Financial Security using Machine Learning for Credit Card Fraud Detection

A PROJECT REPORT

for

INFORMATION SECURITY ANALYSIS AND AUDIT (CSE3501)

in

B.Tech – Information Technology and Engineering

by

KHUSHI AGRAWAL (19BIT0371)- L41+L42

KAMAKSHI BHATT (19BIT0372)- L21+L22

Under the Guidance of

Dr. PRIYA V

Associate Professor, SITE

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

December, 2021

DECLARATION BY THE CANDIDATE

We hereby declare that the project report entitled **"Financial Security using Machine Learning for Credit Card Fraud Detection"** submitted by us to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Information Security Analysis and Audit (CSE3501)** is a record of bonafide project work carried out by us under the guidance of **Dr. PRIYA V** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other course.

Place : Vellore Signature

Date : 10-12-2021

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology & Engineering [SITE]

CERTIFICATE

This is to certify that the project report entitled **"Financial Security using Machine Learning for Credit Card Fraud Detection"** submitted by **KHUSHI AGRAWAL (19BIT0371) and KAMAKSHI BHATT (19BIT0372)** to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Information Security Analysis and Audit (CSE3501)** is a record of bonafide work carried out by them under my guidance.

Dr. Priya V

GUIDE

Associate Professor, SITE

Financial Security using Machine Learning for Credit Card Fraud Detection

Abstract

Fraud detection in credit cards is a data mining problem. Our project tries to predict credit-card frauds using Machine Learning as it is believed to provide better results as compared to the existing techniques used to detect these frauds. It becomes challenging due to two major reasons – first, the profiles of normal and fraudulent behaviour change frequently and second, the credit card fraud data sets are highly skewed. We will be checking the performance by implementation of Decision Tree, Random Forest, SVC, K-Nearest Neighbors and Logistic Regression. Then we will be deploying the best machine learning model according to our accuracy results to a website and show its actual working with the test data set.

Keywords – Credit card, Fraud detection, Machine learning, Under-sampling, Website deployment, Cyber Security

Literature review

Title of the paper	Algorithm used	Methodology applied	Advantages	Issues/Drawbacks	Metrics used
Credit Card Fraud Detection Using AdaBoost and Majority Voting.	Bayesian, Trees, Neural Network, Linear Regression, Logistic Regression, Support Vector Machine	Algorithms are used in conjunction with the AdaBoos and majority voting methods.	The system is very fast due to AdaBoost Technique and effective due to Majority Voting techniques.	Testing credit card FDSs using real data sets is a difficult task. The fraud has to be deducted in real time and the number of false alerts.	The MCC metric has been adopted as a performance measure.
Credit Card Fraud Detection using Local Outlier Factor and Isolation Forest	Local Outlier Factor and Isolation Factor algorithm	The transactions in the dataset are classified as fraud or legit. The two different algorithms are used to detect fraud in the credit card system. Comparisons are made for these algorithms to determine which algorithms give the best results.	Local outlier factor gives a high accuracy rate of 97%	In higher dimensions, the LOF algorithm detection accuracy gets effected. Isolation Forest algorithm has a disadvantage in detecting local anomaly point, which affects the accuracy of algorithm.	Matthews Correlation Coefficient (MCC) , Precision, Recall , F1-score , Support

Machine Learning For Credit Card Fraud Detection System	logistic regression, decision tree and random forest	The comparisons are made for different machine learning algorithms.	High accuracy of >90 is achieved.	The data set is highly imbalanced.	sensitivity, specificity, accuracy and error rate.
A customized classification algorithm for credit card fraud detection	Fraud-BNC, a customized Bayesian Network Classifier (BNC) algorithm, Hyper-Heuristic Evolutionary Algorithm (HHEA)	Two classification approaches are used to deal with class imbalance. Hyper-Heuristic Evolutionary Algorithm (HHEA) is used to automatically generate a customized a Bayesian Network Classifier (BNC) algorithm, Fraud-BNC.The customized algorithm is compared to seven other classification methods.	It improved the current techniques used by the company to quantify fraud by up to 72.64%	Challenges of learning from class-imbalanced data.	F1 measure, economic efficiency.
Sequential fraud detection for prepaid cards using hidden Markov model	clustering algorithm,Baum-Welch algorithm,Viterbi Algorithm.	HMM uses Baum-Welch algorithm and Viterbi algorithms to help to identify abnormal behaviors from an event stream using kl divergence.	The model helps to get high fraud transaction coverage at very low false alarm rate and handling large volumes of transactions, therefore	HMM requires high training time for data preprocessing and analysing.	Thresholds and the mean F-score, precision, and recall

divergence			giving better and faster results in less time.		
Credit Card Fraud Detection using Machine Learning and Data Science	Local Outlier Factor, Isolation Forest Algorithm	The paper uses machine learning algorithms to detect anomalous activities, called outliers. Local Outlier Factor algorithm and Isolation Forest algorithm are used to plot the results	The algorithm reaches over 99.6% accuracy and as the dataset increases the precision also increases.	Precision remains only at 28% when a tenth of the data set is taken into consideration	Precision, recall,f1-score and support.
An Efficient Credit Card Fraud Detection Model Based on Machine Learning Methods	random forest, tree classifiers, artificial neural networks, support vector machine, Naïve Baiyes, logistic regression and gradient boosting classifier	One-class classifiers as well as the Matthews correlation coefficient is used for data pre-processing. Dataset Correlation Matrix is plotted for the whole dataset utilizing classification system (Machine Learning) methodology.	Its feedback approach contributes to enhancing the classifier's detection rate as well as cost-effectiveness.	Misclassified information is an major issue because not all fraudulent activity is captured or recorded.	precision, recall, F1-score, accuracy, and FPR percentage
Deep Convolution Neural Network Model for	Deep Convolution Neural Network (DCNN)	Once the pre-processing is performed, the memory based DCCN technique is applied for the purpose of financial fraud	This algorithm is prone to over-fitting issues. Detection accuracy of 99% was obtained by using the	Sequential prediction issue.	cost, speed and accuracy metrics. matrix, epochs, batch size and optimizers

Credit-Card Fraud Detection and Alert		detection. The existing machine learning models, auto-encoder model and other deep learning models are compared with the proposed model	proposed model.		

Title of the paper	Algorithm used	Methodology applied	Advantages	Issues/Drawbacks	Metrics used
An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection	C5.0 ALGORITHM, ARTIFICIAL NEURAL NETWORK (ANN), NAÏVE BAYES (NB), BAYESIAN BELIEF NETWORK (BBN), LOGISTIC REGRESSION (LR), KNN	A rigorous experimental study with the solutions that handle the imbalance classification problem is conducted. Then these solutions are explored along with the machine learning algorithms used for fraud detection	Their research shows that the approaches usually used to solve imbalance problems can have unpleasant consequences when the imbalance is extreme, for example generating no. of false positives. Even though these methods enhance the classifier's performance, significant fraud cases remain unknown/unrevealed.	Size of the data was a limitation in this research as they could not conduct our study on large amounts of data.	Accuracy, Sensitivity, and AURPC
Credit Card Fraud Detection and Prevention using Machine Learning	Random Forest Algorithm	They use supervised machine learning algorithms such as Random Forest Algorithms to determine online/offline fraud credit card transactions; the data is pre-processed which includes, formatting cleaning sampling. Before this data collection is also done.	We can incorporate various machine learning algorithms as modules in this project and the combination of their results can increase the accuracy of the result, such kind of design of the project is made.	Precision scores are found a little low.	Accuracy, precision
Adaptive Model for Credit Card Fraud Detection	Support Vector Machines (SVM), Fuzzy Association Rules (FAR), Deep Learning (DL) Hybrid Model	The proposed methodology include Updating the database, training the models, inserting new discovered rules, under sampling,	When compared to the usual model proposed in literature survey, this framework, makes a significant contribution by taking into account human behavior factors, and imbalanced data, and allows determining	Other models of machine learning could also be considered and a tool can be made.	Accuracy

		feature selection etc.	unusual transactions that would have not been considered using traditional classic methods/techniques.		
Credit Card Fraud Detection Using Weighted Support Vector Machine	Support Vector Machine (SVM), LR, Random Forest	Three machine learning algorithms, LR, SVM, and RF, will be applied to the data. The results obtained will be used as the benchmark for the more advanced algorithms and then, weighted SVM will be applied to the dataset. Three undersampling techniques are compared and sampling size are going to be optimized.	A new criteria based on the financial loss is developed to judge the performance of classification which is called as financial recovery.	Random undersampling or oversampling are simple, which can help solve the problem of data skewness, but often introduce non-informative or ill-informative sub-structures in data set.	Accuracy, recall, precision, financial recovery
Real-Time Deep Learning Based Credit Card Fraud Detection	Neural Networks	After input data preprocessing and feature selection, bias elimination is performed. Then, then network building and parameter setting takes place. Now, Network training and prediction is done.	The attributes with high correlation levels, both on positive/negative magnitudes are chosen for next level analysis. The ones that exhibit almost zero correlations are removed. This process reduces the size of the dataset to a large extent, and also results in its faster training.	The shortcoming of this model is the slightly low TPR levels.	TPR, TNR, Recall, Precision, Accuracy.
Improved Credit Card Fraud Detection using	NN, SVM, AIS, K-Nearest Neighbour Algorithm, Bayesian	Data collection and pre-processing is carried out and then Credit Card Fraud Detection	Real-time detection of credit card fraud can be stated as one of the main contributions of this project/research paper. Also, ideal	It doesn't focus on location-based frauds. There is also a need of improving the prediction levels to	Accuracy

Machine Learning		Using Bayesian and Neural Networks and other algorithms. At the end Modeling and testing of Fraud Detection Systems is performed where GUI is also included.	algorithms that address 4 main types of frauds were selected through literature survey, experimenting and parameter tuning.	acquire a better prediction than this one.	
An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine	Optimized Light Gradient Boosting Machine known as OLightGBM	After data set is obtained and pre-processing and cleaning is done, feature selection takes place. Then the optimized light gradient boosting classifier is applied and model is evaluated using various performance matrix for getting the results.	The experimental results show that the proposed technique outperformed the other machine learning algorithms and shown the top-level performance in terms of Accuracy, AUC etc.	Because, the total number of fraud transactions is much less than the number of legitimate transactions, the data distribution is unbalanced and it is already shown that the performance of various machine learning algorithms decreases when the data set is unbalanced.	AUC, Precision, Accuracy and F1-score
Credit card Fraud Detection based on Machine Learning Algorithms	Naive Bayes, Logistic regression, J48 and AdaBoost	Online data set is selected and preprocessed. Then some better machine learning algorithms are selected from Literature survey and then they Implement them in python for classifying fraud and non-fraud transaction.	AdaBoost is very quick and takes very less time.	Some more algorithms should have been taken for the survey before implementing the best ones.	Accuracy, Time duration

1. Randhawa, K., Loo, C. K., Seera, M., Lim, C. P., & Nandi, A. K. (2018). Credit Card Fraud Detection Using AdaBoost and Majority Voting. IEEE Access, 6, 14277–14284. https://doi.org/10.1109/access.2018.2806420

The paper used standard models like NB, SVM, and DL as well as hybrid machine learning models such as Ada Boost and majority voting methods to detect credit card fraud. They are applied to credit card database publicly available to evaluate the effectiveness of the model. They then analyzed real-world data from a financial institution. Continuing to test the strength of the algorithms by continuously adding noise to the data samples and ultimately suggest that the multi-voting method achieves good levels of accuracy in detecting credit card fraud by comparing Matthews Correlation Coefficient. The best MCC rating is 0.823, obtained using a majority vote. Achieved rating of 1 of MCC using Ada Boost methods and multiple voting methods on a credit card data set taken from a financial institution.

2. John, H., & Naaz, S. (2019). Credit Card Fraud Detection using Local Outlier Factor and Isolation Forest. International Journal of Computer Sciences and Engineering, 7(4), 1060–1064. https://doi.org/10.26438/ijcse/v7i4.10601064

This paper examines the effectiveness of credit card transactions and makes it into two fraudulent and non-fraudulent categories. Anomalies are created based on these two classes and machine learning skills are used to detect fraudulent transactions. Then using Local Outlier Factor and Isolate Forest the anomalies can be analyzed and compared to improve the performance and to see which algorithm is best. Local outlier factor gives a high accuracy rate of 97% followed by the Isolation forest of 76%.

3. Lakshmi S V S S1 ,Selvani Deepthi Kavila (2018). Machine Learning For Credit Card Fraud Detection System. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 24 (2018) pp. 16819-16824 DOI:10.37622/000000 http://www.ripublication.com/ijaer18/ijaerv13n24_18.pdf

In this paper, a machine learning process such as Logistic regression, Decision Tree and Random forest has been used to detect credit card fraud. Sensitivity, specificity, accuracy and error measurement are used to evaluate the effectiveness of the proposed system. The accuracy of the logistic regression, decision tree and random forest planning are 90.0, 94.3, and 95.5 respectively. To test algorithms, 70% of the database is used for training and 30% is used for testing and validation. By comparing all three methods, it was found that random forest segregation is better than systematic deforestation and decision tree.

