Developing a credit card fraud detection system

Andrei Beiu, Yasin Karyagdi, Kian Sie, Stefan Sprakel, Luna de Vries

29/03/2024

Abstract

Fraudulent transactions are a significant challenge for credit card companies to discover since the proportion of fraudulent over non-fraud transactions is very low. The identification of these dishonest transactions is significant in preventing financial losses and preserving customer trust. In this project, we apply two machine learning techniques to develop a fraud detection system, to predict whether a transaction is fraudulent or not. Specifically, we use Support Vector Machines (SVM) and Logistic Regression (LR) algorithms to build models that spot fraudulent transactions. As said before these algorithms will have to deal with a highly imbalanced dataset. This dataset consists of numeral features consisting of transaction amount, merchant information, and transaction time. By the use of data pre-processing techniques and feature selection, we prepare the dataset for model training. The effectiveness of SVM and Logistic Regression in correctly predicting fraudulent transactions is compared through the metrics of accuracy, precision, recall, and other relevant metrics. Our discoveries offer valuable insight into the capacity of supervised machine learning algorithms to identify fraudulent activities within a significantly imbalanced setting, mirroring real-world circumstances.

1 Introduction

Credit cards are one of the most common forms of payment. With billions of dollars worth of transactions happening daily, a single unwanted transaction could cost a person thousands of dollars. As a result, protocols were needed to prevent unwanted or unauthorized transactions from happening. These ranged from basic ones, such as blocking the physical card when reported lost or stolen, and implementing 2-factor authentication for purchases; to more complex procedures, such as the machine-aided fraud detection methods discussed in this paper.

Although fraudulent transactions make up a very small portion of total transactions, the overall value of these transactions still totals nearly 2 billion euros within the EU alone, as of 2019[16]. Therefore, the goal is to create a fraud detection system that can successfully detect fraudulent transactions. However, because fraudulent transactions make up a small fraction of total transactions, we aim to avoid alerting the customer every time they make a legitimate transaction.

This imbalance in real versus fraudulent transactions is reflected in our dataset, where fraudulent transactions make up roughly only 0,1221% of cases. Although the dataset is synthetic, it is designed to emulate real-world fraud levels, with a total number of $24,380,900$ instances. The research question for this project states: "What machine learning techniques are most effective for predicting fraudulent transactions in a credit card dataset, and how do their performances compare?". We will use the Support Vector Machine (SVM) and Logistic Regression (LR) algorithms to make predictions on our dataset. Both of these algorithms will be employed to create a fraud detection classifier and will be compared to determine which classifier best fits the data. Various techniques were also used to create our final classifier, such as cross-validation and grid search for optimizing performance, as well as data pre-processing methods. These include feature selection, missing values compensation, categorical feature encoding, and numerical feature scaling.

2 Background

Many research has been done on the topic of fraud detection based on machine learning methods exploring various approaches from supervised to unsupervised and hybrid models. Previous studies have examined the effectiveness of different machine learning algorithms and fraud detection systems, highlighting several challenges and areas for improvement. This literature review provides an overview of the current state of research in credit card fraud detection, highlighting key findings, challenges, and opportunities for future research. One significant challenge highlighted in the literature is class imbalance. Because this topic contains high class imbalance as stated previously, researches have to deal with this to develop relevant models. Fraudulent transactions are vastly outnumbered by legitimate ones in transaction datasets. This imbalance causes difficulties for traditional machine learning algorithms in accurately identifying fraudulent transactions, leading to high false negative rates. Researchers have explored various techniques to handle class imbalance, including oversampling methods as synthetic minority oversampling technique (SMOTE) [13] [12], undersampling methods as condensed nearest neighbour (CNN) and random undersampling (RUS) [9] [10], and ensemble methods making use of efficient bagging [7].All methods have proven to successfully handle the highly imbalanced data. In some research a combination of techniques is used to optimize training of the models, but this does not seem to be necessary to obtain good results.

In addition to class imbalance, researchers have encountered obstacles such as data scarcity and privacy concerns. Real-life transaction data are often limited in availability due to privacy regulations and data sensitivity issues, hindering the development of robust fraud detection systems [8]. Moreover, the skewed distribution of data, overlaps between fraudulent and legitimate transactions, and the complexity of categorical data further complicate the fraud detection process [4]. These issues emphasize the importance of advanced pre-processing techniques to improve the effectiveness of fraud detection models.

Despite these challenges, researchers have proposed several promising models and algorithms for credit card fraud detection. Classical algorithms such as Logistic Regression (LR), Random Forests (RF), and Naive Bayes (NB) have shown effectiveness, especially when combined with ensemble methods [9]. Deep learning techniques, including Recurrent Neural Networks (RNNs) and Long Short-term Memory (LSTM) networks, have demonstrated potential in handling complex fraud patterns but often require substantial computational resources [10]. Critical analysis of the literature shows a the importance of addressing class imbalance in credit card fraud detection and handling it properly. Additionally, the complexity and computational requirements of deep learning models raise concerns about practicality and resource constraints in real-world applications.

