Credit Card Fraud Detection

Fraud Detection Problem

In today's digital world, fraud is a big problem. Fraudulent activities like fake transactions can cost companies a lot of money and damage people's trust in online systems.

Traditional ways of spotting fraud, like using simple rules or manual checks, aren't always effective, especially when there are lots of transactions happening all the time.

Fraud Detection Problem

The best way to detect fraud is ML techniques that have indeed revolutionized fraud detection by providing powerful tools to analyze large volumes of transactional data and identify patterns indicative of fraudulent behavior. By leveraging algorithms that can learn from historical data and adapt to new fraud tactics, machine learning-based fraud detection systems can effectively identify suspicious activities in real-time and help mitigate potential risks. machine learning helping businesses and individuals protect themselves against financial losses and maintain trust in digital transactions.

Related Work

1)Fraud Detection in Python :Click here to view

They have seen that Naive Bayes, while it has the lowest precision, has the highest recall and lowest FP rate among the models. The Bayes model has the highest FN rate; however, this error is less serious (in a practical sense, these transactions would be flagged and fraud, and later resolved by the user, which is better than totally missing a fraudulent transaction), so they gave more weight in their decision to the low FP.

Model	Precision	Recall	FP	FN
Naive Bayes	0.1957	0.9993	2	11695
Logistic Regression	0.8887	0.7236	796	261
Neural Network	0.8721	0.7624	684	322
Decision Tree	0.9027	0.7212	803	224
Random Forest	0.8845	0.7462	1096	421

Related Work

2) Credit card Fraud: click here to view

Their goal was to implement 3 different machine learning models in order to classify, to the highest possible degree of accuracy. They implemented a logistic regression model, a k-means clustering model, and a neural network. Logistic regression outperformed both the K-means and neural network. They believed that it is because of how the decision boundary changed with the class weights features. The neural network was next, and K-means performed the poorest.

Model	Accuracy
Neural Network	94.56%
K-Means Clustering	54.27%
Logistic Regression	99.88%

Dataset Information

Obtained from Kaggle, consists of European credit card transactions (September 2013).

DATA VOLUME AND DIVERSITY:

- 284,807 transactions over two days
- 492 fraud cases (0.172% of transactions).

FEATURES:

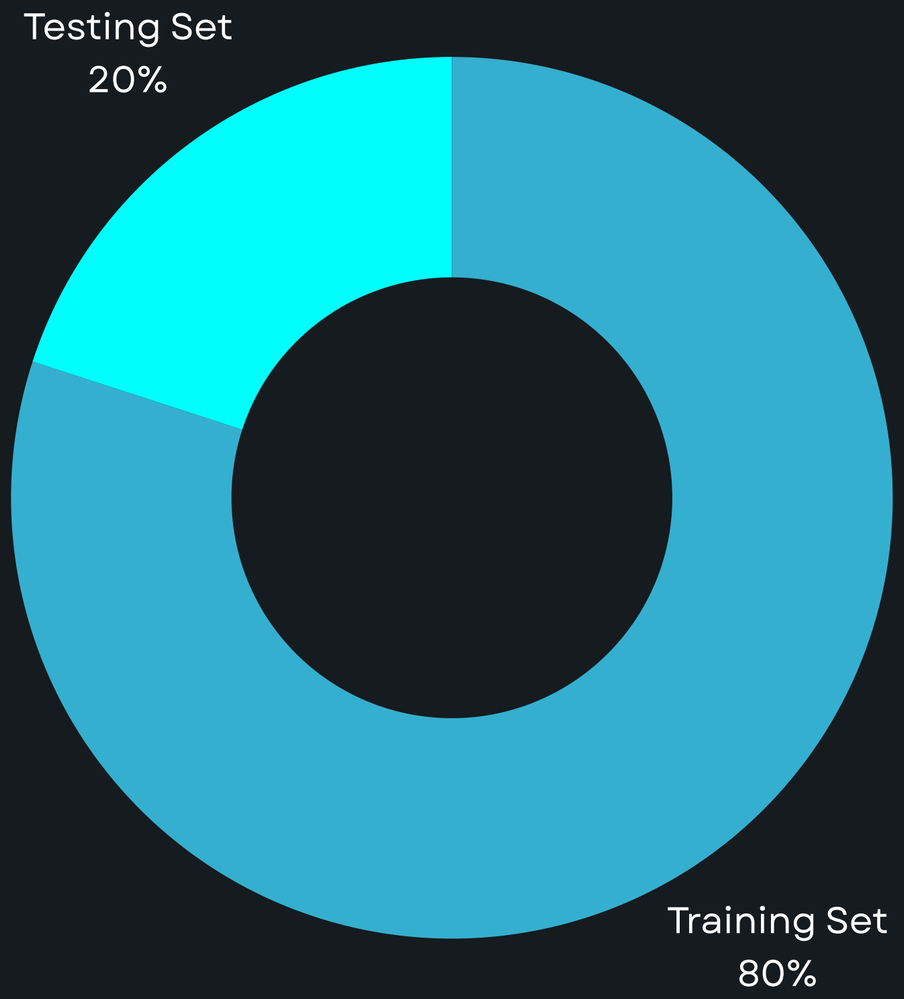
31 columns including transformed numerical features (PCA applied)
Sensitive information was protected, possibly through the use of PCA transformation, 'Time' (in seconds), 'Amount' (transaction amount).

