2022

Homework 1

FINANCIAL TECHNOLOGY

D11949001- MIGUEL BENALCAZAR

(i) Logistic regression (LR), support vector machine (SVM), and random forest (RF) are supervised machine learning algorithm that can be used for classification tasks. Please briefly describe the concept of these three algorithms and build three model to learn the binary classification task.

Logistic Regression (LR)

Logistic regression is a statistical analysis method used to predict dependent binary variable by analyzing the relationship between one or more known independent variables. A logistic regression is a supervised learning model used in classification problems.

In logistic regression, the dependent variable is assumed to be binary, and the independent variables should be independent of one another and linearly related to the log chances. Having a large sample size is important for accurate results, but when the number of predictor variables within the model is high, it is prone to overfitting.

Support Vector Machine (SVM)

The support vector machine is a supervised machine learning algorithm that can be used to perform classification, regression, and outliers' detection. It constructs a hyperplane in multidimensional space to separate different classes by generating an optimal hyperplane to minimize an error iteratively. The main idea of SVM is to find a maximum marginal hyperplane (MMH) that best divides the dataset into classes.

SVM is effective in high dimensional spaces and even effective when the number of dimensions is greater than the number of samples. Using different kernel functions can be specified for the decision functions.

Random Forest (RF)

Random forest is a supervised machine learning method used for classification and regression tasks. A multitude of decision trees are combined into a forest-like structure to make an accurate prediction.

A decision tree popular is a flowchart-like tree structure tool used for classification and prediction. A decision tree is formed by internal nodes which denote a test on an attribute, each brand represents an outcome of the test, and each terminal node holds a class label.

These algorithms were implemented using the scikit-learn library to classify the data as fraud or non-fraud from the provided dataset.

(ii) Plot the confusion matrices for Logistic Regression, Support Vector Machine, and Random Forest

The data was loaded using the library pandas by executing the class creditcardDataset() in the google collab file. In this, all data is divided between "X" which contains the features of the data, and "y" which represents the labels. Moreover, a logistic model, a support vector machine, and a random forest were trained and tested.

The confusion matrix by using the testing set is displayed as follows.

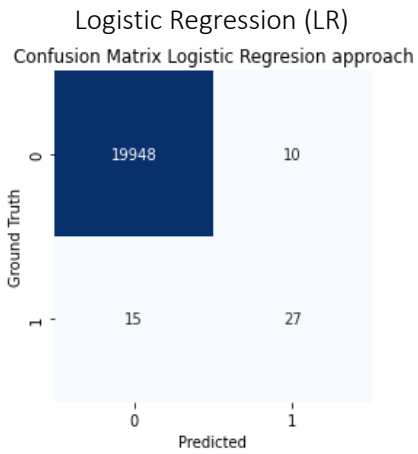


Figure 1. Confusion Matrix - Logistic Regression (LR)

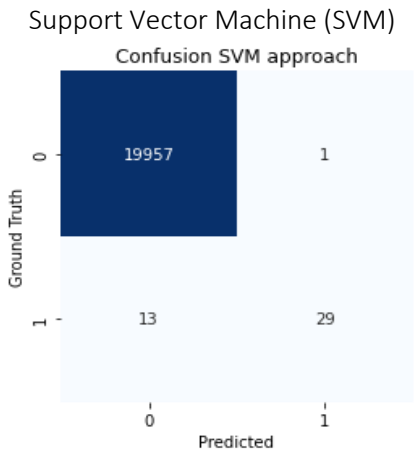


Figure 2. Confusion Matrix - Support Vector Machine (SVM)

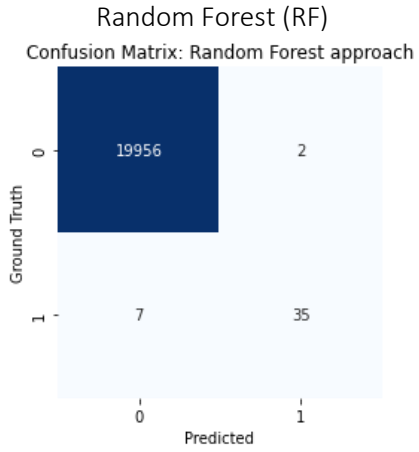


Figure 3. Confusion Matrix - Random Forest (RF)

(iii) Precision, recall, and F1-score are ways to evaluate model performance. For each class, please calculate the corresponding Accuracy, Precision, Recall, and F1-Score on the test set in your report. (hint: use sklearn.metrics.classification report). Which metric do you think is more suitable for this dataset? Please explain why.

Logistic Regression (LR)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.73	0.64	0.68	42
accuracy			1.00	20000
macro avg	0.86	0.82	0.84	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.88%

Support Vector Machine (SVM)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.97	0.69	0.81	42
accuracy			1.00	20000
macro avg	0.98	0.85	0.90	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.93%

Random Forest (RF)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.95	0.83	0.89	42
accuracy			1.00	20000
macro avg	0.97	0.92	0.94	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.97%

$$Accuracy = \frac{TP + TN}{P + N}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The dataset was divided into 80% for the training dataset, and 20% for the testing dataset. Since the data is imbalanced with respect to fraud and non-fraud after the division of the data we have as follows.

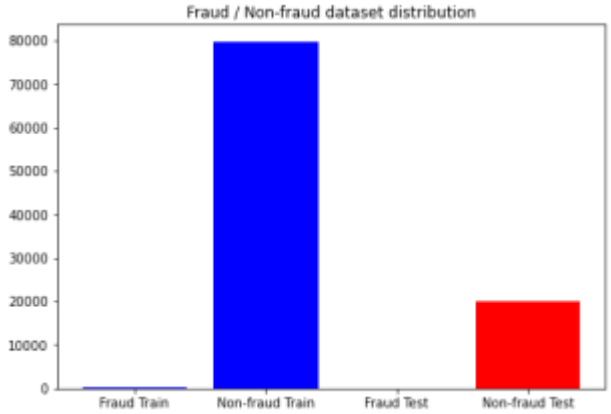


Figure 4. Fraud, non-fraud dataset distribution

For the training dataset, we have 181 fraud samples that represent the 0.23% of the total data in the training dataset, while 79818 non-fraud samples that represent the 99.77% of the total data in the training dataset. For the testing dataset, 42 fraud samples and 19958 non-fraud samples.

It can be seen in figure 4, that the non-fraud class label has a very high number of observations if we compare with the fraud observations.

