Detection of fraudulent credit card activities using Machine learning and Deep learning algorithms

Report submitted to the SASTRA Deemed to be University as
the requirement for the course

CSE300 - MINI PROJECT

Submitted by

Kailash.S
(Reg.no : 124156015, B.Tech Computer Science and engineering - Artificial intelligence and data science)

Shrihari.S
(Reg.no : 124156047, B.Tech Computer Science and engineering - Artificial intelligence and data science)

Tarani Sre.S.G
(Reg.no : 124003337, B.Tech Computer Science and engineering)

May 2023

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act, 1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

<u>Bonafide Certificate</u>

This is to certify that the report titled “ Detection of fraudulent credit card activities using Machine learning and Deep learning algorithms” submitted as a requirement for the course, CSE300 / INT300 / ICT300: MINI PROJECT for B.Tech. is a bonafide record of the work done by Shri.Kailash S (Reg.no : 124156015, B.Tech Computer Science and engineering - Artificial intelligence and data science), Shri. Shrihari.S (Reg.no : 124156047, B.Tech Computer Science and engineering - Artificial intelligence and data science) and Sow.Tarani Sre.S.G (Reg.no : 124003337, B.Tech Computer Science and engineering)during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation :

Date :

Mini Project Viva voce held on ________________

Examiner 1 Examiner 2

Acknowledgements

We would like to thank our Honorable Chancellor Prof. R. Sethuraman for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor Dr. S. Vaidhyasubramaniam and Dr. S. Swaminathan, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to Dr. R. Chandramouli, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to Dr. A. Umamakeswari, Dean, School of Computing,

Dr. S. Gopalakrishnan, Associate Dean, Department of Computer Application, Dr. B.Santhi, Associate Dean, Research, Dr. V. S. Shankar Sriram, Associate Dean, Department of Computer Science and Engineering, Dr. R. Muthaiah, Associate Dean, Department of Information Technology and Information & Communication Technology

Our guide Dr. M.Raja,Senior Assistant professor, School of Computing, was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through project.

Abstract

One of the simple and quick tools for money transactions is the credit card. More people started utilizing it because of how portable and handy it is. The amount of unauthorized misuse has increased along with use. The major goal of this mini project is to identify such frauds. Many machine learning-based algorithms for credit card identification are presented in the pertinent literature, including the Decision Tree, Random Forest, Support Vector Machine, Logistic Regression, and XG Boost. Modern deep learning algorithms must still be used, to cut down on fraud losses. To determine the effective solution, both machine learning and deep learning are compared. European Card Benchmark for Fraud Detection is the name of the dataset being utilised.The dataset was initially subjected to a machine learning technique, which somewhat increased the accuracy of fraud detection. Later, three convolutional neural network-based designs are used to boost the effectiveness of fraud detection. The precision of detection was further improved by adding more layers. By varying the amount of hidden layers, epochs, a thorough empirical investigation has been conducted. In order to reduce the false negative rate, we have also run trials that balanced the data and used deep learning methods. The suggested methods can be successfully used to identify credit card fraud in the real world.

Keywords:

Logistic Regression, Support vector machine, XGBoost, Decision tree, Random forest, Neural networks, Fraud detection,Transactions

Table of Contents

CHAPTER 137
SUMMARY OF BASE PAPER

TITLE : Detection of fraudulent credit card activities using Machine learning and Deep learning algorithms

YEAR OF PUBLICATION : 2022

CONTENT :

Both the owners of credit cards and financial institutions suffer large financial losses as a result of credit card theft.The likelihood of fraudulent transactions can be predicted using machine learning.In the banking sector, machine learning algorithms are useful for making precise predictions.However, because of their poor accuracy, modern deep learning algorithms must still be used to cut down on fraud losses. The article uses Deep Learning and five machine learning classification approaches.A dataset from Kaggle including credit card transactions performed in September 2013 by European cardholders is selected in order to move further with this job.

SYSTEM METHODOLOGY :

Dataset Description :

The dataset contains a cardholder's transactions over a two-day period in September 2018. In total, there were 284,807 transactions, and 492, or 0.172 percent, of those were fraudulent. Principal component analysis (PCA) is used to apply the main component analysis to the bulk of the dataset's features because exposing a consumer's transactional information is regarded as a concern of confidentiality.

Data Pre Processing

Before building a model, data preprocessing is necessary to remove undesirable noise and outliers from the dataset that could lead to inaccurate model training. The process

of cleaning the data and making sure it is prepared for model building starts after gathering the necessary dataset. The target column is one of 31 columns in the dataset that was used. Regarding Time, we don't notice any distinct pattern between fraudulent and non-fraudulent transactions. So, we can eliminate the Time column.
As we can see, the non-fraudulent transactions are dispersed throughout the low to high range of amount, however the fraudulent transactions are primarily concentrated in the lower range of amount.We need to scale only the `Amount` column as all other columns are already scaled by the PCA transformation.

Splitting Dataset into train and test data:

After completing data pre-processing and addressing the imbalanced dataset, the model development stage follows next. The data is split into training and testing data, with the ratio set at 80% training data and 20% testing data, in order to increase accuracy and efficiency for this activity. After splitting, the model is trained using a variety of classification algorithms.Some of the classification methods used for this include Logistic Regression, Decision Tree Classification, Random Forest Classification, K-Nearest Neighbour Classification, and Support Vector Machine.

CHAPTER 2
MERITS AND DEMERITS OF THE BASE PAPER

MERITS :

There are various advantages to the credit card fraud detection dataset, including:

1. **Relevance to the real world:** Because the dataset includes actual credit card transactions, it is pertinent to the real-world issue of identifying credit card fraud. Models developed using this dataset are therefore more likely to succeed when applied to real-world data.

2. **Large sample size:** The dataset has a lot of transactions, which makes it possible to build reliable and precise machine learning models.

3. **Balanced distribution:** The dataset's distribution of fraudulent and non-fraudulent transactions is often balanced, preventing any bias in the machine learning models for either class.

4. **Anonymous Features**: The dataset has features that have been anonymized to safeguard credit card customers' privacy while still giving usable data for detecting fraud.

5. **Accessibility:** The dataset is openly accessible, allowing academics and developers to utilize it to test and enhance fraud detection algorithms.

6. **Standardized Evaluation :** The dataset includes a standardized assessment metric that enables a fair and consistent comparison of various fraud detection techniques.

DEMERITS:

The credit card fraud detection dataset offers numerous advantages, but it also has several drawbacks and shortcomings, such as:

1. **Limited Scope :** Only credit card transactions are included in the dataset, which restricts its usefulness to other forms of financial crime like bank transfers or identity theft.

2. **Classes that are unbalanced:** The dataset is somewhat unbalanced, however it is still unbalanced when compared to actual credit card transactions. Model bias can result from the fact that there are normally many fewer fraudulent transactions than legitimate ones.

3. **Limited data:** The dataset only has a small number of characteristics, which may not be able to fully capture all the necessary data for fraud detection. To increase the accuracy of fraud detection, more data could be required, such as user behavior or contextual information.

4. **Privacy concern :** Sensitive data on credit card customers is included in the dataset, which presents privacy issues. Despite the anonymization of the dataset, there is still a chance of re-identification or data breaches.