IMBALANCED DATASET

Sample Dataset Creation: Balanced dataset by randomly selecting a subset of normal transactions (264 samples) to match fraud cases

Data Splitting

Split using sklearn's train_test_split function.



preprocessing

```python
# Cheking missing values in columns
df_missing_values = df.isnull().sum()

df_missing_values
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        1
V6        1
V7        1
V8        1
V9        1
V10       1
V11       1
V12       1
V13       1
V14       1
V15       1
V16       1
V17       1
V18       1
V19       1
V20       1
V21       1
V22       1
V23       1
V24       1
V25       1
V26       1
V27       1
V28       1
Amount    1
Class     1
```

preprocessing

```
# To handel with the null value, we use Median
df.fillna(df.median(), inplace=True)
print(df)
```

V24	V25	V26	V27	V28
0.066928	0.128539	-0.189115	0.133558	-0.021053
-0.339846	0.167170	0.125895	-0.008983	0.014724
-0.689281	-0.327642	-0.139097	-0.055353	-0.059752
-1.175575	0.647376	-0.221929	0.062723	0.061458
0.141267	-0.206010	0.502292	0.219422	0.215153
...	...	...	...	...
-0.883643	0.335230	0.150161	-0.029612	0.008220
0.475489	0.134602	0.066882	-0.031207	0.020989
0.334513	-0.257496	0.094942	0.487942	0.239537
0.220516	0.517620	-0.312217	0.041102	0.053760
NaN	NaN	NaN	NaN	NaN

```
          V24       V25       V26       V27       V28
 0.066928  0.128539 -0.189115  0.133558 -0.021053
-0.339846  0.167170  0.125895 -0.008983  0.014724
-0.689281 -0.327642 -0.139097 -0.055353 -0.059752
-1.175575  0.647376 -0.221929  0.062723  0.061458
 0.141267 -0.206010  0.502292  0.219422  0.215153
      ...       ...       ...       ...       ...
-0.883643  0.335230  0.150161 -0.029612  0.008220
 0.475489  0.134602  0.066882 -0.031207  0.020989
 0.334513 -0.257496  0.094942  0.487942  0.239537
 0.220516  0.517620 -0.312217  0.041102  0.053760
 0.068728  0.166478 -0.064879  0.011792  0.023610
```

