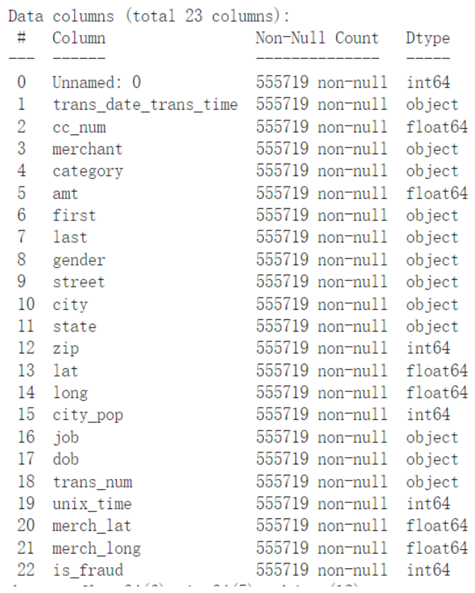
CS-UY 4563
Final Project Write-up

Elaine Li, Terry Mu
NetID: zl4490, xm2204
Email: zl4490@nyu.edu, xm2204@nyu.edu
Section **A**

Experiment Date: April 20, 2024
Submission Date: April 29, 2024

Introduction

This “credit card fraud prediction” project aims to classify fraud and non-fraud credit cards. The dataset from Kaggle has 555719 examples. It consists of 21 features, customer identification number (cc_num), merchant, transaction type (category), transaction amount, cardholder's first name, cardholder's last name, gender, street, city, state, zip, latitude, longitude, city population, date of birth, transaction number, transaction timestamp, merchant’s latitude, and merchant’s longitude, and one target which indicates whether it is a fraud (1) or non-fraud(0), as shown in Figure 1.



```
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  float64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
 19  unix_time              555719 non-null  int64
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64
```

Figure 1: The information on this dataset

The advantage of this dataset is that it doesn’t have null values. However, the data is extremely imbalanced. There are 553574 non-fraud examples and only 2145 fraud examples, around 0.386% of the examples.

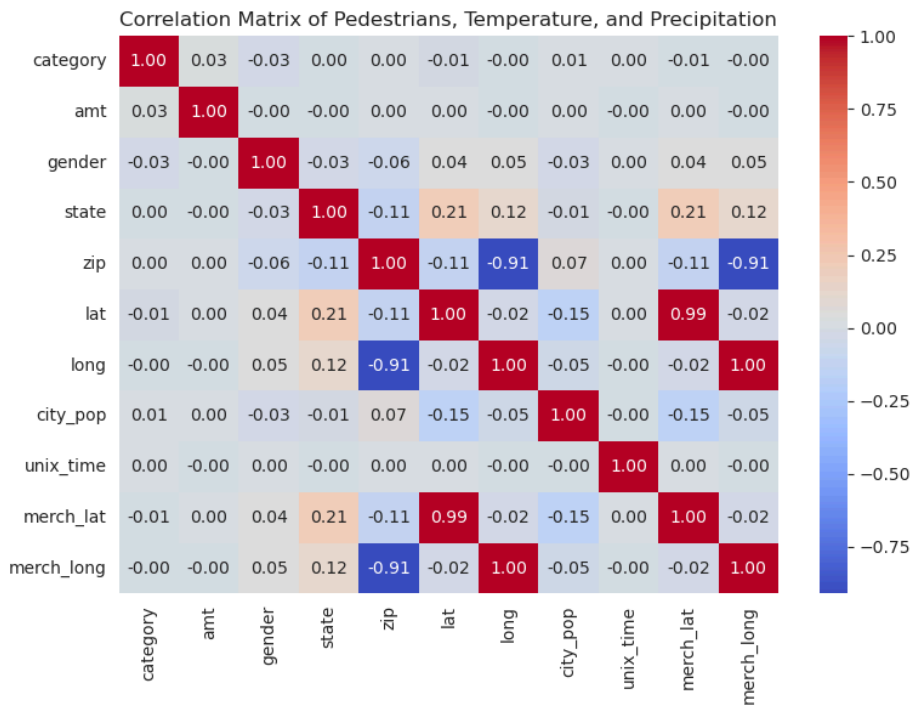
We utilized three models to train the data: logistic regression, Support Vector Machine (SVM), and neural network. For each model, we attempted different transformation methods, regularization types, and regularization parameters to see how accuracy and recall are affected.