5. **Limited diversity:** Because only particular locations, sectors, or client segments may be represented in the dataset, it may not be representative of all credit card transactions.

6. **Limited updates:** As fraudsters develop new strategies and tactics, the dataset may become old. The dataset must be continually updated and enlarged to stay up with new fraud patterns.

CHAPTER 3
SOURCE CODE

Importing and Exploratory Data Analysis

In [1]:
```
# Importing the libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

# Reading the dataset
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[1]:

	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

In [2]:
```
df.shape
```

Out[2]:
```
(284807, 31)
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
```

In [3]:
```
df.describe()
```

Out[3]:

	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	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-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

In [4]:
```
# Cheking percent of missing values in columns
df_missing_columns = (round(((df.isnull().sum()/len(df.index))*100),2).to_frame('null')).sort_values('null', ascending=False)
df_missing_columns
```

Out[4]:

	null
Time	0.0
V16	0.0
Amount	0.0
V28	0.0
V27	0.0
V26	0.0
V25	0.0
V24	0.0
V23	0.0
V22	0.0
V21	0.0
V20	0.0
V19	0.0
V18	0.0
V17	0.0
V15	0.0
V1	0.0
V14	0.0

Checking the distribution of the classes

In [5]:
```
classes = df['Class'].value_counts()
classes
```

Out[5]:
```
0    284315
1       492
Name: Class, dtype: int64
```

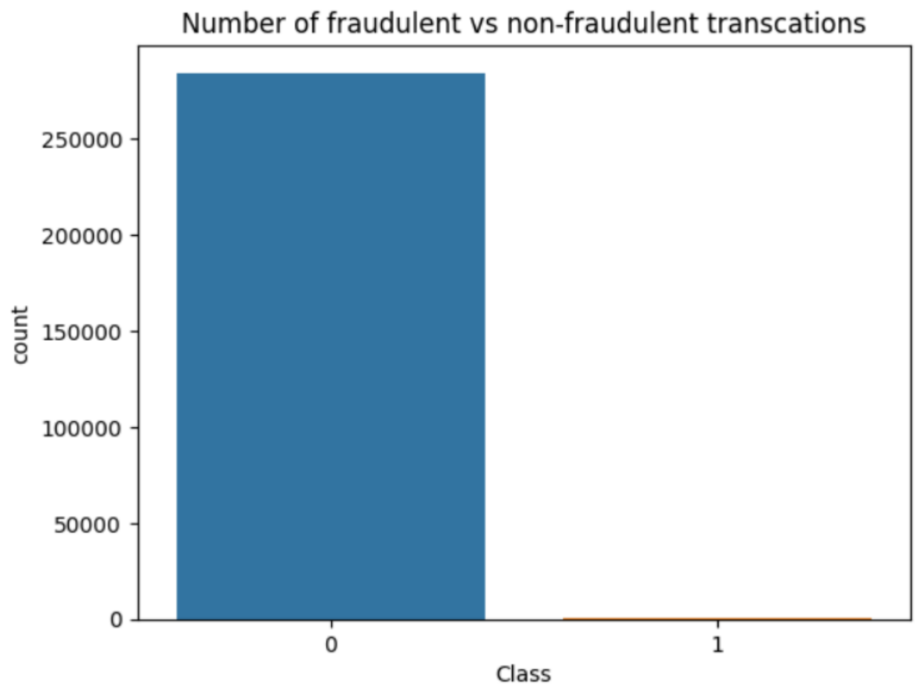
In [6]:
```
valid_transc_percent = round((classes[0]/df['Class'].count()*100),2)
valid_transc_percent
```

Out[6]: 99.83

In [7]:
```
fraud_transc_percent = round((classes[1]/df['Class'].count()*100),2)
fraud_transc_percent
```

Out[7]: 0.17

In [8]:
```
# Bar plot for the number of fraudulent vs non-fraudulent transcations
sns.countplot(x='Class', data=df)
plt.title('Number of fraudulent vs non-fraudulent transcations')
plt.show()
```



Train-Test Split

In [12]:
```
# Import library
from sklearn.model_selection import train_test_split
# Putting feature variables into X
X = df.drop(['Class'], axis=1)
# Putting target variable to y
y = df['Class']
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)
```

Feature Scaling

In [13]:
```
# Standardization method
from sklearn.preprocessing import StandardScaler
# Instantiate the Scaler
scaler = StandardScaler()
# Fit the data into scaler and transform
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
X_train.head()
```

Out[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23
201788	134039.0	2.023734	-0.429219	-0.691061	-0.201461	-0.162486	0.283718	-0.674694	0.192230	1.124319	...	-0.171390	-0.195207	-0.477813	0.340513
179369	124044.0	-0.145286	0.736735	0.543226	0.892662	0.350846	0.089253	0.626708	-0.049137	-0.732566	...	0.206709	-0.124288	-0.263560	-0.110568
73138	54997.0	-3.015846	-1.920606	1.229574	0.721577	1.089918	-0.195727	-0.462586	0.919341	-0.612193	...	0.842838	0.274911	-0.319550	0.212891
208679	137226.0	1.851980	-1.007445	-1.499762	-0.220770	-0.568376	-1.232633	0.248573	-0.539483	-0.813368	...	-0.196551	-0.406722	-0.899081	0.137370
206534	136246.0	2.237844	-0.551513	-1.426515	-0.924369	-0.401734	-1.438232	-0.119942	-0.449263	-0.717258	...	-0.045417	0.050447	0.125601	0.215531

5 rows × 30 columns

In [14]:
```
# Transform the test set
X_test['Amount'] = scaler.transform(X_test[['Amount']])
X_test.head()
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23
49089	43906.0	1.229452	-0.235478	-0.627166	0.419877	1.797014	4.069574	-0.896223	1.036103	0.745991	...	-0.057922	-0.170060	-0.288750	-0.130270
154704	102638.0	2.016893	-0.088751	-2.989257	-0.142575	2.675427	3.332289	-0.652336	0.752811	1.962566	...	-0.147619	-0.184153	-0.089661	0.087188
67247	52429.0	0.535093	-1.469185	0.868279	0.385462	-1.439135	0.368118	-0.499370	0.303698	1.042073	...	0.437685	0.028010	-0.384708	-0.128376
251657	155444.0	2.128486	-0.117215	-1.513910	0.166456	0.359070	-0.540072	0.116023	-0.216140	0.680314	...	-0.227278	-0.357993	-0.905085	0.223474
201903	134084.0	0.558593	1.587908	-2.368767	5.124413	2.171788	-0.500419	1.059829	-0.254233	-1.959060	...	0.249457	-0.035049	0.271455	0.381606

