Detection of Fraudulent Transactions

Ayoub HAIDA

April 3, 2023

1 Detection of Fraudulent Transactions

The objective of this case study is to employ different models based on classification techniques for the purpose of identifying whether a given transaction is a regular payment or a fraudulent one.

2 Problem Definition

In this case, the response variable takes a value of 1 in case the given transaction is fraud and 0 otherwise.

The dataset provided contains information about credit card transactions carried out by European cardholders in September 2013. The data pertains to transactions that took place over a period of two days, during which there were a total of 492 fraudulent transactions out of 284,807. It is important to note that this dataset is highly imbalanced, with only 0.172% of all transactions being identified as frauds. The main objective is to predict instances of fraud. The response variable, 'Class', takes on a value of 1 if the transaction is fraudulent and 0 otherwise. The features in the dataset are a result of PCA transformation and may not be immediately interpretable based on their names.

Here, you can find the dataset details as well as download it: https://www.kaggle.com/mlg-ulb/creditcardfraud

3 Loading the Packages and the Data

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot
from pandas import read_csv, set_option
from pandas.plotting import scatter_matrix
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
↪GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
```

```
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier,␣
 ↪RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.optimizers import SGD
from pickle import dump
from pickle import load
```

[2]:
```
# load dataset
dataset = read_csv('/kaggle/input/creditcardfraud/creditcard.csv')
```

[3]:
```
import warnings
warnings.filterwarnings('ignore')
```

4 Exploratory Data Analysis

4.1 Stats

[4]:
```
# shape
dataset.shape
```

[4]:
```
(284807, 31)
```

[5]:
```
set_option('display.width', 100)
dataset.head()
```

[5]:
```
   Time        V1        V2        V3        V4        V5        V6        V7
V8        V9  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
0.098698  0.363787
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
0.085102 -0.255425
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
0.247676 -1.514654
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
0.377436 -1.387024
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941
-0.270533  0.817739

   …       V21       V22       V23       V24       V25       V26       V27
V28  Amount  \
```

```
0  … -0.018307  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558
-0.021053  149.62
1  … -0.225775 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983
0.014724    2.69
2  …  0.247998  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353
-0.059752  378.66
3  … -0.108300  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723
0.061458  123.50
4  … -0.009431  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422
0.215153   69.99

   Class
0      0
1      0
2      0
3      0
4      0

[5 rows x 31 columns]
```

[6]:
```
set_option('display.max_rows', 500)
dataset.dtypes
```

[6]:
```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
```

```
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

[7]:
```python
pd.set_option('display.float_format', '{:.3f}'.format)
dataset.describe()
```

[7]:
```
            Time         V1          V2          V3          V4          V5
V6          V7  \
count 284807.000 284807.000 284807.000 284807.000 284807.000 284807.000
284807.000 284807.000
mean   94813.860      0.000      0.000     -0.000      0.000      0.000
0.000     -0.000
std    47488.146      1.959      1.651      1.516      1.416      1.380
1.332      1.237
min        0.000    -56.408    -72.716    -48.326     -5.683   -113.743
-26.161    -43.557
25%    54201.500     -0.920     -0.599     -0.890     -0.849     -0.692
-0.768     -0.554
50%    84692.000      0.018      0.065      0.180     -0.020     -0.054
-0.274      0.040
75%   139320.500      1.316      0.804      1.027      0.743      0.612
0.399      0.570
max   172792.000      2.455     22.058      9.383     16.875     34.802
73.302    120.589

              V8         V9  …         V21         V22         V23         V24
V25  \
count 284807.000 284807.000  … 284807.000 284807.000 284807.000 284807.000
284807.000
mean       0.000     -0.000  …      0.000     -0.000      0.000      0.000
0.000
std        1.194      1.099  …      0.735      0.726      0.624      0.606
0.521
min      -73.217    -13.434  …    -34.830    -10.933    -44.808     -2.837
-10.295
25%       -0.209     -0.643  …     -0.228     -0.542     -0.162     -0.355
-0.317
50%        0.022     -0.051  …     -0.029      0.007     -0.011      0.041
0.017
75%        0.327      0.597  …      0.186      0.529      0.148      0.440
0.351
```

```
max       20.007     15.595  …     27.203     10.503     22.528      4.585
7.520

             V26        V27        V28     Amount      Class
count 284807.000 284807.000 284807.000 284807.000 284807.000
mean       0.000     -0.000     -0.000     88.350      0.002
std        0.482      0.404      0.330    250.120      0.042
min       -2.605    -22.566    -15.430      0.000      0.000
25%       -0.327     -0.071     -0.053      5.600      0.000
50%       -0.052      0.001      0.011     22.000      0.000
75%        0.241      0.091      0.078     77.165      0.000
max        3.517     31.612     33.848  25691.160      1.000

[8 rows x 31 columns]
```

To begin with, we will examine the number of instances of fraud as compared to non-fraud cases in the dataset.