Preprocessing

Of the 21 features, only 8 features are selected at last. We first checked among the object type (categorical) data to find those that have more than 100 unique values. For those object features that have too many unique entries, we can say that the target has little to do with those features. For example, if we have 10,000 examples, and each example has a different 'city' feature, then we can say that the target's pattern is not related to the ‘city’. So, we can safely drop such features.

Then, we encoded the categorical features such as category, gender, and state by converting those features from object to categorical/ordinal first and then using *cat.code* to assign each category a numerical value.

To further clean the data, we examined the duplicate features by looking at the features that have a high correlation with each other. We visualized a correlation matrix of all the left features (Figure 2). It was shown that the cardholder’s longitude (long) merchant’s longitude (merch_long) cardholder’s latitude (lat) and the merchant’s latitude (merch_lat) correlate between 1 and 0.99. Zip Code (zip) has a correlation of -0.91 with both long and merch_long. Therefore, we determined that duplication existed among those features which could lead to overfitting in future model training. Therefore, we removed long, lat, and zip and only retained merch_long and merch_lat.



	category	amt	gender	state	zip	lat	long	city_pop	unix_time	merch_lat	merch_long
category	1.00	0.03	-0.03	0.00	0.00	-0.01	-0.00	0.01	0.00	-0.01	-0.00
amt	0.03	1.00	-0.00	-0.00	0.00	0.00	-0.00	0.00	-0.00	0.00	-0.00
gender	-0.03	-0.00	1.00	-0.03	-0.06	0.04	0.05	-0.03	0.00	0.04	0.05
state	0.00	-0.00	-0.03	1.00	-0.11	0.21	0.12	-0.01	-0.00	0.21	0.12
zip	0.00	0.00	-0.06	-0.11	1.00	-0.11	-0.91	0.07	0.00	-0.11	-0.91
lat	-0.01	0.00	0.04	0.21	-0.11	1.00	-0.02	-0.15	0.00	0.99	-0.02
long	-0.00	-0.00	0.05	0.12	-0.91	-0.02	1.00	-0.05	-0.00	-0.02	1.00
city_pop	0.01	0.00	-0.03	-0.01	0.07	-0.15	-0.05	1.00	-0.00	-0.15	-0.05
unix_time	0.00	-0.00	0.00	-0.00	0.00	0.00	-0.00	-0.00	1.00	0.00	-0.00
merch_lat	-0.01	0.00	0.04	0.21	-0.11	0.99	-0.02	-0.15	0.00	1.00	-0.02
merch_long	-0.00	-0.00	0.05	0.12	-0.91	-0.02	1.00	-0.05	-0.00	-0.02	1.00

Figure 2. Correlation Matrix

After the selection of features was done, we split the data into 272302 examples for the training set and 116701 examples for the validation set in this project, and then X_train and X_val were scaled by the standard scaler.

As we discovered before, the dataset contains an extremely small portion of fraud examples. Therefore, we took advantage of undersampling and oversampling to balance the data. Specifically, we utilized SMOTE oversampling for both logistic regression and neural network, and random undersampling for SVM as it was more time-consuming.

Model Training

Logistic Regression

In the logistic regression model, we tried 13 different parameter settings.

As for transformation, in addition to no transformation, we applied polynomial transformation with a power of 2 to see how prediction changed when the number of features increased that could better fit non-linearly separable data. In addition, L1 and L2 regularization with lambda value (C) = [0, 0.1, 1, 0.5, 1.5, 2] was added to the model.