5 rows × 30 columns

Logistic Regression

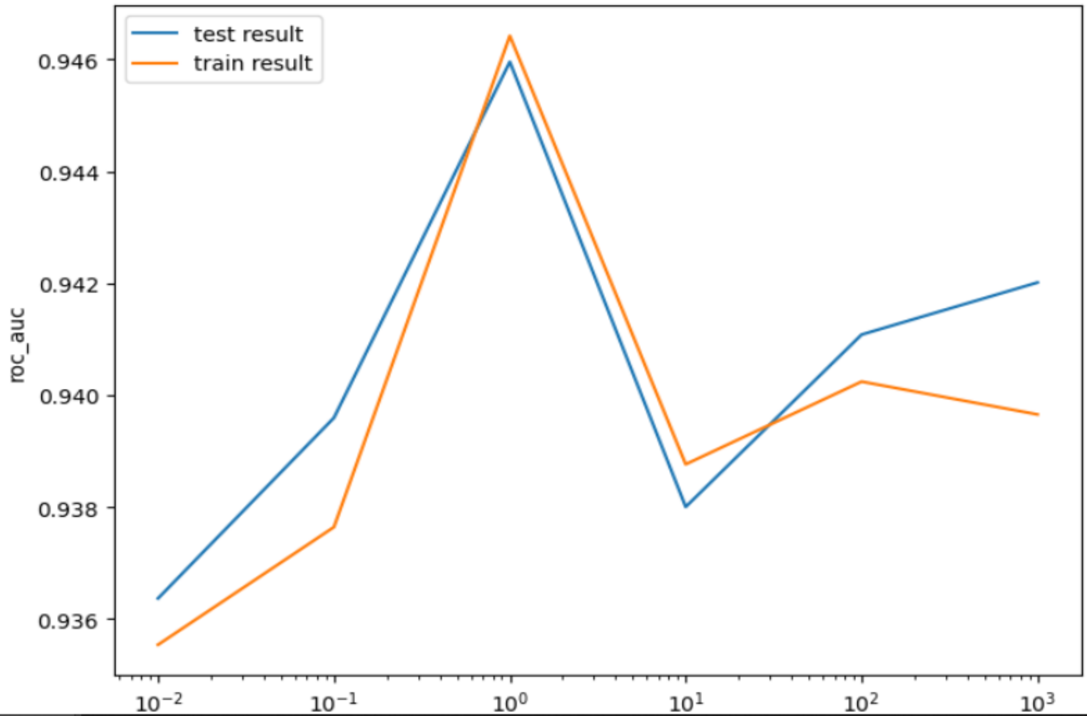
In [15]:
```
# Importing scikit logistic regression module
from sklearn.linear_model import LogisticRegression
# Impoting metrics
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report

# Importing libraries for cross validation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
# Creating KFold object with 5 splits
folds = KFold(n_splits=5, shuffle=True, random_state=4)

# Specify params
params = {"C": [0.01, 0.1, 1, 10, 100, 1000]}

# Specifing score as recall as we are more focused on acheiving the higher sensitivity than the accuracy
model_cv = GridSearchCV(estimator = LogisticRegression(),
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# Fit the model
model_cv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

Out[15]:

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [17]:
```
# plot of C versus train and validation scores

plt.figure(figsize=(8, 6))
plt.plot(cv_results['param_C'], cv_results['mean_test_score'])
plt.plot(cv_results['param_C'], cv_results['mean_train_score'])
plt.xlabel('C')
plt.ylabel('roc_auc')
plt.legend(['test result', 'train result'], loc='upper left')
plt.xscale('log')
```



In [20]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))
# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9991529329149202
F1-Score:- 0.7359781121751026
```

In [21]:
```
y_test_pred = logistic_imb_model.predict(X_test)
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56840    26]
 [   34    62]]
```

In [22]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# F1 score
print("F1-Score:-", f1_score(y_test, y_test_pred))
```

```
Accuracy:- 0.9989466661985184
F1-Score:- 0.6739130434782609
```

Decision Tree

In [23]:
```
# Importing decision tree classifier
from sklearn.tree import DecisionTreeClassifier
# Create the parameter grid
param_grid = {
    'max_depth': range(5, 15, 5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
}


# Instantiate the grid search model
dtree = DecisionTreeClassifier()

grid_search = GridSearchCV(estimator = dtree,
                           param_grid = param_grid,
                           scoring= 'roc_auc',
                           cv = 3,
                           verbose = 1)

# Fit the grid search to the data
grid_search.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

Out[23]:
```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [25]:
```
# Model with optimal hyperparameters
dt_imb_model = DecisionTreeClassifier(criterion = "gini",
                                      random_state = 100,
                                      max_depth=5,
                                      min_samples_leaf=100,
                                      min_samples_split=100)

dt_imb_model.fit(X_train, y_train)
```

Out[25]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_leaf=100, min_samples_split=100,
                       random_state=100)
```

In [26]:
```
y_train_pred = dt_imb_model.predict(X_train)
confusion = metrics.confusion_matrix(y_train, y_train_pred)
print(confusion)
```

```
[[227374     75]
 [   114    282]]
```

In [27]:
```
TP = confusion[1,1]
TN = confusion[0,0]
FP = confusion[0,1]
FN = confusion[1,0]
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9991704887094297
F1-Score:- 0.7490039840637449
```

In [28]:
```
y_test_pred = dt_imb_model.predict(X_test)
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

```
[[56836    30]
 [   40    56]]
```

In [29]:
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# F1 score
print("F1-Score:-", f1_score(y_test, y_test_pred))
```

```
Accuracy:- 0.9987711105649381
F1-Score:- 0.6153846153846155
```

Random Forest

```
# Importing random forest classifier
from sklearn.ensemble import RandomForestClassifier
```

```
param_grid = {
    'max_depth': range(5,10,5),
    'min_samples_leaf': range(50, 150, 50),
    'min_samples_split': range(50, 150, 50),
    'n_estimators': [100,200,300],
    'max_features': [10, 20]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf,
                           param_grid = param_grid,
                           cv = 2,
                           n_jobs = -1,
                           verbose = 1,
                           return_train_score=True)

# Fit the model
grid_search.fit(X_train, y_train)
```

```
Fitting 2 folds for each of 24 candidates, totalling 48 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed: 101.0min finished
```

```
# Predictions on the train set
y_train_pred = rfc_imb_model.predict(X_train)
```

```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train, y_train)
print(confusion)
```

```
[[227449      0]
 [     0    396]]
```

```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_train, y_train_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9993460466545239
Sensitivity:- 1.0
Specificity:- 1.0
F1-Score:- 0.7983761840324763
```

Prediction on the test set

```
# Predictions on the test set
y_test_pred = rfc_imb_model.predict(X_test)
```