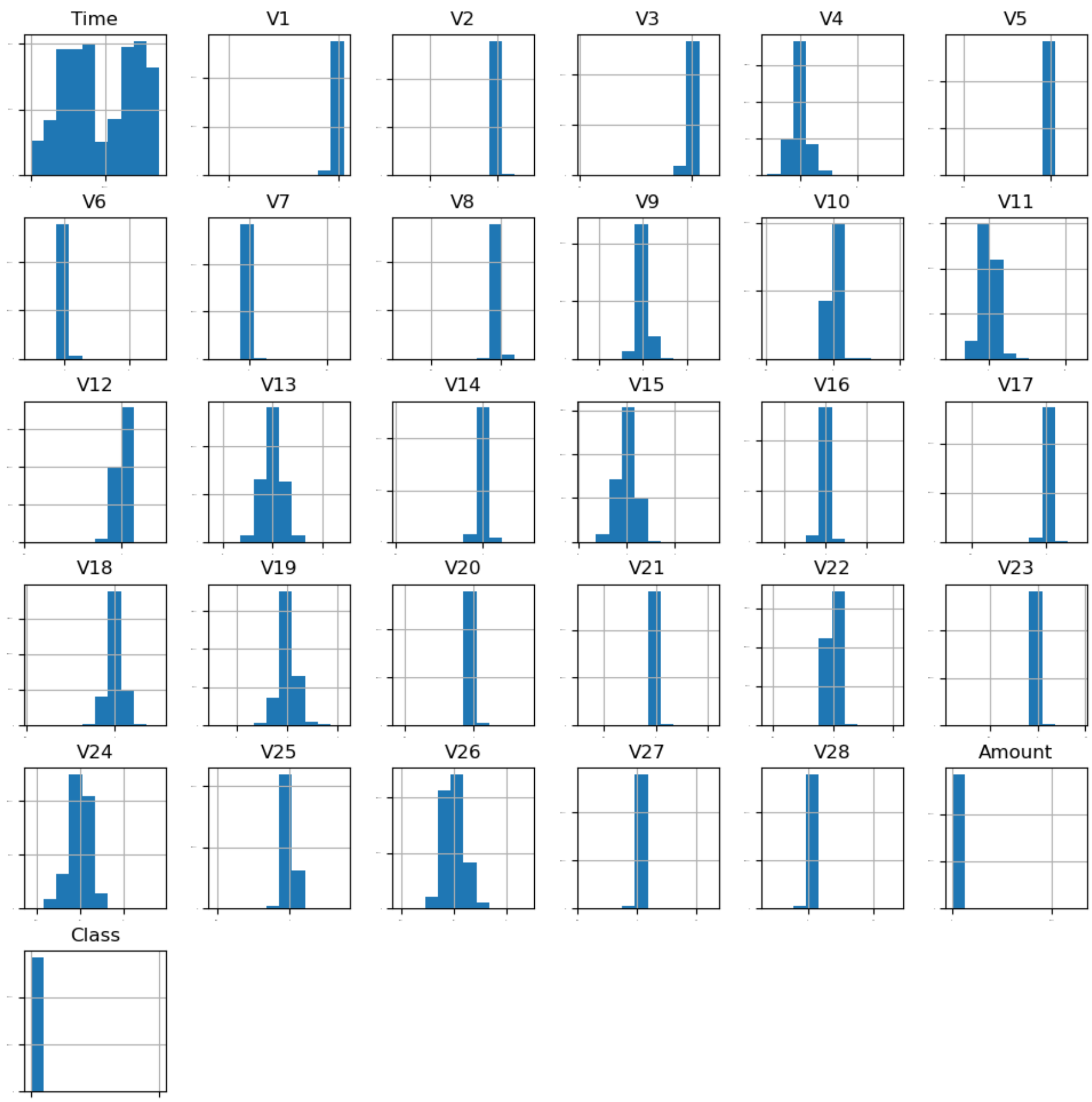
[8]:
```python
class_names = {0:'Not Fraud', 1:'Fraud'}
print(dataset.Class.value_counts().rename(index = class_names))
```

```
Not Fraud    284315
Fraud           492
Name: Class, dtype: int64
```

The distribution of instances in the dataset is skewed, with a significant majority of the transactions being classified as non-fraudulent.