The accuracy is not a suitable metric to evaluate this model since it is the total number of correct predictions by the classifier divided by the total number of predictions. This metric is good enough for a balanced dataset but for this case, the TN samples will shrink the model's misclassification. Therefore, accuracy will show a bias toward the TN without considering the minority class or frauds. This effect can be seen in the results of the accuracy from the methods used to classify, all of the accuracies are relatively close to 100 but if we take a look over the confusion matrices, we can see that the TP are not even close to the total fraud samples which is 42.

For credit card fraud detection, it is vital to identify frauds correctly. Precision on the other hand represents how many are actually positive from all of the predicted positives while Recall represents the ability of the classifier to find all TP. $F_1 - score$ combines the Recall and Precision in one metric by taking their harmonic mean. In this way, the two metrics are compared to each other. Therefore, $F_1 - score$ is the suitable metric to see how good is the classification by the model.

Based on $F_1 - score$ we can conclude that the classification made by the Random Forest (RF) model is better that the classification performed by the logistic regression (LR) and the Support Vector Machine (SVM).

(iv) Please construct a DNN model with Pytorch for binary classification according to the cross-entropy error function.
Minimize the error function $J(\theta)$ by running the error backpropagation algorithm using the Adam Optimizer. You should decide the following hyperparameters: number of hidden layers, number of hidden units, learning rate, number of iterations, and mini-batch size. Please try to perform a grid search over the variables mentioned above and show the best-performing setting for your model in the report. You also have to show your (a) training accuracy, (b) test accuracy, (c) training loss, (d) test loss (Figure 2 is an example.), (e) confusion matrices, and (f) Accuracy, Precision, Recall, and F1-Score in the report.

The following are the hyperparameters used to perform the grid search to choose a good model within the following eight models.

Models

DNN1((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (l2): Linear(in_features=32, out_features=32, bias=True) (act2): ReLU() (l3): Linear(in_features=32, out_features=32, bias=True) (act3): ReLU() (l4): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())	**DNN2**((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (l2): Linear(in_features=32, out_features=64, bias=True) (act2): ReLU() (l3): Linear(in_features=64, out_features=32, bias=True) (act3): ReLU() (l4): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())

DNN3((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (l2): Linear(in_features=32, out_features=32, bias=True) (act2): ReLU() (l3): Linear(in_features=32, out_features=32, bias=True) (act3): ReLU() (l4): Linear(in_features=32, out_features=32, bias=True) (act4): ReLU() (l5): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())	DNN4((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (l2): Linear(in_features=32, out_features=64, bias=True) (act2): ReLU() (l3): Linear(in_features=64, out_features=128, bias=True) (act3): ReLU() (l4): Linear(in_features=128, out_features=128, bias=True) (act4): ReLU() (l5): Linear(in_features=128, out_features=1, bias=True) (act_out): Sigmoid())
DNN5((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (l2): Linear(in_features=32, out_features=64, bias=True) (act2): ReLU() (l3): Linear(in_features=64, out_features=128, bias=True) (act3): ReLU() (l4): Linear(in_features=128, out_features=64, bias=True) (act4): ReLU() (l5): Linear(in_features=64, out_features=32, bias=True) (act5): ReLU() (l6): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())	DNN6((l1): Linear(in_features=30, out_features=32, bias=True) (act1): ReLU() (drop1): Dropout(p=0.2, inplace=False) (l2): Linear(in_features=32, out_features=32, bias=True) (act2): ReLU() (drop2): Dropout(p=0.2, inplace=False) (l3): Linear(in_features=32, out_features=32, bias=True) (act3): ReLU() (drop3): Dropout(p=0.2, inplace=False) (l4): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())
DNN7((l1): Linear(in_features=30, out_features=64, bias=True) (act1): ReLU() (drop1): Dropout(p=0.2, inplace=False) (l2): Linear(in_features=64, out_features=128, bias=True) (act2): ReLU() (drop2): Dropout(p=0.2, inplace=False) (l3): Linear(in_features=128, out_features=128, bias=True) (act3): ReLU() (drop3): Dropout(p=0.2, inplace=False) (l4): Linear(in_features=128, out_features=128, bias=True) (act4): ReLU() (drop4): Dropout(p=0.2, inplace=False) (l5): Linear(in_features=128, out_features=64, bias=True) (act5): ReLU() (drop5): Dropout(p=0.2, inplace=False) (l6): Linear(in_features=64, out_features=32, bias=True) (act6): ReLU() (drop6): Dropout(p=0.2, inplace=False) (l7): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())	DNN8((l1): Linear(in_features=30, out_features=32, bias=True) (norm1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (act1): ReLU() (drop1): Dropout(p=0.2, inplace=False) (l2): Linear(in_features=32, out_features=32, bias=True) (norm2): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (act2): ReLU() (drop2): Dropout(p=0.2, inplace=False) (l3): Linear(in_features=32, out_features=32, bias=True) (norm3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True) (act3): ReLU() (drop3): Dropout(p=0.2, inplace=False) (l4): Linear(in_features=32, out_features=1, bias=True) (act_out): Sigmoid())

The learning rate tested was 0.01 and 0.001, the max epochs were set to 1000 and the batch size was set to 10, 100, 500, 5000, and 10000. All of the previous data were combined with each other to get the best model with the best hyperparameters. The following table displays the results of all possible combinations using the testing dataset.