4. de Sá, A. G., Pereira, A. C., & Pappa, G. L. (2018). A customized classification algorithm for credit card fraud detection. Engineering Applications of Artificial Intelligence, 72, 21–29. https://doi.org/10.1016/j.engappai.2018.03.011

This paper generates a custom BNC database algorithm for real-world credit card fraud. The customized algorithm is compared to seven other methods of classification using the F1 rating and economic

efficiency metrics. The audit is based on the metrics of both categories and those used by financial professionals to assess fraud rates. The proposed algorithm improved the current techniques currently used by the company to quantify fraud detection in PagSeguro by up to 72.64%.

5. Robinson, W. N., & Aria, A. (2018). Sequential fraud detection for prepaid cards using hidden Markov model divergence. Expert Systems with Applications, 91, 235–251. https://doi.org/10.1016/j.eswa.2017.08.043

This paper detects credit card fraud during transactions using the Hidden Markov Model. Model customer transaction pattern is analyzed and any standard pattern deviation is considered a fraudulent transaction. It makes managing management much easier and tries to eliminate difficulties. The paper also explained how they could detect incoming transactions by many additional security measures such as MAC address acquisition and shipping verification, so it provides improved security and better access to fraudulent transactions.

6. S P Maniraj, Aditya Saini, Shadab Ahmed, & Swarna Deep Sarkar. (2019). Credit Card Fraud Detection using Machine Learning and Data Science. International Journal of Engineering Research And, 08(09). https://doi.org/10.17577/ijertv8is090031

This paper presents a model used to determine whether a new transaction is fraudulent or not. Data sets are analyzed and processed in advance with the distribution of multiple detection algorithms such as Local Outlier Factor and Isolate Forest algorithm in PCA Credit Card converted data. Precision, recall,f1-score and support are used as metrics.

7. Naresh Kumar Trivedi, Sarita Simaiya, Umesh Kumar Lilhore, Sanjeev Kumar Sharma. (2020). An Efficient Credit Card Fraud Detection Model Based on Machine Learning Methods. International Journal of Advanced Science and Technology, 29(05), 3414 - 3424. https://www.researchgate.net/profile/Dr-Kumar-92/publication/341932015_An_Efficient_Credit_Card_Fraud_Detection_Model_Based_on_Machine_Learning_Methods/links/5ee4a477458515814a5b891e/An-Efficient-Credit-Card-Fraud-Detection-Model-Based-on-Machine-Learning-Methods.pdf

This paper introduces an effective way to detect credit card fraud including a response system, based on a machine learning process. Its response approach contributes to improving the detection rate of the separator and the cost-effectiveness. The effectiveness of these methods is always tested depending on the size of the test for the performance of the various separators. The random forest shows better results compared to other machine learning classifiers.

8. Chen, J. I. Z., & Lai, K. L. (2021). Deep Convolution Neural Network Model for Credit-Card Fraud Detection and Alert. June 2021, 3(2), 101–112. https://doi.org/10.36548/jaicn.2021.2.003

The paper uses the Convolution Neural Network process for the purpose of detecting financial fraud. Results are enhanced by the hyperparameters used by the model. The algorithm determines whether the transaction is fraudulent or not and tests various parameters to ensure the validity of the result. Existing machine learning models, auto-encoder models and other in-depth learning models are compared to the proposed performance test model by capturing real-time credit card fraud data.

9. S. Makki, Z. Assaghir, Y. Taher, R. Haque, M. Hacid and H. Zeineddine, "An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection," in IEEE Access, vol. 7, pp. 93010-93022, 2019, doi: 10.1109/ACCESS.2019.2927266.

This research paper shows that imbalanced classification methods are not useful, most of the times when there is extremely imbalanced dataset. It also shows that the existing techniques result in a huge number of false alarms, which are very expensive to financial organisations. This may lead to wrong results as well as increasing the occurrence of fraud cases. They have surveyed eight machine learning techniques presented in the literature that are used for fraud detection and classification. They have selected the most efficient methods according to three evaluation metrics. Then, they have compared them with class imbalance approaches applied to these same algorithms. They studied the added value of these methods by comparing the imbalanced classification methods to the original methods. This comparison helped them understand the limitations of these approaches. They reported the results in detail and discussed the limitations of the solutions that they experimented in this paper.

10. S. Abinayaa, H. Sangeetha, R. A. Karthikeyan, K. Saran Sriram, D. Piyush, Credit Card Fraud Detection and Prevention using Machine Learning, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-4, April, 2020

According to the authors, this research shows the most common methods of fraud alongside their detection methods and algorithms, and examined recent research works present in this field.
They have used random forest algorithm in the proposed system to classify the credit card data set.
Random Forest Algorithm is a (Machine Learning) Classification and Regression algorithm.
This research paper also explains in how machine learning can be implemented in combination with the random forest algorithm, pseudo code, in order to obtain better results in the detection of credit card fraud. The method's effectiveness is measured based on various evaluation metrics such as accuracy and precision.

11. SADGALI, Imane; SAEL, Naoual; BENABBOU, Faouzia. Adaptive Model for Credit Card Fraud Detection. **International Journal of Interactive Mobile Technologies (iJIM)**, [S.l.], v. 14, n. 03, p. pp. 54-65, feb. 2020

This research paper provides a flexible approach to credit card fraud detection that exploits the effectiveness of strategies that provide high accuracy and look at the type of transaction and customer profile. The proposal of the authors in this research is a multi-level framework, which includes the banking security feature, the customer profile and the activity profile itself.

It uses different detection algorithms to increase and achieve high-level accuracy and four-component design to handle data storage too. Thus, in short a conceptual framework, to detect credit card fraud is proposed in the paper.

12. Zhang, D., Bhandari, B., & Black, D. (2020). Credit Card Fraud Detection Using Weighted Support Vector Machine. *Applied Mathematics-a Journal of Chinese Universities Series B, 11*, 1275-1291.

According to the authors in this research paper, they are performing credit card fraud detection using weighted support vector machine algorithm. A new criterion to judge classification algorithm is developed, which considers the cost of misclassification, and several undersampling techniques are compared by this new criterion. Then the weighted support vector machine (SVM) algorithm considering the financial cost of misclassification is proposed, which show that they are more effective in detecting credit card fraud than traditional methods. This weighted support vector machine algorithm uses transaction balances as fraudulent transactions weights, and a same for nonfraudulent transactions and the results with accuracy precision etc. evaluation metrics shows that this strategy significantly improves the effectiveness of credit card fraud detection.

13. Devi, V., & Ravi, D.G. (2020). Real-Time Deep Learning Based Credit Card Fraud Detection. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE 03, MARCH 2020

In this paper, a deep-learning based architecture for quick and efficient credit card fraud detection is presented. The architecture is composed of feature selection, bias elimination and network training and prediction as the major phases or steps of it. Correlation based feature selection aims to identify the highly significant features and bias elimination results in standardized data formation. A deep learning based model is built for the prediction process using Keras. The resultant predictions were observed to exhibit top level and well-functioning outcome according to the evaluation metrics such as recall, precision, accuracy etc.

14. Ramya, M., S. Kumar and K. Raja. “Improved Credit Card Fraud Detection using Machine Learning.” *International journal of engineering research and technology* 8 (2020)

This research paper proposes a novel credit-card fraud detection system by detecting 4 types of fraudulent transactions using optimal algorithms and by analysing the related problems identified by past researchers. By focusing on real-time credit card fraud detection using predictive analytics and also an API module the end user is informed over the Graphical User Interface of the real-time system the minute a fraudulent transaction occurs. This part of this project allows the credit card fraud investigation team to make their decision to move to the next step as soon as a suspicious transaction is determined through the process.

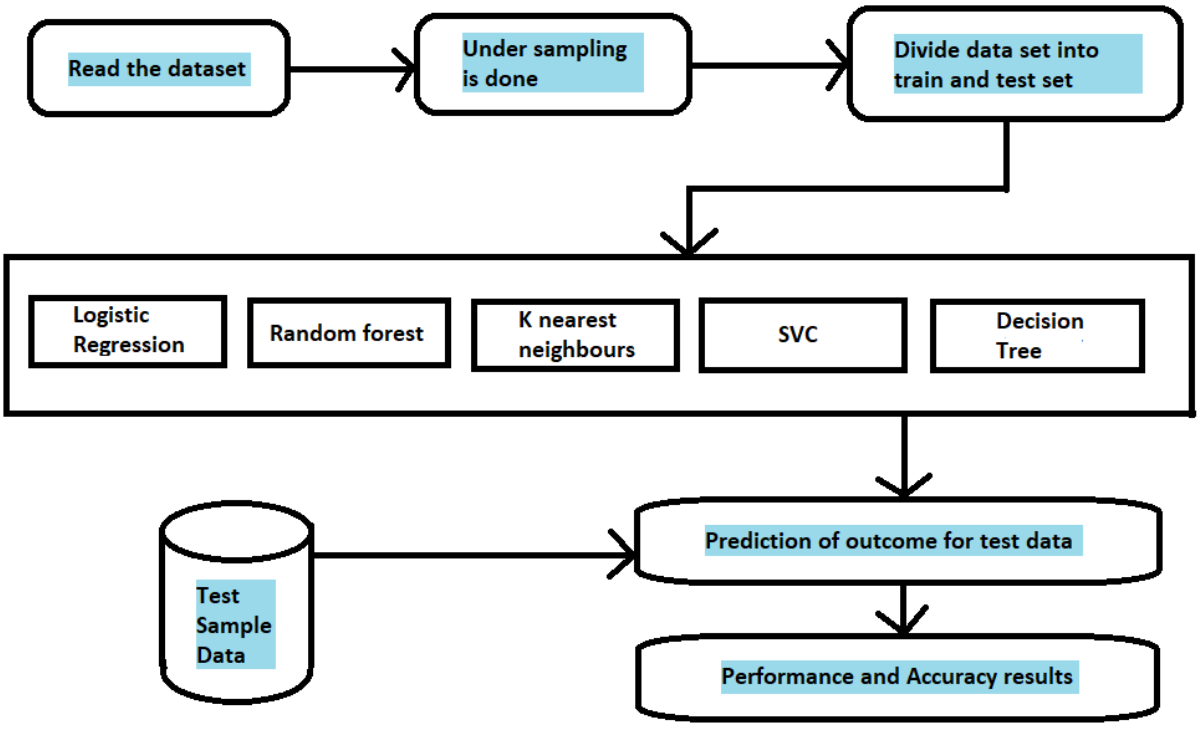
15. A. Taha and S. J. Malebary, "An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine," in IEEE Access, vol. 8, pp. 25579-25587, 2020, doi: 10.1109/ACCESS.2020.2971354.

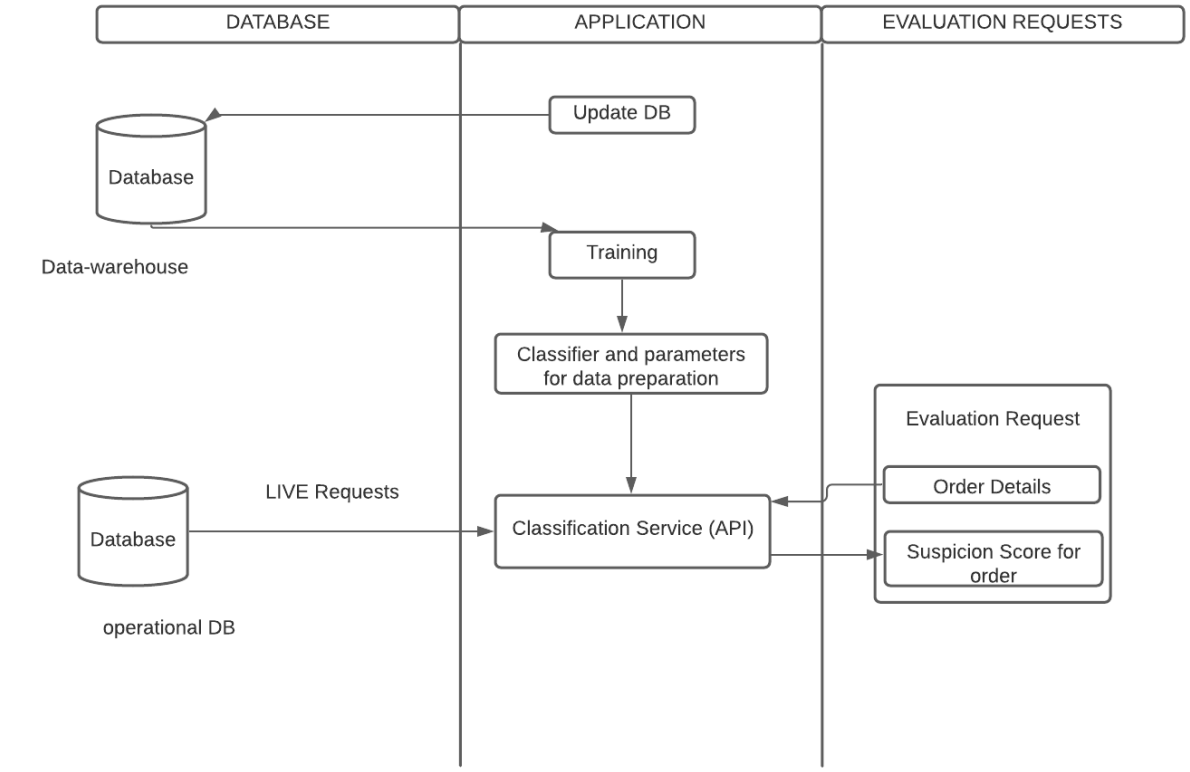
According to the authors, in this project work the main contribution of their research is an intelligent method for credit card fraud detection using an optimized light gradient boosting machine in which a bayesian-based hyper parameter optimization algorithm is utilized to improvise the parameters of the Optimized Light Gradient Boosting Machine- OLightGBM. The execution of the proposed intelligent technique is assessed based on two real-world data sets and compared with some machine learning algorithms/techniques using performance evaluation metrics such as AUC, F1-score, precision and Accuracy.