Table 1. Logistic Regression Parameter settings

Trail #	C (lambda)	transformation	regularization
1	0.1	non	l2
2	1	non	l2
3	0.5	non	l2
4	1.5	non	l2
5	2	non	l2
6	0.1	non	l1
7	1	non	l1
8	0	non	non
9	0.1	poly, power=2	l2
10	1	poly, power=2	l2
11	0.1	poly, power=2	l1
12	1	poly, power=2	l1
13	0	poly, power=2	non

SVM

In the SVM model, we used sklearn’s SVC and tried 12 different parameter settings.

Linear, polynomial, and radial basis function (RBF) kernels were utilized for transformation in this model. Similar to the polynomial kernel, RBF is also good at determining non-linear decision boundaries, but it is more effective in capturing complex relationships in data than the polynomial kernel. We made C = [1, 10, 0.1, 0.5, 5, 50] as our hyperparameter for regularization.

Table 2. SVM Parameter Settings

Trail #	C (lambda)	kernel (trans)
1	1	rbf
2	10	rbf
3	0.1	rbf
4	1	linear
5	10	linear
6	0.1	linear
7	1	poly
8	10	poly
9	0.1	poly
10	0.5	poly
11	5	poly
12	50	poly

Neural Network

In the neural network, we also tried 12 different parameter settings.

For the neural network, we use the API provided by Keras under TensorFlow.Our neural network adopts a four-layer structure: an input layer, a hidden layer with 64 nodes, a hidden layer with 32 nodes, and an output layer with one node. The activation equations of both hidden layers are RELU, and the activation equation of the output layer is the sigmoid equation. The whole training process is divided into 10 epochs, and each epoch is divided into 16955 steps.

Table 3. Neural Network Parameter Settings

Trail #	C (lambda)	transformation	regularization
1	0	non	non
2	0.001	non	l2
3	0.1	non	l2
4	0.005		l2
5	0.5		l2
6	0.001	non	l1
7	0.01 (0.1 caused err)	non	l1
8	0	poly, power=2	non
9	0.001	poly, power=2	l2
10	0.1	poly, power=2	l2
11	0.001	poly, power=2	l1
12	0.01 (0.1 caused err)	poly, power=2	l1

Result

We displayed the accuracy, precision, recall, and loss for both the training set and validation set to examine the performance. However, in this project, we preferred to use validation recall to evaluate how well the model did. As the positive values, which means fraud, are only 0.386% of the entire data, it is improper to simply measure the performance using the entire sample. For instance, even though all the positive values are predicted falsely, the accuracy can reach around 99.7% according to equation (1).

$$Accuracy = \frac{\#TP+\#TN}{\#TP+\#FP+\#TN+\#FN} \qquad (1)$$

On the other hand, recall in this case is a better measurement. Throughout the project, we aim to use a set of information such as geographic location, price, and type to determine if the transaction has a tendency to fraud. With this goal in mind, it is easy to see that the cost of a false alarm is much smaller than the cost of a misjudgment, i.e., "it is actually a fraud but the prediction is that there is no fraud". Like a fire alarm, there is no problem with a false fire alarm, the big deal is that everyone leaves the house and comes back, a false alarm; but if the fire alarm is not triggered when there is a real fire, then there is a big problem. Similarly, if it's a false alarm, then all the user needs to do is to check the status of the credit card to make sure there's nothing wrong with it, and then it's fine, but if it's a misdiagnosis of fraud as non-fraud, then the user is going to be in for some very serious property damage. Therefore, in this case, we should emphasize recall, which is the ratio of the number of correctly predicted frauds to the number of actual frauds (2); we would like to see this value to be 1 because that would mean that there are no cases of misclassification of frauds as non-frauds (the number of false negatives).

$$Recall = \frac{\#TP}{\#TP+\#FN} \qquad (2)$$

In the following tables, the blue cells represent the highest validation recall and green cells represent the highest validation accuracy.

Logistic Regression

As shown in Table 4, we obtained the highest validation recall 77.8% when no transformation was performed and the highest validation accuracy 97.1% when using polynomial transformation. In logistic regression, other parameters such as regularization type and the hyperparameter C did not affect the result. (reason?)