Furthermore, the evaluation of fraud detection systems is crucial for assessing their performance and identifying areas for improvement. Researchers have employed various performance metrics to evaluate the effectiveness of different algorithms and models. However, the interpretation of these metrics can be influenced by factors such as class imbalance and dataset characteristics, highlighting the importance of comprehensive evaluation methodologies [6]. While significant progress has been made in credit card fraud detection, challenges such as class imbalance persist and require ongoing research efforts. By leveraging advanced pre-processing techniques and algorithmic innovations, researchers aim to develop more robust and adaptable fraud detection systems capable of accurately identifying fraudulent transactions while minimizing false positives and negatives. However, addressing challenges such as class imbalance, data scarcity, privacy concerns, and evaluation ambiguity requires careful consideration of researchers.

To approach the problem, several different models would be viable to use. The most common approaches in practice make use of a combination of multiple different models. For instance, a combination of neural networks, decision trees, and logistic regressions[11]. There are many benefits to this approach, such as increased accuracy, lower mean absolute error, liming over fitting as well as working well at scale on large datasets, which is what makes it so appealing in practice for large companies. However, they also come with some drawbacks; most notably that they are costly to run. As such, for this project, the focus will be on single models. Among those Logistic Regression, Support Vector Machines (SVM), and Decision Trees are the most common models used. For Logistic Regression one of the main reasons for its popularity is due to its efficiency, as well as the fact that it is well suited for problems with binary outcomes, such as in the case of fraud detection. SVMs, similarly, are also well suited to binary classification, and posses good generalization capacity. Decision trees on the other hand share few of the same advantages, however, they are simple to understand and interpret, in part due to it not being a black box model. They are also relatively efficient and fairly versatile. The main drawback of Decision Trees lies in their tendency towards over fitting, as well that it is more unstable compared to other alternatives.

3 Approach

In this section, we outline the approach taken in our research, focusing on the description of the dataset and features, the explanation of data pre-processing steps for features, and details about SVM and LR setup and parameter tuning.

3.1 Dataset

Firstly, we describe the dataset used in our study. The dataset comprises synthetic credit card transactions, designed to mimic real-world scenarios. The dataset consists of $24,351,143$ legitimate transactions and $29,757$ fraudulent transactions. This makes our data a highly imbalanced dataset. To overcome the problems introduced by data imbalance, resampling techniques are often applied to the original data set to adjust imbalance and to create unbiased prediction.[14]. Later on we will make use of undersampling on the training data to deal with the class imbalance.

Each transaction contains 13 features capturing various aspects of the transaction process, including user information and transaction details such as Year, Month, Day, Time, Amount, Use of Chip, Merchant Name, Merchant State, Zip Code, and Merchant Category Code (MCC). The target variable in the data is the variable $IsFraud?$, which will output $'Yes'$ when the transaction is fraudulent, and $'No'$ when the transaction is normal. The features Use of Chip, Zip Code, Merchant Name, Merchant State, and MCC are nominal. These are categorical variables where there is no specific order in the different labels. The other features, Year, Month, Day, Time, and Amount are numeric. The target variable $IsFraud?$ is a binary feature, this is a categorical feature with only two possible outcomes, in this case Yes versus No.

3.2 Pre-processing

Secondly, to prepare the features to be used for training the SVM and LR model we perform multiple data pre-processing steps. First of all, the features are filtered to make a simpler model with fewer features that are often easier to interpret and understand. Having fewer features in SVM simplifies the construction of the decision boundary, aiding in its visualization and comprehension. This reduction diminishes susceptibility to noise and irrelevant data, enhancing the model's robustness and accuracy. The features that are in the final dataset are $Time$, $Amount$, $UseChip$, and MCC. The feature $UseChip$ describes whether the transaction was done online or physically by swipe or by chip. This feature was chosen because it was discovered that the fraudulent transactions were done primarily online. The MCC is a special 4-number code that represents the industry type for example cruise lines or hotels, where a transaction occurred. The MCC by itself doesn't bring much information, but in combination with Amount the model can find suspicious overcharging or undercharging per industry. For example, if we look at Figure 1 we can see that the red dots (fraud cases) are either above or below blue dots (non-fraud cases) which indicates whether there was overcharging or undercharging.

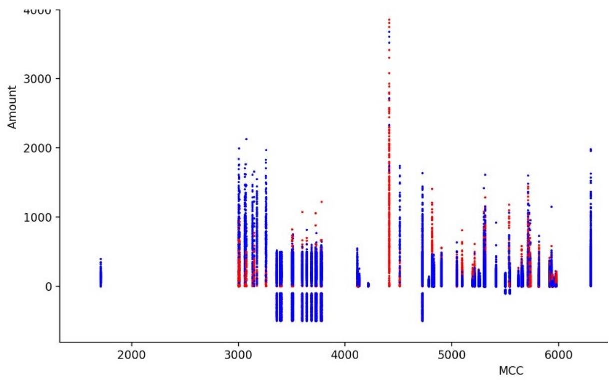


Figure 1: Amount vs MCC

Finally, the Time feature shows at which hour the transaction was made. It is suspected that fraud cases may occur during the night hours for example 3 A.M. to potentially avoid detection. Therefore this feature is relevant.