Model	batch_size	learning_rate	epoch	AUROC	accuracy	precision	recall	f1_score	tp	tn	fp	fn
DNN1	10	0.001	165	0.928521323	99.96	94.73684211	85.71428571	0.9	36	19956	2	6
	100	0.001	13	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.001	7	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	5000	0.001	3	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7
	10	0.01	25	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	100	0.01	6	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	10000	0.01	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.01	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	5000	0.01	2	0.904511379	99.91	77.27272727	80.95238095	0.790697674	34	19948	10	8
DNN2	10	0.001	113	0.868947409	99.925	88.57142857	73.80952381	0.805194805	31	19954	4	11
	100	0.001	19	0.928471218	99.95	90	85.71428571	0.87804878	36	19954	4	6
	10000	0.001	3	0.904536431	99.915	79.06976744	80.95238095	0.8	34	19949	9	8
	500	0.001	6	0.928421113	99.94	85.71428571	85.71428571	0.857142857	36	19952	6	6
	5000	0.001	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10	0.01	22	0.940225664	99.925	78.72340426	88.0952381	0.831460674	37	19948	10	5
	100	0.01	7	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.01	5	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.01	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	5000	0.01	4	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7
DNN3	10	0.001	108	0.928496271	99.955	92.30769231	85.71428571	0.888888889	36	19955	3	6
	100	0.001	30	0.916616561	99.955	94.59459459	83.33333333	0.886075949	35	19956	2	7
	10000	0.001	3	0.928345955	99.925	80	85.71428571	0.827586207	36	19949	9	6
	500	0.001	6	0.92839606	99.935	83.72093023	85.71428571	0.847058824	36	19951	7	6
	5000	0.001	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10	0.01	12	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	100	0.01	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	500	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	5000	0.01	6	0.5	99.79	0	0	0	0	19958	0	42
DNN4	10	0.001	36	0.928345955	99.925	80	85.71428571	0.827586207	36	19949	9	6
	100	0.001	11	0.928496271	99.955	92.30769231	85.71428571	0.888888889	36	19955	3	6
	10000	0.001	2	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7
	500	0.001	7	0.904636642	99.935	87.17948718	80.95238095	0.839506173	34	19953	5	8
	5000	0.001	3	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	10	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	100	0.01	4	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	500	0.01	3	0.928345955	99.925	80	85.71428571	0.827586207	36	19949	9	6
	5000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
DNN5	10	0.001	43	0.916391088	99.91	76.08695652	83.33333333	0.795454545	35	19947	11	7
	100	0.001	15	0.916566456	99.945	89.74358974	83.33333333	0.864197531	35	19954	4	7
	10000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.001	3	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	5000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10	0.01	14	0.5	99.79	0	0	0	0	19958	0	42
	100	0.01	3	0.916391088	99.91	76.08695652	83.33333333	0.795454545	35	19947	11	7
	10000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	500	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	5000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
DNN6	10	0.001	56	0.904536431	99.915	79.06976744	80.95238095	0.8	34	19949	9	8
	100	0.001	14	0.928345955	99.925	80	85.71428571	0.827586207	36	19949	9	6
	10000	0.001	2	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7
	500	0.001	4	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	5000	0.001	3	0.92829585	99.915	76.59574468	85.71428571	0.808988764	36	19947	11	6
	10	0.01	19	0.5	99.79	0	0	0	0	19958	0	42
	100	0.01	6	0.5	99.79	0	0	0	0	19958	0	42
	10000	0.01	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.01	6	0.916366035	99.905	74.46808511	83.33333333	0.786516854	35	19946	12	7
	5000	0.01	3	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
DNN7	10	0.001	3	0.5	99.79	0	0	0	0	19958	0	42
	100	0.001	7	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.001	3	0.904511379	99.91	77.27272727	80.95238095	0.790697674	34	19948	10	8
	500	0.001	4	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7
	5000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	100	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	10000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	500	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
	5000	0.01	2	0.5	99.79	0	0	0	0	19958	0	42
DNN8	10	0.001	156	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	100	0.001	33	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	500	0.001	14	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	5000	0.001	2	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10	0.01	26	0.928345955	99.925	80	85.71428571	0.827586207	36	19949	9	6
	100	0.01	7	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	10000	0.01	5	0.916441193	99.92	79.54545455	83.33333333	0.813953488	35	19949	9	7
	500	0.01	11	0.928320902	99.92	78.26086957	85.71428571	0.818181818	36	19948	10	6
	5000	0.01	3	0.916416141	99.915	77.77777778	83.33333333	0.804597701	35	19948	10	7

Figure 5. Grid Search Results

The DNN model and its hyperparameters were chosen based on the f1-score metric and the result is the following.

Model: DNN1, batch_size: 10, learning_rate: 0.001 , n_epochs = 165

```
gridSearch.best_f1_score
```

	file	file_name	batch_size	learning_rate	n_epoch	epoch	AUROC	accuracy	precision	recall	f1_score	tp	tn	fp	fn
0	DNN1	DNN1_lr0001_epoch1000_bz10.pkl	10	0.001	1000	165	0.928521	99.96	94.736842	85.714286	0.9	36	19956	2	6

Figure 6. Model DNN1, f1-score

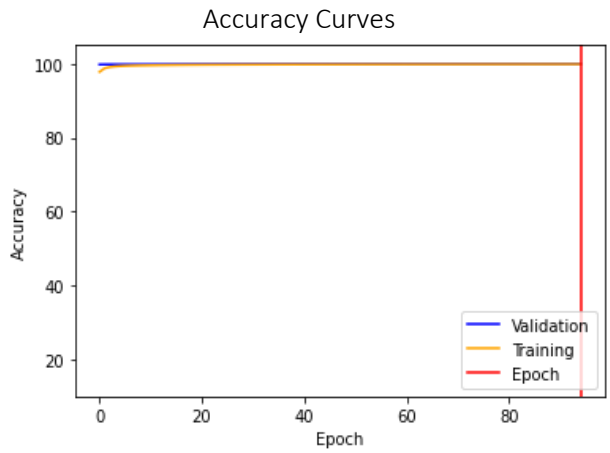


Figure 7. Train/Validation accuracy DNN1

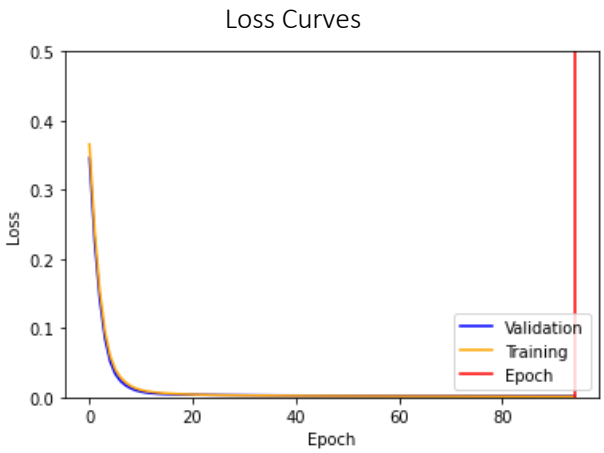


Figure 8. Train/Validation loss DNN1

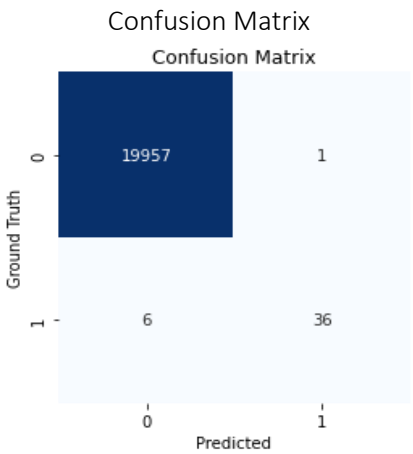


Figure 9. Confusion Matrix DNN1

Metrics

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.97	0.86	0.91	42
accuracy			1.00	20000
macro avg	0.99	0.93	0.96	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.965%

(v) You have to plot the receiver operating characteristic curve (ROC, as shown in Figure 3) and precision-recall curve (PRC, as shown in Figure 3) with their area-under-curve (AUROC and AUPRC) for DNN, LR, SVM, and random forest on the test set.