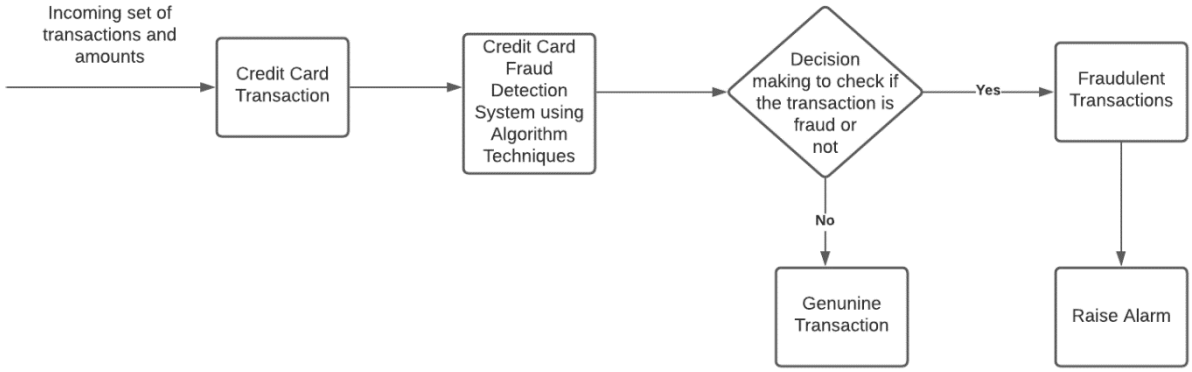
16. Naik, Het and Prashasti Kanikar. "Credit card Fraud Detection based on Machine Learning Algorithms." *International Journal of Computer Applications* 182 (2019): 8-12.

The number of online bank transactions is increasing day by day, number of fraudulent transactions are also increasing quickly. In this research paper to to reduce such fraud transactions, various machine learning algorithms like Naive Bayes, Logistic regression (LR), J48 and AdaBoost etc. are surveyed upon and comparative study is done. These algorithms are applied and tested using an online dataset which is collected. According to the analysis done it can be concluded that Logistic regression and AdaBoost algorithms perform better in fraud detection and also on the basis of the evaluation metrics accuracy and time taken scores shown in the research paper.

System Architecture [High level design and low-level design diagrams]







Explanation of the Architecture

The architecture explains that first we read the dataset in our project and then we perform under sampling to avoid the accuracy paradox to take place. Our original dataset is highly imbalanced as most of our transactions are non-fraud. If we use this data frame as the base for our predictive models and analysis, we might get a lot of errors and our algorithms will probably overfit since it will assume that most transactions are not fraud. But we don't want our model to assume, we want our model to detect patterns that give signs of fraud. According to this it means prediction will get a very high accuracy score without detecting a fraud transaction.
When we apply any classification algorithm directly on the data set we will get the results dealing with a problem called accuracy paradox. The ACCURACY PARADOX is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analytics. This is because a simple model may have a high level of accuracy but be too crude to be useful.

So, then after performing under sampling we divide the dataset into train and test and build the 5 models namely Logistic Regression, Random Forest, K Nearest Neighbours, SVC and Decision Tree. Then we calculate the prediction outcomes for test data, we test the sample data and finally get the Performance and Accuracy results.

Data set description

Link for the data set:
https://www.kaggle.com/mlg-ulb/creditcardfraud

The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numeric input variables which are the result of a PCA transformation. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. There are no "Null" values, so we don't have to work on ways to replace values.

```
In [4]: dataset = pd.read_csv('creditcard.csv')

        dataset.describe()
        dataset.head()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

Algorithms/Pseudo code

Step 1: Import Libraries such as numpy, panda, seaborn and matplotlib.
Step 2: Read the creditcard.csv file for the dataset (dt).
Step 3: The distributions show that the features are heavily skewed and we solve this issue by performing under sampling on dt.
Step 4: Scale the columns amount and time using Robust Scaler.
Step 5: Shuffle dt and create the subsamples to have equal fraud and non-fraud transactions.
Step 6: Implement classifiers namely Logistic Regression, Decision Tree, KNN, SVC and Random Forest on the balanced dt.
Step 7: Get the accuracy, precision etc. scores and perform metrics analysis.
Step 8: Deploy the most efficient algorithm to a website using flask.
Step 9: Enter the input in the website and display results as “Fraud” or “Non- Fraud” to show it as a running website and instantly giving results according to the algorithm deployed.

Explanation of the Algorithms/Pseudo code

In our project the comparison is made for different machine learning algorithms such as Logistic Regression, Decision Trees, Random Forest, to determine which algorithm suits best and can be adapted by credit card merchants for identifying fraud transactions. The algorithm given above explains that first we Read the Dataset. Then Random Sampling is done on the data set to make it balanced. Now, the dataset is divided into two parts i.e., Train dataset and Test dataset. Accuracy and performance metrics has been calculated to know the efficiency for different algorithms such as KNN, Decision Tree, Random Forest, Logistic Regression and SVC. And then we retrieve the best algorithm based on efficiency for the given dataset. Finally we deploy the most efficient algorithm into a website using flask and display the results as per the inputs given by the user to display the running functionality of the website for credit card fraud detection.

Implementation Code

Part 1 - Developing machine learning model

Working of the project

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Dataset

```
dataset.describe()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	..
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15	..
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	..
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	..
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	..
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	..
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	..
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	..

8 rows × 31 columns

```
dataset.describe()
```

Out[2]:

V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0e+05	...	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
0e-15	...	1.473120e-16	8.042109e-16	5.282512e-16	4.456271e-15	1.426896e-15	1.701640e-15	-3.662252e-16	-1.217809e-16	88.349619	0.001727
2e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
7e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
6e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
3e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
0e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
9e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

```
dataset.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

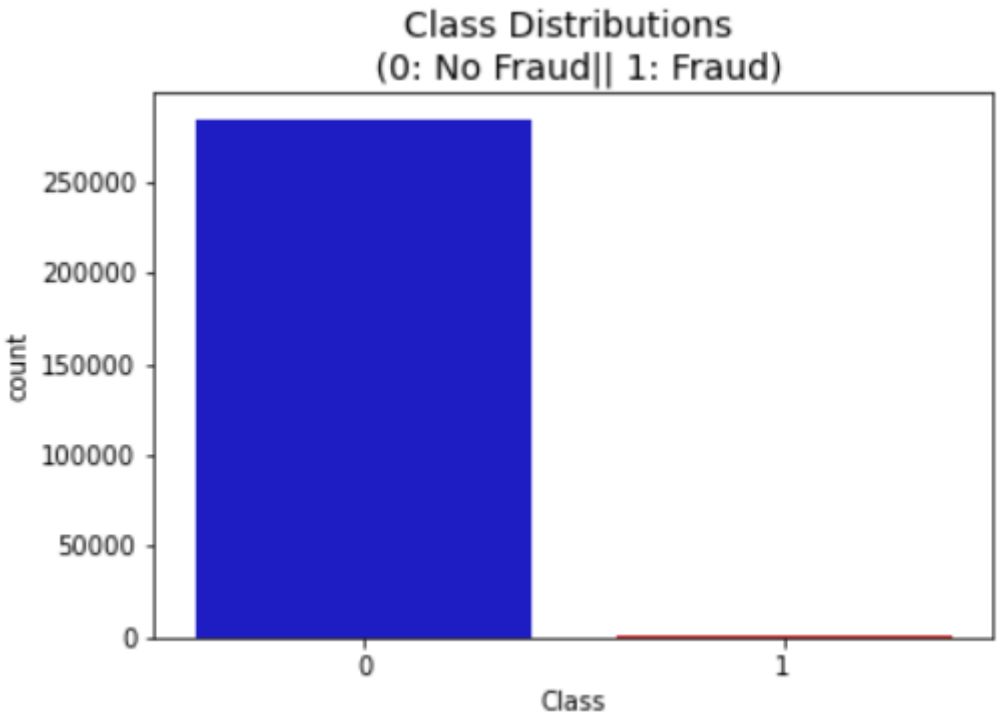
5 rows × 31 columns

```
dataset.head()
```

Out[3]:

	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
5	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
4	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
0	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
1	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

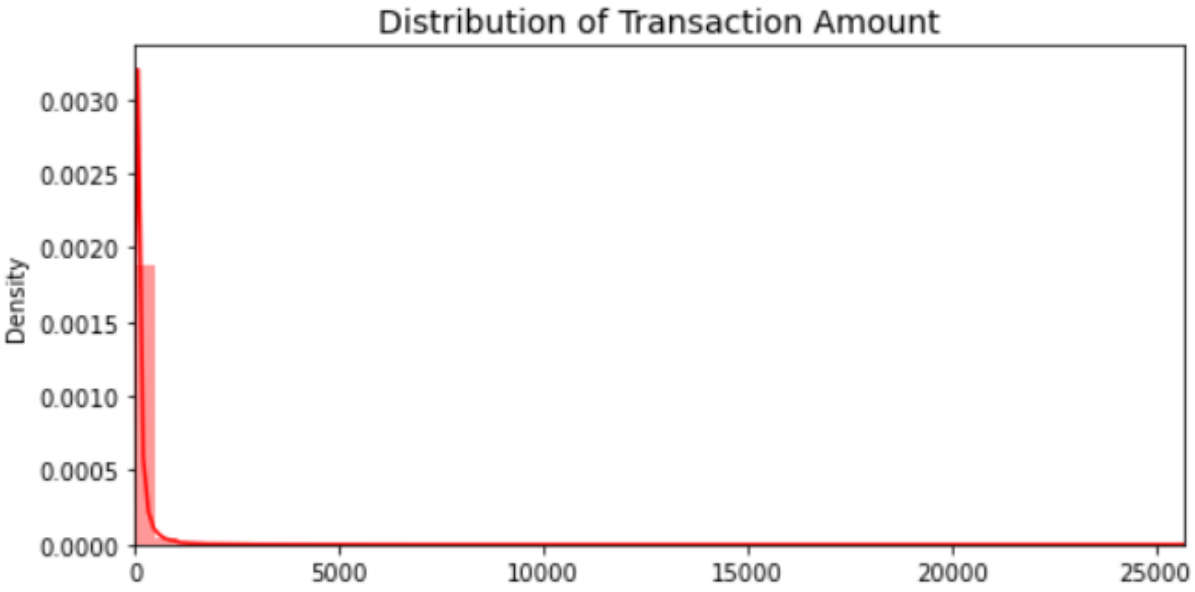
Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification. Directly running prediction on the dataset is not a good idea.

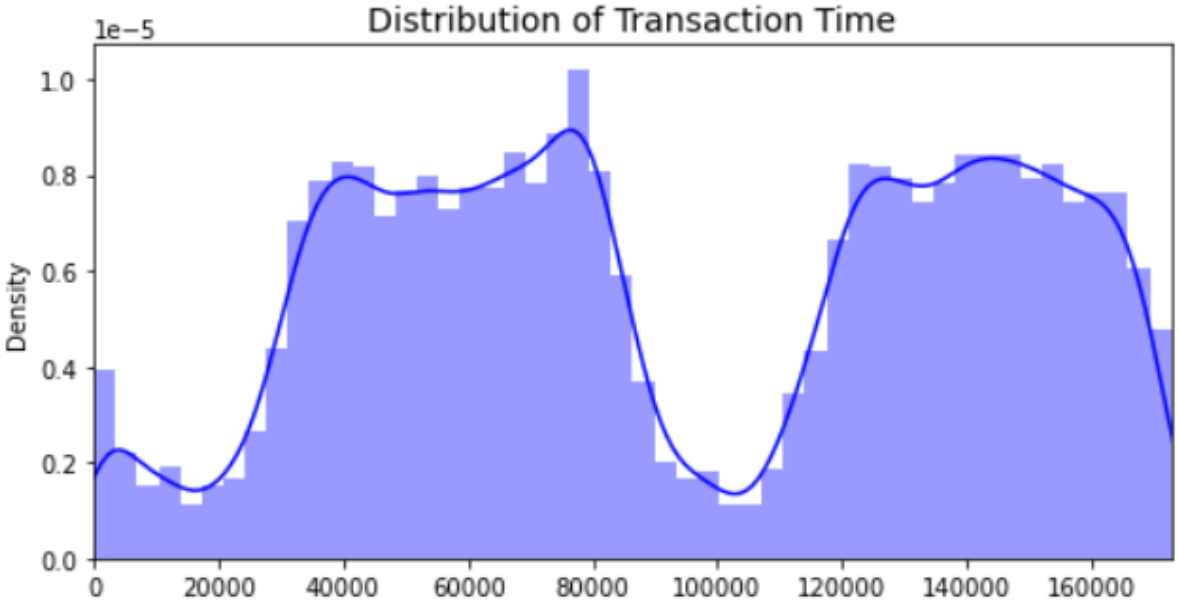


No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset

When we apply any classification algorithm directly on the dataset we will get the results dealing with a problem called accuracy paradox. The ACCURACY PARADOX is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analytics. This is because a simple model may have a high level of accuracy but be too crude to be useful. Precision and recall are better measures in such cases. The underlying issue is that there is a class imbalance between the positive class and the negative class. Prior probabilities for these classes need to be accounted for in error analysis.

Also, by seeing the distributions, we can have an idea how skewed these features are. There are techniques that can help the distributions be less skewed which we will be implementing.





To handle this kind of problem one better way is to class distribution, i.e., sampling minority classes. In the sampling minority, class training examples can be increased in proportion to the majority class to raise the chance of correct prediction by the algorithm.

Scaling and Distributing

In this phase, we will first scale the columns composed of Time and Amount. Time and amount should be scaled as the other columns.
On the other hand, we need to also create a subsample of the data frame in order to have an equal amount of Fraud and Non-Fraud cases, helping our algorithms better understand patterns that determine whether a transaction is a fraud or not.
In this scenario, our subsample will be a data frame with a 50/50 ratio of fraud and

non-fraud transactions. Meaning our sub-sample will have the same amount of fraud and non-fraud transactions.

Using the original data frame will cause the following issues:
1. Overfitting: Our classification models will assume that in most cases there are no frauds. What we want for our model is to be certain when a fraud occurs.
2. Wrong Correlations: By having an imbalance data frame we are not able to see the true correlations between the class and features.

We use Robust Scalar to scale time and amount, the reason being that they scale features using statistics that are robust to outliers.

Standardization of a dataset is a common requirement for many machine learning estimators. Typically, this is done by removing the mean and scaling to unit variance.
However, outliers can often influence the sample mean / variance in a negative way.
In such cases, the median and the interquartile range often give better results.
As Robust Scaler, Quantile Transformer is robust to outliers in the sense that adding or removing outliers in the training set will yield approximately the same transformation on held out data. But contrary to Robust Scaler, Quantile Transformer will also automatically collapse any outlier by setting them to the apriori defined range boundaries (0 and 1).
Splitting the Data (Original DataFrame)

Summary:
● There are 492 cases of fraud in our dataset so we can randomly get 492 cases of non-fraud to create our new sub data frame.
● We concatenate the 492 cases of fraud and non-fraud, creating a new sub-sample.
Scaled amount and scaled time are the columns with scaled values:

In [6]:
```
from sklearn.preprocessing import StandardScaler, RobustScaler
# RobustScaler is less prone to outliers.
#std_scaler = StandardScaler()
rob_scaler = RobustScaler()