preprocessing

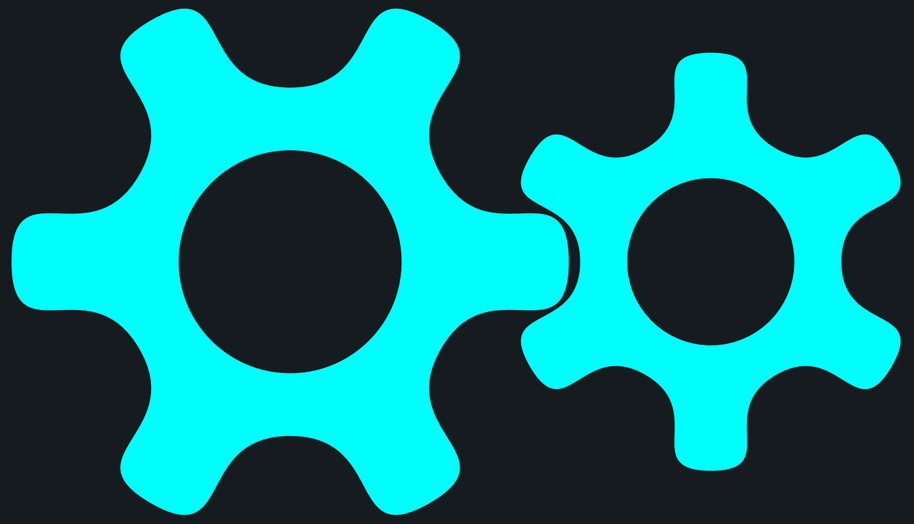
```
df_missing_values = df.isnull().sum()

df_missing_values
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
```

Models

We will focus on using supervised machine learning algorithms to classify fraudulent transactions. by evaluating the effectiveness of four machine learning models:

- Random Forest
- Decision Tree
- Logistic Regression
- Linear SVM

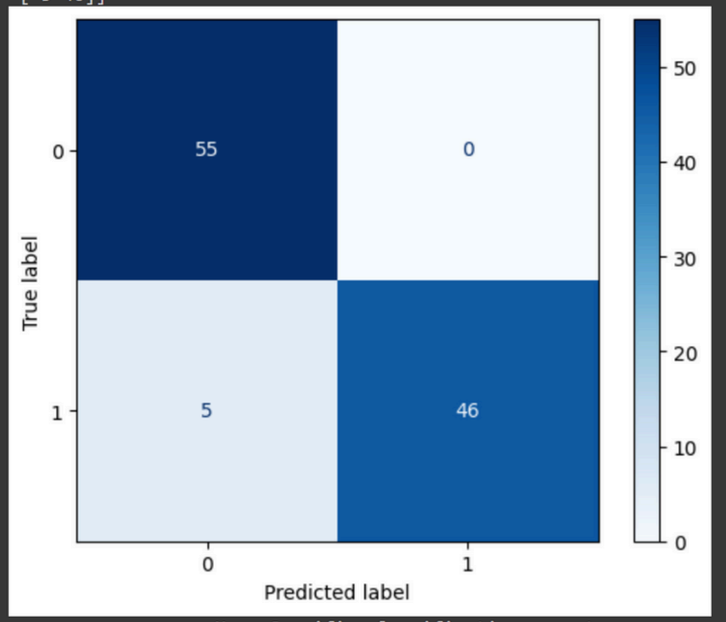


Models

Random Forest

```
model_rf = RandomForestClassifier(n_estimators=20, max_depth=5,n_jobs=-1,random_state=0)
model_rf.fit(X_train, y_train)
y_pred_rf = model_rf.predict(X_test)

# Confusion matrix
print("------------------ Confusion Matrix --------------------")
c_matrix = metrics.confusion_matrix(y_test, y_pred_rf)
print(c_matrix)

cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix)
cm_display.plot(cmap=plt.cm.Blues)
plt.show()


print("------------------ More Specific classification_report --------------------")

# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_pred_rf))
#Precision
print("Precision:",precision_score(y_test, y_pred_rf))
#Recall
print("Recall:", recall_score(y_test, y_pred_rf))
# F1 score
print("F1-Score:-", f1_score(y_test, y_pred_rf))

X_train_prediction = model_rf.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
print('Accuracy on Training data : ', training_data_accuracy)

results.loc[len(results)] = ['Random Forest', metrics.accuracy_score(y_test, y_pred_rf),precision_score(y_test, y_pred_rf),recall_score(y_t
```

```
------------------ Confusion Matrix --------------------
[[55  0]
 [ 5 46]]
```



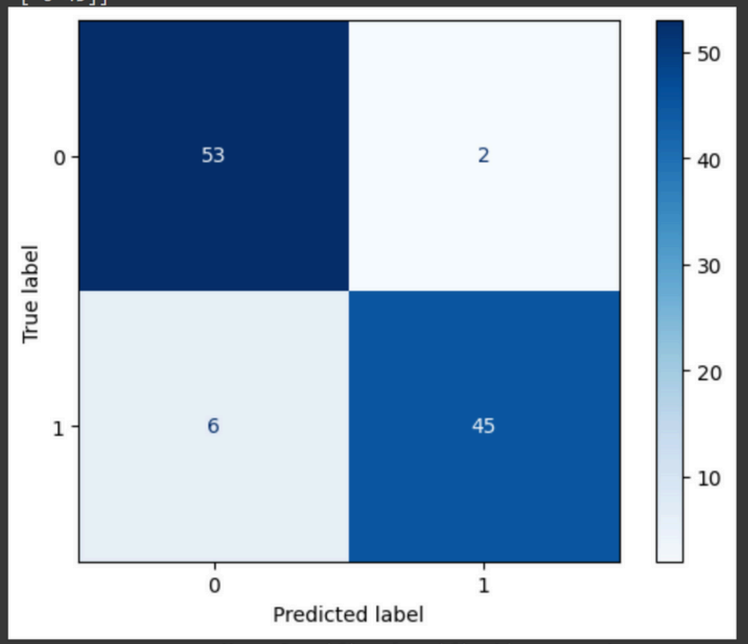

```
------------------ More Specific classification_report --------------------
Accuracy:- 0.9528301886792453
Precision: 1.0
Recall: 0.9019607843137255
F1-Score:- 0.9484536082474228
Accuracy on Training data :  0.976303317535545
```

Models

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression(random_state=0)
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

# Confusion matrix
print("------------------ Confusion Matrix --------------------")
c_matrix = metrics.confusion_matrix(y_test, y_pred_lr)
print(c_matrix)

cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix)
cm_display.plot(cmap=plt.cm.Blues)
plt.show()

print("------------------ More Specific classification_report --------------------")

# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_pred_lr))
#Precision
print("Precision:",precision_score(y_test, y_pred_lr))
#Recall
print("Recall:", recall_score(y_test, y_pred_lr))
# F1 score
print("F1-Score:-", f1_score(y_test, y_pred_lr))


X_train_prediction = model_lr.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
print('Accuracy on Training data : ', training_data_accuracy)

results.loc[len(results)] = ['Logistic Regression', metrics.accuracy_score(y_test, y_pred_lr),precision_score(y_test, y_pred_lr),
```

