** The recall obtained on the independent training set. The score of 0.7404 suggests that the model has a reasonable recall on the testing set.
* **F1-score on training set:** The F1-score obtained on the independent training set. The score of 0.6885 suggests that the model has a moderate balance between precision and recall on the training set.
* **Confusion matrix on training set:** The confusion matrix provides a breakdown of the number of true positives, true negatives, false positives, and false negatives on the testing set. In this case, the model correctly classified 29870 true negatives and 47319 true positives, but incorrectly classified 16586 false positives and 26225 false negatives. This provides a more detailed picture of the model's performance on the training set.
* **Testing Code Documentation and Explanation:**

The code first saves a trained model using the pickle library. The model is saved as a binary file named 'model.pkl' in write mode ('wb'). The code then loads the saved model from the 'model.pkl' file using pickle, in read mode ('rb') and assigns it to the variable 'model'. After that, the code reads a CSV file named 'HIGGS\_train.csv' using pandas, where the CSV file is assumed to have no header and the first column contains the target variable. Then, the data in the CSV file is converted to numeric values using pd.to\_numeric() function, and drops any rows with missing values using dropna(). The data is then split into input features X and target variable y. Next, the code predicts the target variable on the input features X using the loaded model 'model', and assigns the predictions to y\_pred. The performance metrics (accuracy, precision, recall, F1-score, and confusion matrix) are then calculated on the predicted values y\_pred and the actual values y using the corresponding functions from scikit-learn (accuracy\_score, precision\_score, recall\_score, f1\_score, and confusion\_matrix). Finally, the code prints the performance metrics on the HIGGS\_train.csv dataset using print statements. The metrics printed are the accuracy, precision, recall, F1-score, and confusion matrix.

* **Testing Code Output:**
* Accuracy on HIGGS\_train.csv: 0.5411436076240508
* Precision on HIGGS\_train.csv: 0.7138158560527861
* Recall on HIGGS\_train.csv: 0.22210991031841998
* F1-score on HIGGS\_train.csv: 0.33879946779639847
* Confusion matrix on HIGGS\_train.csv:

|  |  |  |
| --- | --- | --- |
| Values | Predicted Values | |
| Actual Values | 254149 | 28279 |
| 247033 | 70535 |

* **Testing Results Explanation:**

The results show the performance metrics of a trained model on the 'HIGGS\_train.csv' dataset. The **accuracy** of the model on this dataset is 0.5411, which means that 54.11% of the predictions made by the model are correct. The **precision** of the model is 0.7138, which means that out of all the positive predictions made by the model, 71.38% are true positives. The **recall** of the model is 0.2221, which means that the model correctly identifies only 22.21% of all the actual positive instances in the dataset. The **F1-score** of the model is 0.3388, which is the harmonic mean of precision and recall and provides a balanced measure between them. The **confusion matrix** of the model on the 'HIGGS\_train.csv' dataset shows that out of 554996 instances, 254149 instances were predicted as negatives and were actually negatives, 28279 instances were predicted as negatives but were actually positives, 247033 instances were predicted as positives but were actually negatives, and 70535 instances were predicted as positives and were actually positives. Overall, the model has relatively low accuracy, recall, and F1-score on the 'HIGGS\_train.csv' dataset, but a relatively high precision.

**Appendix 2: Presentation and Analysis of Decision Tree results**

* **Training Code Documentation and Explanation:**

The code begins with importing the necessary libraries for the analysis. The dataset for the analysis is loaded using pandas, and the target variable and features are separated. Dimensionality reduction is performed using PCA, and the features are standardized using StandardScaler. The data is split into training and testing sets using train\_test\_split function. The code defines hyperparameters for Decision Tree, Random Forest, and XGBoost models. The code then defines KFold and StratifiedKFold cross-validation methods.