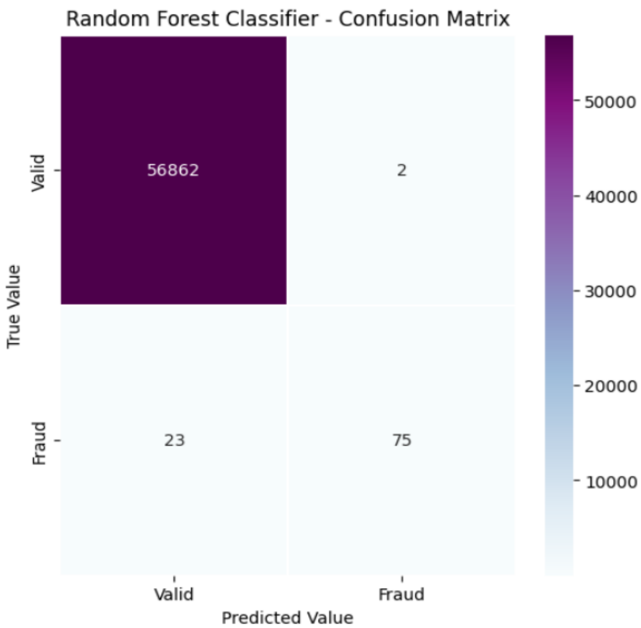
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_test, y_test_pred)
print(confusion)
```

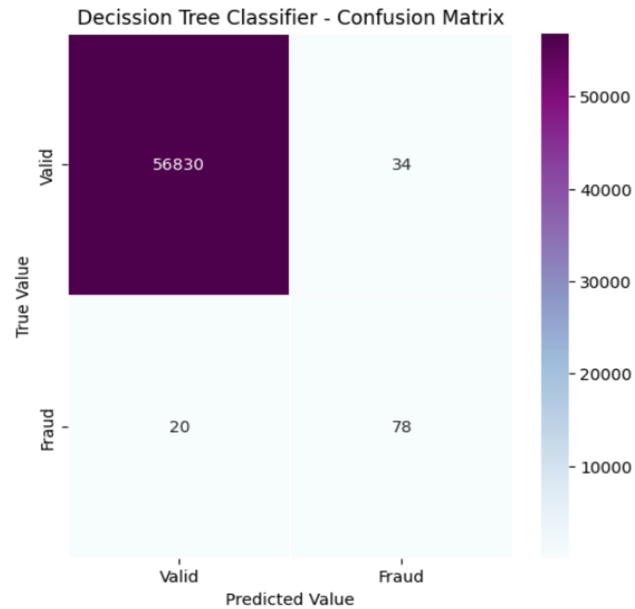
```
[[56841    25]
 [   36    60]]
```

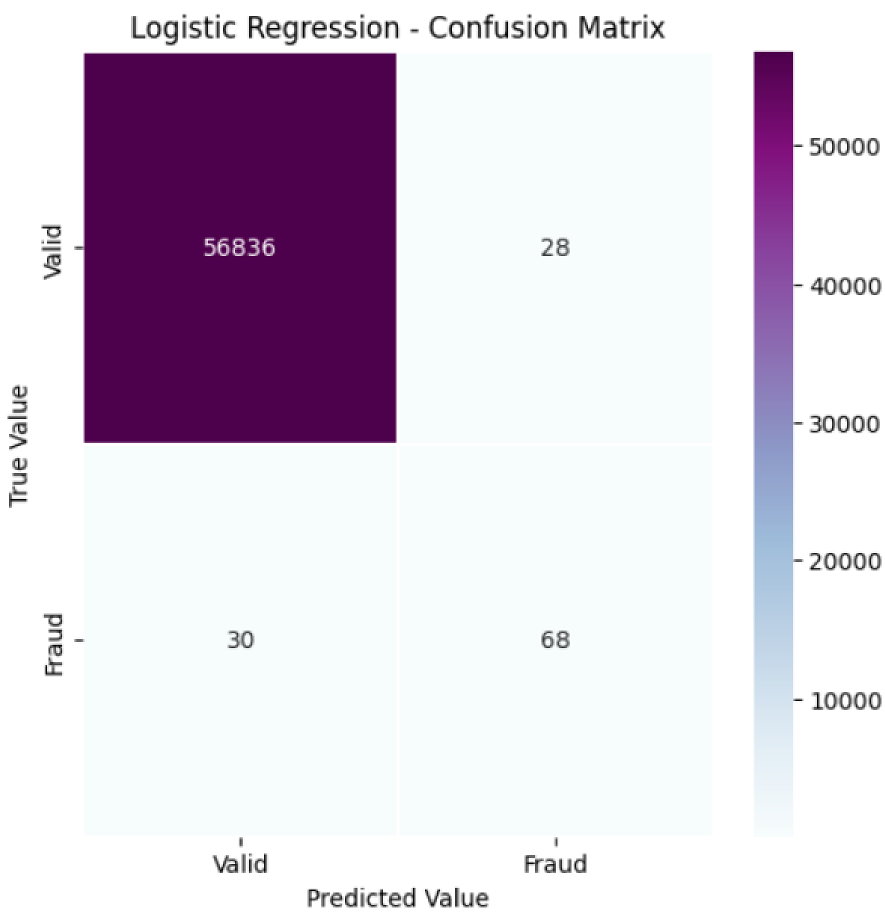
```
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

```
# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_test_pred))

# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))

# Specificity
print("Specificity:-", TN / float(TN+FP))

# F1 score
print("F1-Score:-", f1_score(y_train, y_train_pred))
```

```
Accuracy:- 0.9989291106351603
Sensitivity:- 0.625
Specificity:- 0.9995603699926142
F1-Score:- 0.7983761840324763
```


Random Forest Classifier - Confusion Matrix
56862
2
23
75
Valid
Fraud
True Value
Predicted Value
50000
40000
30000
20000
10000


Decission Tree Classifier - Confusion Matrix
56830
34
20
78
Valid
Fraud
True Value
Predicted Value
50000
40000
30000
20000
10000


Logistic Regression - Confusion Matrix
56836
28
30
68
Valid
Fraud
True Value
Predicted Value
50000
40000
30000
20000
10000

K - Nearest Neighbour

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='ball_tree',n_neighbors = 5,metric='euclidean')
knn.fit(X_train, y_train)
```

Out[8]:
```
KNeighborsClassifier(algorithm='ball_tree', metric='euclidean')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [9]:
```
y_pred = knn.predict(X_test)
y_pred[5000:6000]
```

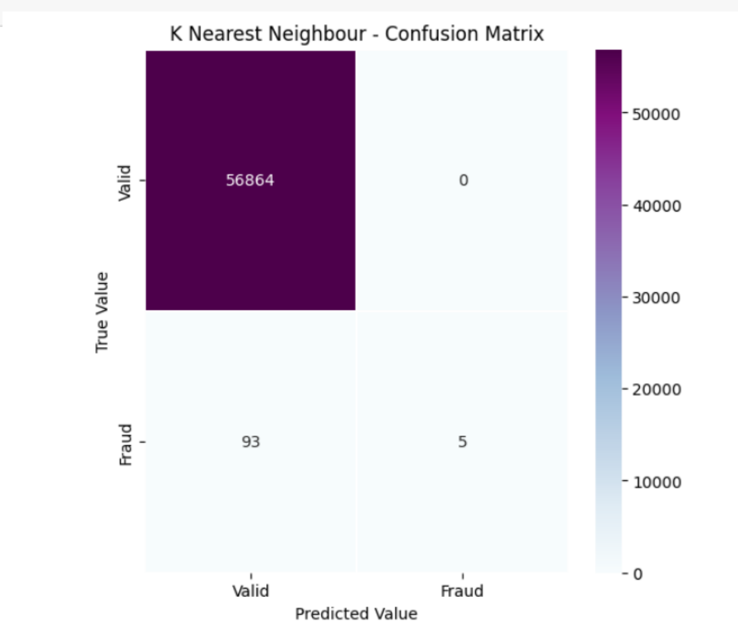
In [10]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[56864     0]
 [   93     5]]
```

In [11]:
```
acc_knn =accuracy_score(y_test, y_pred)*100
print("The accuracy is",acc_knn,"%")
```