4.2 Visualization

[9]:
```python
# histograms
dataset.hist(sharex=False, sharey=False, xlabelsize=1, ylabelsize=1,␣
 ↪figsize=(12,12))
pyplot.show()
```



4.3 Data Preparation

[10]:
```
print('Null Values =',dataset.isnull().values.any())
```

```
Null Values = False
```

There is no null in the data.

[11]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

bestfeatures = SelectKBest( k=10)
bestfeatures
```

```
Y= dataset["Class"]
X = dataset.loc[:, dataset.columns != 'Class']
fit = bestfeatures.fit(X,Y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)

featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

```
   Specs      Score
17   V17 33979.169
14   V14 28695.548
12   V12 20749.822
10   V10 14057.980
16   V16 11443.349
3     V3 11014.508
7     V7 10349.605
11   V11  6999.355
4     V4  5163.832
18   V18  3584.381
```

While certain features in the dataset may be important for detecting fraud, the process of selecting these features has not been emphasized or prioritized.

5 Algorithms and Models

5.1 Train-Test-Split and Evaluation Metrics

[14]:
```
Y= dataset["Class"]
X = dataset.loc[:, dataset.columns != 'Class']
validation_size = 0.2
seed = 7
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y,␣
 ↪test_size=validation_size, random_state=seed)
scoring = 'accuracy'
```

[16]:
```
num_folds = 10
seed = 7
```

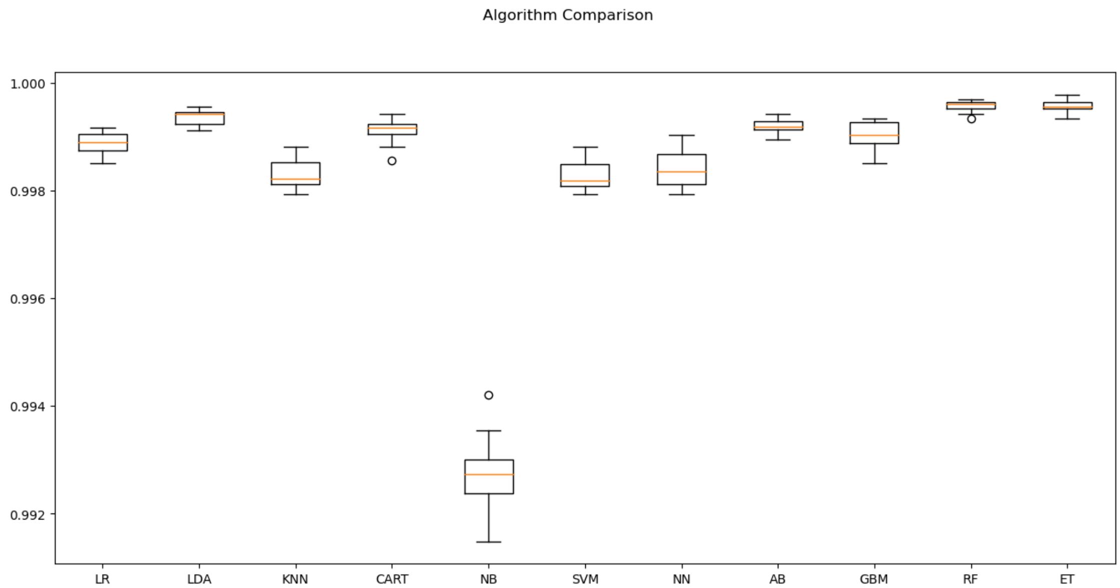
[17]:
```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('NN', MLPClassifier()))
```

```
models.append(('AB', AdaBoostClassifier()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('ET', ExtraTreesClassifier()))
```

[18]:
```
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,␣
 ↪scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.998885 (0.000205)
LDA: 0.999364 (0.000136)
KNN: 0.998310 (0.000290)
CART: 0.999109 (0.000245)
NB: 0.992745 (0.000733)
SVM: 0.998280 (0.000278)
NN: 0.998429 (0.000361)
AB: 0.999192 (0.000154)
GBM: 0.999008 (0.000285)
RF: 0.999570 (0.000107)
ET: 0.999565 (0.000122)
```