AUROC

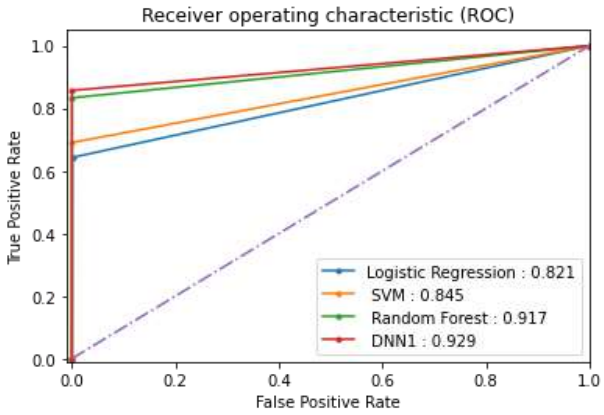


Figure 10. AUROC Curves (LR, SVM, RF, DNN1)

AUPRC

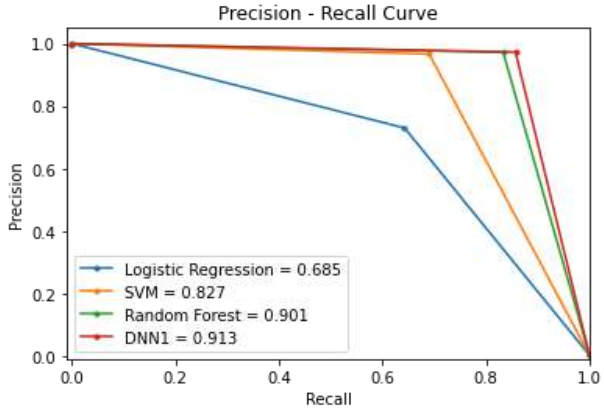


Figure 11. AUPRC Curves (LR, SVM, RF, DNN1)

(vi) Please use the metric you choose in (iii) to compare these four models (LR, SVM, RF, DNN) and briefly describe what you found.

Logistic Regression (LR)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.73	0.64	0.68	42
accuracy			1.00	20000
macro avg	0.86	0.82	0.84	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.88%

Support Vector Machine (SVM)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.97	0.69	0.81	42
accuracy			1.00	20000
macro avg	0.98	0.85	0.90	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.93%

Random Forest (RF)

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.95	0.83	0.89	42
accuracy			1.00	20000
macro avg	0.97	0.92	0.94	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.97%

DNN1

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.97	0.86	0.91	42
accuracy			1.00	20000
macro avg	0.99	0.93	0.96	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.965%

The metric chosen to compare the classification of the model in part (iii) was the f1-score which represents the relation between precision and recall in harmonic mean. Random Forest (RF) shows an f1-score of 0.89 while DNN1 shows an f1-score of 0.91 for fraud. Both show better results than the f1-score of the logistic regression and the support vector machine with 0.68 and 0,81 respectively.

Since f1-score is related to precision and recall, we can compare these values too. Recall from RF is 0.83 and DNN1 is 0.86, which means that the ability of the classifier to find all positive instances from the DNN1 is slightly better than the RF for fraud. On the other hand, DNN1 shows a better precision with 0.97 if we compare it with the rest. It means that DNN1 has a better ability to identify frauds among all the predicted fraud cases than the other models. Each model's confusion matrix reflects this. Moreover, we can also notice that while the recall increases the precision decrease and vice versa.

(vii) Let's observe the number of fraud vs. non-fraud cases in the dataset. The dataset is imbalanced, with most of the transactions being non-fraud. This bias in the training dataset can influence many machine learning algorithms, leading to ignoring the minority class entirely. The two main approaches: (1) Undersampling: Randomly resampling an imbalanced dataset are to delete examples from the majority class (2) Oversampling: Duplicate examples from the minority class. First, please use the random undersampling method to deal with your training dataset and bring the non-fraud transactions to the same amount as fraud transactions (we want a 50/50 ratio). Second, use this training dataset to train the LR, RF, SVM, and DNN again. Whether the performance improves after resampling in your case?

1. Oversampling

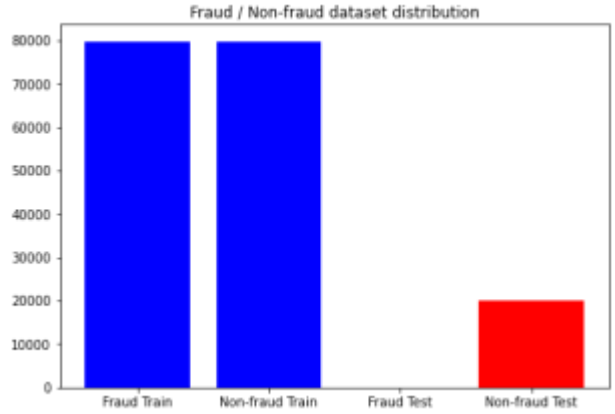
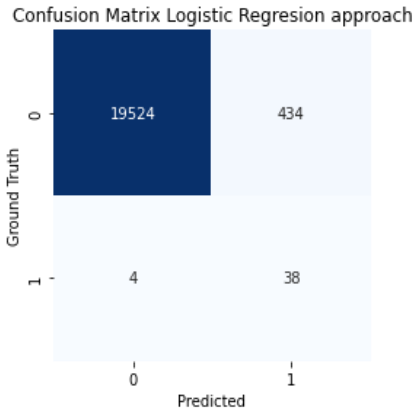