Now, that these features have been selected, the subsequent steps entail addressing missing values, encoding categorical features, and scaling numerical features. The $Time$ feature, represented in the format $06:45$, will be transformed into $Hours$, denoting the number of hours in a day. The $Amount$ feature represents the transaction amount, indicated with a \$ symbol preceding the numerical value. The sole pre-processing step for this feature involves removing the dollar sign. We utilize a pre-processing technique known as one-hot encoding for the $UseChip$ feature. This method transforms categorical inputs into multiple numerical features. The $UseChip$ feature, with options including online, swipe, or chip, is split into three separate binary features. For each transaction, if it involves a chip, the $Chip$ feature receives a value of 1, while $Online$ and $Swipe$ features receive values of 0. At last, our target variable $IsFraud$? is a categorical feature as well. To convert this to a numerical feature we provide a 1 whenever a transaction is fraudulent and a 0 when it is not.

3.3 SVM

In this section, we elaborate on in what way we will use our SVM model, by first explaining what the SVM algorithm does.

A SVM or Support Vector Machine[25], is a supervised learning model that makes use of a hyperplane to classify instances. The basic principle in SVM is to find the optimal hyperplane that best separates the different classes in the feature space. To increase accuracy, the ideal method is to try and maximize the distance between the hyperplane and the 'support vectors'. In this case, the support vector refers to the positive and negative instances closest to the hyperplane. The distance between the support vectors and the hyperplane is called the margin. The larger the margin, the higher the ability of the model to correctly separate the two classes. Furthermore, we also can decide on either a hard margin or a soft margin. For soft margin, this entails allowing instances to appear within the bounds of the margin, without creating a new one. This allows us to keep the distance of the margin high while dealing with a few outliers. In the case of hard margin, however, if a point lands within the bounds, the margin and hyperplane instead get modified. This creates more constraints and minimizes the margin. In some cases this can lead to better classification for specific instances, however, it greatly reduces the generalization capabilities of the model. To address the high imbalance in the dataset we assign weights to the different target values of the SVM classifier. These weights are the proportions of the classes from all the instances.

Leveraging the sckicit-learn[3] library in Python, which we will discuss in Section 4, we implement SVM for binary classification of fraudulent and legitimate transactions. In our project, we use the linear kernel for the SVM classifier. This kernel was chosen since it gave us the highest precision and accuracy when undergoing cross-validation and undersampling. The hyperparameter for the SVM classifier is the parameter C. This hyperparameter stands for the regularization parameter in SVM, which plays a crucial role in determining the balance between fitting the training data closely and maintaining a smooth decision boundary. Essentially, the value of C regulates the degree to which misclassification of training examples is penalized[27]. Parameter tuning is a critical aspect of optimizing the SVM model's performance. This can be done by for instance Grid Search and Cross Validation. Grid Search is an approach where a combination of pre-defined hyperparameter choices is used to find the combination of hyperparameters that yields the best model performance[17]. To prevent the risk of overfitting our data and obtain a more reliable estimate of the model's performance, we use cross-validation during parameter tuning. This involves splitting the dataset into multiple folds, training the model on a subset (fold), and evaluating its performance on the remaining unseen data[17].

3.4 Logistic Regression

Similarly to SVM, Logistic Regression[26] is a supervised learning model, primarily used for binary classification. However, unlike SVM, Logistic Regression operates on independent variables to determine the classification instead of the support vectors used previously. Additionally, while a decision boundary is still part of the classification process to determine our final labels, how we approach it differs from SVM. If we use the Logistic Regression formula, we obtain a value representing the probability of an instance being part of a given class or not within the range [0,1] rather than simply finding a decision threshold based on the points of our data. We derive our decision threshold on this range, where our threshold represents the minimum probability we accept to assign it to a given class. The Logistic Formula (shown below) contains attributes $f(x)$, which is the predicted output probability for item x, which represents our instance. From the remaining terms b_0 represents the bias or the intercept, while b_1 represents a coefficient with which to multiply x.

$$f(x) = \frac{e^{b_0+b_1x}}{1+e^{b_0+b_1x}}$$

4 Experimental Setup

In this section we want to give an overview of the environment we will use in this project, the way we split our data, and how we will measure our models performance.

4.1 Computational Environment

Our preferred tool for performing experiments is JupyterHub, a highly extensible, feature-rich notebook authoring application and editing environment [21]. We apply the Python programming language within JupyterHub for its adaptability, extensive libraries [20], and ease of use in data analysis and machine learning tasks [18]. In addition to Python's standard library, we rely on various other libraries and packages to support different parts of our research. One notable example is scikit-learn [3], a widely-used library that offers machine learning algorithms, including Support Vector Machines (SVM) for classification tasks[5]. We employed scikit-learn for tasks such as data pre-processing, model training, and evaluation. Another essential tool was Pandas[2], which serves as an essential foundation for conducting practical, real-world data analysis tasks in Python[19]. We used pandas for tasks such as loading, using DataFrame, and pre-processing the dataset, for example, the one-hot coding with the $UseChip$ feature. We utilized Matplotlib[1], a powerful visualization library in Python, to create informative plots and visualizations for our research. This library enabled us to visualize various aspects of the data, such as model performance metrics, which proved instrumental in interpreting our results effectively. Moreover, Numpy[15], an essential library for numerical computations in Python and a basis for libraries like scikit-learn [22], played a pivotal role in our research endeavors. We used Numpy for tasks such as array manipulation and numerical computations essential for both model training and evaluation.