[20]:
```
# compare algorithms
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
fig.set_size_inches(15,7)
pyplot.show()
```



[21]:
```
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
```

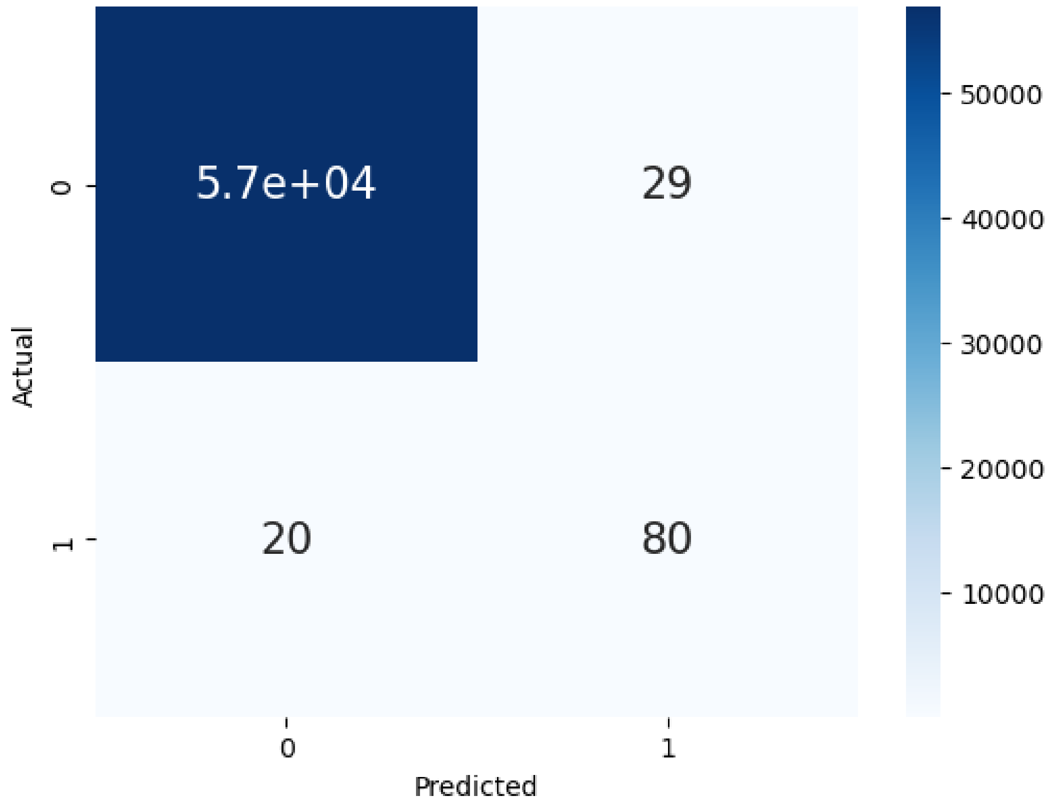
[21]:
```
DecisionTreeClassifier()
```

[22]:
```
rescaledValidationX = X_validation
predictions = model.predict(rescaledValidationX)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9991397773954567
[[56833    29]
 [   20    80]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56862
           1       0.73      0.80      0.77       100

    accuracy                           1.00     56962
   macro avg       0.87      0.90      0.88     56962
weighted avg       1.00      1.00      1.00     56962
```

