*Semester Project: Credit-Card Fraud Detection* ***using Supervised Learning***

**Problem Statement:**

Credit card fraud is a common problem that costs millions of dollars to financial institutions and cardholders. Fraudsters use various techniques to deceive banks and individuals, making it challenging to detect fraudulent transactions. Therefore, there is a need to develop a reliable system that can identify fraudulent transactions and minimize the losses incurred.

**Methodology**: The methodology of our project is in the following order: -

1. Data collection: The credit card transaction dataset will be obtained from public sources such as Kaggle.
2. Data preprocessing: The dataset will be prepared for model testing and training, which includes processing the source file into a format which can be worked on, preparing subsets of data for test and train data, and to account for imbalanced data and any other challenges.
3. Model building: The logistic regression algorithm will be implemented using Python's Scikit-learn library to build a fraud detection model. Other libraries will also be used for implementation of various other models for evaluation purposes.
4. Model evaluation: The performance of the model will be evaluated using metrics like accuracy, precision, recall, F1 score, and ROC-AUC. We will also compare the model's performance with other machine learning approaches which are Decision Trees, Random Forests, and Neural Networks.
5. Model tuning: The model's hyperparameters will be tuned to further improve its accuracy.
6. Implementation: Implement the final model as part of a Jupyter notebook.

**Disclaimer:** Due to the use of frequent output from certain parts of code in this project, the project will utilize **Jupyter** **Notebook**. Also, the following dependencies will be used to implement the project: **Pandas, Numpy, Tensorflow, Tabulate and Sci-kit (Sklearn).**

**Note**: In our proposal, we had chosen the implementation to be Command-Line Interface (CLI) but as the project has progressed, we believe it better to utilize Jupyter due to its ease-of-use.

**Background**

Supervised machine learning algorithms derive patterns and relationships from a labeled training dataset. Supervised learning problems can be further classified into regression and classification problems.

Classification: In a classification problem, the output variable is a category, such as “red” or “blue,” “disease” or “no disease,” “true” or “false,” etc.

Regression: In a regression problem, the output variable is a real continuous value, such as “dollars” or “weight.”

In this project, we will be using a training dataset, and a test dataset and attempt to classify certain test data as fraudulent or non-fraudulent through the training of our model. For this purpose, we will utilize the logistic regression model as helps to calculate the possibility of a particular event taking place (i.e., a transaction being fraudulent or not).

Logistic regression is a statistical method that is used in machine learning models where the dependent variable is binary. It is used to describe data and the relationship between one dependent variable and one or more independent variables.

The Logistic Regression model utilizes a sigmoid function, which outputs a value from between zero to 1. This can be easily utilized to represent the ‘probability’ of certain data being classified as a response variable.

We will evaluate the performance of the Logistic Regression Model by comparing its performance with that of other Machine Learning approaches using the following metrics:

Accuracy score: Percentage of correct predictions made by the model.

Precision score: Proportion of true positives from samples that the model predicted as positive.

Recall: Measures the proportion of true positives among all samples that are actually positive.

F1 score: Combines precision and recall into a single score.

ROC-AUC: Measure of the area under the receiver operating characteristic (ROC) curve, which is a plot of the true positive rate (sensitivity) vs the false positive rate (1-specificity). The ROC curve shows how well the model can distinguish between positive and negative samples.

The project also implements logging in the notebook for the tabulation of results.

The step-by-step execution of the project is available in the Jupyter Notebook (.ipynb) file.

**Week 1 - Data collection and preprocessing**

For data collection, we will use Kaggle for this project. Kaggle is a very useful repository of datasets as well as a hub for data scientists.

The dataset we have chosen is: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>

The Kaggle link states the following regarding the variables and features of the dataset:

*“Features V1, V2, … V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset … Feature 'Class' is the response variable, and it takes value 1 in case of fraud and 0 otherwise.”*

The original transaction details have been obscured using Principal Component Analysis (an unsupervised learning algorithm). PCA converts the observations of correlated features into a set of linearly uncorrelated features. i.e. it anonymizes the individual transactions to hide the identity of the user, while still retaining features of the dataset.

The Class feature identifies whether the transaction was fraudulent or not (i.e. it is binary), indicating that the dataset is labeled/classified.

The actual dataset is in comma-separated value (.csv) format, so we will use Pandas to create a Pandas Dataframe of the credit card dataset which we can work on in Python.

Next, we check what are the features of the dataset (i.e. columns). We find that there is Time, V1-V28, Amount and Class features, with Class being integer type and the rest being Float type.

Using pd.value\_counts on the ‘Class’ column, we find that the distribution of transactions is very unbalanced i.e. 284315 legit vs 492 fraud. The dataset is highly unbalanced. We cannot use the dataset as is due to the risk of invalid accuracy score of models. So, we will now perform under-sampling on the dataset.

We built a new dataset that comprises of 492 transactions randomly sampled from the ‘0’ Class and the existing 492 transactions from the ‘1’ Class. Now our dataset is balanced and ready.

Now we need to create training and test splits of our dataset. For this purpose, we will use the built-in function of sci-kit - train\_test\_split(). An 80-20 split was specified for the data training and testing process. To ensure that the results remain reproducible, we use the random\_state parameter to set the random seed for the splitting of data.

**Week 2 – Model Building and Logging**

Utilizing the Sci-Kit library, we implemented the Logistic Regression, Decision Tree, and Random Forest models. To ensure that the results are reproducible for models that incorporate randomness, we used the random\_state parameter to set the random seed for the algorithms.

Similarly, we implemented an Artificial Neural Network using the TensorFlow library with Keras, where the network is comprised of 2 hidden layers, and one output layer.