Figure 12. Dataset Oversampled

Logistic Regression

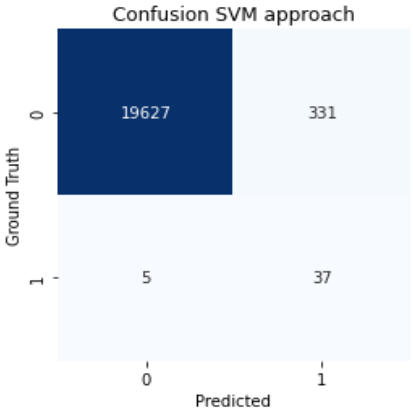


	precision	recall	f1-score	support
no fraud	1.00	0.98	0.99	19958
fraud	0.08	0.90	0.15	42
accuracy			0.98	20000
macro avg	0.54	0.94	0.57	20000
weighted avg	1.00	0.98	0.99	20000

accuracy = 97.81%

Figure 13. Confusion matrix RL oversampling

SVM

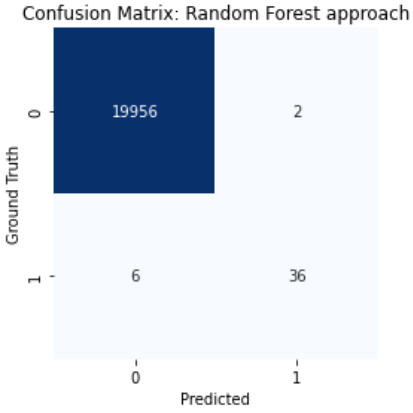


	precision	recall	f1-score	support
no fraud	1.00	0.98	0.99	19958
fraud	0.10	0.88	0.18	42
accuracy			0.98	20000
macro avg	0.55	0.93	0.59	20000
weighted avg	1.00	0.98	0.99	20000

accuracy = 98.32%

Figure 14. Confusion Matrix S VM oversampling

RF

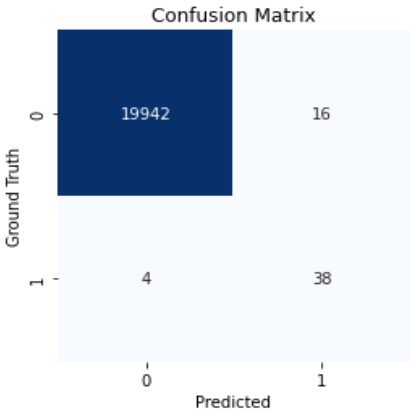


	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.95	0.86	0.90	42
accuracy			1.00	20000
macro avg	0.97	0.93	0.95	20000
weighted avg	1.00	1.00	1.00	20000

accuracy = 99.96%

Figure 15. Confusion matrix RF oversampling

DNN1



	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.70	0.90	0.79	42
accuracy			1.00	20000
macro avg	0.85	0.95	0.90	20000
weighted avg	1.00	1.00	1.00	20000

accuracy = 99.9 %

Figure 16. Confusion matrix DNN1 oversampling

AUROC

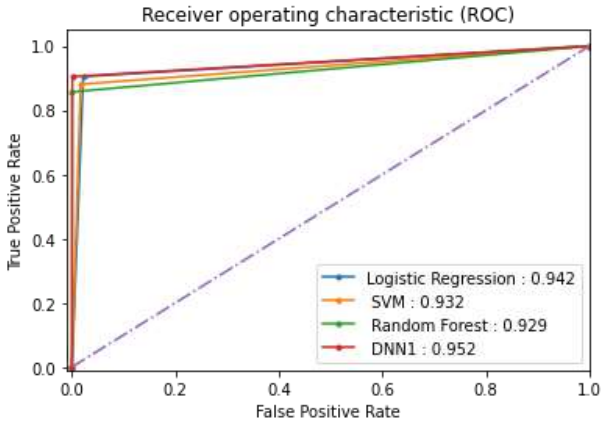


Figure 17. AUROC Curves (LR, SVM, RF, DNN1) oversampling

AUPRC

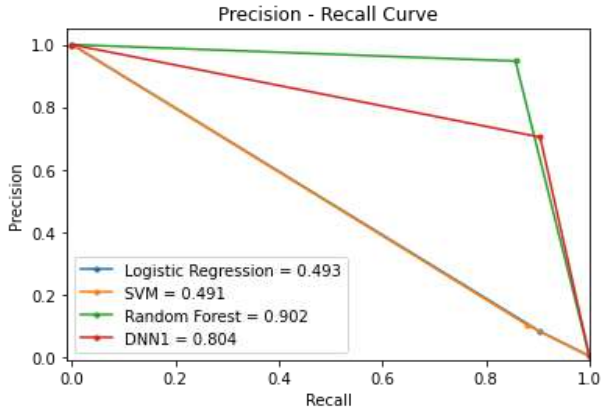


Figure 18. AUPRC Curves (LR, SVM, RF, DNN1) oversampling

2. Undersampling

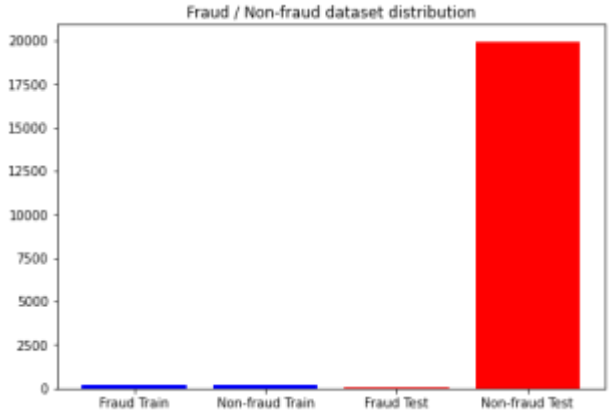
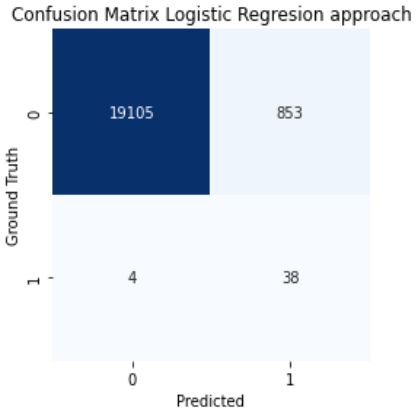