[23]:
```
df_cm = pd.DataFrame(confusion_matrix(Y_validation, predictions), columns=np.
 ↪unique(Y_validation), index = np.unique(Y_validation))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
```

```python
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```



Despite achieving good results, it is worth noting that 20 out of 100 fraud cases are not detected. Therefore, it is crucial to prioritize the metric of recall, which minimizes false negatives.

5.2 Model Tuning

[24]:
```python
scoring = 'recall'
```

[25]:
```python
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('NN', MLPClassifier()))
models.append(('AB', AdaBoostClassifier()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('ET', ExtraTreesClassifier()))
```

[ ]:
```
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold,␣
 ↪scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.631287 (0.116650)
LDA: 0.758283 (0.045450)
KNN: 0.023882 (0.019671)
CART: 0.749286 (0.070769)
NB: 0.659465 (0.080686)
SVM: 0.000000 (0.000000)
```

Given the LDA has the best recall out of all the models, it is used to ealuate the test set

[ ]:
```
model = LinearDiscriminantAnalysis()
model.fit(X_train, Y_train)
```

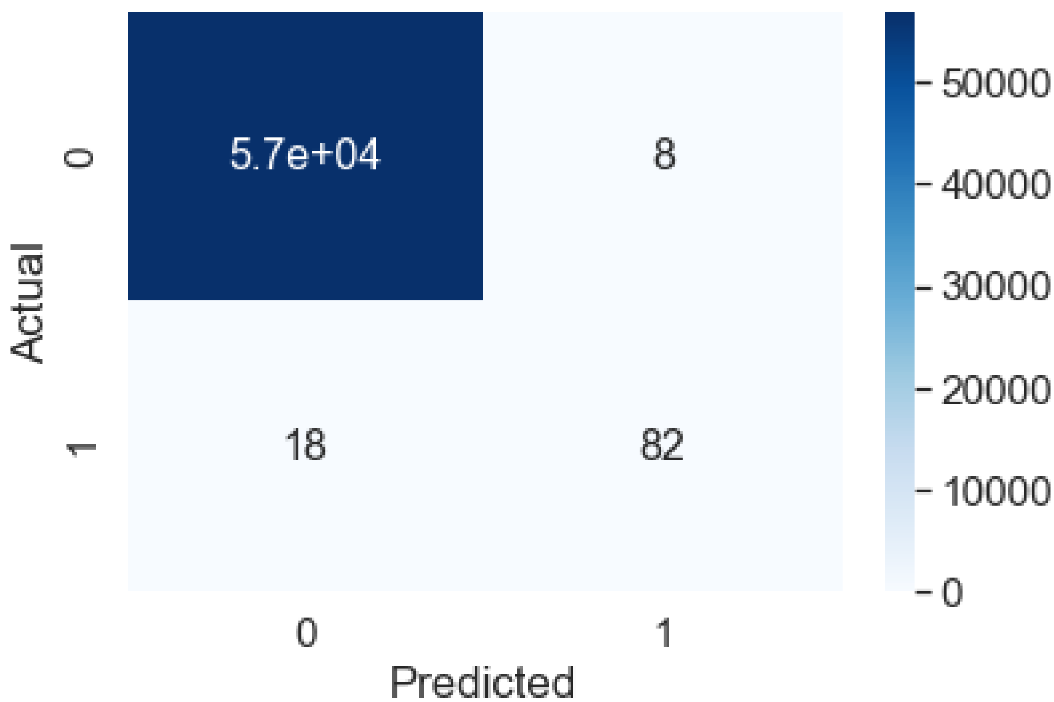
[185]:
```
rescaledValidationX = X_validation
predictions = model.predict(rescaledValidationX)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9995435553526912

[[56854     8]

 [   18    82]]

              precision    recall  f1-score   support



           0       1.00      1.00      1.00     56862

           1       0.91      0.82      0.86       100



    accuracy                           1.00     56962

   macro avg       0.96      0.91      0.93     56962
```

```
weighted avg       1.00      1.00      1.00     56962
```

[186]:
```python
df_cm = pd.DataFrame(confusion_matrix(Y_validation, predictions), columns=np.
 ↪unique(Y_validation), index = np.unique(Y_validation))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```

[186]: <matplotlib.axes._subplots.AxesSubplot at 0x20b99399978>



[204]:
```python
Y_train.head()
```

[204]:
```
44828     0
221877    0
278826    0
149792    0
226041    0
Name: Class, dtype: int64
```

[41]:
```python
df = pd.concat([X_train, Y_train], axis=1)

fraud_df = df.loc[df['Class'] == 1]
```

```python
non_fraud_df = df.loc[df['Class'] == 0][:492]# fraud classes = 492.

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

df_new = normal_distributed_df.sample(frac=1, random_state=42)
Y_train_new= df_new["Class"]
X_train_new = df_new.loc[:, dataset.columns != 'Class']

dataset.head()
```