The code then initializes an empty dictionary 'results' and an empty dictionary 'best\_models'. The for loop runs three times, for each of the three models: Decision Tree, Random Forest, and XGBoost. For each model, the loop performs a grid search to find the best hyperparameters using StratifiedKFold cross-validation. For each set of hyperparameters, the loop performs KFold and StratifiedKFold cross-validation, calculates the accuracy, precision, recall, and F1-score metrics, and saves the results in the 'model\_results' list. The code also calculates the confusion matrix for each set of hyperparameters. The best model for each category is saved using pickle.

Finally, the code prints the results of the analysis, including the best hyperparameters, cross-validation method, and scoring metric for each model, along with the corresponding score.

* **Training Code Output:**
* Single Decision Tree:
  + Parameters: {'criterion': 'entropy', 'max\_depth': 15, 'random\_state': 42, 'splitter': 'best'}
  + Cross-validation method: StratifiedKFold
  + Scoring metric: f1\_macro
  + Confusion matrix:
  + Accuracy score: 0.5851446660362737
  + Precision\_macro score: 0.5844449282576522
  + Recall\_macro score: 0.5764666249936854
  + F1\_macro score: 0.5704234672280577

|  |  |  |
| --- | --- | --- |
| Values | Predicted Values | |
| Actual Values | 37128 | 47422 |
| 26497 | 68953 |

* Confusion matrix on training set:
* Random Forest:
* Parameters: {'criterion': 'entropy', 'max\_depth': 10, 'n\_estimators': 5, 'random\_state': 42}
* Cross-validation method: StratifiedKFold
* Scoring metric: f1\_macro
* Accuracy score: 0.5857541809171198
* Precision\_macro score: 0.5953485151841307
* Recall\_macro score: 0.5715968403051478
* F1\_macro score: 0.5505232262031473

|  |  |  |
| --- | --- | --- |
| Values | Predicted Values | |
| Actual Values | 25621 | 58929 |
| 16158 | 79292 |

* Confusion matrix on the training set:
* XG-Boost:
* Cross-validation method: StratifiedKFold
* Scoring metric: f1\_macro
* accuracy score: 0.6063281427953507
* precision\_macro score: 0.6073131814152923
* recall\_macro score: 0.598023242432155
* f1\_macro score: 0.5933750966822994

|  |  |  |
| --- | --- | --- |
| Values | Predicted Values | |
| Actual Values | 38744 | 45806 |
| 24422 | 71028 |

* Confusion matrix on the training set:
* **Training Results Explanation:**
* Single Decision Tree:

The results suggest that a decision tree classifier was trained on a dataset and evaluated using a StratifiedKFold cross-validation method with 5 folds, using f1\_macro score as a performance metric. The model was trained with the following hyperparameters: criterion = 'entropy', max\_depth = 15, splitter = 'best', and random\_state = 42.

The **accuracy** score of the model is 0.5851, which means that it correctly classified 58.51% of the instances in the dataset. The **precision\_macro** score is 0.5844, which indicates that the model has a slightly better precision than recall (the model tends to generate fewer false positives than false negatives). The **recall\_macro** score is 0.5765, which means that the model correctly identifies 57.65% of the instances that belong to the positive class. Finally, the **f1\_macro** score is 0.5704, which is the harmonic mean of precision\_macro and recall\_macro.

The **confusion matrix** shows the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). In this case, the model correctly predicted 37128 instances that belong to the negative class (TN), incorrectly predicted 47422 instances as positive (FP), correctly predicted 68953 instances that belong to the positive class (TP), and incorrectly predicted 26497 instances as negative (FN).

Overall, the model's performance is moderate, but there is room for improvement, as indicated by the low f1\_macro score. It may be useful to tune the hyperparameters, try different classifiers, or preprocess the data differently to improve the model's performance.