Figure 19. Dataset Undersampling

Logistic Regression

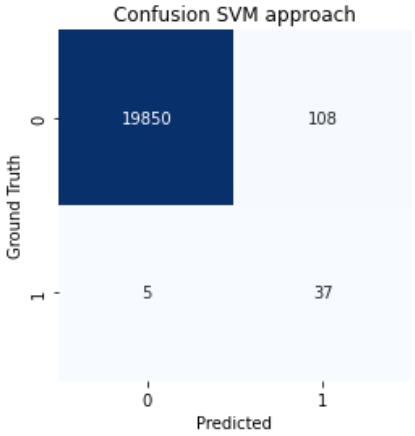


	precision	recall	f1-score	support
no fraud	1.00	0.96	0.98	19958
fraud	0.04	0.90	0.08	42
accuracy			0.96	20000
macro avg	0.52	0.93	0.53	20000
weighted avg	1.00	0.96	0.98	20000

accuracy = 95.71%

Figure 20. Confusion matrix RL undersampling

SVM

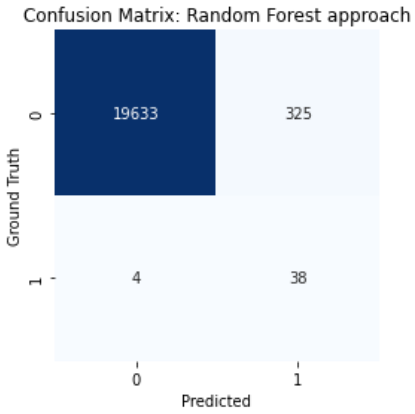


	precision	recall	f1-score	support
no fraud	1.00	0.99	1.00	19958
fraud	0.26	0.88	0.40	42
accuracy			0.99	20000
macro avg	0.63	0.94	0.70	20000
weighted avg	1.00	0.99	1.00	20000

accuracy = 99.44 %

Figure 21. Confusion matrix SVM undersampling

RF

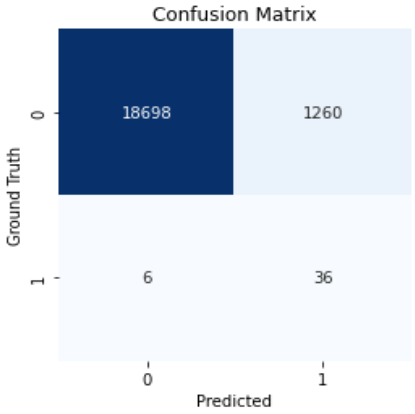


	precision	recall	f1-score	support
no fraud	0.92	0.98	0.95	45
fraud	0.98	0.91	0.94	45
accuracy			0.94	90
macro avg	0.95	0.94	0.94	90
weighted avg	0.95	0.94	0.94	90

accuracy = 98.36%

Figure 22. Confusion matrix RF undersampling

DNN1



	precision	recall	f1-score	support
no fraud	1.00	0.94	0.97	19958
fraud	0.03	0.86	0.05	42
accuracy			0.94	20000
macro avg	0.51	0.90	0.51	20000
weighted avg	1.00	0.94	0.97	20000

accuracy = 93.670%

Figure 23. Confusion matrix DNN1 undersampling

AUROC

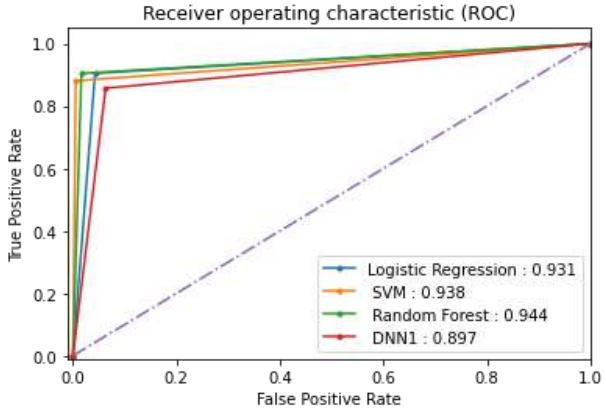


Figure 24. AUROC Curves (LR, SVM, RF, DNN1) undersampling

AUPRC

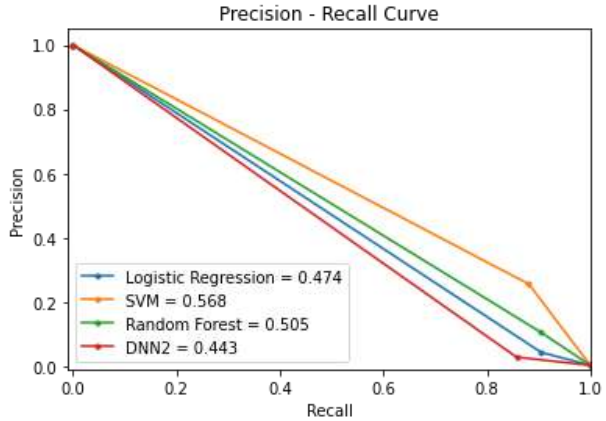


Figure 25. . AUPRC Curves (LR, SVM, RF, DNN1) undersampling

In the oversampling method, exact copies from minority class are randomly distributed all over the training dataset. This may increase the likelihood of overfitting, especially for higher over-sampling rates. Furthermore, the size of the dataset increased which demands more computational resources and more storage space.

In the case of the undersampling method, since random samples from the majority class are deleted, it ends up reducing the number of samples of the majority class. In this case, decisive data could be deleted leading to potential data loss.

After verifying the results obtained in both oversampling and undersampling, we can conclude that neither the oversampling nor undersampling method helped to improve the performance of the models if we compare them with the models trained with the original dataset. As it was mentioned, the oversampling is just duplicating data from the minority class so models end up covering replicated samples, while the other deletes samples from the majority class which could delete important samples. The f1-score is degraded by lower precision in both methods, although recall has improved slightly but not significantly.

However, if we needed to choose a method between oversampling and undersampling, we could say that oversampling performs slightly better than undersampling. An interesting finding is that using oversampling and undersampling helps to improve the detection of True Positives and, therefore, improve the detection of False Negatives.

(viii) Follow (vii). Do you know other methods to deal with imbalanced data? (you can explain your ideas or directly implement them, the latter gets more points.)

Focal Loss

Besides of the oversampling and undersampling methods, we can address the imbalance dataset by the error-aware loss. The use of the focal loss was proposed first for dense object task. It enables training highly accurate dense object detectors with an imbalance between foreground and background classes at 1:1000 scale.

Focal loss is a Cross-Entropy Loss that weighs the contribution of each sample to the loss based on the classification error. The main idea of focal loss is that if a sample is classified correctly, its contribution to the loss decreases. Therefore, the problem of class imbalance is addressed correctly by focusing in the minor classes. Focal Loss comes represented by the following equation:

$$FL(p_t) = -\alpha_t(1 - p_t)^{\gamma}\log(p_t)$$

Where $p_t = \begin{cases} p, y = 1 \\ (1-p), otherwise \end{cases}$

α and γ are hyperparameters that need to be tuned. $\gamma \geq 0$ is a focusing parameter that adjusts the rate to down weight the easily classified examples. α is a balancing parameter used to increase the importance of minority class examples. The modulation of these parameters will reduce the impact of the majority class on loss.