```
------------------ Confusion Matrix --------------------
[[53  2]
 [ 6 45]]
```



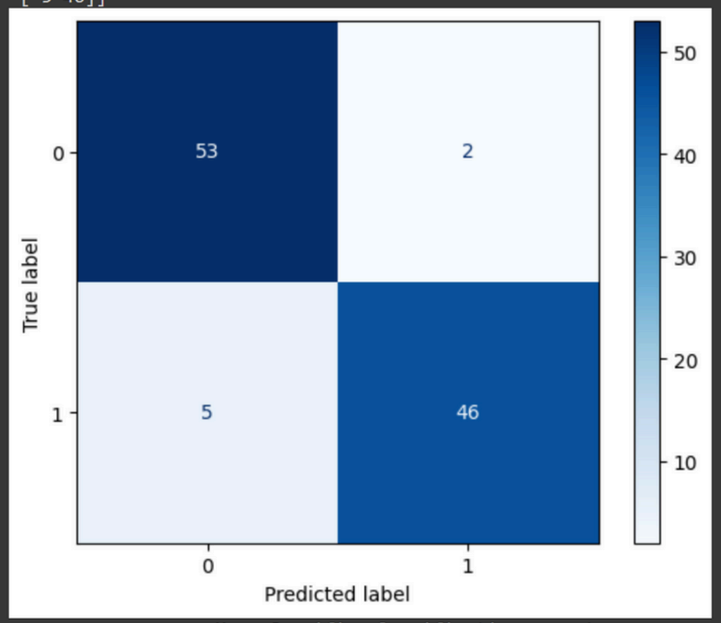

```
------------------ More Specific classification_report --------------------
Accuracy:- 0.9245283018867925
Precision: 0.9574468085106383
Recall: 0.8823529411764706
F1-Score:- 0.9183673469387754
Accuracy on Training data :  0.957345971563981
```

Models

Decision Tree

```python
# DecisionTree Classifier
from sklearn.tree import DecisionTreeClassifier
model_tree = DecisionTreeClassifier(max_depth=3,random_state=0)
model_tree.fit(X_train, y_train)
# tree best estimator
y_pred_clf = model_tree.predict(X_test)


# Confusion matrix
print("------------------ Confusion Matrix --------------------")
c_matrix = metrics.confusion_matrix(y_test, y_pred_clf)
print(c_matrix)

cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix)
cm_display.plot(cmap=plt.cm.Blues)
plt.show()


print("------------------ More Specific classification_report --------------------")

# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_pred_clf))
#Precision
print("Precision:",precision_score(y_test, y_pred_clf))
#Recall
print("Recall:", recall_score(y_test, y_pred_clf))
# F1 score
print("F1-Score:-", f1_score(y_test, y_pred_clf))


X_train_prediction = model_tree.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
print('Accuracy on Training data : ', training_data_accuracy)

results.loc[len(results)] = ['Decision Tree ', metrics.accuracy_score(y_test, y_pred_clf),precision_score(y_tes
```

```
------------------ Confusion Matrix --------------------
[[53  2]
 [ 5 46]]
```



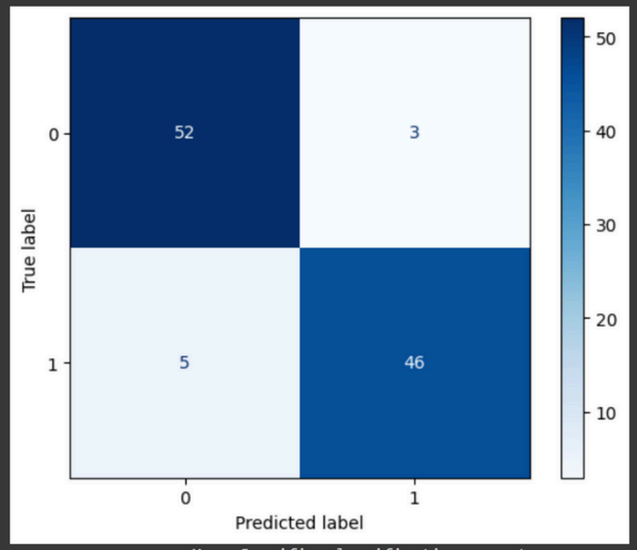

```
------------------ More Specific classification_report --------------------
Accuracy:- 0.9339622641509434
Precision: 0.9583333333333334
Recall: 0.9019607843137255
F1-Score:- 0.9292929292929293
Accuracy on Training data :  0.9549763033175356
```

Models

Linear SVM