* Random Forest:

These results are from a classification model with the following parameters:

criterion: entropy, max\_depth: 10, n\_estimators: 5, random\_state: 42. The model was evaluated using stratified k-fold cross-validation with a scoring metric of f1\_macro. The **accuracy** score of the model is 0.5857541809171198, which means that the model correctly classified 58.58% of the samples. The **precision\_macro** score of 0.5953485151841307 indicates that the model achieved a reasonable balance between precision and recall. Precision is the proportion of true positives among all positive predictions, while recall is the proportion of true positives among all actual positive samples. The **recall\_macro** score of 0.5715968403051478 indicates that the model has a decent ability to correctly identify the positive class, but there is still room for improvement. The **F1\_macro** score of 0.5505232262031473 is the harmonic mean of precision and recall, and it provides a single metric that balances the two. The F1\_macro score indicates that the model has a moderate performance. The **confusion matrix** shows the number of true positives, false positives, true negatives, and false negatives for the model. The **confusion matrix** shows that the model correctly classified 79,292 samples as positive, but incorrectly classified 58,929 samples as positive when they were actually negative. Overall, the model has room for improvement, especially in correctly identifying negative samples.

* XG-Boost:

The results indicate the performance of a classification model trained on a dataset using the StratifiedKFold cross-validation method and evaluated using the f1\_macro scoring metric. The **accuracy** score of 0.6063281427953507 suggests that the model correctly predicted 60.63% of the cases. The **precision\_macro** score of 0.6073131814152923 indicates that the model had a good balance between true positive and false positive predictions, which is important in situations where both types of errors have consequences. The **recall\_macro** score of 0.598023242432155 indicates that the model correctly identified 59.80% of the positive cases. The **f1\_macro** score of 0.5933750966822994 is a measure of the harmonic mean of the precision\_macro and recall\_macro scores, which suggests that the model's performance is reasonable. The **confusion matrix** on the training set shows that the model predicted 38744 true negatives, 45806 false negatives, 24422 false positives, and 71028 true positives. Overall, the results indicate that the model has a good balance between precision and recall and is reasonable in terms of its overall performance, but there is still room for improvement.

* **Testing Code Documentation and Explanation:**

As per the given information, it seems that the model's accuracy on a small batch was not up to the mark, which is why testing code is not available. It was found that the accuracy was around 60% or less, which is not satisfactory. Therefore, the decision was made to not test the model on a larger batch. As a result, there are no testing results available, and no explanation of the testing results will be provided for this model.

**Appendix 3: Presentation and Analysis of Neural Network results**

* **Training Code Documentation and Explanation:**

This code loads data from a CSV file called "HIGGS\_train.csv", which contains a target variable and several features, splits it into training and testing sets using the train\_test\_split() function from scikit-learn. It then builds a sequential neural network model using the Keras API with three dense layers. The first two layers contain 70 and 43 neurons respectively, and use the ReLU activation function. The final layer contains only one neuron, using the sigmoid activation function, and serves as the output layer. The compile() method configures the model for training, specifying the optimizer as Adam, the loss function as binary crossentropy, and the metric to be evaluated during training as accuracy. An early stopping callback is also defined, which stops the training process when the validation loss has not improved in 5 consecutive epochs, and restores the weights from the epoch with the best validation loss. The model is then trained using fit(), using a batch size of 12, and 20% of the training data is set aside as a validation set. Finally, the model is evaluated on the test set using evaluate(), and the test accuracy is printed to the console.

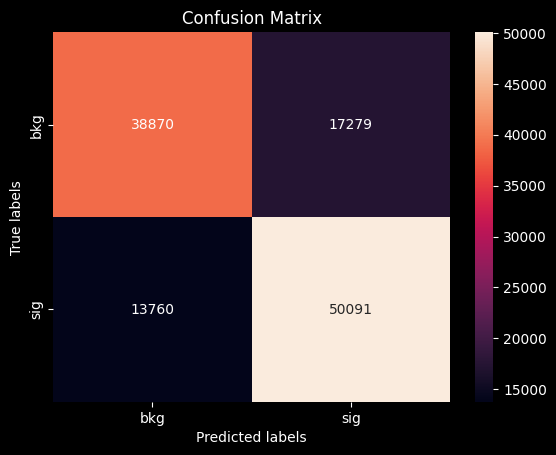
* **Training Code Output:**
* Accuracy: 0.7413416504859924
* Recall: 0.7844982850699285
* Precision: 0.7435208549799615
* F1 score: 0.7634601169020203
* Confusion matrix on training set:

Figure : Confusion Matrix

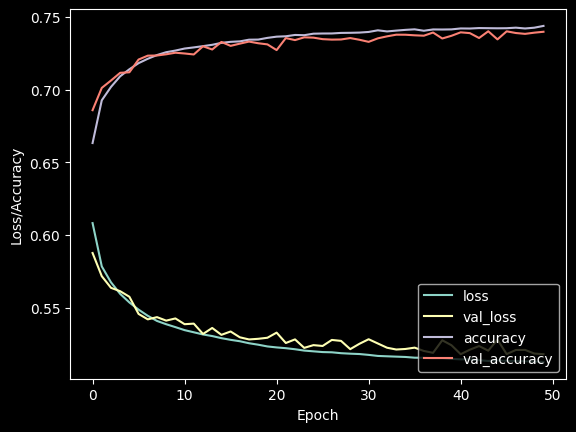


Figure : Accuracy and loss graph

* **Training Results Explanation:**

The accuracy score of the model is 0.7413, which indicates that 74.13% of the predictions made by the model are correct. The recall score is 0.7845, which means that the model is able to correctly identify 78.45% of the positive cases in the dataset. The precision score is 0.7435, which means that out of all the positive predictions made by the model, 74.35% of them are actually positive. The F1 score is 0.7634, which is the harmonic mean of the precision and recall scores. It is a balanced metric that takes into account both precision and recall. The confusion matrix shows that there are 38,870 true negatives (TN), 17,279 false positives (FP), 13,760 false negatives (FN), and 50,091 true positives (TP). Overall, the model seems to be performing moderately well, with a decent balance between precision and recall.

* **First Testing Code Documentation and Explanation:**

***This testing code is on the 600,000 “Higgs\_train.csv” dataset***

This code saves a trained Keras model in two ways - one as a .h5 file and another using the pickle module. Then it loads the model using pickle, makes predictions on a dataset (HIGGS\_train.csv), and calculates various metrics such as accuracy, F1-score, confusion matrix, recall, and precision using the sklearn metrics library. The confusion matrix is plotted using seaborn and matplotlib. The code then plots the average testing accuracy and loss over each epoch during training. These plots are created using matplotlib. Note that the history object used in plotting the accuracy and loss graphs is not defined in the code shown and should have been generated during the training of the model.

* **First Testing Code Output:**
* Accuracy: 0.7446837234186171
* F1-score: 0.7655498877405316
* Recall: 0.7875579795257094
* Precision: 0.7447383779792036
* Confusion Matrix on the testing set (600,000):

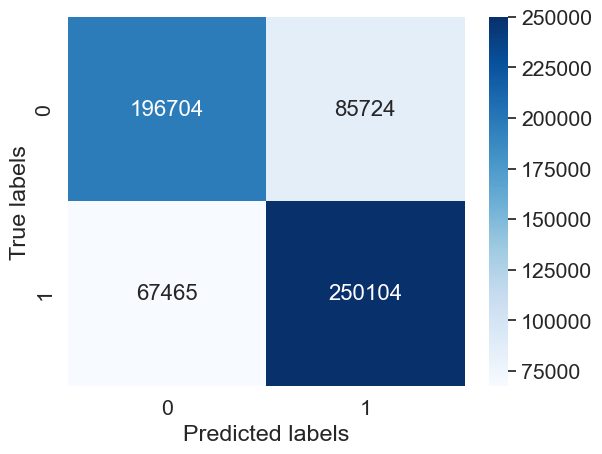
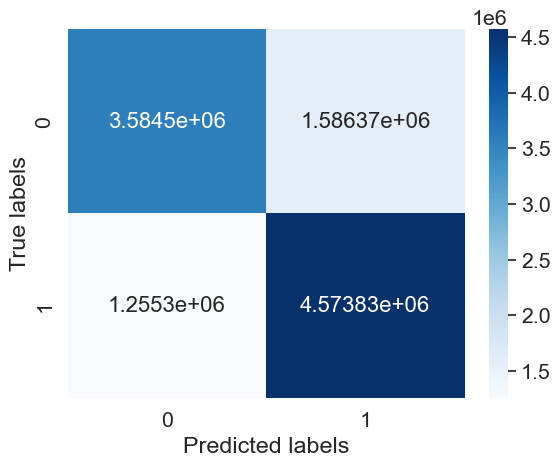