Table 4. Results of Logistic Regression

Trail #	C	transformation	regularization	train accuracy	train precision	train recall	train loss	Val accuracy	Val precision	Val recall	val loss
1	0.1	non	l2	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.333
2	1	non	l2	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
3	0.5	non	l2	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
4	1.5	non	l2	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
5	2	non	l2	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
6	0.1	non	l1	0.846	0.921	0.758	5.535	0.935	0.045	0.778	2.331
7	1	non	l1	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
8	0	non	non	0.846	0.921	0.758	5.536	0.935	0.045	0.778	2.334
9	0.1	poly, power=2	l2	0.852	0.963	0.732	5.338	0.971	0.096	0.763	1.043
10	1	poly, power=2	l2	0.852	0.963	0.732	5.339	0.971	0.096	0.763	1.043
11	0.1	poly, power=2	l1	0.852	0.963	0.732	5.338	0.971	0.096	0.763	1.043
12	1	poly, power=2	l1	0.852	0.963	0.732	5.339	0.971	0.096	0.763	1.043
13	0	poly, power=2	non	0.852	0.963	0.732	5.339	0.971	0.096	0.763	1.043

SVM

According to Table 5, both validation recall and accuracy occurred when applying polynomial transformation. The highest validation recall was 73.7% with C=50, while the highest validation accuracy was 98.3% with C=0.1.

It is observed that when the transformation is complex, specifically for RBF and polynomials, when the regularization parameter increases, the validation recall also increases. It might be because the higher regularization alleviates the overfitting of the model. As C increases, more penalties would be given for large coefficients. Therefore, this model could better generalize the unseen data in validation, resulting in a decrease in variance and an increase in bias.

Table 5. Results of SVM

Trail #	C	kernel (trans)	train accuracy	train precision	train recall	train loss	Val accuracy	Val precision	Val recall	val loss
1	1	rbf	0.865	0.973	0.750	4.876	0.979	0.119	0.708	0.760
2	10	rbf	0.877	0.983	0.877	4.421	0.970	0.086	0.721	1.081
3	0.1	rbf	0.857	0.955	0.750	5.144	0.968	0.080	0.705	1.145
4	1	linear	0.850	0.932	0.756	5.389	0.950	0.053	0.721	1.789
5	10	linear	0.851	0.932	0.757	5.377	0.950	0.053	0.721	1.789
6	0.1	linear	0.850	0.932	0.756	5.389	0.951	0.054	0.721	1.775
7	0.1	poly	0.753	0.967	0.525	8.888	0.983	0.118	0.521	0.597
8	0.5	poly	0.830	0.953	0.694	6.124	0.966	0.071	0.666	1.234
9	1	poly	0.850	0.952	0.737	5.424	0.958	0.062	0.705	1.503
10	5	poly	0.864	0.953	0.766	4.899	0.949	0.052	0.721	1.839
11	10	poly	0.864	0.951	0.768	4.899	0.948	0.051	0.726	1.890
12	50	poly	0.864	0.951	0.767	4.911	0.946	0.050	0.737	1.948

Neural Network

In the neural network model, no transformation and L1 regularization with C=0.001 gave the significantly highest validation recall of 95.6%. Polynomial transformation with no regularization resulted in the highest validation accuracy of 98.7% (table 5).

Through Table 5, we can observe that for trials with the same transformation (or no transformation) and regularization type, as lambda increases, the validation recall decreases. This might be due to the underfitting caused by the high regularization parameter. A higher C leads to a stronger regularization, hence decreasing the model complexity. Since the validation recall is higher when C is smaller, it can be concluded that the data has a complex pattern. However, since the model works best (the highest val_recall) with no transformation, the data pattern might not be too complex.