4.2 Dataset Splitting

To start the training of our SVM and LR classifier we first need to split our dataset into training and test data. Since our dataset has around 24 million instances, we can assume our dataset is large enough to provide sufficient training data while having a reasonable-sized test set. Therefore, we decided on a split of 80% training data and 20% test data. To do this we first put our features in a random variable X and our target variable in y. Then, we use the *train_test_split* function from scikit-learn to split the X and y into training and test sets. Since our SVM classifier has a hyperparameter C and the LR classifier hyperparameter α, the training data is split into a training and validation set, where 70% of the whole data is the training data and 10% of the whole data is the validation set.

4.3 Model Performance

When evaluating the performance of our model, it's important to define certain terms such as precision, recall, as well as ROC and AUC. Precision[24] refers to the number of true positives, that is, the number of items that were labeled as positive that are also genuinely positive. In our context, this would be the number of items identified as fraud that were correctly labeled as such. Recall[24], on the other hand, represents the number of positive items that were correctly identified. That is, from the items that we know to be positive, what is the proportion of items that got labeled correctly as positive? Accuracy, meanwhile, is the proportion of correctly classified predictions out of the total number of predictions. So, this metric takes the true positives and true negatives and divides them by the total number of predictions. The formulas for precision, recall, and accuracy are as follows:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number of Predictions}}$$

The ROC (Receiver Operating Characteristics)[23] space graphs the TPR (True Positive Rate) against the FPR (False Positive Rate). The ROC curve thus allows us to see where our TPR is the highest, and the FPR the lowest. Additionally, it is possible to compare the performance of multiple models by making use of AUC(Area Under the Curve)[23], by taking the ROC curves for each model, and plot them on top of each other. From there we can say that the ROC that has a greater surface area under the curve performed better compared to the other. The precision-recall space graphs the precision of our model against our recall. There often has to be a tradeoff between high precision and high recall. The AUPRC (Area Under the Precision-Recall Curve) shows this precision-recall balance.

5 Results

When looking at the results of our experiment, we will look at metrics such as accuracy, recall, precision, and the ROC-AUC score. As seen in Section 4.3, some of these metrics include variables such as True Positives and False Negatives, so we will first explain what these mean in our case. When a transaction is labeled as fraudulent we call this a positive case and a non-fraudulent negative. So when we talk about a True Positive, it means a fraudulent transaction that is correctly predicted. A False Negative is a fraudulent transaction wrongly predicted as a non-fraudulent transaction.

5.1 Hyperparameter Tuning

The first step we took to train our SVM model was to use cross-validation to look for the best hyperparameter for our model. As discussed in Section 3.3 C has a role in maximizing the margin and minimizing the classification error. With the use of JupyterHub and Python, we produced code for the cross-validation of this hyperparameter. Since our dataset is highly imbalanced we chose to undersample our data such that the majority class (non-fraud cases) are 75% of our training data and the minority class (fraudulent cases) are 25%. Since we use the cross-validation method and we do not want to lose any information on our data, we implement the undersampling method in each fold of the cross-validation.

This approach can be used in our case since our dataset has so many instances. After performing the code for the cross-validation, which can be found in the Appendix, we see that our optimal C hyperparameter is of size 0.1.

For our Logistic Regression (LR) model we use similar steps to come up with our hyperparameter C. With the combination of undersampling and cross-validation, it is concluded that our optimal hyperparameter is equal to 100. Such a high hyperparameter indicates that our regularization strength is low and could cause overfitting later on.

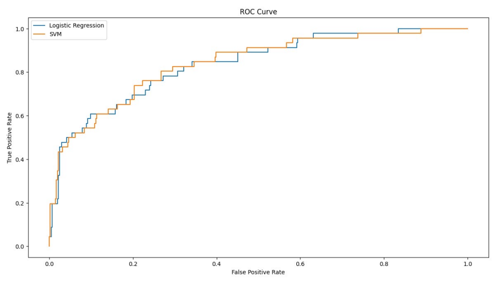
5.2 Evaluation Metrics on Validation Set

After the hyperparameter tuning, we see how our model does by looking at the metrics discussed in Section 4.3 on our training and validation set. Table 1 shows the resulting scores. The first metric is

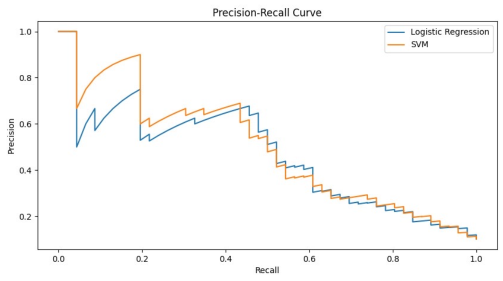
	Accuracy	Precision	Recall	ROC-AUC	AUPRC
SVM	0.9022	0.7500	0.0652	0.5314	0.1445
LR	0.9066	0.6429	0.1957	0.5916	0.2080

Table 1: Table displaying the metric values obtained from predictions made on our validation set for SVM and LR.

accuracy with a score of 0.9022 for the SVM classifier and 0.9066 for LR classifier. This suggests that both models are performing relatively well in terms of overall prediction correctness, SVM a bit more than LR. However, since our dataset is imbalanced, accuracy might not be the most reliable metric. For the precision metric, we get a value of 0.7500 for the SVM classifier and a lower value of 0.6429 for the LR classifier. This implies that a smaller fraction of the predicted fraud cases are true positives for the LR classifier. The third metric is the recall score which in our case is 0.0652 for the SVM and 0.1957 for