dataset['scaled_amount'] = rob_scaler.fit_transform(dataset['Amount'].values.reshape(-1,1))
dataset['scaled_time'] = rob_scaler.fit_transform(dataset['Time'].values.reshape(-1,1))

dataset.drop(['Time','Amount'], axis=1, inplace=True)

scaled_amount = dataset['scaled_amount']
scaled_time = dataset['scaled_time']

dataset.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
dataset.insert(0, 'scaled_amount', scaled_amount)
dataset.insert(1, 'scaled_time', scaled_time)

# Amount and Time are Scaled!

dataset.head()
```

Out[6]:

```
dataset.head()
```

	scaled_amount	scaled_time	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Class
0	1.783274	-0.994983	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	...	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0
1	-0.269825	-0.994983	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	...	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0
2	4.983721	-0.994972	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	...	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0
3	1.418291	-0.994972	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	...	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0
4	0.670579	-0.994960	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	...	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0

5 rows × 31 columns

Out[6]:

	scaled_amount	scaled_time
0	1.783274	-0.994983
1	-0.269825	-0.994983
2	4.983721	-0.994972
3	1.418291	-0.994972
4	0.670579	-0.994960

5 rows × 31 columns

The main goal is to fit the model either with the data frames that were under sample and oversample (in order for our models to detect the patterns), and test it on the original testing set.
One of the most common and simplest strategies to handle imbalanced data is to under sample the majority class.
Oversampling the minority class can result in overfitting problems if we oversample before cross-validating.

In [7]:

```
print('No Frauds', round(dataset['Class'].value_counts()[0]/len(dataset) * 100,2), '% of the dataset')
print('Frauds', round(dataset['Class'].value_counts()[1]/len(dataset) * 100,2), '% of the dataset')

X = dataset.drop('Class', axis=1)
y = dataset[['Class']]


#from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
for train_index, test_index in sss.split(X, y):
    print("Train:", train_index, "Test:", test_index)
    original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
    original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
```

```
No Frauds 99.83 % of the dataset
Frauds 0.17 % of the dataset
Train: [265518 180305  42664 ...  29062  13766  17677] Test: [263020  11378 147283 ... 274532 269819  64170]
Train: [ 72227 114282  16818 ... 264471 191914 284017] Test: [202638  32978 128121 ... 244024 127667  48318]
Train: [ 20895 114622 167683 ... 244502 178972 218506] Test: [284352  82483  90981 ... 171224 168807 271602]
Train: [122248 181660 194400 ... 104631 277586  29432] Test: [225673  63348  68025 ... 279451  77554  76043]
Train: [241684 223467 136928 ...  86495 160550  49633] Test: [157557 204860  83760 ... 251478 178967 216850]
```

In [8]:

```
# We already have X_train and y_train for undersample data thats why I am using original to distinguish and to not overwrite these variables.
# original_Xtrain, original_Xtest, original_ytrain, original_ytest = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the Distribution of the labels

# Turn into an array
original_Xtrain = original_Xtrain.values
original_Xtest = original_Xtest.values
original_ytrain = original_ytrain.values
original_ytest = original_ytest.values

# See if both the train and test label distribution are similarly distributed
train_unique_label, train_counts_label = np.unique(original_ytrain, return_counts=True)
test_unique_label, test_counts_label = np.unique(original_ytest, return_counts=True)
print('-' * 100)

print('Label Distributions: \n')
print(train_counts_label/ len(original_ytrain))
print(test_counts_label/ len(original_ytest))
```

```
----------------------------------------------------------------------------------------------------
Label Distributions:

[0.99827075 0.00172925]
[0.99827955 0.00172045]
```

Under Sampling:

In this phase of the project, we implement "Under Sampling" which basically consists of removing data in order to have a more balanced dataset and thus avoiding our model's overfitting.

- The first thing we have to do is determine how imbalanced is our class (use "value_counts()" on the class column to determine the amount for each label)
- Once we determine how many instances are considered fraud transactions (Fraud = "1") , we should bring the non-fraud transactions to the same amount as fraud transactions (assuming we want a 50/50 ratio), this will be equivalent to 492 cases of fraud and 492 cases of non-fraud transactions.
- After implementing this technique, we have a sub-sample of our data frame with a 50/50 ratio with regards to our classes. Then the next step we will implement is to shuffle the data to see if our models can maintain a certain accuracy every time, we run this script.

In [9]:
```
dataset = dataset.sample(frac=1)

# amount of fraud classes 492 rows.
fraud_df = dataset.loc[dataset['Class'] == 1]
non_fraud_df = dataset.loc[dataset['Class'] == 0][:497]

normal_distributed_df = pd.concat([fraud_df, non_fraud_df])

# Shuffle dataframe rows
new_df = normal_distributed_df.sample(frac=1, random_state=42)

new_df.head()
new_df.describe()
```

In [10]:
```
print('Distribution of the Classes in the subsample dataset')
print(new_df['Class'].value_counts()/len(new_df))

sns.countplot('Class', data=new_df, palette=colors)
plt.title('Equally Distributed Classes', fontsize=14)
plt.show()
```

```
Distribution of the Classes in the subsample dataset
0    0.502528
1    0.497472
Name: Class, dtype: float64
```

Now that we have our data frame correctly balanced, we can go further with our analysis and data pre-processing.

Now that we have sampled our data set, we apply our supervised learning models.

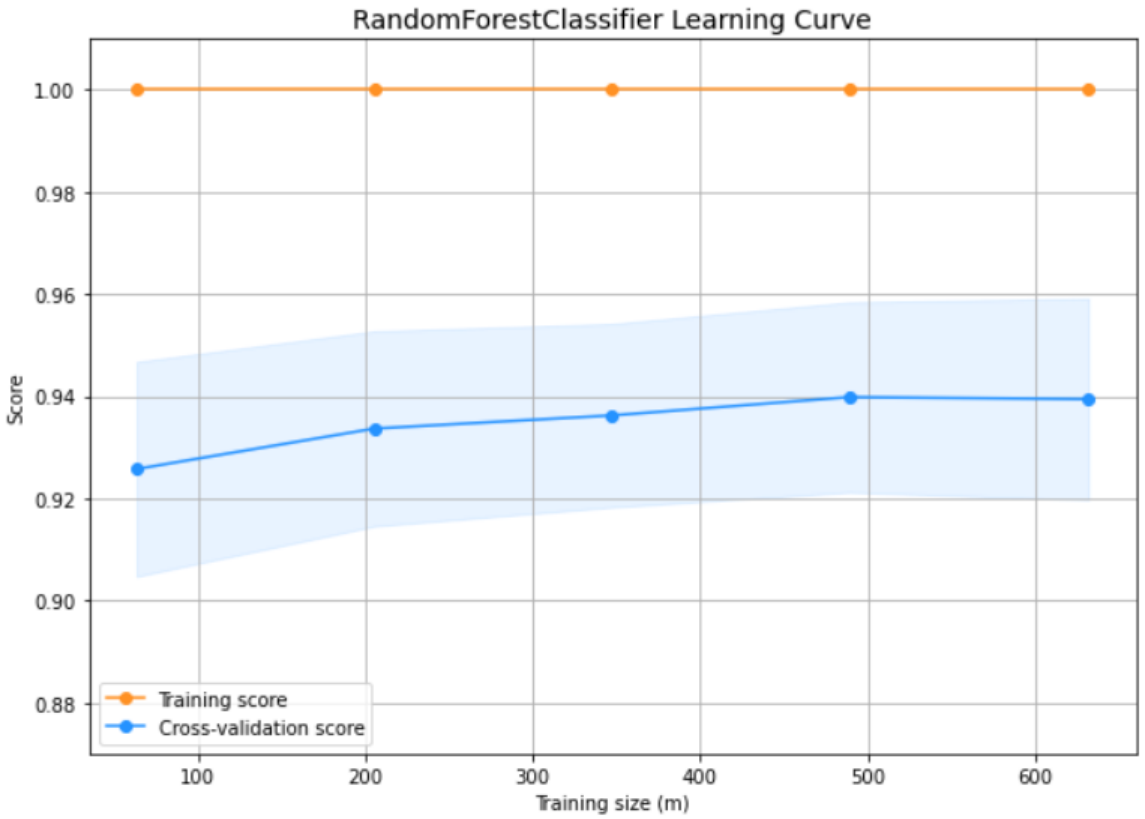
In [14]:

```
# Classifier Libraries
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


classifiers = {
    "RandomForestClassifier": RandomForestClassifier(n_estimators=300),
    "LogisiticRegression": LogisticRegression(),
    "KNearest": KNeighborsClassifier(),
    "Support Vector Classifier": SVC(),
    "DecisionTreeClassifier": DecisionTreeClassifier(),