Table 6. Results of Neural Network

Trail #	C	transformation	regularization	train accuracy	train precision	train recall	train loss	Val accuracy	Val precision	Val recall	val loss
1	0	non	non	0.978	0.987	0.970	0.055	0.986	0.193	0.826	0.045
2	0.001	non	l2	0.970	0.966	0.973	0.116	0.964	0.090	0.901	0.129
3	0.005	non	l2	0.947	0.962	0.931	0.178	0.963	0.087	0.895	0.169
4	0.1	non	l2	0.855	0.953	0.747	0.373	0.963	0.076	0.763	0.273
5	0.5	non	l2	0.837	0.898	0.761	0.437	0.914	0.035	0.785	0.432
6	0.001	non	l1	0.941	0.937	0.946	0.188	0.935	0.054	0.956	0.219
7	0.01	non	l1	0.854	0.941	0.756	0.391	0.952	0.060	0.769	0.370
8	0	poly, power =2	non	0.993	0.989	0.996	0.024	0.987	0.200	0.774	0.052
9	0.001	poly, power =2	l2	0.977	0.967	0.986	0.097	0.966	0.091	0.870	0.122
10	0.1	poly,	l2	0.888	0.939	0.831	0.303	0.946	0.056	0.809	0.258

		power =2									
11	0.001	poly, power =2	l1	0.961	0.968	0.955	0.156	0.967	0.095	0.870	0.144
12	0.01	poly, power =2	l1	0.857	0.856	0.859	0.329	0.856	0.023	0.846	0.360

To sum up, the best model in this project is the neural network model no matter whether the performance measurement is accuracy or recall. The best recall is 95.6% and the best accuracy is 98.7%. The data shown in the tables also shows that both logistic regression and neural network generated the highest recall when no transformation was performed. Therefore, the feature data tends to be linearly separable.

Analysis of weights

The following figures are the weights of Trail 7 of the logistic regression and the weights of Trail 6 of the SVM (Figure 3 and Figure 4) with no transformation (The neural network has too many parameters to visualize compare and analyze afterward, so I won't put a graph of the parameters of the neural network here).