(a) ROC curve



(b) Precision-Recall curve

Figure 2: ROC- and Precision-Recall curves on the validation set of the SVM and LR classifier

the LR classifier. This value indicates that the models are capturing only a very small proportion of the actual fraudulent transactions (true positives). So, a big part of the actual fraudulent transactions are predicted as non-fraud, and thus false negatives. The ROC-AUC score is 0.5314 for the SVM model and 0.5916 for the LR model. This is the metric that measures how well the classifier can tell positive and negative examples apart. This score shows that the models have an okay way of doing this, better than guessing randomly, but there's still some room for improvement. We again get low values for our last metric, the AUPRC score, which for the SVM classifier has a value of 0.1445 and for the LR classifier a value of 0.2080. These scores suggest an imbalance between precision and recall, which we already have seen in our results. This highlights the challenges in correctly identifying fraudulent transactions while keeping the false positive rate low. We can see that for each metric the difference between the SVM and LR classifiers is not big, the biggest difference being in the recall of the models. Since this is only the representation of the training and validation set it does not give us a final evaluation on which model is best to predict fraudulent transactions.

In Figure 2 we project the ROC- and Precision-Recall curves that are produced from predictions on the validation set. In both curves the SVM and LR classifier lines intersect on multiple points. From these graphs it becomes clear that both classifier have a similar pattern in predicting the fraudulent transactions.

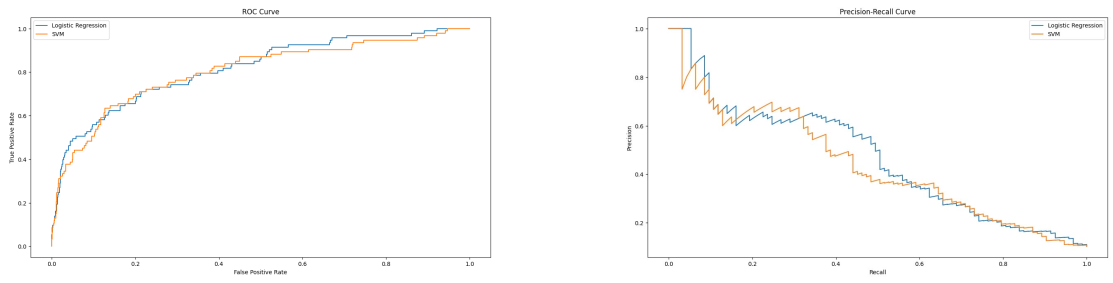
5.3 Final Evaluation Metrics on Test Set

For the final evaluation of our dataset, we will train the classifiers with the training set and predict using the test set. In the previous computations, the classifiers have never seen the test set before predicting their values, so this will be completely new information. This way, we can see how far our classifiers succeeded in predicting fraudulent transactions in a highly imbalanced dataset. Also, when classifying the training set we undersampled our training data, this however is not the case when using the test set since this should be a representative of the real world.

	Accuracy	Precision	Recall	ROC-AUC	AUPRC
SVM	0.9011	0.8333	0.0538	0.5263	0.1426
LR	0.9022	0.6666	0.1075	0.5507	0.1639

Table 2: Table displaying the metric values obtained from predictions made on our test set for SVM and LR.

The first metric is accuracy with a score of 0.9011 for the SVM classifier and 0.9022 for the LR classifier. This suggests that both models are performing relatively well in terms of overall prediction correctness, LR a bit more than SVM. For the precision metric, we get a value of 0.8333 for the SVM classifier and a lower value of 0.6666 for the LR classifier. This implies that the fraction of true positives from the predicted fraud cases are higher for the SVM than the LR model. The third metric is the recall score which in our case is 0.0538 for the SVM and 0.1075 for the LR classifier. Both these values are very low. This indicates that the classifiers miss a lot of the positive instances, resulting in a high number of false negatives. The ROC-AUC score is 0.5263 for the SVM model and 0.5507 for the LR model. This



(a) ROC curve (b) Precision-Recall curve

Figure 3: ROC- and Precision-Recall curves on the test set of the SVM and LR classifier

score shows that the models have an okay way of doing this, better than guessing randomly, but there's still some room for improvement. We again get low values for our last metric, the AUPRC score, which for the SVM classifier has a value of 0.1426 and for the LR classifier a value of 0.1639. This suggests that the classifier struggled with finding a nice balance between precision and recall. Since our dataset is highy imbalanced and there are a lot more non-fraud cases than fraudulent, this could mean that our classifier may prioritize maximizing precision at the cost of recall. For each metric the difference between the SVM and LR classifiers is not very big, the biggest difference being in the precision of the models.

In Figure 3, the ROC- and Precision-Recall curves are shown for the predictions on the test set. The results are quite similar to the ones obtained in Section 5.2 when testing on the validation set. We see that, again, for each curve the SVM and LR classifier lines are intertwined. For the ROC curve, we see that the lines lie above the diagonal, which proves that both the classifiers perform better than random guessing. Our Precision-Recall curve starts in the top left corner, where precision is 1 and recall is 0. It has a decreasing line that ends in the point $(1, 0)$. This line indicates that the model achieves perfect precision in the beginning, however when our recall increases the precision value decreases to 0. This pattern suggests that the model initially makes correct positive predictions but this diminishes when trying to capture more positive instances.