}

for key, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    #metrics
    print("Classifiers: ", classifier.__class__.__name__)
    cm=confusion_matrix(y_test, y_pred)
    print(cm)

    training_score = cross_val_score(classifier, X_train, y_train, cv=5)
    cv = ShuffleSplit(n_splits=100, test_size=0.2, random_state=42)
    x=plot_learning_curve(classifier, X_train, y_train,key, (0.87, 1.01), cv=cv, n_jobs=4)
    x.show()
    f1=f1_score(y_test, y_pred)
    acc=accuracy_score(y_test, y_pred)
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    pr=precision_score(y_test, y_pred)
    print("tn, fp, fn, tp")
    t=tp+tn+fn+fp
    print(tn/t, fp/t, fn/t, tp/t)
    print("specificity: "+str(tn/(tn+fp)))
    print("sensitivity: "+str(tp/(tp+fn)))
    print(key)
    print("accuracy_score:",acc)
    print("precision_score:",pr)
    print("f1_score:",f1)
```

Results

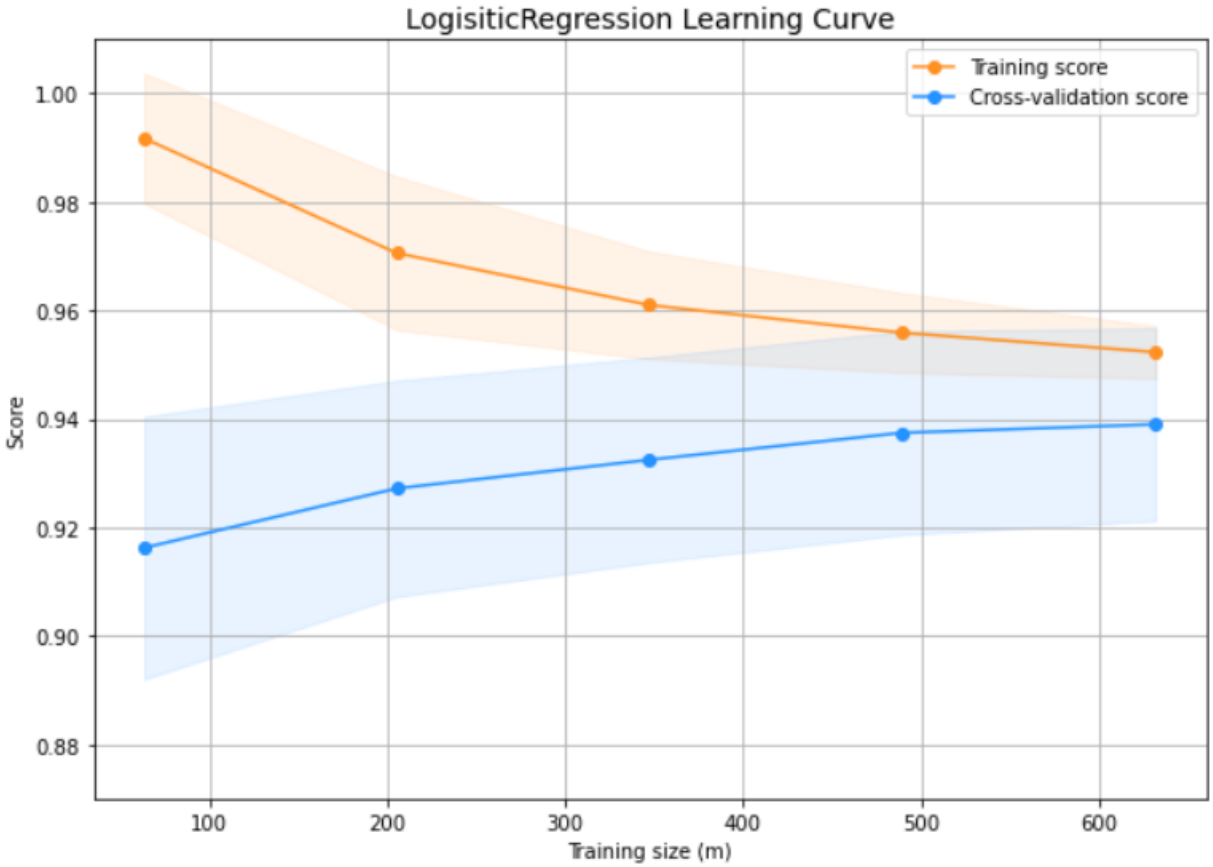
The following screenshots display the results of supervised learning models namely Random Forest, Logistic Regression, KNN, SVC and Decision Tree.

```
Classifiers:  RandomForestClassifier
[[89  2]
 [ 9 98]]
```



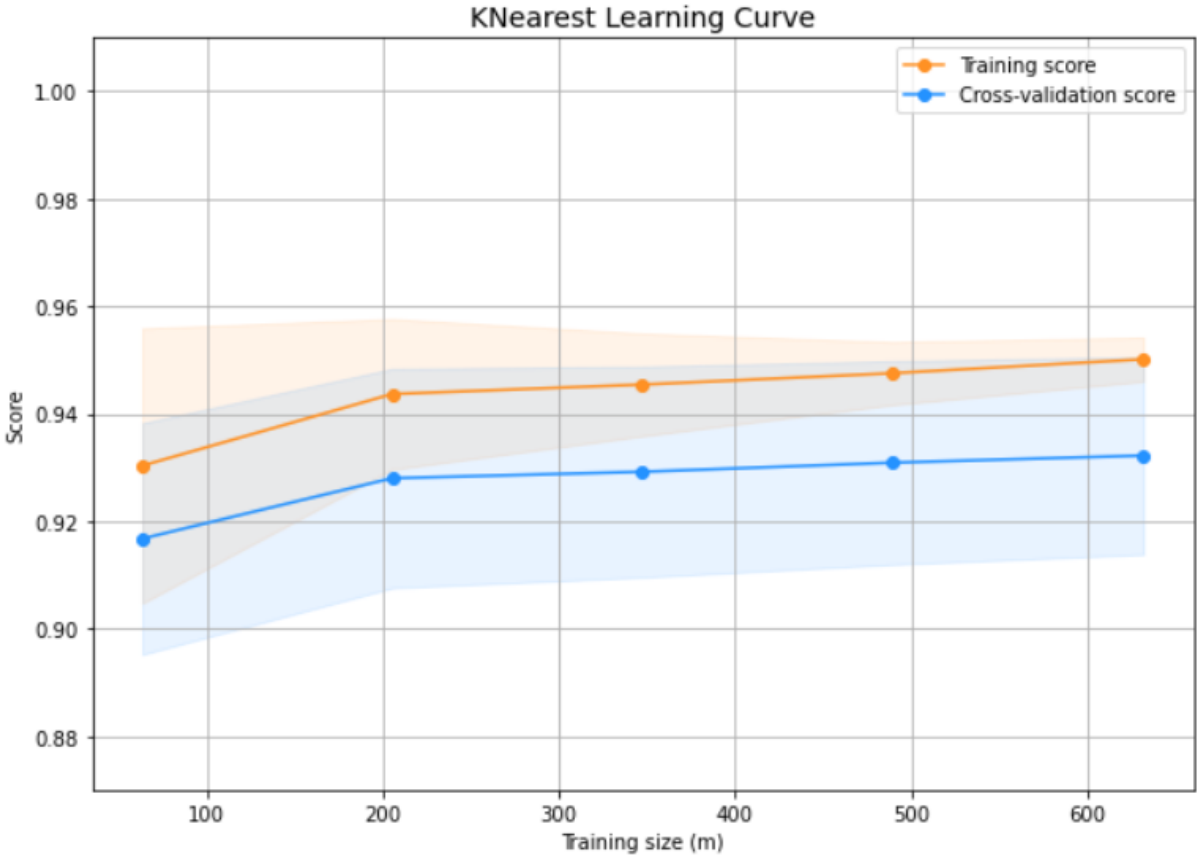

```
tn, fp, fn, tp
0.4494949494949495 0.010101010101010102 0.045454545454545456 0.494949494949495
specificity: 0.978021978021978
sensitivity: 0.9158878504672897
RandomForestClassifier
accuracy_score: 0.9444444444444444
precision_score: 0.98
f1_score: 0.9468599033816426
```

```
Classifiers:  LogisticRegression
[[89  2]
 [ 8 99]]
```



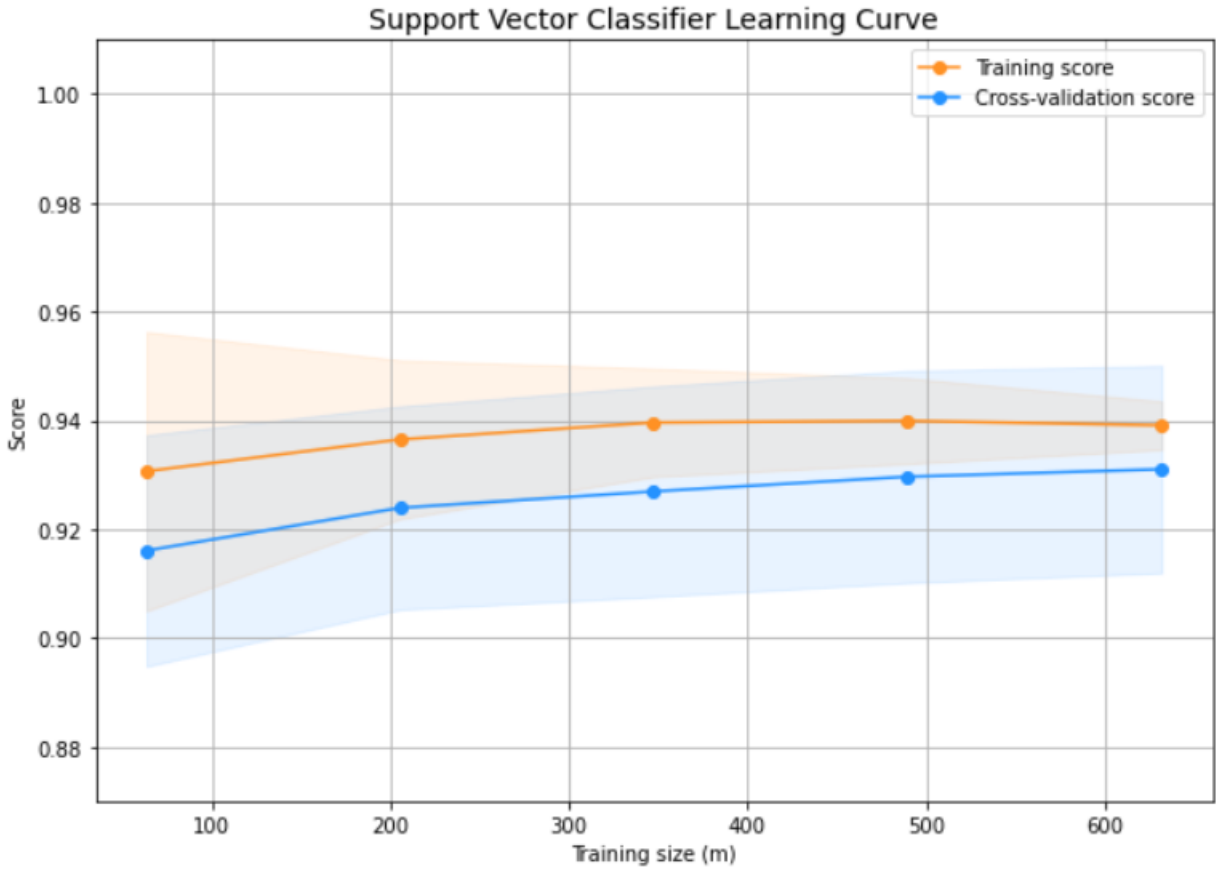

```
tn, fp, fn, tp
0.4494949494949495 0.010101010101010102 0.04040404040404041 0.5
specificity: 0.978021978021978
sensitivity: 0.9252336448598131
LogisiticRegression
accuracy_score: 0.9494949494949495
precision_score: 0.9801980198019802
f1_score: 0.9519230769230769
```

```
Classifiers:  KNeighborsClassifier
[[91  0]
 [12 95]]
```



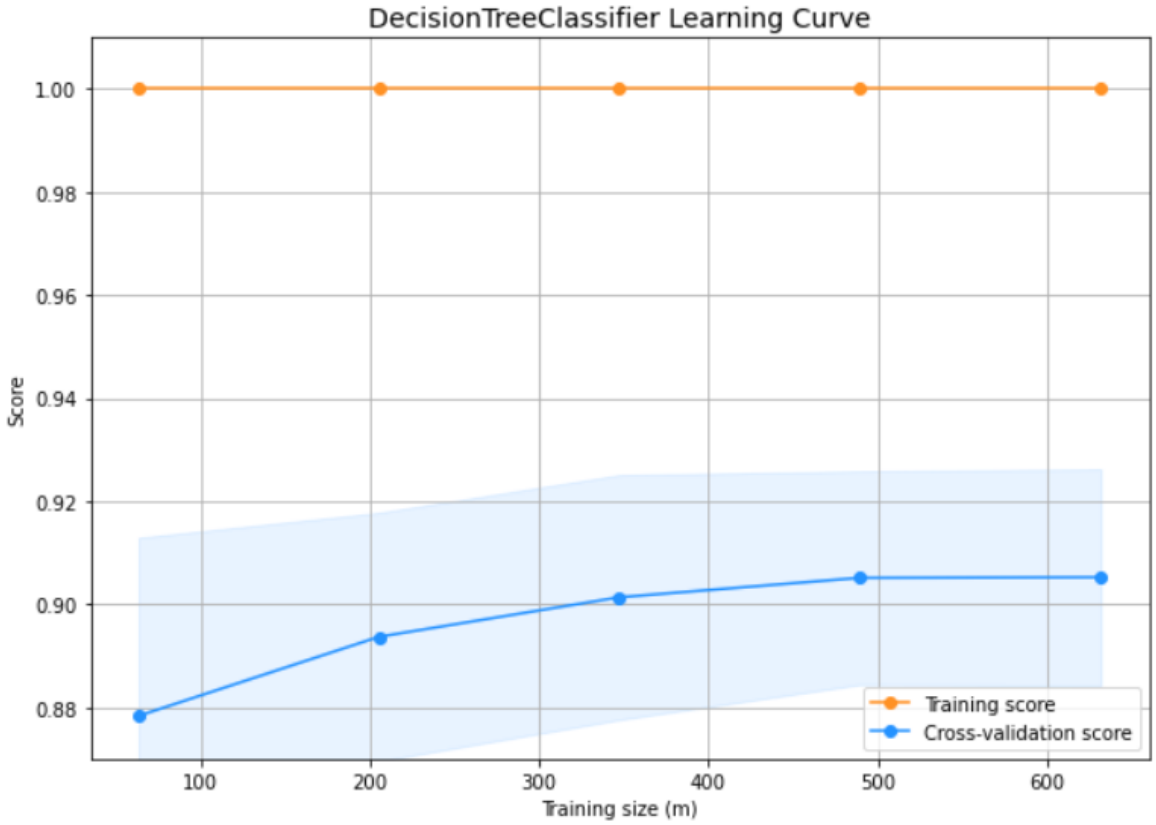