```
The accuracy is 99.83673326077034 %
```

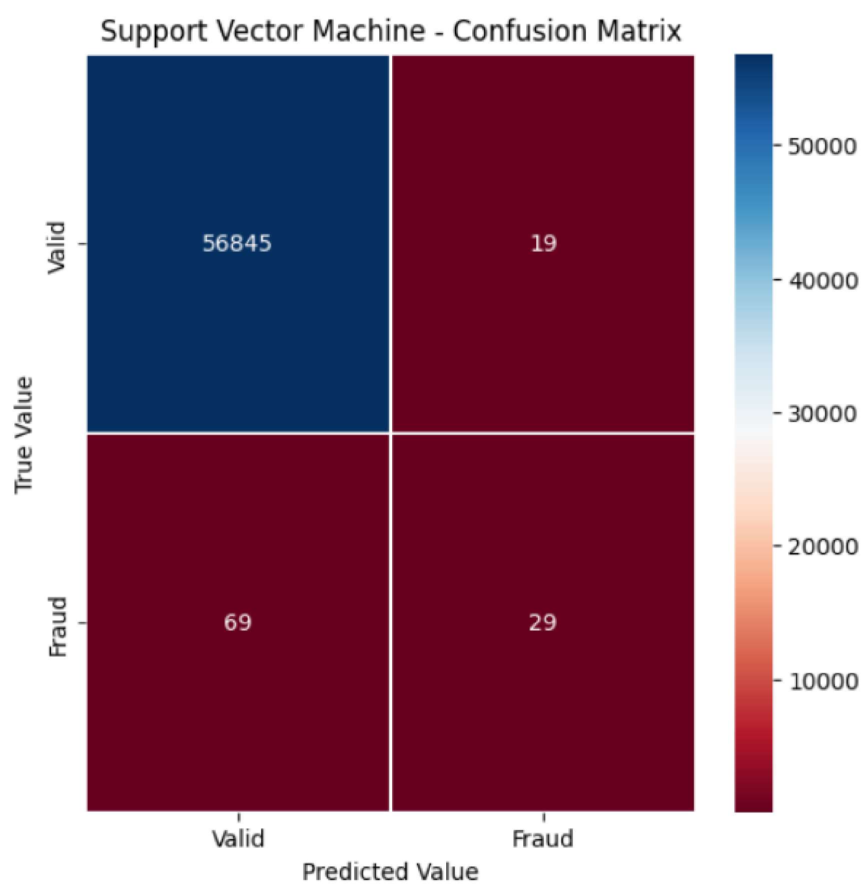
In [12]:
```
Y_pred1 = knn.predict(X_test)
conf_matrix1 = confusion_matrix(y_test, Y_pred1)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix1),annot=True, fmt='d',
            linewidths= 0.05 ,cmap='BuPu',xticklabels= labels, yticklabels= labels)

plt.title('K Nearest Neighbour - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```



Support Vector Machine

In [ ]:
```
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Create an SVM model
model = svm.SVC(kernel='linear', C=1.0, random_state=42)
# Train the SVM model
model.fit(X_train, y_train)
# Evaluate the SVM model on the testing set
y_pred = model.predict(X_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
print(accuracy_score(y_test,y_pred))
```

In [17]:
```
y_pred = model.predict(X_test)
conf_matrix1 = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 6))
labels= ['Valid', 'Fraud']

sns.heatmap(pd.DataFrame(conf_matrix1),annot=True, fmt='d',
            linewidths= 0.05 ,cmap='RdBu',xticklabels= labels, yticklabels= labels)

plt.title('Support Vector Machine - Confusion Matrix')
plt.ylabel('True Value')
plt.xlabel('Predicted Value')
plt.show()
```



Fraud Detection using CNN

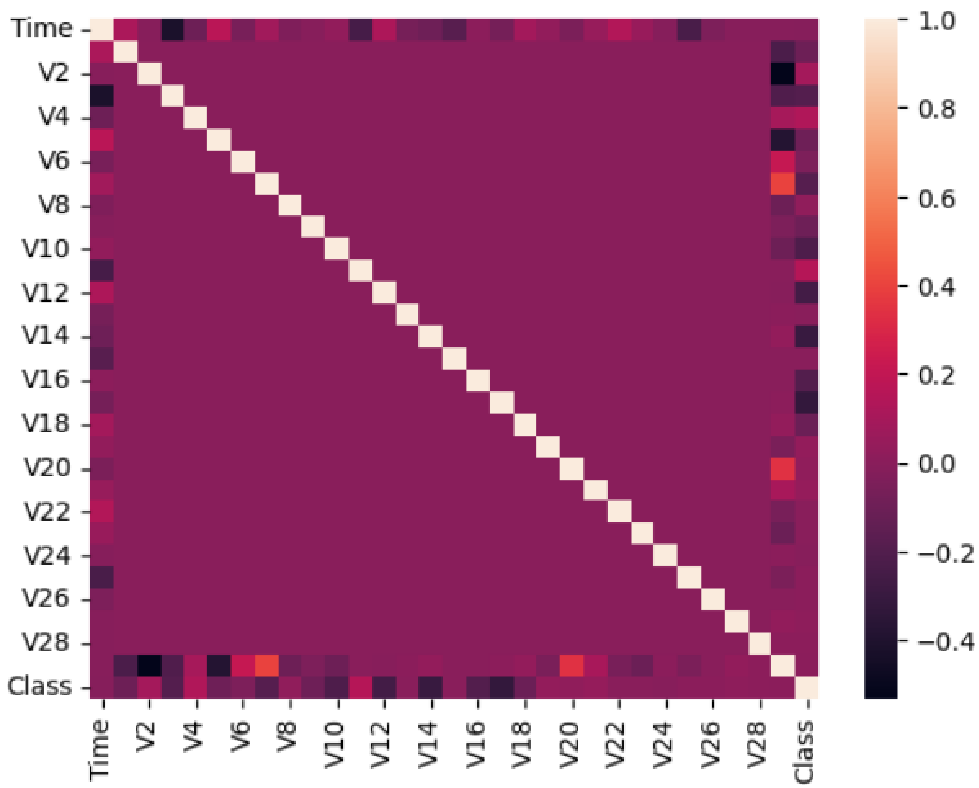
In [2]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout, BatchNormalization, Embedding
from tensorflow.keras.layers import Conv1D, MaxPooling1D
from keras.layers import SpatialDropout1D
from tensorflow.keras.optimizers import Adam
print(tf.__version__)
```

```
2.11.0
```

In [3]:
```
import pandas as pd
import numpy as np
import  seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Correlation Heatmap

In [7]:
```
dataplot=sns.heatmap(data.corr())
plt.show()
```



Balancing the dataset using Downsampling (making number of frauds and non frauds equal)

In [8]:
```
data['Class'].value_counts()
```
Out[8]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [9]:
```
non_fraud = data[data['Class']==0]
fraud = data[data['Class']==1]
```

In [10]:
```
non_fraud.shape, fraud.shape
```
Out[10]:
```
((284315, 31), (492, 31))
```

In [11]:
```
non_fraud = non_fraud.sample(fraud.shape[0])
non_fraud.shape
```
Out[11]:
```
(492, 31)
```

In [12]:
```
data = fraud.append(non_fraud, ignore_index=True)
data
```

In [13]:
```
data['Class'].value_counts()
```
Out[13]:
```
1    492
0    492
Name: Class, dtype: int64
```

In [14]:
```
X = data.drop('Class', axis = 1)
y = data['Class']
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0, stratify = y)
```