This Focal loss was improved by Sergievskiy that introduced a novel function named reduced focal loss (RFL) function for object detection in satellite imagery. This function modified the focal loss function to reduce

the contribution of well-classified examples and soften the response of the loss function to the minority samples. The equation for the RFL is described as follows:

$$RFL(p_t) = -f_r(p_t, th) \ \log(p_t)$$

Where $f_r(p_t, th) = \begin{cases} 1, & p_t < th \\ (1-p_t)^\gamma, & p_t \geq th \end{cases}$

$(1-p_t)^\gamma$ is the modulating factor.

The RFL was implemented at the end of the google collab.

The RFL is in the class FocalLoss and for the experiment the following model was used.

DNNT(
(l1): Linear(in_features=32, out_features=32, bias=True)
(norm1): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(act1): ReLU()
(drop1): Dropout(p=0.3, inplace=False)
(l2): Linear(in_features=32, out_features=16, bias=True)
(norm2): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
(act2): ReLU()
(drop2): Dropout(p=0.3, inplace=False)
(l3): Linear(in_features=16, out_features=1, bias=True)
(act_out): Sigmoid()

For the Focal logistic implementation, another class was included FocalLoss(nn.Module). In this class the Reduced Focal Loss (RFL) was defined. The results are displayed as follows:

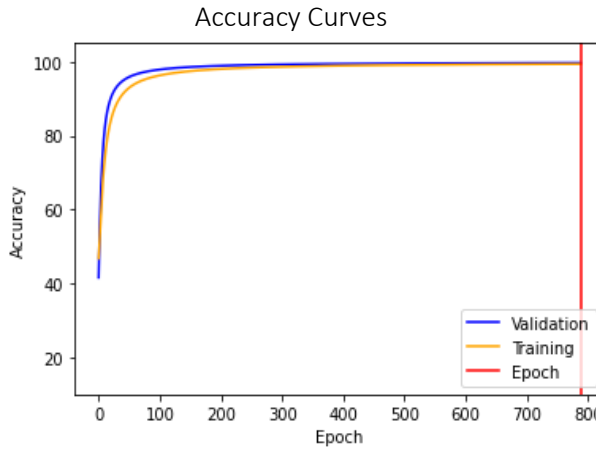


Figure 26. Train/Validation accuracy DNNT

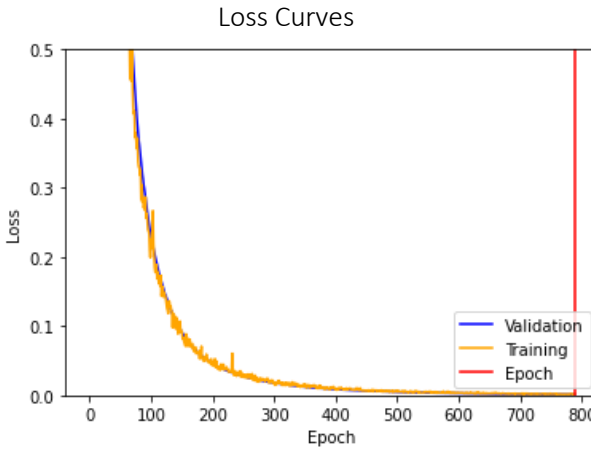
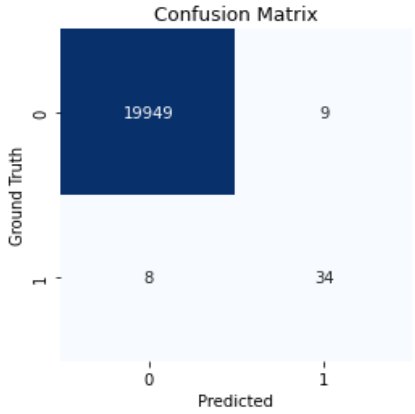


Figure 27. Train/Validation loss DNNT

Confusion Matrix



Metrics

	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.79	0.81	0.80	42
accuracy			1.00	20000
macro avg	0.90	0.90	0.90	20000
weighted avg	1.00	1.00	1.00	20000

Accuracy: 99.915%

Figure 28. Confusion Matrix DNN1

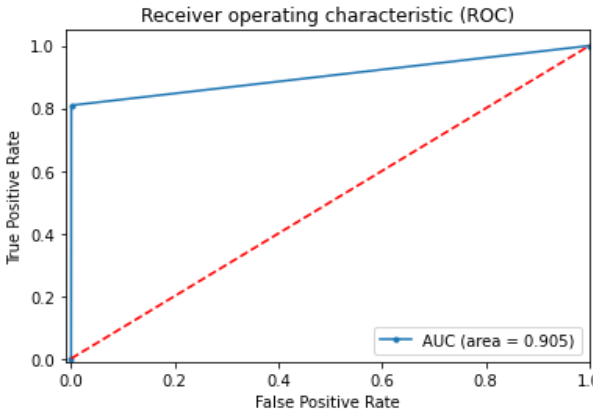


Figure 29. AUROC Focal Loss DNNT

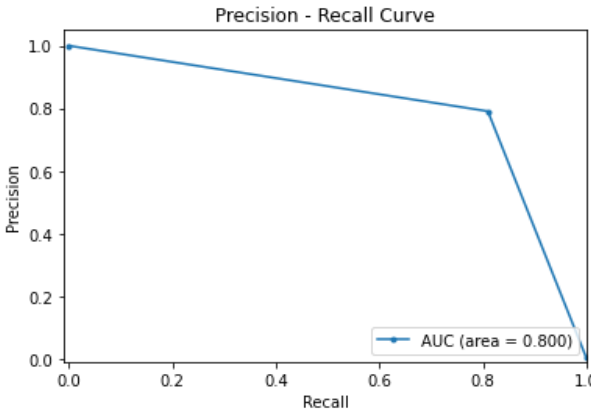


Figure 30. AUPRC Focal Loss DNNT

SMOTE: Synthetic Minority Over-sampling technique

This method was also implemented at the end of the google collab.
SMOTE is an oversampling technique that works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space, and drawing a new sample at a point along that line.
Linear interpolation is used to generate virtual training records for minorities. For each example in the minority class, one or more of the k-nearest neighbors are randomly selected. Following the oversampling process, the data can be reconstructed, then classified using several classification models.