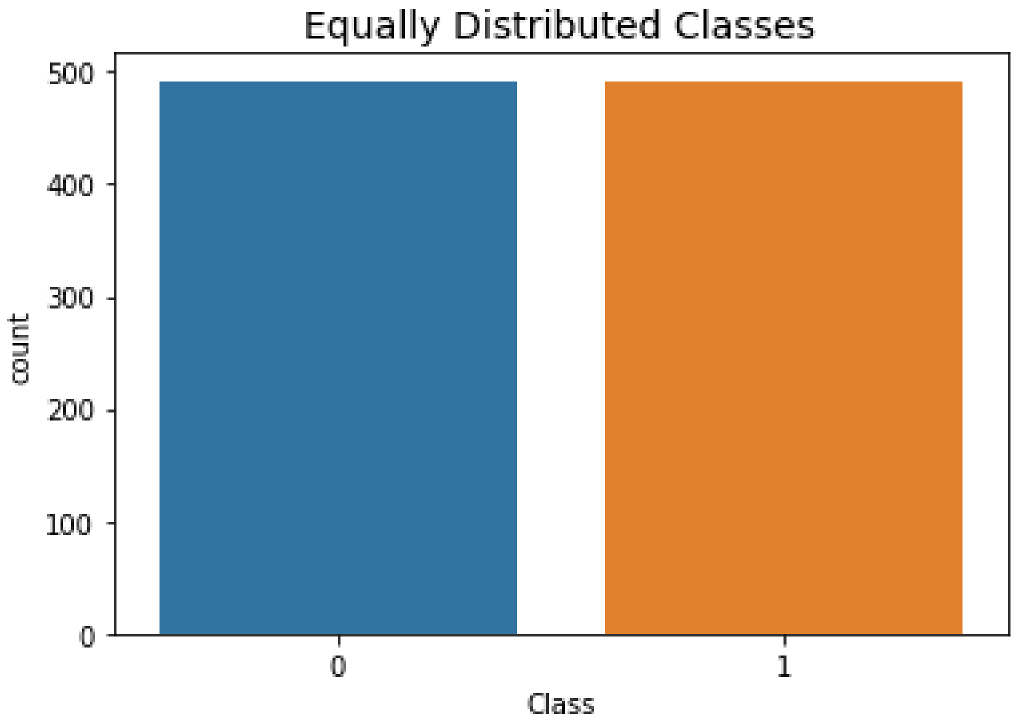
[41]:

```
   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9  …
V21    V22    V23  \
0   0.0 -1.360 -0.073  2.536  1.378 -0.338  0.462  0.240  0.099  0.364  …
-0.018  0.278 -0.110
1   0.0  1.192  0.266  0.166  0.448  0.060 -0.082 -0.079  0.085 -0.255  …
-0.226 -0.639  0.101
2   1.0 -1.358 -1.340  1.773  0.380 -0.503  1.800  0.791  0.248 -1.515  …
0.248  0.772  0.909
3   1.0 -0.966 -0.185  1.793 -0.863 -0.010  1.247  0.238  0.377 -1.387  …
-0.108  0.005 -0.190
4   2.0 -1.158  0.878  1.549  0.403 -0.407  0.096  0.593 -0.271  0.818  …
-0.009  0.798 -0.137

     V24    V25    V26    V27    V28  Amount  Class
0  0.067  0.129 -0.189  0.134 -0.021  149.62      0
1 -0.340  0.167  0.126 -0.009  0.015    2.69      0
2 -0.689 -0.328 -0.139 -0.055 -0.060  378.66      0
3 -1.176  0.647 -0.222  0.063  0.061  123.50      0
4  0.141 -0.206  0.502  0.219  0.215   69.99      0

[5 rows x 31 columns]
```

[42]:

```python
print('Distribution of the Classes in the subsample dataset')
print(df_new['Class'].value_counts()/len(df_new))
sns.countplot('Class', data=df_new)
pyplot.title('Equally Distributed Classes', fontsize=14)
pyplot.show()
```

```
Distribution of the Classes in the subsample dataset

1    0.5

0    0.5

Name: Class, dtype: float64
```



[43]:
```
scoring='accuracy'
```

[44]:
```
# spot check the algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('NN', MLPClassifier()))
models.append(('AB', AdaBoostClassifier()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RF', RandomForestClassifier()))
models.append(('ET', ExtraTreesClassifier()))
```

[45]:
```
EnableDLModelsFlag = 1
if EnableDLModelsFlag == 1 :
    def create_model(neurons=12, activation='relu', learn_rate = 0.01,␣
 ↪momentum=0):
        # create model
        model = Sequential()
```

```
        model.add(Dense(X_train.shape[1], input_dim=X_train.shape[1],␣
↪activation=activation))
        model.add(Dense(32, activation=activation))
        model.add(Dense(1, activation='sigmoid'))
        optimizer = SGD(lr=learn_rate, momentum=momentum)
        model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
        return model
    models.append(('DNN', KerasClassifier(build_fn=create_model, epochs=50,␣
↪batch_size=10, verbose=0)))
```