In [17]:
```
X_train.shape, X_test.shape
```
Out[17]:
```
((787, 30), (197, 30))
```

In [18]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [20]:
```
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

In [19]:
```
X_train.shape
```
Out[19]:
```
(787, 30)
```

In [21]:
```
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [22]:
```
X_train.shape, X_test.shape
```
Out[22]:
```
((787, 30, 1), (197, 30, 1))
```

Building the CNN model

In [23]:
```
epochs = 20
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d (Conv1D)             (None, 29, 32)            96        
                                                                 
 batch_normalization (BatchN  (None, 29, 32)           128       
 ormalization)                                                   
                                                                 
 dropout (Dropout)           (None, 29, 32)            0         
                                                                 
 conv1d_1 (Conv1D)           (None, 28, 64)            4160      
                                                                 
 batch_normalization_1 (Batc  (None, 28, 64)           256       
 hNormalization)                                                 
                                                                 
 dropout_1 (Dropout)         (None, 28, 64)            0         
                                                                 
 flatten (Flatten)           (None, 1792)              0         
                                                                 
 dense (Dense)               (None, 64)                114752    
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 1)                 65        
                                                                 
=================================================================
Total params: 119,457
Trainable params: 119,265
Non-trainable params: 192
_________________________________________________________________
```

Compiling the model

In [24]:
```
model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
```
```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,
cy.Adam.
```

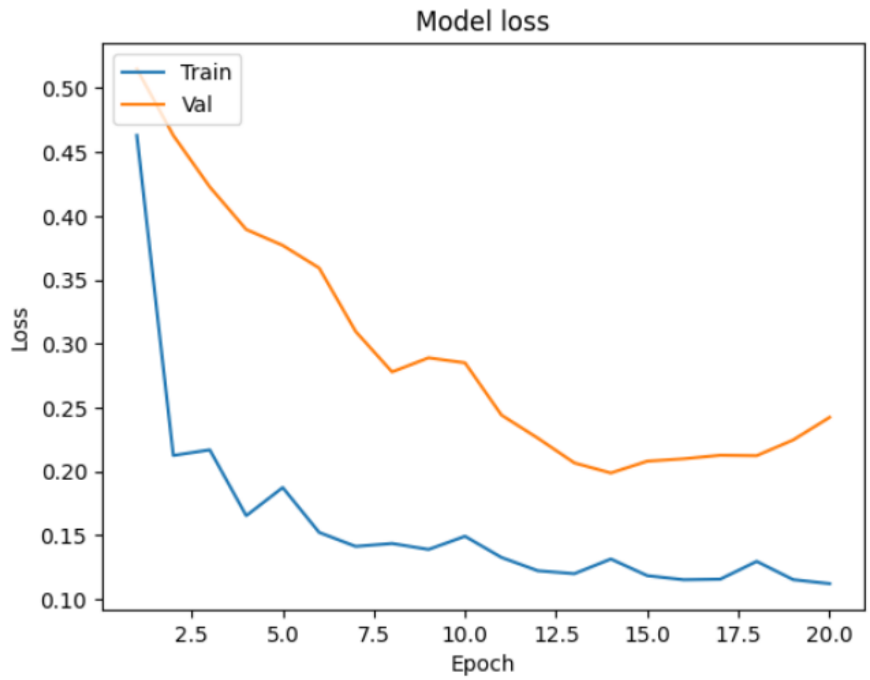
In [25]:
```
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/20
25/25 [==============================] - 7s 87ms/step - loss: 0.4630 - accuracy: 0.8564 - val_loss: 0.5150 - val_accuracy: 0.91
37
Epoch 2/20
25/25 [==============================] - 1s 22ms/step - loss: 0.2125 - accuracy: 0.9377 - val_loss: 0.4630 - val_accuracy: 0.92
39
Epoch 3/20
25/25 [==============================] - 0s 19ms/step - loss: 0.2169 - accuracy: 0.9390 - val_loss: 0.4227 - val_accuracy: 0.91
88
Epoch 4/20
25/25 [==============================] - 0s 16ms/step - loss: 0.1654 - accuracy: 0.9365 - val_loss: 0.3894 - val_accuracy: 0.92
39
Epoch 5/20
25/25 [==============================] - 1s 21ms/step - loss: 0.1874 - accuracy: 0.9352 - val_loss: 0.3769 - val_accuracy: 0.92
89
Epoch 6/20
25/25 [==============================] - 1s 21ms/step - loss: 0.1523 - accuracy: 0.9466 - val_loss: 0.3592 - val_accuracy: 0.93
91
Epoch 7/20
25/25 [==============================] - 0s 14ms/step - loss: 0.1414 - accuracy: 0.9517 - val_loss: 0.3096 - val_accuracy: 0.92
89
Epoch 8/20
25/25 [==============================] - 0s 19ms/step - loss: 0.1436 - accuracy: 0.9504 - val_loss: 0.2779 - val_accuracy: 0.93
40
Epoch 9/20
25/25 [==============================] - 1s 23ms/step - loss: 0.1390 - accuracy: 0.9568 - val_loss: 0.2889 - val_accuracy: 0.91
88
Epoch 10/20
25/25 [==============================] - 1s 21ms/step - loss: 0.1493 - accuracy: 0.9428 - val_loss: 0.2850 - val_accuracy: 0.91
37
Epoch 11/20
25/25 [==============================] - 1s 22ms/step - loss: 0.1327 - accuracy: 0.9492 - val_loss: 0.2441 - val_accuracy: 0.92
39
Epoch 12/20
25/25 [==============================] - 1s 28ms/step - loss: 0.1223 - accuracy: 0.9644 - val_loss: 0.2260 - val_accuracy: 0.92
89
Epoch 13/20
25/25 [==============================] - 0s 17ms/step - loss: 0.1200 - accuracy: 0.9530 - val_loss: 0.2067 - val_accuracy: 0.91
37
```