```
tn, fp, fn, tp
0.4595959595959596 0.0 0.06060606060606061 0.4797979797979798
specificity: 1.0
sensitivity: 0.8878504672897196
KNearest
accuracy_score: 0.9393939393939394
precision_score: 1.0
f1_score: 0.9405940594059405
```

```
Classifiers:  SVC
[[90  1]
 [12 95]]
```




```
tn, fp, fn, tp
0.45454545454545453 0.005050505050505051 0.06060606060606061 0.4797979797979798
specificity: 0.989010989010989
sensitivity: 0.8878504672897196
Support Vector Classifier
accuracy_score: 0.9343434343434344
precision_score: 0.9895833333333334
f1_score: 0.9359605911330049
```

```
Classifiers:  DecisionTreeClassifier
[[ 83   8]
 [  5 102]]
```




```
tn, fp, fn, tp
0.41919191919191917 0.04040404040404041 0.025252525252525252 0.5151515151515151
specificity: 0.9120879120879121
sensitivity: 0.9532710280373832
DecisionTreeClassifier
accuracy_score: 0.9343434343434344
precision_score: 0.9272727272727272
f1_score: 0.9400921658986175
```

In order to compare various techniques, we calculate the true positive, true negative, false positive and false negative generated by a system or an algorithm and use these in quantitative measurements to evaluate and compare performance of different systems.
True Positive (TP) is the number of transactions that were fraudulent and were also classified as fraudulent by the system. True Negative (TN) is the number of transactions that were legitimate and were also classified as legitimate. False Positive (FP) is the number of transactions that were legitimate but were wrongly classified as fraudulent transactions. False Negative (FN) is the number of transactions that were fraudulent but were wrongly classified as legitimate transactions by the system.

Basic Metrics for sampled data distribution

Metrics	Random Forest	Decision Tree	KNearest	SVC	Logistic Regression
True Positive	98	102	95	95	99
False Positive	2	8	0	1	2
False Negative	9	5	12	12	8
True Negative	89	83	91	90	89

Accuracy Results for sampled data distribution

Metrics	Random Forest	Decision Tree	KNearest	SVC	Logistic Regression
Accuracy	0.944	0.934	0.93	0.934	0.949
Sensitivity	0.915	0.953	0.88	0.887	0.925
Specificity	0.978	0.912	1.0	0.989	0.978
Precision	0.98	0.927	1.0	0.98	0.9801
F1 score	0.946	0.940	0.940	0.935	0.951

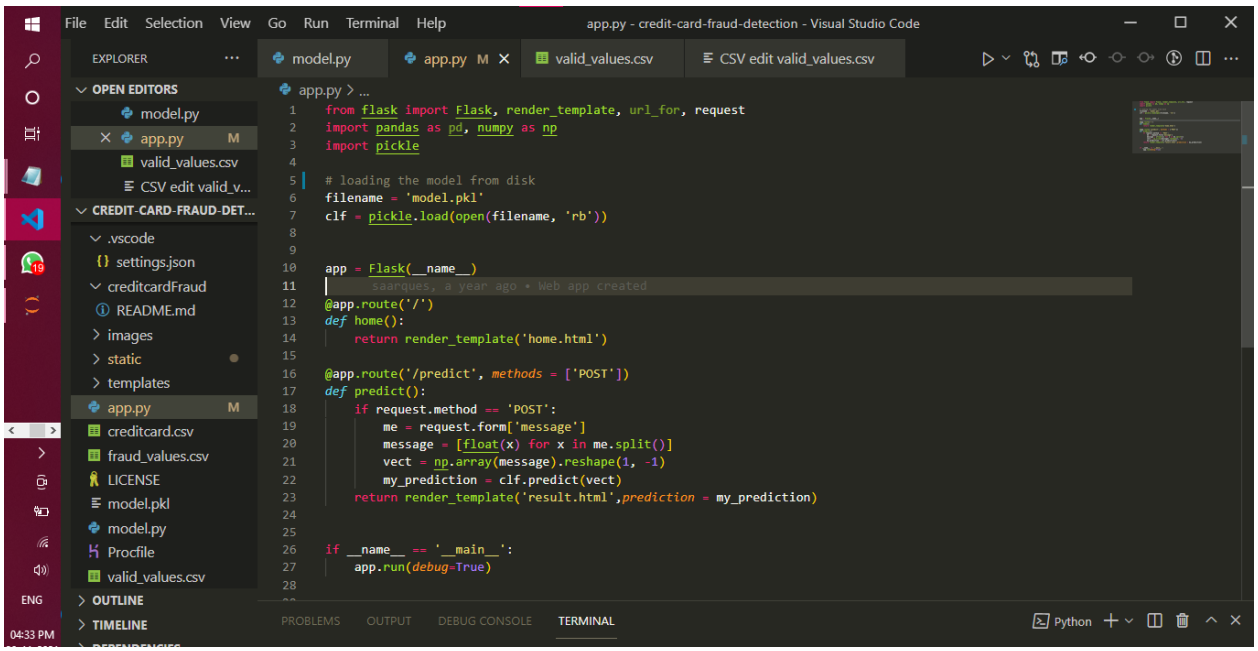
From the above table it is clear that Logistic regression gives the highest accuracy followed by Random Forest and SVC.
Thus, we use the logistic regression model to deploy a web app for our project.

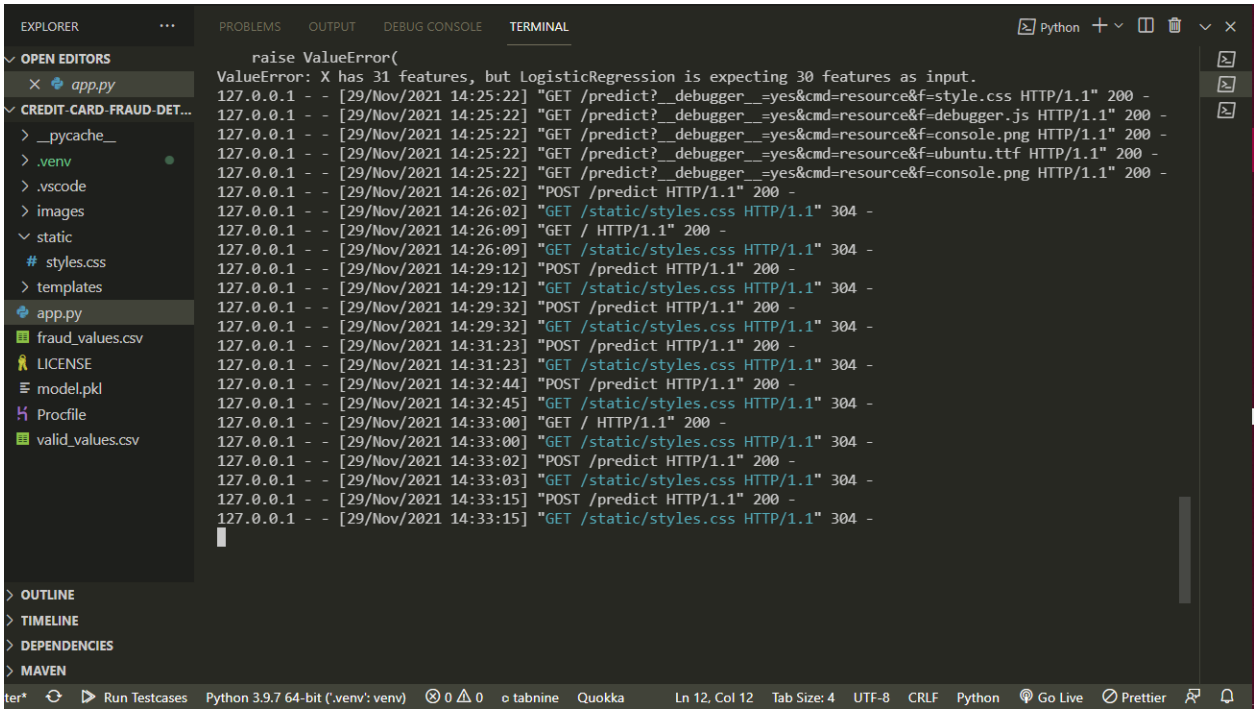
Part 2 - Deploying machine learning model

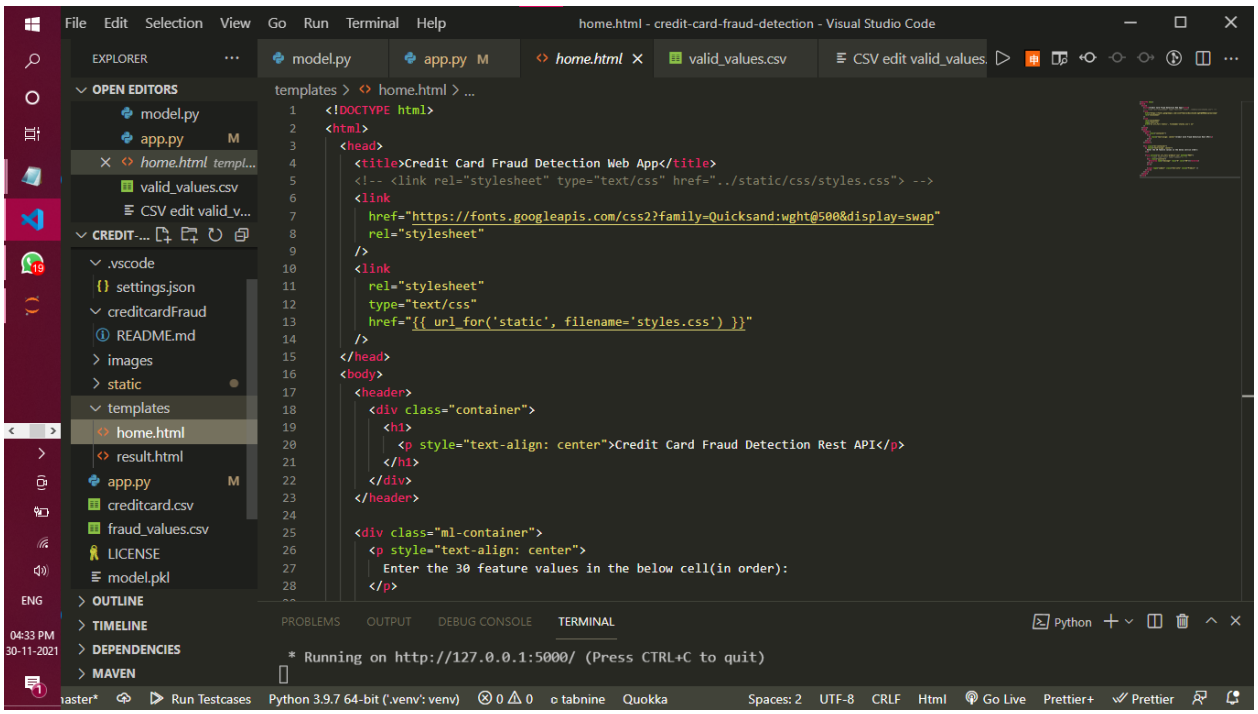
Web App Production

Templates - Files required for rendering purpose
Static - CSS styles
App - Main file which will run our Web App
Test Data - fraud_values.csv and valid_values.csv

Code files


File Edit Selection View Go Run Terminal Help
app.py - credit-card-fraud-detection - Visual Studio Code
EXPLORER
model.py
app.py M
valid_values.csv
CSV edit valid_values.csv
OPEN EDITORS
CREDIT-CARD-FRAUD-DET...
.vscode
settings.json
creditcardFraud
README.md
images
static
templates
app.py
creditcard.csv
fraud_values.csv
LICENSE
model.pkl
model.py
Procfile
valid_values.csv
OUTLINE
TIMELINE
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
from flask import Flask, render_template, url_for, request
import pandas as pd, numpy as np
import pickle
# loading the model from disk
filename = 'model.pkl'
clf = pickle.load(open(filename, 'rb'))
app = Flask(__name__)
saarques, a year ago • Web app created
@app.route('/')
def home():
return render_template('home.html')
@app.route('/predict', methods = ['POST'])
def predict():
if request.method == 'POST':
me = request.form['message']
message = [float(x) for x in me.split()]
vect = np.array(message).reshape(1, -1)
my_prediction = clf.predict(vect)
return render_template('result.html',prediction = my_prediction)
if __name__ == '__main__':
app.run(debug=True)


EXPLORER
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
OPEN EDITORS
app.py
CREDIT-CARD-FRAUD-DET...
__pycache__
.venv
.vscode
images
static
styles.css
templates
app.py
fraud_values.csv
LICENSE
model.pkl
Procfile
valid_values.csv
raise ValueError(
ValueError: X has 31 features, but LogisticRegression is expecting 30 features as input.
127.0.0.1 - - [29/Nov/2021 14:25:22] "GET /predict?__debugger__=yes&cmd=resource&f=style.css HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:25:22] "GET /predict?__debugger__=yes&cmd=resource&f=debugger.js HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:25:22] "GET /predict?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:25:22] "GET /predict?__debugger__=yes&cmd=resource&f=ubuntu.ttf HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:25:22] "GET /predict?__debugger__=yes&cmd=resource&f=console.png HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:26:02] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:26:02] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:26:09] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:26:09] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:29:12] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:29:12] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:29:32] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:29:32] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:31:23] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:31:23] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:32:44] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:32:45] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:33:00] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:33:00] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:33:02] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:33:03] "GET /static/styles.css HTTP/1.1" 304 -