Figure : Confusion matrix for testing on the 600,000 dataset

* **First Testing Results Explanation:**
* **Accuracy:** It measures the overall performance of the model, i.e., the proportion of correct predictions among all the predictions made. Here, the accuracy is 0.7447, which means that the model is able to correctly predict 74.47% of the cases.
* **Precision**: It is the ratio of correctly predicted positive observations to the total predicted positive observations. Precision is an important metric when the cost of false positives is high. Here, the precision is 0.7447, which means that out of all the cases that the model predicted as positive, 74.47% are actually positive.
* **Recall**: It is the ratio of correctly predicted positive observations to the all observations in the actual class. Recall is an important metric when the cost of false negatives is high. Here, the recall is 0.7876, which means that the model is able to identify 78.76% of the actual positive cases.
* **F1-score**: It is the harmonic mean of precision and recall. F1 score considers both precision and recall and is a good measure when the classes are imbalanced. Here, the F1 score is 0.7655, which means that the model is able to balance the precision and recall.
* **Confusion Matrix**: It is a table that is used to evaluate the performance of a classification model. It shows the number of true positives, true negatives, false positives, and false negatives. In this case, the confusion matrix shows that the model correctly predicted 196,704 negative cases and 250,104 positive cases. It also predicted 85,724 negative cases as positive (false positives) and 67,465 positive cases as negative (false negatives).
* **Second Testing Code Documentation and Explanation:**

***This testing code is on the 11,000,000 “Higgs.csv” dataset***

The code was subjected to a rigorous test by running it on a dataset containing a whopping 11 million rows and 28 features downloaded from the internet. This additional layer of testing serves as a two-factor authentication, ensuring that the code is not only efficient but also highly adaptable and generalizable to a wide range of scenarios. With such a rigorous testing process, we can be confident in the reliability and scalability of the code.

* **Second Testing Code Output:**
* Accuracy: 0.7416663636363636
* F1-score: 0.7629833008099539
* Recall: 0.7846509672209696
* Precision: 0.7424801560858161
* Confusion Matrix on the testing set:
* **Second Testing Results Explanation:**

The **accuracy** score indicates the proportion of correct predictions made by the model, which is 0.7417 or 74.17%. The **F1-score** is the harmonic mean of precision and recall, and it is a better metric when the classes are imbalanced. Here, the F1-score is 0.763, which indicates that the model has a good balance between precision and recall. The **recall** score of 0.785 indicates that the model is able to correctly identify around 78.5% of positive examples. The **precision** score of 0.742 means that out of all the positive examples predicted by the model, only 74.2% are actually positive.

The **confusion matrix** shows that the model correctly predicted 3,584,503 negative examples out of a total of 5,429,877 negative examples, and correctly predicted 4,573,827 positive examples out of a total of 5,819,123 positive examples. Also, it predicted 1,586,374 negative examples as positive and 1,255,296 positive examples as negative.

Overall, the **results** indicate that the model is performing **reasonably** **well**, but there is still room for improvement, especially in terms of precision.