6 Conclusion

Looking at our results we see challenges posed by class imbalance and cost imbalance in credit card fraud detection. While certain metrics gain significance due to class imbalance, it's crucial to acknowledge the potential consequences of false positives, especially in time-sensitive transactions. For example, someone might need to send an urgent transaction, and flagging it as fraud could have devastating effects. For this reason, real-world implementations should include human oversight over the system and let customers quickly appeal any false positives. On the other hand, ignoring fraudulent transactions can be as devastating.

Comparing the results between Support Vector Machines (SVM) and Logistic Regression (LR) we see the importance of precision and recall in evaluating model performance. The higher precision observed in SVM suggests its effectiveness in identifying non-fraudulent cases. Although SVM exhibits slightly lower recall than LR, the difference lacks statistical significance. Nevertheless, it's crucial to take a range of factors into account, including feature complexity and computational efficiency, when making a decision between SVM and LR. Although our results may suggest that SVM might be a better model suited for fraud detection than LR it would be incorrect. This is because LR might become better once more different features are added. As the computation time of SVM skyrockets once more features are used. In conclusion, further research has to be done to explore the performance between LR and SVM with a wider array of features. Additionally, researching the optimal balance between precision and recall for credit card fraud is also necessary.

7 Future work

There are a couple of points to make in light of follow-up projects, the first one being the fine-tuning of final machine learning models using this dataset. Because much research has been done in the field of credit card fraud detection, a large comparison study between various methods might provide some clear insights into the performance of various approaches and the power of ensemble methods. This study should implement a uniform method of pre-processing the data to handle class imbalance and justify the decision process.

Another big challenge in this field is the data collection of credit card transactions. Due to privacy and regulatory issues, few types of information can be used to build a machine learning model that classifies fraudulent transactions. Therefore, a more political and social type of research can be carried out to examine the difficulty of acquiring sensitive data and handling it properly. This might lead to more features available to extract relevant information from. Complementary to this is exploring advanced pre-processing techniques to enhance feature selection, missing value compensation, and categorical feature encoding. This goes hand in hand with investigating the application of more sophisticated feature engineering methods to extract meaningful information from transaction data. It might be a rigid field of research, but any progress or enhancements might lead to an increase in model power.

Finally, the last interesting aspect for future work is the implementation of these developed models. This has always been the goal for creating, fine-tuning, and optimizing these machine learning models, for them to aid humans in achieving the goal of correctly classifying fraudulent transactions. The cost balance of misclassifying instances should be studied more extensively to gain information about the actions that will be taken from the model results. A framework could be created that indicates different actions that could be taken in different types of situations. This is highly dependent on the assessment of the cost balance, which should be derived carefully to justify the actions.

References

[1] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55.

[2] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] M. Zareapoor, K. Seeja, and M. A. Alam, "Analysis on credit card fraud detection techniques: Based on certain design criteria," *International journal of computer applications*, vol. 52, no. 3, 2012.

[5] L. Buitinck, G. Louppe, M. Blondel, *et al.*, *Api design for machine learning software: Experiences from the scikit-learn project*, 2013. DOI: 10.48550/arXiv.1309.0238. arXiv: 1309.0238 [cs.LG].

[6] J. West and M. Bhattacharya, "An investigation on experimental issues in financial fraud mining," in *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, 2016, pp. 1796–1801. DOI: 10.1109/ICIEA.2016.7603878.

[7] S. Akila and U. S. Reddy, "Risk based bagged ensemble (rbe) for credit card fraud detection," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*, IEEE, 2017, pp. 670–674. DOI: 10.1109/ICICI.2017.8365220.

[8] M. Rafalo, "Real-time fraud detection in credit card transactions," *Data Science Warsaw.*, 2017.

[9] A. Mishra and C. Ghorpade, "Credit card fraud detection on the skewed data using various classification and ensemble techniques," in *2018 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, IEEE, 2018, pp. 1–5. DOI: 10.1109/SCEECS.2018.8546939.

[10] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, "Deep learning detecting fraud in credit card transactions," in *2018 systems and information engineering design symposium (SIEDS)*, IEEE, 2018, pp. 129–134. DOI: 10.1109/SIEDS.2018.8374722.

[11] E. Kim, J. Lee, H. Shin, *et al.*, "Champion-challenger analysis for credit card fraud detection: Hybrid ensemble and deep learning," *Elsevier Ltd.*, vol. 128, Aug. 2019. DOI: 10.1016/j.eswa.2019.03.042.

[12] A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga, and N. Kuruwitaarachchi, "Real-time credit card fraud detection using machine learning," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, IEEE, 2019, pp. 488–493. DOI: 10.1109/CONFLUENCE.2019.8776942.

[13] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla, "Credit card fraud detection-machine learning methods," in *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, IEEE, 2019, pp. 1–5. DOI: 10.1109/INFOTEH.2019.8717766.

[14] D. B. Dongfang Zhang Basu Bhandari, "Credit card fraud detection using weighted support vector machine," *Applied Mathematics*, vol. 11, no. 12, Dec. 2020. DOI: 10.4236/am.2020.1112087..