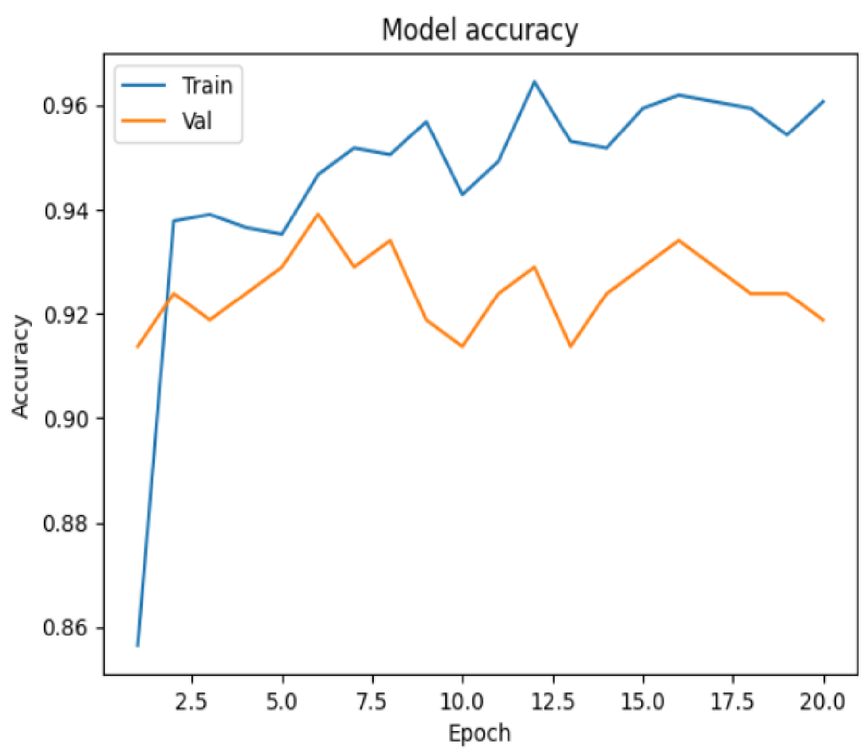
Plotting Accuracy Loss graph

In [26]:
```
def plot_learningCurve(history, epoch):
  # Plot training & validation accuracy values
  epoch_range = range(1, epoch+1)
  plt.plot(epoch_range, history.history['accuracy'])
  plt.plot(epoch_range, history.history['val_accuracy'])
  plt.title('Model accuracy')
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()

  # Plot training & validation loss values
  plt.plot(epoch_range, history.history['loss'])
  plt.plot(epoch_range, history.history['val_loss'])
  plt.title('Model loss')
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['Train', 'Val'], loc='upper left')
  plt.show()
```

In [27]:
```
plot_learningCurve(history, epochs)
```



Model loss
Train
Val
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Loss
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch



Model accuracy
Train
Val
0.96
0.94
0.92
0.90
0.88
0.86
Accuracy
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epoch

Increasing number of epochs to 50

In [28]:
```
epochs = 50
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))
```

In [29]:
```
model.summary()
```

In [29]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_2 (Conv1D)           (None, 29, 32)            96

 batch_normalization_2 (Batc  (None, 29, 32)           128
 hNormalization)

 dropout_3 (Dropout)         (None, 29, 32)            0

 conv1d_3 (Conv1D)           (None, 28, 64)            4160

 batch_normalization_3 (Batc  (None, 28, 64)           256
 hNormalization)

 dropout_4 (Dropout)         (None, 28, 64)            0

 flatten_1 (Flatten)         (None, 1792)              0

 dense_2 (Dense)             (None, 64)                114752

 dropout_5 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 119,457
Trainable params: 119,265
Non-trainable params: 192
_________________________________________________________________
```

Adding Max Pool layer

In [26]:
```
epochs = 50
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(MaxPool1D(2))
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool1D(2))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
plot_learningCurve(history, epochs)
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.le
gacy.Adam.
```

```
Epoch 1/50
25/25 [==============================] - 4s 30ms/step - loss: 0.5877 - accuracy: 0.7408 - val_loss: 0.5620 - val_accuracy: 0.
8477
Epoch 2/50
25/25 [==============================] - 0s 11ms/step - loss: 0.3311 - accuracy: 0.8704 - val_loss: 0.5066 - val_accuracy: 0.
8629
Epoch 3/50
25/25 [==============================] - 0s 13ms/step - loss: 0.3050 - accuracy: 0.8844 - val_loss: 0.4616 - val_accuracy: 0.
8883
Epoch 4/50
25/25 [==============================] - 0s 13ms/step - loss: 0.2693 - accuracy: 0.9136 - val_loss: 0.4165 - val_accuracy: 0.
9036
Epoch 5/50
25/25 [==============================] - 0s 13ms/step - loss: 0.2259 - accuracy: 0.9174 - val_loss: 0.3729 - val_accuracy: 0.
9086
```

Increasing the number of layers

Architecture of 14 layers

In [27]:
```
epochs = 100
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))


model.add(Dense(1, activation='sigmoid'))
```

In [28]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_6 (Conv1D)           (None, 29, 32)            96

 batch_normalization_6 (Batc  (None, 29, 32)           128
 hNormalization)

 dropout_9 (Dropout)         (None, 29, 32)            0

 conv1d_7 (Conv1D)           (None, 28, 64)            4160

 batch_normalization_7 (Batc  (None, 28, 64)           256
 hNormalization)

 dropout_10 (Dropout)        (None, 28, 64)            0

 flatten_3 (Flatten)         (None, 1792)              0

 dense_6 (Dense)             (None, 64)                114752

 dropout_11 (Dropout)        (None, 64)                0

 dense_7 (Dense)             (None, 100)               6500

 dense_8 (Dense)             (None, 50)                5050

 dense_9 (Dense)             (None, 25)                1275

 dense_10 (Dense)            (None, 1)                 26

=================================================================
Total params: 132,243
Trainable params: 132,051
Non-trainable params: 192
```

In [29]:
```
model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.lega
cy.Adam.
```

In [30]:
```
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/100
25/25 [==============================] - 5s 33ms/step - loss: 0.3987 - accuracy: 0.8272 - val_loss: 0.5518 - val_accuracy: 0.
8985
Epoch 2/100
25/25 [==============================] - 0s 14ms/step - loss: 0.2461 - accuracy: 0.9123 - val_loss: 0.4877 - val_accuracy: 0.
9340
Epoch 3/100
25/25 [==============================] - 0s 16ms/step - loss: 0.2070 - accuracy: 0.9327 - val_loss: 0.4258 - val_accuracy: 0.
9188
Epoch 4/100
25/25 [==============================] - 0s 15ms/step - loss: 0.1922 - accuracy: 0.9288 - val_loss: 0.4292 - val_accuracy: 0.
7614
Epoch 5/100
25/25 [==============================] - 0s 16ms/step - loss: 0.1903 - accuracy: 0.9314 - val_loss: 0.3944 - val_accuracy: 0.
8883
Epoch 6/100
25/25 [==============================] - 0s 15ms/step - loss: 0.1675 - accuracy: 0.9403 - val_loss: 0.3480 - val_accuracy: 0.
9188
Epoch 7/100
```

Architecture of 17 layers

In [47]:
```
epochs = 100
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(100, activation='relu'))
model.add(Dense(50,activation='relu'))
model.add(Dense(25,activation='relu'))

model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_10"
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1d_22 (Conv1D)          (None, 29, 32)            96        
                                                                 
 batch_normalization_22 (Bat  (None, 29, 32)           128       
 chNormalization)                                                
                                                                 
 dropout_29 (Dropout)        (None, 29, 32)            0         
                                                                 
 conv1d_23 (Conv1D)          (None, 28, 64)            4160      
                                                                 
 batch_normalization_23 (Bat  (None, 28, 64)           256       
 chNormalization)                                                
                                                                 
 dropout_30 (Dropout)        (None, 28, 64)            0         
                                                                 
 conv1d_24 (Conv1D)          (None, 27, 64)            8256      
                                                                 
 batch_normalization_24 (Bat  (None, 27, 64)           256       
 chNormalization)                                                
                                                                 
 dropout_31 (Dropout)        (None, 27, 64)            0         
                                                                 
 flatten_7 (Flatten)         (None, 1728)              0         
                                                                 
 dense_28 (Dense)            (None, 64)                110656    
                                                                 
 dropout_32 (Dropout)        (None, 64)                0         
                                                                 
 dense_29 (Dense)            (None, 3)                 195       
                                                                 
 dense_30 (Dense)            (None, 1)                 4         
                                                                 
=================================================================
Total params: 124,007
Trainable params: 123,687
Non-trainable params: 320
_________________________________________________________________
```

```
In [50]: model.compile(loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
In [51]: history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/100
25/25 [==============================] - 2s 20ms/step - loss: 0.1232 - accuracy: 0.9733 - val_loss: 0.5547 - val_accuracy: 0.
9188
Epoch 2/100
25/25 [==============================] - 0s 10ms/step - loss: 0.1233 - accuracy: 0.9809 - val_loss: 0.5359 - val_accuracy: 0.
9239
Epoch 3/100
25/25 [==============================] - 0s 10ms/step - loss: 0.1172 - accuracy: 0.9822 - val_loss: 0.5835 - val_accuracy: 0.
9137
Epoch 4/100
25/25 [==============================] - 0s 10ms/step - loss: 0.1287 - accuracy: 0.9746 - val_loss: 0.5435 - val_accuracy: 0.
9239
Epoch 5/100
25/25 [==============================] - 0s 11ms/step - loss: 0.1063 - accuracy: 0.9848 - val_loss: 0.6482 - val_accuracy: 0.
9137
Epoch 6/100
25/25 [==============================] - 0s 10ms/step - loss: 0.1308 - accuracy: 0.9771 - val_loss: 0.5573 - val_accuracy: 0.
9239
Epoch 7/100
```