The original SMOTE paper suggested combining SMOTE with random undersampling of majority classes.

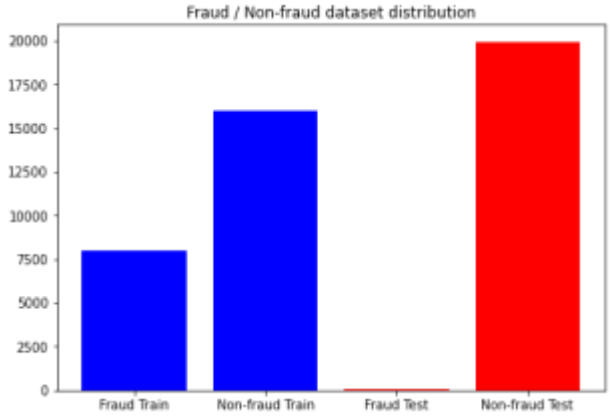
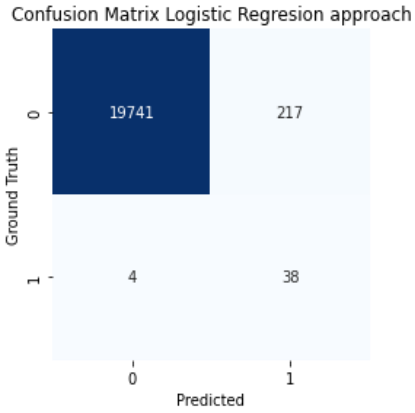


Figure 31. SMOTE Dataset

Logistic Regression

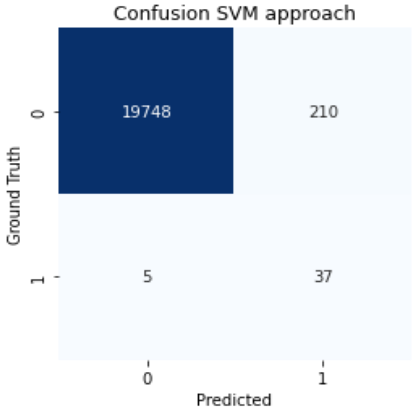


	precision	recall	f1-score	support
no fraud	1.00	0.99	0.99	19958
fraud	0.15	0.90	0.26	42
accuracy			0.99	20000
macro avg	0.57	0.95	0.63	20000
weighted avg	1.00	0.99	0.99	20000

accuracy 98.89%

Figure 32. Confusion matrix RL SMOTE

SVM

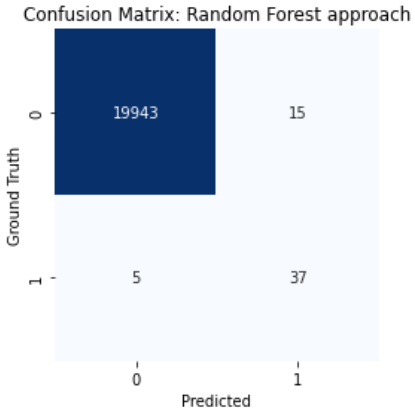


	precision	recall	f1-score	support
no fraud	1.00	0.99	0.99	19958
fraud	0.15	0.88	0.26	42
accuracy			0.99	20000
macro avg	0.57	0.94	0.63	20000
weighted avg	1.00	0.99	0.99	20000

accuracy = 98.92%

Figure 33. Confusion matrix - SVM SMOTE

RF

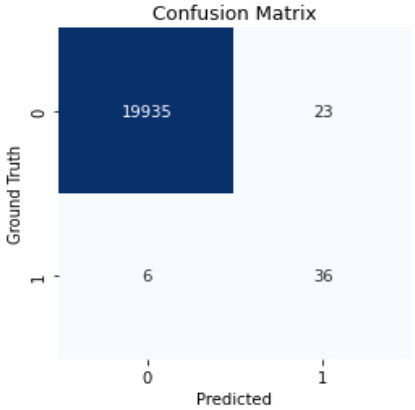


	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.71	0.88	0.79	42
accuracy			1.00	20000
macro avg	0.86	0.94	0.89	20000
weighted avg	1.00	1.00	1.00	20000

accuracy = 99.9%

Figure 34. Confusion matrix RF SMOTE

DNN1



	precision	recall	f1-score	support
no fraud	1.00	1.00	1.00	19958
fraud	0.61	0.86	0.71	42
accuracy			1.00	20000
macro avg	0.80	0.93	0.86	20000
weighted avg	1.00	1.00	1.00	20000

accuracy = 99.855%

Figure 35. Confusion matrix DNN1 SMOTE

AUROC

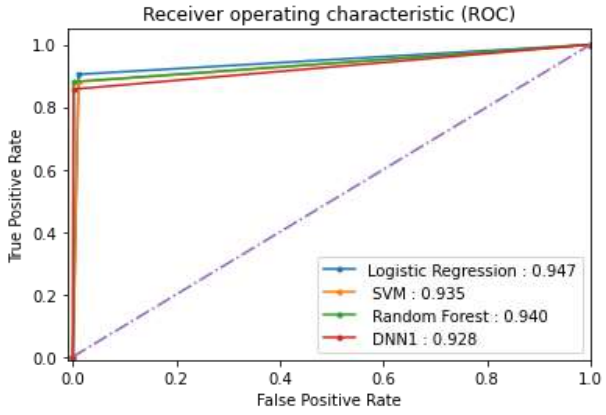


Figure 36. AUROC Curves (LR, SVM, RF, DNN1) SMOTE

AUPRC

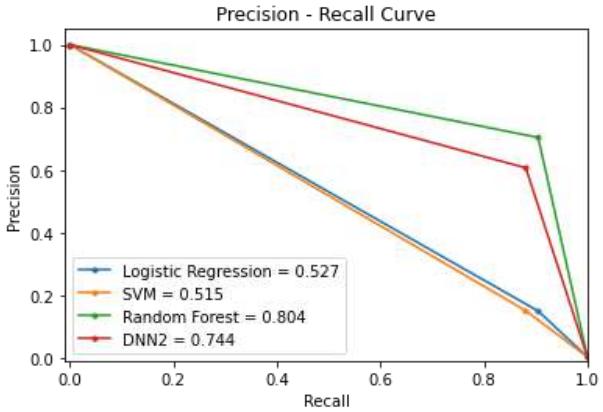


Figure 37. . AUPRC Curves (LR, SVM, RF, DNN1) SMOTE