```
from sklearn.svm import LinearSVC
classifier = clf = LinearSVC(C=1.0,random_state=0)
classifier.fit(X_train, y_train)
y_pred_SVM = classifier.predict(X_test)



# Confusion matrix
print("------------------ Confusion Matrix --------------------")
c_matrix = metrics.confusion_matrix(y_test, y_pred_SVM)
print(c_matrix)

cm_display = ConfusionMatrixDisplay(confusion_matrix=c_matrix)
cm_display.plot(cmap=plt.cm.Blues)
plt.show()

print("------------------ More Specific classification_report --------------------")

# Accuracy
print("Accuracy:-",metrics.accuracy_score(y_test, y_pred_SVM))
#Precision
print("Precision:",precision_score(y_test, y_pred_SVM))
#Recall
print("Recall:", recall_score(y_test, y_pred_SVM))
# F1 score
print("F1-Score:-", f1_score(y_test, y_pred_SVM))


X_train_prediction = classifier.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, y_train)
print('Accuracy on Training data : ', training_data_accuracy)

results.loc[len(results)] = ['Linear SVM ', metrics.accuracy_score(y_test, y_pred_SVM),precision_score(y_test, y_
```

```
------------------ Confusion Matrix --------------------
[[52  3]
 [ 5 46]]
/usr/local/lib/python3.10/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning:

Liblinear failed to converge, increase the number of iterations.
```




```
------------------ More Specific classification_report --------------------
Accuracy:- 0.9245283018867925
Precision: 0.9387755102040817
Recall: 0.9019607843137255
F1-Score:- 0.92
Accuracy on Training data :  0.966824644549763
```

Analysis

```
[ ]  results.sort_values(by="Accuracy", ascending=False)
```

	Model Name	Accuracy	Precision	Recall	F1-score
0	Random Forest	0.952830	1.000000	0.901961	0.948454
2	Decision Tree	0.933962	0.958333	0.901961	0.929293
1	Logistic Regression	0.924528	0.957447	0.882353	0.918367
3	Linear SVM	0.924528	0.938776	0.901961	0.920000

Analysis

Random Forest:

- Strengths:
  - Achieved the highest overall performance (accuracy, precision, recall, F1-score).
  - Robust to outliers and handles high dimensional data well.
- Weaknesses:
  - Complex to interpret
  - Requires more computational power to train compared to simpler models like Logistic Regression or Linear SVM.

Decision Tree:

- Strengths:
  - High accuracy and F1-score, similar to Random Forest.
  - Interpretable
- Weaknesses:
  - Prone to overfitting, especially on smaller datasets. May not perform well with unseen data.
  - Might be slightly less accurate than Random Forest in terms of precision and recall.

Analysis

Logistic Regression:

- Strengths:
  - Simpler to train and interpret compared to Random Forest and Decision Tree.
  - Requires less computational power to train.
- Weaknesses:
  - Lower overall performance (accuracy, precision, recall, F1-score) compared to Random Forest and Decision Tree.
  - Might miss more fraudulent transactions (lower recall) due to its simpler modeling approach.

Linear SVM:

- Strengths:
  - Similar strengths to Logistic Regression - simpler to train and interpret, requires less computational power.
- Weaknesses:
  - Similar weaknesses to Logistic Regression - lower overall performance compared to Random Forest and Decision Tree, might miss more fraudulent transactions.

Analysis

The most efficient model is :

Random Forest emerges as the strongest candidate for this fraud detection task based on several factors:

- **Highest Overall Performance:** It achieves the best accuracy, precision, recall, and F1-score among the evaluated models.
- **Balance between Precision and Recall:** It offers a good balance between identifying fraudulent transactions (high recall) and avoiding false positives (high precision).
- **Handling Complexities:** It can handle potentially complex relationships within the data due to its ensemble nature.