Architecture of 20 layers

```
In [31]: epochs = 100
model = Sequential()
model.add(Conv1D(32, 2, activation='relu', input_shape = X_train[0].shape))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Conv1D(64, 2, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(25, activation='relu'))

model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_4"
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_8 (Conv1D)           (None, 29, 32)            96

 batch_normalization_8 (Batc  (None, 29, 32)           128
 hNormalization)

 dropout_12 (Dropout)        (None, 29, 32)            0

 conv1d_9 (Conv1D)           (None, 28, 64)            4160

 batch_normalization_9 (Batc  (None, 28, 64)           256
 hNormalization)

 dropout_13 (Dropout)        (None, 28, 64)            0

 conv1d_10 (Conv1D)          (None, 27, 64)            8256

 batch_normalization_10 (Bat  (None, 27, 64)           256
 chNormalization)

 dropout_14 (Dropout)        (None, 27, 64)            0

 conv1d_11 (Conv1D)          (None, 26, 64)            8256

 batch_normalization_11 (Bat  (None, 26, 64)           256
 chNormalization)

 dropout_15 (Dropout)        (None, 26, 64)            0

 flatten_4 (Flatten)         (None, 1664)              0

 dense_11 (Dense)            (None, 64)                106560

 dropout_16 (Dropout)        (None, 64)                0

 dense_12 (Dense)            (None, 100)               6500

 dense_13 (Dense)            (None, 50)                5050

 dense_14 (Dense)            (None, 25)                1275

 dense_15 (Dense)            (None, 1)                 26
```

```
=================================================================
Total params: 141,075
Trainable params: 140,627
Non-trainable params: 448
_________________________________________________________________
```

```
In [32]: model.compile(optimizer=Adam(lr=0.0001), loss = 'binary_crossentropy', metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.lega
cy.Adam.
```

```
In [33]: history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 2/100
25/25 [==============================] - 1s 24ms/step - loss: 0.2991 - accuracy: 0.8971 - val_loss: 0.5551 - val_accuracy: 0.
8071
Epoch 3/100
25/25 [==============================] - 1s 23ms/step - loss: 0.2367 - accuracy: 0.9098 - val_loss: 0.4415 - val_accuracy: 0.
9188
Epoch 4/100
25/25 [==============================] - 1s 24ms/step - loss: 0.2317 - accuracy: 0.9136 - val_loss: 0.3537 - val_accuracy: 0.
9137
Epoch 5/100
25/25 [==============================] - 1s 25ms/step - loss: 0.2033 - accuracy: 0.9314 - val_loss: 0.2989 - val_accuracy: 0.
9188
Epoch 6/100
25/25 [==============================] - 1s 21ms/step - loss: 0.2117 - accuracy: 0.9288 - val_loss: 0.2739 - val_accuracy: 0.
9137
Epoch 7/100
25/25 [==============================] - 1s 23ms/step - loss: 0.1817 - accuracy: 0.9314 - val_loss: 0.2734 - val_accuracy: 0.
9137
Epoch 8/100
```

CHAPTER 4
RESULTS

Comparison of machine learning classification algorithms using the below metrics:

Sr No	Algorithm Name	Accuracy Score (%)	F1 Score (%)
1.	Decision tree algorithm	99.93	81.05
2.	KNN algorithm	99.95	85.71
3.	Logistic regression algorithm	99.91	73.56
4.	SVM Algorithms	99.93	77.71
5.	Random forest tree algorithm	99.92	77.27
6.	XG Boost	99.94	84.49

It is clear from the above table that the K Nearest Neighbour classification algorithm performs the best with an accuracy of 99.95%.

CHAPTER 5
CONCLUSION AND FUTURE WORKS

5.1 Conclusion

With an accuracy of 99.95, the K Nearest Neighbour method outperforms all other algorithms. We have employed dataset downsampling to address an unbalanced dataset for CNN.

5.2 Future Works

Investigating new data sources that can be utilized to spot fraud is one potential subject for future research. For instance, using information from social media, online buying patterns, and location data can offer more details that can be used to spot fraudulent actions.

The accuracy and robustness of fraud detection systems can frequently be improved by combining multiple machine learning models. Future research might investigate the creation of ensemble models that can successfully combine the advantages of various techniques to enhance overall performance.

CHAPTER 6
REFERENCES

1. Y. Abakarim, M. Lahby, and A. Attioui, ‘‘An efficient real time model for credit card fraud detection based on deep learning,’’ in Proc. 12th Int. Conf. Intell. Systems: Theories Appl., Oct. 2018, pp. 1–7, doi: 10.1145/3289402.3289530.
2. H. Abdi and L. J. Williams, ‘‘Principal component analysis,’’ Wiley Inter-discipl. Rev., Comput. Statist., vol. 2, no. 4, pp. 433–459, Jul. 2010, doi: 10.1002/wics.101.
3. V. Arora, R. S. Leekha, K. Lee, and A. Kataria, ‘‘Facilitating user authorization from imbalanced data logs of credit cards using artificial intelligence,’’ Mobile Inf. Syst., vol. 2020, pp. 1–13, Oct. 2020, doi: 10.1155/2020/8885269.
4. A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, ‘‘Performance analysis of feature selection methods in software defect prediction: A search method approach,’’ Appl. Sci., vol. 9, no. 13, p. 2764, Jul. 2019, doi: 10.3390/app9132764.
5. B. Bandaranayake, ‘‘Fraud and corruption control at education system level: A case study of the Victorian department of education and early childhood development in Australia,’’ J. Cases Educ. Leadership, vol. 17, no. 4, pp. 34–53, Dec. 2014, doi: 10.1177/1555458914549669.
6. J. Błaszczyński, A. T. de Almeida Filho, A. Matuszyk, M. Szelg¸, and R. Słowiński, ‘‘Auto loan fraud detection using dominance-based rough set approach versus machine learning methods,’’ Expert Syst. Appl., vol. 163, Jan. 2021, Art. no. 113740, doi: 10.1016/j.eswa.2020.113740.