[46]:

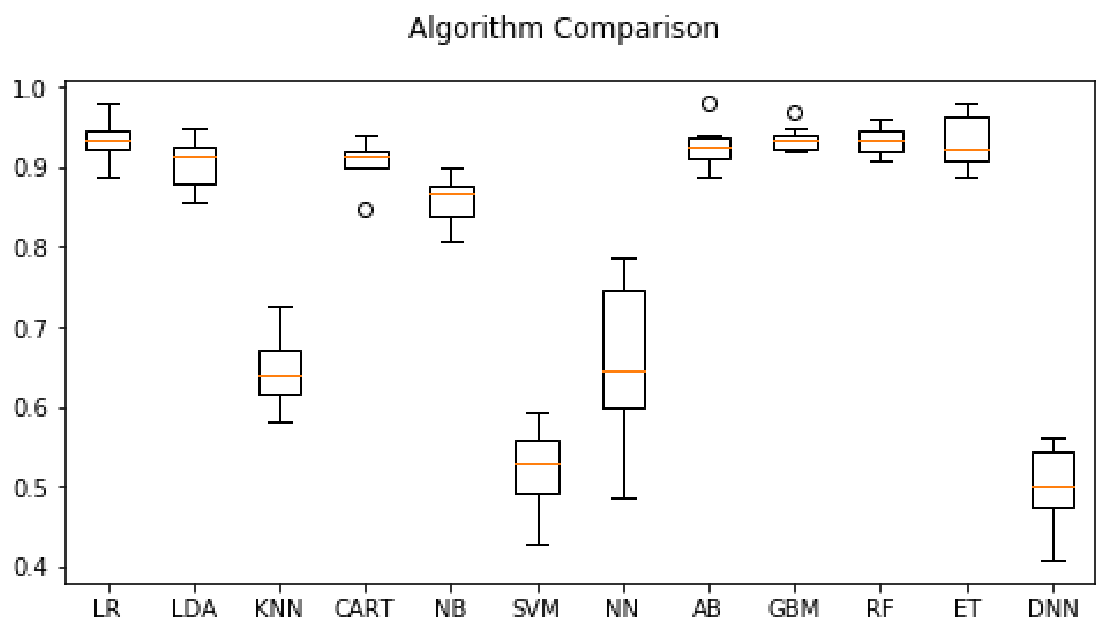
```
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=num_folds, random_state=seed)
    cv_results = cross_val_score(model, X_train_new, Y_train_new, cv=kfold,␣
↪scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.931911 (0.024992)

LDA: 0.905473 (0.027422)

KNN: 0.648258 (0.044550)

CART: 0.907565 (0.022669)

NB: 0.860771 (0.027234)

SVM: 0.522356 (0.048395)

NN: 0.648712 (0.100137)

AB: 0.924830 (0.024068)

GBM: 0.934982 (0.015132)

RF: 0.932931 (0.015859)

ET: 0.931962 (0.031043)

WARNING:tensorflow:From D:\Anaconda\lib\site-
packages\tensorflow\python\framework\op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
```

```
version.

Instructions for updating:

Colocations handled automatically by placer.

WARNING:tensorflow:From D:\Anaconda\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.

Instructions for updating:

Use tf.cast instead.

DNN: 0.498011 (0.050742)
```

[47]:

```
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
fig.set_size_inches(8,4)
pyplot.show()
```



Since GBM has been determined as the top-performing model among all other models, a grid search is conducted to optimize the GBM model's performance by varying the number of estimators and maximum depth parameters.