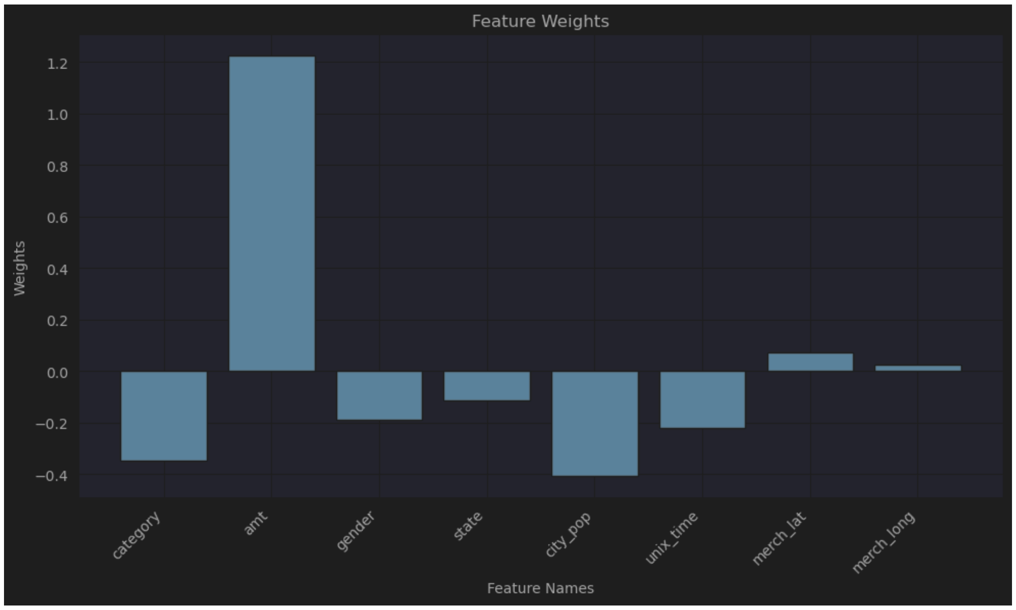


Figure 3: The weights of the Trail 7 of the logistic regression model

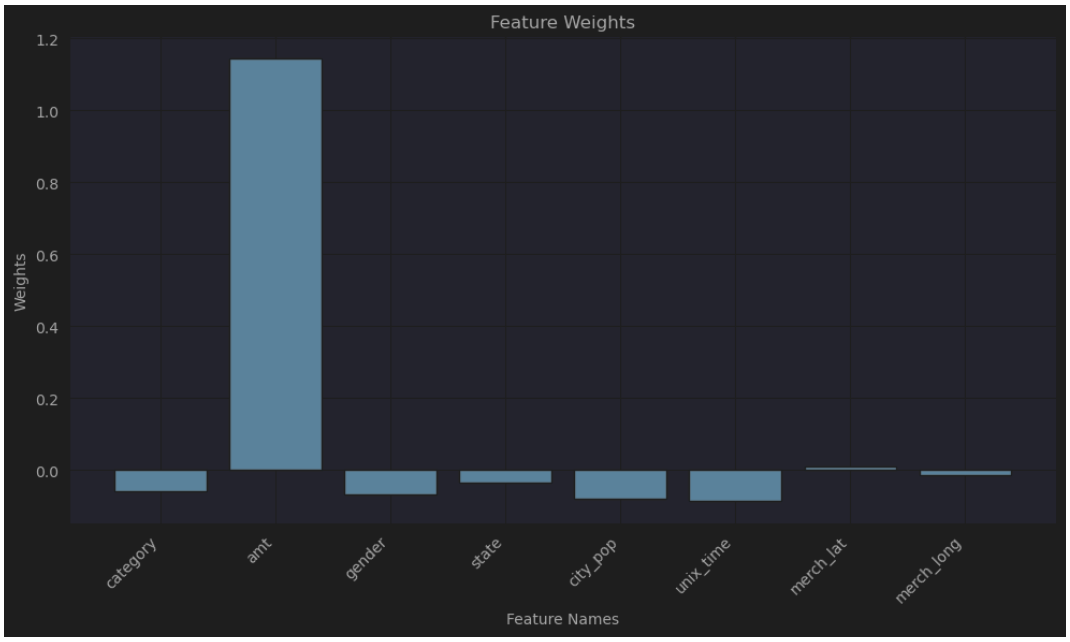


Figure 4: The weights of Trail 6 of the SVM model

From Figure 3 and Figure 4, it can be seen that the weights did not change tremendously between the two models: both models have a maximum weight between 1.0 to 1.2, which corresponds to the amount of money in the transaction. In both models, the latitude and the longitude of the merchant does not have a significant weight. Comparing the two models, although the weights of the other features are somewhat slightly changed, these weights, in both models, do not play a dominant role. In summary, the amount of money in a transaction determines whether or not the transaction will be fraudulent. This is reasonable because in reality, for a transfer of a significant amount, it is likely to be sent by a transferor who has been subjected to a criminal scam (such as those claiming to have investments with unrealistic benefits). The merchant's longitude and latitude did not play an important role in determining the result because scammers can pop up anywhere.

Conclusion

About This Project

From the tables listed above, we can find the best validation accuracy and the best validation recall of each of the logistic regression, SVM, and neural networks, as shown in Table 7.

Table 7. The best accuracy (row 2) and recall (row 3) of each group

Logistic Regression	SVM	Neural Networks
0.971	0.983	0.987
0.778	0.737	0.956

From the table, it can be seen that all of the three models have a high accuracy. However, we cannot conclude that the models perform well directly from the high accuracy. Considering the imbalance of the dataset (only less than 1% of data are 1's and the rest are 0's), if someone codes another predictor that is so naive that will always predict 0 to each example, then they will still have an accuracy of 99.7%, which can also be seen from the confusion matrix (Figure 5).