[15] C. R. Harris, K. J. Millman, and S. J. van der Walt etc., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2.

[16] E. C. Bank, "Seventh report on card fraud: A country-by-country and regional perspective on card fraud," European Central Bank, Tech. Rep. ISBN 978-92-899-4874-6, ISSN 2315-0033, , QB-BI-21-001-EN-N, 2021. DOI: 10.2866/793033.

[17] H. Liu. "An evaluation of hyperparameter tuning methods in svm." (2021), [Online]. Available: https://math.ucsd.edu/sites/math.ucsd.edu/files/undergrad/honors-program/honors-theses/2020-2021/Huanxi_Liu_Honors_Thesis.pdf (visited on 03/23/2024).

[18] M. Ranjan, K. Barot, V. Khairnar, *et al.*, "Python: Empowering data science applications and research," *Journal of Operating Systems Development Trends*, vol. 10, pp. 27–33, Aug. 2023. DOI: 10.37591/joosdt.v10i1.576.

[19] I. H. b. O. © 2024 pandas via NumFOCUS. "About pandas." (), [Online]. Available: https://pandas.pydata.org/about/. (accessed: 23.03.2024).

[20] P. S. F. © Copyright 2001-2024. "The python tutorial." (), [Online]. Available: https://docs.python.org/3/tutorial/index.html#the-python-tutorial. (accessed: 23.03.2024).

[21] P. J. © Copyright 2018-2024. "Jupyterlab documentation." (), [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/index.html. (accessed: 23.03.2024).

[22] ©. 2. N. team. All rights reserved. "Numphy." (), [Online]. Available: https://numpy.org. (accessed: 23.03.2024).

[23] A. Bhandari. "Guide to auc roc curve in machine learning : What is specificity?" (), [Online]. Available: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/.

[24] G. F. Courses. "Classification: Precision and recal." (), [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

[25] S. Ray. "Learn how to use support vector machines (svm) for data science." (), [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[26] A. Saini. "A beginner's guide to logistic regression." (), [Online]. Available: https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/.

[27] S. Yıldırım. "Hyperparameter tuning for support vector machines — c and gamma parameters." (), [Online]. Available: https://towardsdatascience.com/hyperparameter-tuning-for-support-vector-machines-c-and-gamma-parameters-6a5097416167. (accessed: 23.03.2024).

8 Information Sheet

MLVU information sheet 29-03-2024

Group number

51

Authors

Student Name	Student Number
Andrei Beiu	2777096
Yasin Karyagdi	2745681
Kian Sie	2685871
Stefan Sprakel	2710263
Luna de Vries	2666266

Software used *We used the JupyterHub environment using Python. The libraries we used are Numphy, Sklearn, Pandas and Matplotlib.*

Use of AI tools *The AI tool we have used during this project is ChatGPT. This tool has been used during the writing process of this report for generating synonyms of words to improve readability.*

Link to code (optional) *We added our code in the appendix. We do have a link for our dataset on this website.*

Group disagreements -

9 Appendix

Listing 1: Code for data pre-processing

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv('card.csv')

#Making dataset with only features: Time, Use Chip (3), MCC, and Amount
new_df = df.drop(['User', 'Card', 'Year', 'Month', 'Day', 'Merchant Name',
         'Merchant City', 'Merchant State', 'Zip', 'Errors?'], axis=1)

#Convert Time variable into Hours
new_df['Time'] = pd.to_datetime(new_df['Time'], format='%H:%M')
new_df['Hours'] = new_df['Time'].dt.hour + new_df['Time'].dt.minute / 60
new_df = new_df.drop(['Time'], axis=1)

# Convert 'Use Chip' column to numeric values
# Perform one-hot encoding on the 'Use Chip' column
one_hot_encoded = pd.get_dummies(new_df['Use Chip'], prefix='Use_Chip')
# Convert True/False to 1/0
one_hot_encoded = one_hot_encoded.astype(int)
# Concatenate the one-hot encoded columns with the dataset
new_df = pd.concat([new_df, one_hot_encoded], axis=1)
# Drop the original 'Use Chip' column
new_df.drop('Use Chip', axis=1, inplace=True)

#Converting the Amount feature to one without Dollar sign and only integer
new_df['Amount'] = new_df['Amount'].str.replace('$\$$', '').astype(float)

# Replace 'Yes' with 1 and 'No' with 0 in the target variable
new_df['Is Fraud?'] = new_df['Is Fraud?'].replace({'Yes': 1, 'No': 0})
```

```
new_df.to_csv('newcard.csv', index = False)
```

Listing 2: Code for cross-validation process

```
# Extracting features (X) and target variable (y)
X = new_df.drop(columns=['Is-Fraud?'])  # Drop the target column to get features
y = new_df['Is-Fraud?']  # Get the target column

from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from sklearn.svm import SVC

Split the dataset into training, validation, and test sets
while maintaining class distribution
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_val, y_train, y_val =
train_test_split(X_train, y_train, test_size=0.125, random_state=42, stratify=y_train)
#training set 70%, test set 20%, validation set 10%

from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.svm import SVC
from imblearn.under_sampling import RandomUnderSampler