[48]:
```
n_estimators = [20,180,1000]
max_depth= [2, 3,5]
param_grid = dict(n_estimators=n_estimators, max_depth=max_depth)
model = GradientBoostingClassifier()
kfold = KFold(n_splits=num_folds, random_state=seed)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring,␣
 ↪cv=kfold)
grid_result = grid.fit(X_train_new, Y_train_new)

#Print Results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
ranks = grid_result.cv_results_['rank_test_score']
for mean, stdev, param, rank in zip(means, stds, params, ranks):
    print("#%d %f (%f) with: %r" % (rank, mean, stdev, param))
```

```
Best: 0.936992 using {'max_depth': 5, 'n_estimators': 1000}

#3 0.931911 (0.016958) with: {'max_depth': 2, 'n_estimators': 20}

#6 0.929878 (0.017637) with: {'max_depth': 2, 'n_estimators': 180}

#9 0.924797 (0.021358) with: {'max_depth': 2, 'n_estimators': 1000}

#6 0.929878 (0.020476) with: {'max_depth': 3, 'n_estimators': 20}

#3 0.931911 (0.011120) with: {'max_depth': 3, 'n_estimators': 180}

#3 0.931911 (0.017026) with: {'max_depth': 3, 'n_estimators': 1000}

#8 0.928862 (0.022586) with: {'max_depth': 5, 'n_estimators': 20}

#2 0.934959 (0.015209) with: {'max_depth': 5, 'n_estimators': 180}

#1 0.936992 (0.012639) with: {'max_depth': 5, 'n_estimators': 1000}
```

[49]:
```
model = GradientBoostingClassifier(max_depth= 5, n_estimators = 1000)
model.fit(X_train_new, Y_train_new)
```

[49]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
```

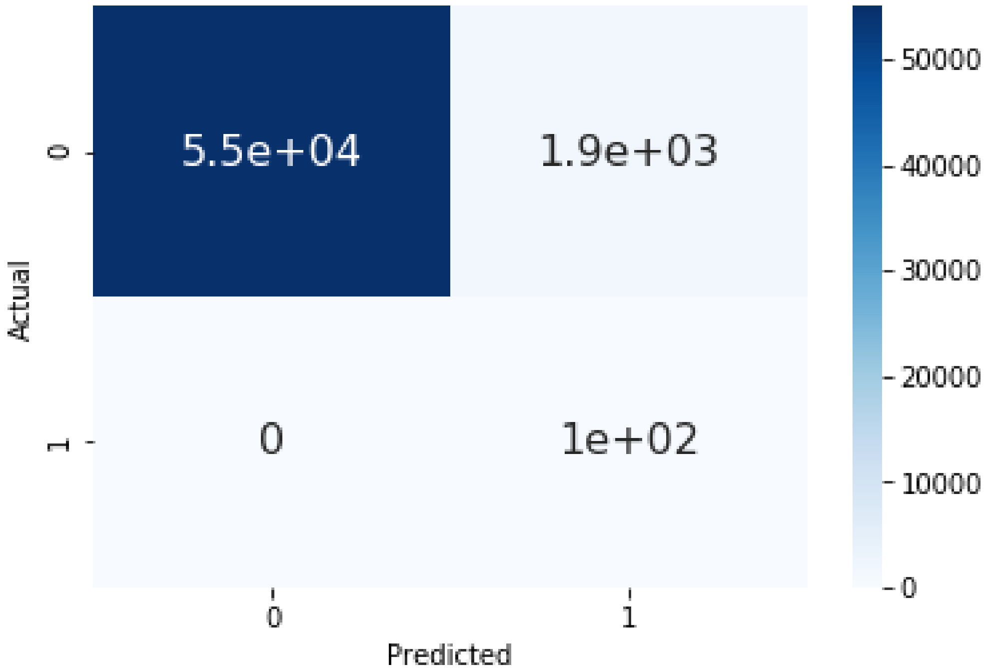
```
                             min_weight_fraction_leaf=0.0, n_estimators=1000,
                             n_iter_no_change=None, presort='auto',
                             random_state=None, subsample=1.0, tol=0.0001,
                             validation_fraction=0.1, verbose=0,
                             warm_start=False)
```

[50]:
```
predictions = model.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.9668199852533268

[[54972  1890]

 [    0   100]]

              precision    recall  f1-score   support



           0       1.00      0.97      0.98     56862

           1       0.05      1.00      0.10       100



    accuracy                           0.97     56962

   macro avg       0.53      0.98      0.54     56962

weighted avg       1.00      0.97      0.98     56962
```

[51]:
```
df_cm = pd.DataFrame(confusion_matrix(Y_validation, predictions), columns=np.
 ↪unique(Y_validation), index = np.unique(Y_validation))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
sns.heatmap(df_cm, cmap="Blues", annot=True,annot_kws={"size": 16});
```

[51]: <matplotlib.axes._subplots.AxesSubplot at 0x26e0cc0bb70>



The performance of the model on the test set is highly satisfactory, as it demonstrates superior results with no instances of fraud being missed.