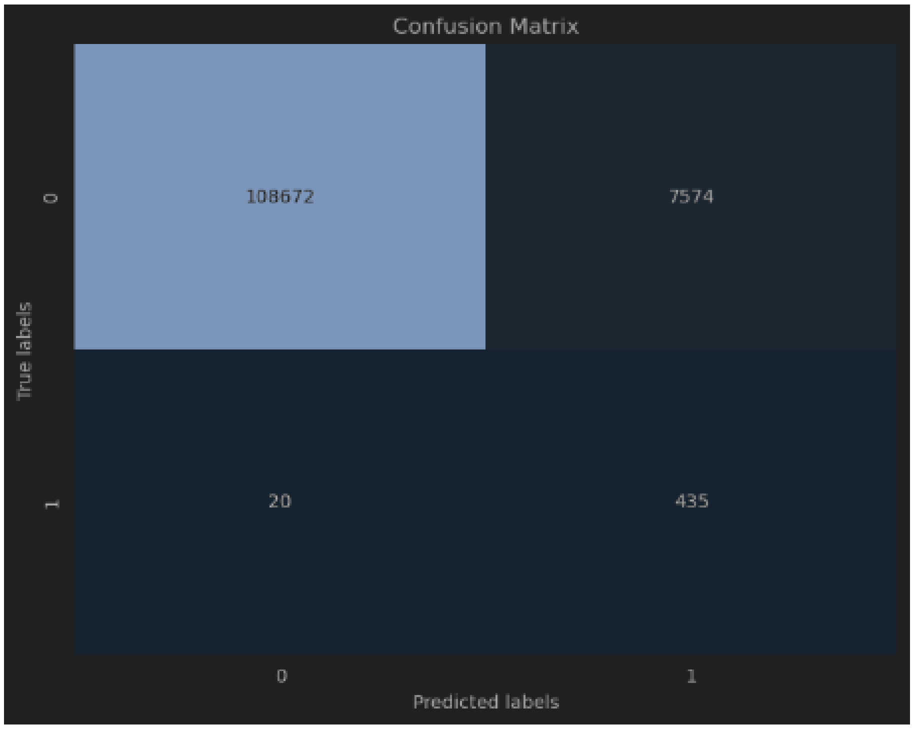


Figure 5: Confusion matrix of the trial with the best recall (Trail 8 of neural networks).

Combined with the importance of recall, as we mentioned before, if we were to make a summary, we should say that only neural networks satisfy our need to train this model because it did best in lowering the number of false negatives. In other words, in practice, training and using a neural network to predict will maximize the protection of user property security.

Further Improvements and Lessons Learned

Though this experiment was successful, some places can still be improved in the future. Firstly, because of the device, training SVMs with a training set that is too large will take too long, which leads us to use RandomUnderSampler to balance the training set used to train the SVMs; using UNDER sampler will make the training set too small, leading to unsatisfactory results.

Therefore, SVM, a powerful tool, should have been utilized more in this project. Therefore, when training models in the future, more appropriate devices should be selected; if capable, it is even more important to use those with graphics acceleration. In addition, trying neural networks with more layers and more neurons is also a good way to improve validation recall. In addition, while we were working on this project, we found some models that are more suitable for this dataset but more complex, such as random forests and decision trees. We should also learn these models and use more powerful tools to achieve better results.

We did learn a lot from this project. The most important one is that we put the theory into practice, and gained a better understanding of logistic regression, SVM, and neural networks. Also, if we become a dataset producer in the future, we need to learn the advantages of this dataset: no null values; of course, we also need to avoid the disadvantages of this dataset: we have to try to ensure that our dataset of data balance so that we can make our dataset more usable, convenient for the user.

Works Cited

Brownlee, Jason. “SMOTE for Imbalanced Classification with Python.” *Machine Learning Mastery*, 16 Jan. 2020, machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/.

“Credit Card Fraud Prediction.” *Www.kaggle.com*, www.kaggle.com/datasets/kelvinkelue/credit-card-fraud-prediction/data.

“RandomUnderSampler — Version 0.9.0.” *Imbalanced-Learn.org*, imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html.

scikit-learn. “Sklearn.linear_model.LogisticRegression — Scikit-Learn 0.21.2 Documentation.” *Scikit-Learn.org*, 2014, scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

---. “Sklearn.svm.SVC — Scikit-Learn 0.22 Documentation.” *Scikit-Learn.org*, 2019, scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.