# Define accuracy score for cross-validation
scorer = make_scorer(accuracy_score)

# Define cross-validation strategy
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

#Define undersampler, 75% non fraud, 25% fraud
undersampler = RandomUnderSampler(sampling_strategy=0.33, random_state=42)

# Define the parameter grid
param_grid = {'C': [0.1, 1, 10]}

# Initialize an empty list to store cross-validation scores
cv_scores = []

# Loop through each combination of C
for C in param_grid['C']:
        #SVM classifier with linear kernel
        svm_classifier = SVC(kernel='linear', C=C, random_state=42)

        # Perform cross-validation with under-sampling
        fold_accuracies = []
        for train_index, val_index in cv.split(X_train, y_train):
            # Get the training and validation data for this fold
            X_train_fold, X_val_fold =
            X_train.iloc[train_index], X_train.iloc[val_index]
            y_train_fold, y_val_fold =
            y_train.iloc[train_index], y_train.iloc[val_index]

            # Undersample the majority class in the training fold
            X_train_under , y_train_under =
            undersampler.fit_resample(X_train_fold , y_train_fold)

            # Train the SVM classifier on the under-sampled training fold
            svm_classifier.fit(X_train_under , y_train_under)

            # Predict the labels for the validation fold
            y_val_pred = svm_classifier.predict(X_val_fold)

            # Evaluate the classifier on the validation fold
            fold_accuracy = accuracy_score(y_val_fold , y_val_pred)
            #precision = precision_score(y_val_fold , y_val_pred)

            # Store the accuracy in the list
            fold_accuracies.append(fold_accuracy)

        # Calculate the mean accuracy across all folds
        mean_accuracy = sum(fold_accuracies) / len(fold_accuracies)
        # Append the mean accuracy to the list of cross-validation scores
        cv_scores.append((gamma, mean_accuracy))

# Find the best combination of C based on mean accuracy
best_params = max(cv_scores)
best_C , best_accuracy = best_params

print("Best Parameter C:" , best_C)
print("Best Mean Accuracy:" , best_accuracy)
```

Listing 3: Code for the final run with the test set

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
%matplotlib inline

df = pd.read_csv('newcard.csv')

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score , recall_score ,
accuracy_score , roc_auc_score
from sklearn.metrics import average_precision_score
from sklearn.metrics import roc_curve , precision_recall_curve

# Extracting features (X) and target variable (y)
X = new_df.drop(columns=['Is Fraud?'])  # Drop the target column to get features
y = new_df['Is Fraud?']  # Get the target column


#Split the dataset into training , validation , and test sets
while maintaining class distribution
X_train , X_test , y_train , y_test =
train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
#For this final run X_train becomes 80% again
```

```
# Initialize and train the SVM classifier with C = 0.1
svm_classifier = SVC(kernel='linear', C=0.1, random_state=42)
svm_classifier.fit(X_train, y_train)

# Initialize and train the Logistic Regression classifier with C = 100
LR_classifier = LogisticRegression(C = 100, max_iter=1000)
LR_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred_svm = svm_classifier.predict(X_test)
y_pred_lr = LR_classifier.predict(X_test)

# Calculate precision
precision_svm = precision_score(y_test, y_pred_svm)
precision_lr = precision_score(y_test, y_pred_lr)

# Calculate recall
recall_svm = recall_score(y_test, y_pred_svm)
recall_lr = recall_score(y_test, y_pred_lr)

# Calculate accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# Calculate ROC-AUC score
roc_auc_svm = roc_auc_score(y_test, y_pred_svm)
roc_auc_lr = roc_auc_score(y_test, y_pred_lr)

# Compute the average precision score
auprc_svm = average_precision_score(y_test, y_pred_svm)
auprc_lr = average_precision_score(y_test, y_pred_lr)

# Print the results
print("Logistic Regression Metrics:")
print("Precision:", precision_lr)
print("Recall:", recall_lr)
print("Accuracy:", accuracy_lr)
print("ROC-AUC:", roc_auc_lr)
print("AUPRC:", auprc_lr)

print("\nSVM Metrics:")
print("Precision:", precision_svm)
print("Recall:", recall_svm)
print("Accuracy:", accuracy_svm)
print("ROC-AUC:", roc_auc_svm)
print("AUPRC:", auprc_svm)

#Plotting ROC curves
# Get predicted probabilities from the model
lr_probs = LR_classifier.predict_proba(X_test)[:, 1]
svm_probs = svm_classifier.predict_proba(X_test)[:, 1]

# Compute ROC curve for logistic regression
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
```

```
# Compute ROC curve for SVM
svm_fpr, svm_tpr, _ = roc_curve(y_test, svm_probs)

# Plot ROC curves
plt.figure(figsize=(10, 5))
plt.plot(lr_fpr, lr_tpr, linestyle='--', label='Logistic Regression')
plt.plot(svm_fpr, svm_tpr, linestyle='--', label='SVM')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()

#Plotting precision-recall curve
# Compute precision-recall curve for logistic regression
lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_probs)

# Compute precision-recall curve for SVM
svm_precision, svm_recall, _ = precision_recall_curve(y_test, svm_probs)

# Plot Precision-Recall curves
plt.figure(figsize=(10, 5))
plt.plot(lr_recall, lr_precision, linestyle='--', label='Logistic Regression')
plt.plot(svm_recall, svm_precision, linestyle='--', label='SVM')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```