For the model tuning, evaluation, and logging, we used the score functions already built-in to the Sci-Kit and TensorFlow libraries to retrieve the various metrics like Accuracy, Precision and so on. To tabulate these results for each model, we saved them in lists and formulated a table using the **Tabulate** library.

Furthermore, we also assigned Experiment numbers in case the models are run again with a different random\_ state value for comparison purpose.

**Week 3 – Model Testing and Tuning**

As now we can log the results, it is time to test and tune our various models to find the best set of hyperparameters for use on each of the models.

To help in selecting the best set of hyperparameters, we will utilize the GridSearchCV class from the Sci-Kit Library. GridSearchCV exhaustively searches over a defined parameter grid to find the optimal combination of hyperparameters for a given model.

For the Logistic Regression model, we used the following hyperparameters:

* ‘**penalty**’: controls the type of regularization applied to the model, with options of ‘l2’ and ‘none’.
* ‘**C**’: controls the trade-off between fitting the training data well and having small parameter values, with a range of values [0.1, 0.5, 1, 1.5, 1.9].

For the Decision Tree model, we used the following hyperparameters:

* ‘**max\_depth**’: sets the maximum depth of the tree, with range of values [2, 4, 6, 8, 10, 12].
* ‘**min\_samples\_split**’: specifies the minimum number of samples required to split a node in the tree, with a range of values [2, 4, 6, 8, 10, 12].
* ‘**criterion**’: parameter with ‘gini’ and ‘entropy’ measures to choose the best one for measuring the quality of a split in the decision tree.

For the Random Forest model, we used the following hyperparameters:

* ‘**n\_estimators**’: number of trees in forest, with this set of values [50, 100, 200].
* ‘**max\_depth**’: sets the maximum depth of each tree in the forest from [None, 10, 20].
* ‘**min\_samples\_split**’: sets the minimum number of samples required to split an internal node with values of [2, 5].
* ‘**min\_samples\_leaf**’: this sets the minimum number of samples required to be at a leaf node with values of [1, 2].

For our neural network model, we used the following hyperparameters:

* '**batch\_size**': the number of samples to be used in each batch during training, with this set of values [16, 32].
* '**epochs**': the number of times the entire dataset is passed through the network during training, with values of [100, 200, 400, 800].

The choice of hyperparameters was the most time-consuming task, especially for the neural network, as GridSearchCV would test all possible combinations of parameters so every new set of hyperparameters would take significant time to test. By testing different combinations of hyperparameters, we hope to find the best combination which gives the best performance.

**Week 4 – Model Results and Evaluation**

In this section, we will tune the models using GridSearchCV and use the best hyperparameters found by GridSearchCV to evaluate the results. The following hyperparameters performed best:

Logistic Regression:

Best Hyperparameters: {'C': 0.1, 'penalty': 'l2'}

Best Mean Cross-validated Score: 0.9263

Decision Trees:

Best Hyperparameters: {'criterion': 'gini', 'max\_depth': 2, 'min\_samples\_split': 2}

Best Mean Cross-validated Score: 0.9238

Random Forests:

Best Hyperparameters: {'max\_depth': None, 'min\_samples\_leaf': 1, 'min\_samples\_split': 5, 'n\_estimators': 100}

Best Mean Cross-validated Score: 0.9415

Neural Network:

Best Hyperparameters: ['batch\_size': 32, 'epochs': 400]

Best Mean Cross-validated Score: 0.8958

The following are the results of the metrics we used to evaluate the performance of models:

|  |  |  |  |
| --- | --- | --- | --- |
| Experiment Name | Score | Training Data | Test Data |
| Logistic Regression 1 | Accuracy | 0.93774 | 0.95939 |
|  | Precision | 0.96748 | 0.97872 |
| Recall | 0.90609 | 0.93878 |
| F1 | 0.93578 | 0.95833 |
| ROC-AUC | 0.93778 | 0.95929 |
| Decision Tree 1 | Accuracy | 0.9352 | 0.93401 |
|  | Precision | 0.94545 | 0.91262 |
| Recall | 0.92386 | 0.95918 |
| F1 | 0.93453 | 0.93532 |
| ROC-AUC | 0.93521 | 0.93414 |
| Random Forest 1 | Accuracy | 0.99619 | 0.94416 |
|  | Precision | 1 | 0.95789 |
| Recall | 0.99239 | 0.92857 |
| F1 | 0.99618 | 0.94301 |
| ROC-AUC | 0.99619 | 0.94408 |
| ANN 1 | Accuracy | 0.88818 | 0.91878 |
|  | Precision | 0.98113 | 1 |
| Recall | 0.79188 | 0.83673 |
| F1 | 0.8764 | 0.91111 |
| ROC-AUC | 0.88831 | 0.91837 |

We can see that the **Logistic Regression** model has the highest accuracy, F1 score, and ROC-AUC score on the test data, indicating excellent performance in distinguishing between legit and fraudulent transactions.

Notably, for **training data,** the **Random Forests** model performed the best in all metrics, indicating that if the objective of the model was to identify training data, then the Random Forest model would be the best choice for binary classification.

Whereas, on **test data**, the **Neural Network** achieved perfect precision indicating that it did not falsely identify even a single legitimate transaction as fraudulent. However, it had lower recall leading to a higher number of false negatives.

Overall, **Logistic Regression** outperformed all models in F1 and ROC-AUC scores. That is not to say the other models are unsuited for binary classification as all models have good scores (> 0.9) in F1 and ROC-AUC. There are other factors to consider also, such as no. of features in the data, and the complexity of the model. So, for our credit card transaction dataset, Logistic Regression performed the best.