127.0.0.1 - - [29/Nov/2021 14:33:15] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2021 14:33:15] "GET /static/styles.css HTTP/1.1" 304 -
OUTLINE
TIMELINE
DEPENDENCIES
MAVEN
Run Testcases
Python 3.9.7 64-bit ('.venv': venv)
tabnine
Quokka
Ln 12, Col 12
Tab Size: 4
UTF-8
CRLF
Python
Go Live
Prettier

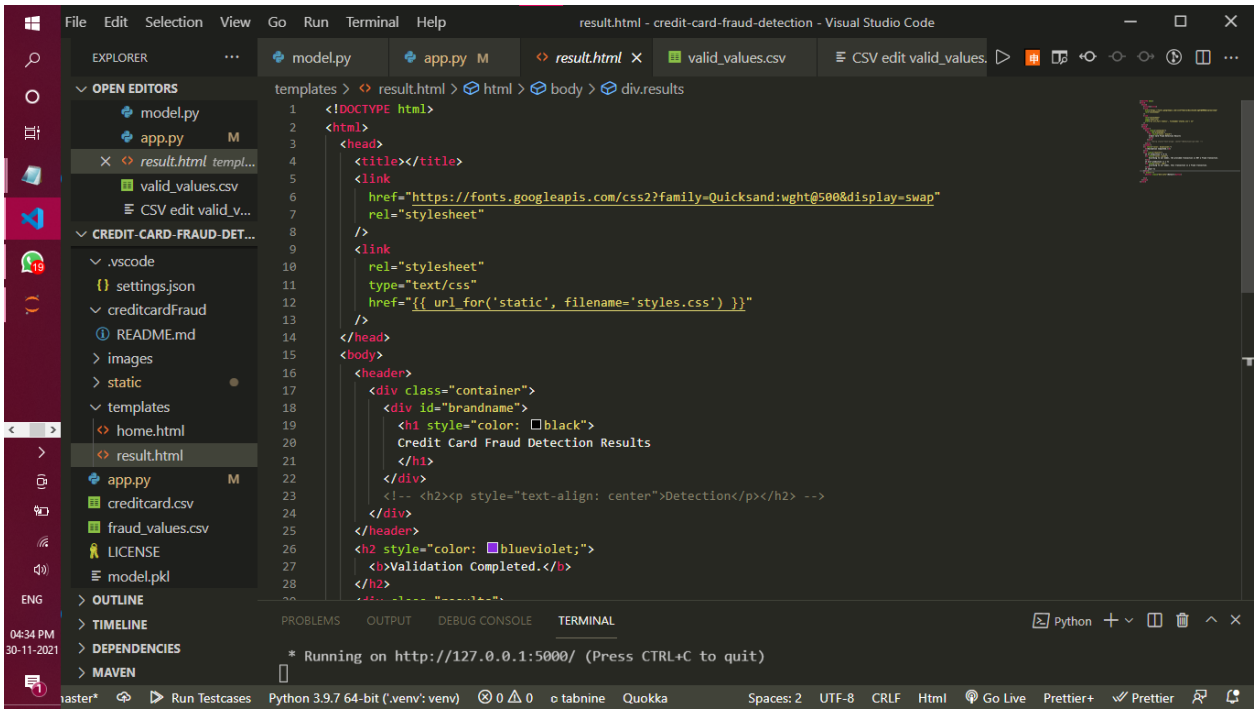

File Edit Selection View Go Run Terminal Help
home.html - credit-card-fraud-detection - Visual Studio Code
EXPLORER
model.py
app.py M
home.html
valid_values.csv
CSV edit valid_values
templates > home.html > ...
<!DOCTYPE html>
<html>
<head>
<title>Credit Card Fraud Detection Web App</title>
<!-- <link rel="stylesheet" type="text/css" href="../static/css/styles.css"> -->
<link
href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display=swap"
rel="stylesheet"
/>
<link
rel="stylesheet"
type="text/css"
href="{{ url_for('static', filename='styles.css') }}"
/>
</head>
<body>
<header>

<h1>
Credit Card Fraud Detection Rest API
</h1>

</header>

Enter the 30 feature values in the below cell(in order):

OPEN EDITORS
CREDIT-...
.vscode
settings.json
creditcardFraud
README.md
images
static
templates
home.html
result.html
app.py
creditcard.csv
fraud_values.csv
LICENSE
model.pkl
OUTLINE
TIMELINE
DEPENDENCIES
MAVEN
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Run Testcases
Python 3.9.7 64-bit ('.venv': venv)
tabnine
Quokka
Spaces: 2
UTF-8
CRLF
Html
Go Live
Prettier+
Prettier
ENG
04:33 PM
30-11-2021

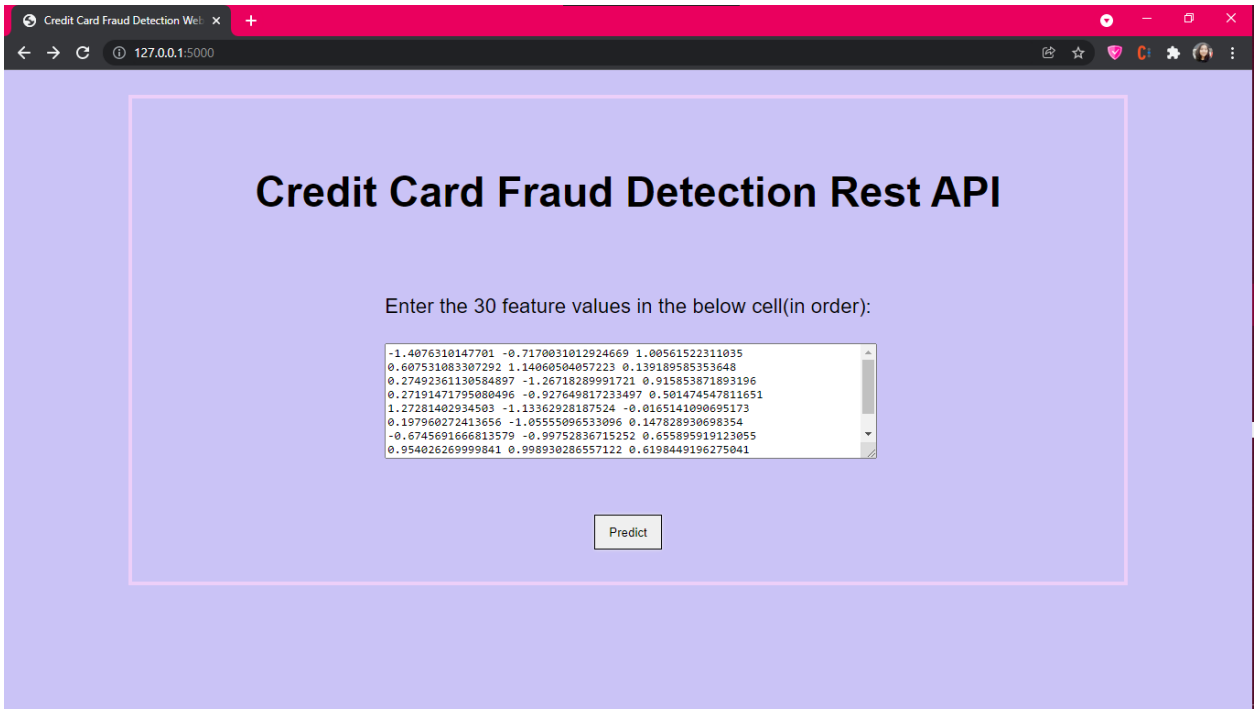

File Edit Selection View Go Run Terminal Help
result.html - credit-card-fraud-detection - Visual Studio Code
EXPLORER
model.py
app.py M
result.html
valid_values.csv
CSV edit valid_values
templates > result.html > html > body > div.results
<!DOCTYPE html>
<html>
<head>
<title></title>
<link
href="https://fonts.googleapis.com/css2?family=Quicksand:wght@500&display=swap"
rel="stylesheet"
/>
<link
rel="stylesheet"
type="text/css"
href="{{ url_for('static', filename='styles.css') }}"
/>
</head>
<body>
<header>

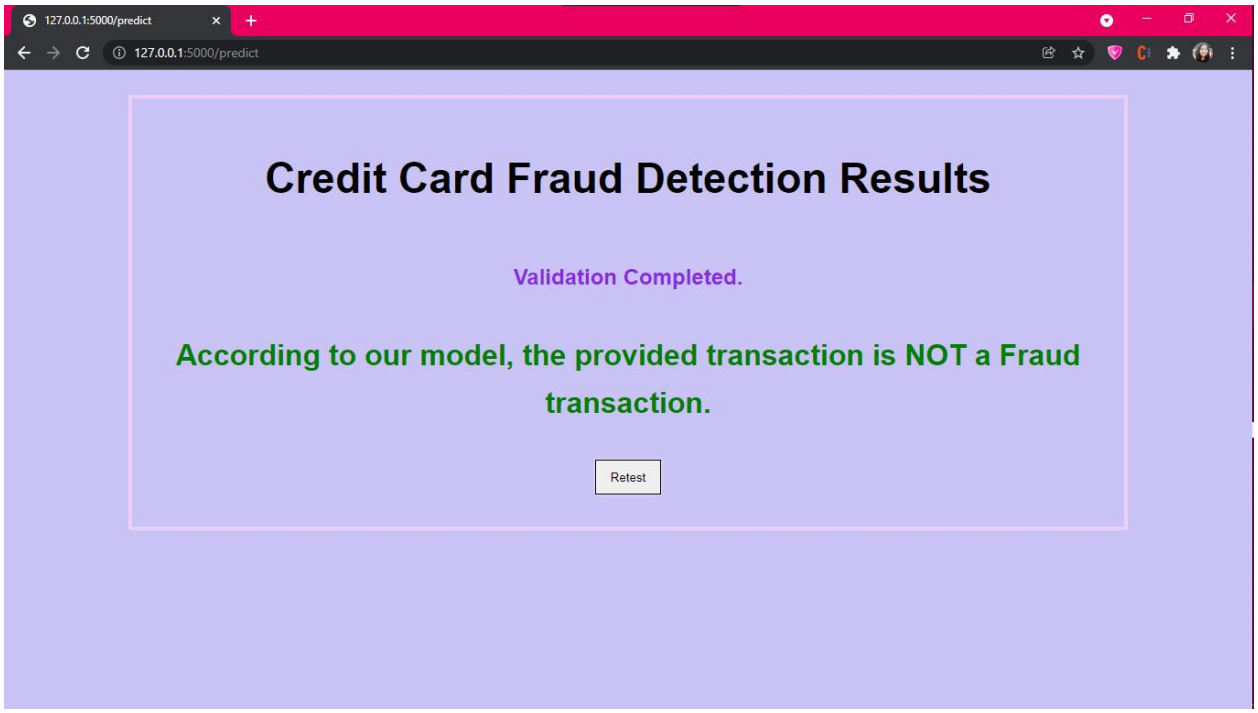
<h1 style="color: black">
Credit Card Fraud Detection Results
</h1>

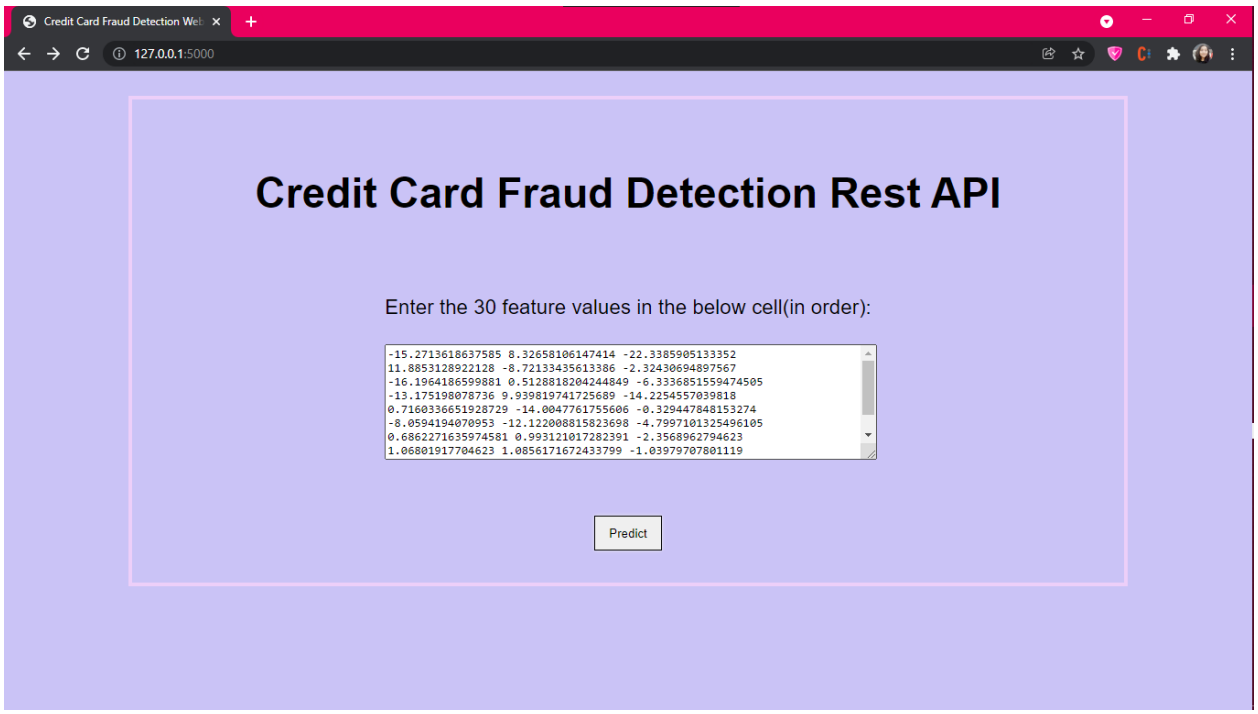
<!-- <h2> Detection </h2> -->

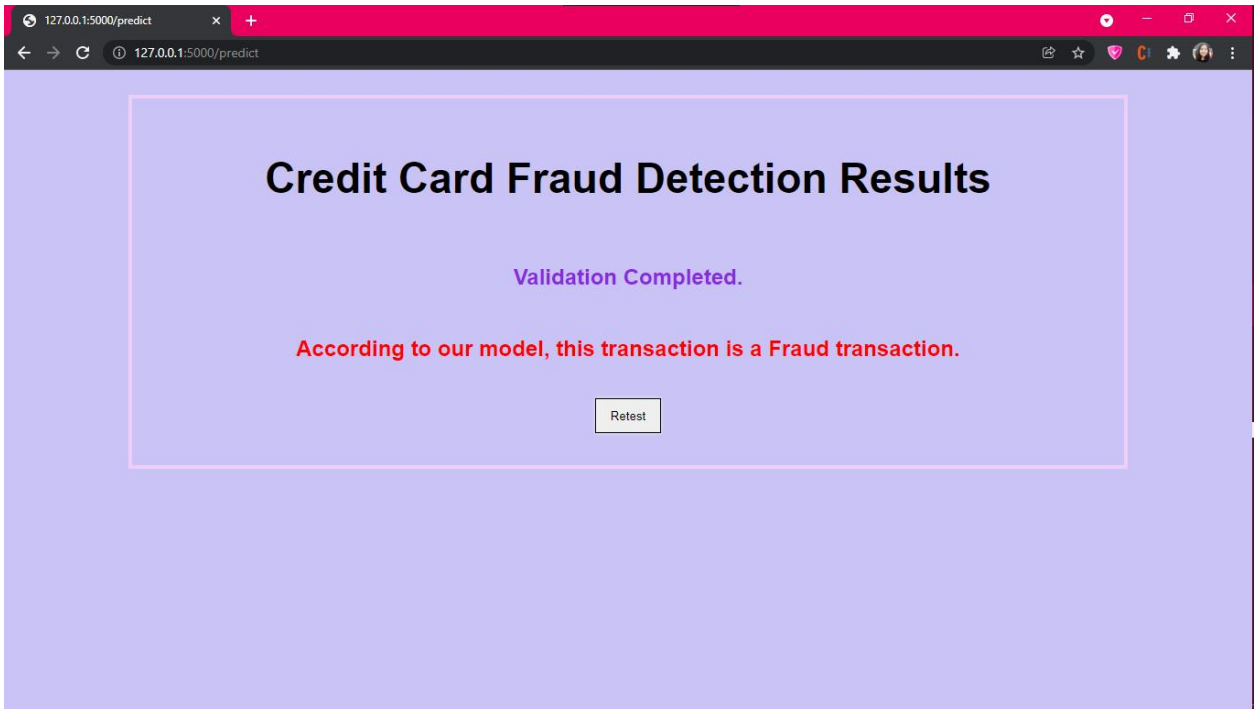
</header>
<h2 style="color: blueviolet;">
Validation Completed.
</h2>
OPEN EDITORS
CREDIT-CARD-FRAUD-DET...
.vscode
settings.json
creditcardFraud
README.md
images
static
templates
home.html
result.html
app.py
creditcard.csv
fraud_values.csv
LICENSE
model.pkl
OUTLINE
TIMELINE
DEPENDENCIES
MAVEN
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
Run Testcases
Python 3.9.7 64-bit ('.venv': venv)
tabnine
Quokka
Spaces: 2
UTF-8
CRLF
Html
Go Live
Prettier+
Prettier
ENG
04:34 PM
30-11-2021

Screenshots of website






Credit Card Fraud Detection Wel
127.0.0.1:5000
Credit Card Fraud Detection Rest API
Enter the 30 feature values in the below cell(in order):
-15.2713618637585 8.32658106147414 -22.3385905133352
11.8853128922128 -8.72133435613386 -2.32430694897567
-16.1964186599881 0.5128818204244849 -6.3336851559474505
-13.175198078736 9.939819741725689 -14.2254557039818
0.7160336651928729 -14.0047761755606 -0.329447848153274
-8.0594194070953 -12.122008815823698 -4.7997101325496105
0.6862271635974581 0.993121017282391 -2.3568962794623
1.06801917704623 1.0856171672433799 -1.03979707801119
Predict


127.0.0.1:5000/predict
127.0.0.1:5000/predict
Credit Card Fraud Detection Results
Validation Completed.
According to our model, this transaction is a Fraud transaction.
Retest

Comparative Study with existing system

We have surveyed a total of 16 papers and what we can majorly infer from them is that in one of the papers it faced challenges of learning from class-imbalanced data. The data set was highly imbalanced. The total number of fraud transactions was much less than the number of legitimate transactions, the data distribution was unbalanced and it is already shown that the performance of various machine learning algorithms decreases when the data set is unbalanced. This is something which we are taking care of in our project.
Precision scores are found a little low in some papers. The results of our work show higher precision scores to improve upon this drawback found in one of the papers.
Size of the data was a limitation in one research as they could not conduct our study on large amounts of data. Some more algorithms should have been taken for the survey before implementing the best ones. Other models of machine learning could also be considered and a tool can be made.
In our project we have taken five algorithms to survey the performance on our dataset and found the best performing algorithm to deploy a web app to detect credit card fraud for user entered values.

Conclusion

Credit Cards are a great tool to pay money easily, but as with all the other monetary payment tools, reliability is an issue here too as it is subject to breach and other frauds. To encounter this problem, a solution is needed to identify the patterns in the transactions and identify the ones which are fraud, so that finding such transactions beforehand in future will be very easy. Machine Learning is a great tool to do this work since Machine Learning helps us in finding patterns in the data. Machine Learning can help produce great results if provided enough data. Also, with further advances in the technology, Machine Learning too will advance with time, it will be easy for a person to predict if a transaction is fraud or not much more accurately with the advances.

Future Work

With an increasing number of bank fraudulency and cyber-crime cases, the need for a secure testing system is on rise. And this is a direct solution to this problem. It can be extended to a duplex verification of not only a customer(debit-ant) but also of the seller(credit-ant). It can be taken and used on a regular basis just like OTP. It can be used to even assess past transactions in a database to find whether certain transactions were fraudulent or not and also would be able to produce evidence in such cases. Also. optimization techniques on the proposed models and testing on new models can be done.

References

1. Randhawa, K., Loo, C. K., Seera, M., Lim, C. P., & Nandi, A. K. (2018). Credit Card Fraud Detection Using AdaBoost and Majority Voting. IEEE Access, 6, 14277–14284. https://doi.org/10.1109/access.2018.2806420

2. John, H., & Naaz, S. (2019). Credit Card Fraud Detection using Local Outlier Factor and Isolation Forest. International Journal of Computer Sciences and Engineering, 7(4), 1060–1064. https://doi.org/10.26438/ijcse/v7i4.10601064

3. Lakshmi S V S S1, Selvani Deepthi Kavila (2018). Machine Learning for Credit Card Fraud Detection System. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 24 (2018) pp. 16819-16824 DOI:10.37622/000000 http://www.ripublication.com/ijaer18/ijaerv13n24_18.pdf

4. de Sá, A. G., Pereira, A. C., & Pappa, G. L. (2018). A customized classification algorithm for credit card fraud detection. Engineering Applications of Artificial Intelligence, 72, 21–29. https://doi.org/10.1016/j.engappai.2018.03.011

5. Robinson, W. N., & Aria, A. (2018). Sequential fraud detection for prepaid cards using hidden Markov model divergence. Expert Systems with Applications, 91, 235–251. https://doi.org/10.1016/j.eswa.2017.08.043

6. S P Maniraj, Aditya Saini, Shadab Ahmed, & Swarna Deep Sarkar. (2019). Credit Card Fraud Detection using Machine Learning and Data Science. International Journal of Engineering Research And, 08(09). https://doi.org/10.17577/ijertv8is090031

7. Naresh Kumar Trivedi, Sarita Simaiya, Umesh Kumar Lilhore, Sanjeev Kumar Sharma. (2020). An Efficient Credit Card Fraud Detection Model Based on Machine Learning Methods. International Journal of Advanced Science and Technology, 29(05), 3414 - 3424.

8. Chen, J. I. Z., & Lai, K. L. (2021). Deep Convolution Neural Network Model for Credit-Card Fraud Detection and Alert. June 2021, 3(2), 101–112. https://doi.org/10.36548/jaicn.2021.2.003

9. S. Makki, Z. Assaghir, Y. Taher, R. Haque, M. Hacid and H. Zeineddine, "An Experimental Study with Imbalanced Classification Approaches for Credit Card Fraud Detection," in IEEE Access, vol. 7, pp. 93010-93022, 2019, doi: 10.1109/ACCESS.2019.2927266.

10. S. Abinayaa, H. Sangeetha, R. A. Karthikeyan, K. Saran Sriram, D. Piyush, Credit Card Fraud Detection and Prevention using Machine Learning, International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-4, April, 2020

11. SADGALI, Imane; SAEL, Naoual; BENABBOU, Faouzia. Adaptive Model for Credit Card Fraud Detection. **International Journal of Interactive Mobile Technologies (iJIM)**, [S.l.], v. 14, n. 03, p. pp. 54-65, feb. 2020

12. Zhang, D., Bhandari, B., & Black, D. (2020). Credit Card Fraud Detection Using Weighted Support Vector Machine. *Applied Mathematics-a Journal of Chinese Universities Series B, 11*, 1275-1291.

13. Devi, V., & Ravi, D.G. (2020). Real-Time Deep Learning Based Credit Card Fraud Detection. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 9, ISSUE 03, MARCH 2020

14. Ramya, M., S. Kumar and K. Raja. “Improved Credit Card Fraud Detection using Machine Learning.” *International journal of engineering research and technology* 8 (2020)

15. Taha and S. J. Malebary, "An Intelligent Approach to Credit Card Fraud Detection Using an Optimized Light Gradient Boosting Machine," in IEEE Access, vol. 8, pp. 25579-25587, 2020, doi: 10.1109/ACCESS.2020.2971354.

16. Naik, Het and Prashasti Kanikar. “Credit card Fraud Detection based on Machine Learning Algorithms.” *International Journal of Computer Applications* 182 (2019): 8-12.