Project Exhibition - 1

Hybrid Machine Learning Algorithm for Credit Card Fraud Detection

Under the Guidance of Dr. Sandip Mal

Our Team

ANJALI KOLHATKAR

21BCE11171

SHEIKH MOHD SHOAIB

21BCE11346

SHUBHAM SINGH

21BCE10456

HARSHITA NIMJE

21BCE10524

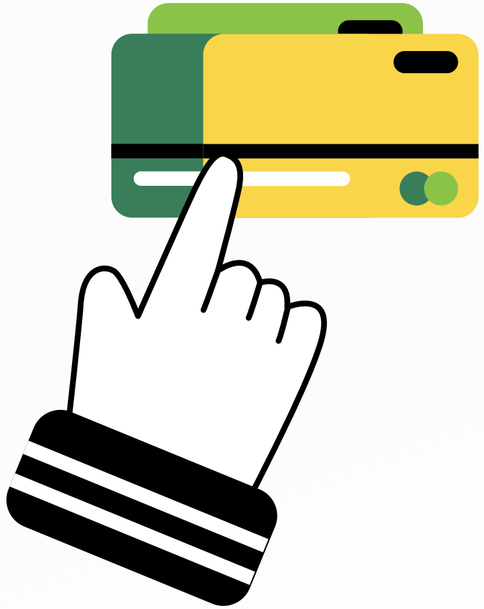
AJINKIYA FULLPATIL

21BCE11160

Content

What is Credit Card Fraud?

- Credit card fraud refers to the **unauthorized or deceptive use** of someone else's credit card information for financial gain.
- It encompasses various illegal activities in which a person's credit card details, such as the **card number, expiration date, and security code, are used by fraudsters** to make unauthorized transactions or gain access to funds. This can include making unauthorized purchases, withdrawing cash, or transferring funds, all without the cardholder's consent.
- Credit card fraud is a widespread problem that can result in financial losses for individuals and damage to the reputation of financial institutions.



Why Credit Card Detection is Necessary ?

- Losses related to credit card fraud will grow to **$43 billion** within five years and climb to **$408.5 billion** globally within the next decade, according to a recent Nilson Report — meaning that credit card fraud detection has become more important than ever.

- In 2021, credit card fraud ranked as the **second most common type of identity theft in the U.S.**

ML Algorithms For Fraud Detection

Credit card fraud detection depends on various machine learning algorithms to identify fraud transactions. Here are some commonly used ML algorithms for credit card fraud detection:

Logistic Regression:

A classic binary classification algorithm that's used to predict whether a transaction is fraudulent or not based on historical data.

Random Forest:

An ensemble learning method that combines multiple decision trees to improve accuracy. It's effective at handling imbalanced datasets common in fraud detection.

Support Vector Machine (SVM):

SVM is used to classify transactions into fraudulent and non-fraudulent categories. It can handle high-dimensional data effectively.

Convolution Neural Networks:

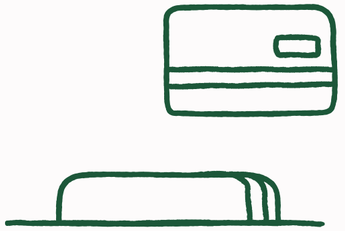
Deep learning models like Convolutional Neural Networks (CNN) are used for pattern recognition and fraud detection.

Naive Bayes:

Naive Bayes classifiers are probabilistic models that can estimate the likelihood of a transaction being fraudulent.

Gradient Boosting:

Gradient boosting is a machine learning technique that combines multiple weak models, like decision trees, to create a strong predictive model. It's an iterative process where each model corrects the mistakes made by the previous ones.



Accuracy of Different Models

Algorithm	Accuracy	
Decision Tree	94%	❌
SVM Model	95.99%	❌
Logistic Regression	97.2%	❌
Gradient Boosting	99.801%	❌
CNN	99.901%	✅
Random Forest	99.90%	✅

Hybrid Model For fraud Detection

Hybrid Model

Hybrid machine learning combines different approaches such as supervised and unsupervised learning, reinforcement learning, or deep learning to address complex problems. It can help to overcome the limitations of individual machine learning algorithms.

Random Forest + CNN

Hybrid Model

Hybrid Model

Random Forest Model

+

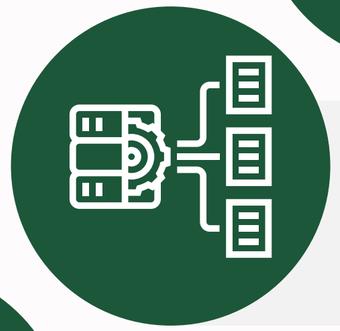
CNN Model

Hybrid architectures can be more interpretable. This is because normal machine learning algorithms are often easier to interpret than deep learning algorithms. By combining a deep learning model with a traditional machine learning algorithm, it is possible to create a hybrid architecture that is both accurate and interpretable.

Architecture

1st Step
Data Source

2nd Step
Data Analysis And Processing



3rd Step
Data Modelling

4th Step
Hybrid Model Making

5th Step
Evaluation And Results

Process Flow

01

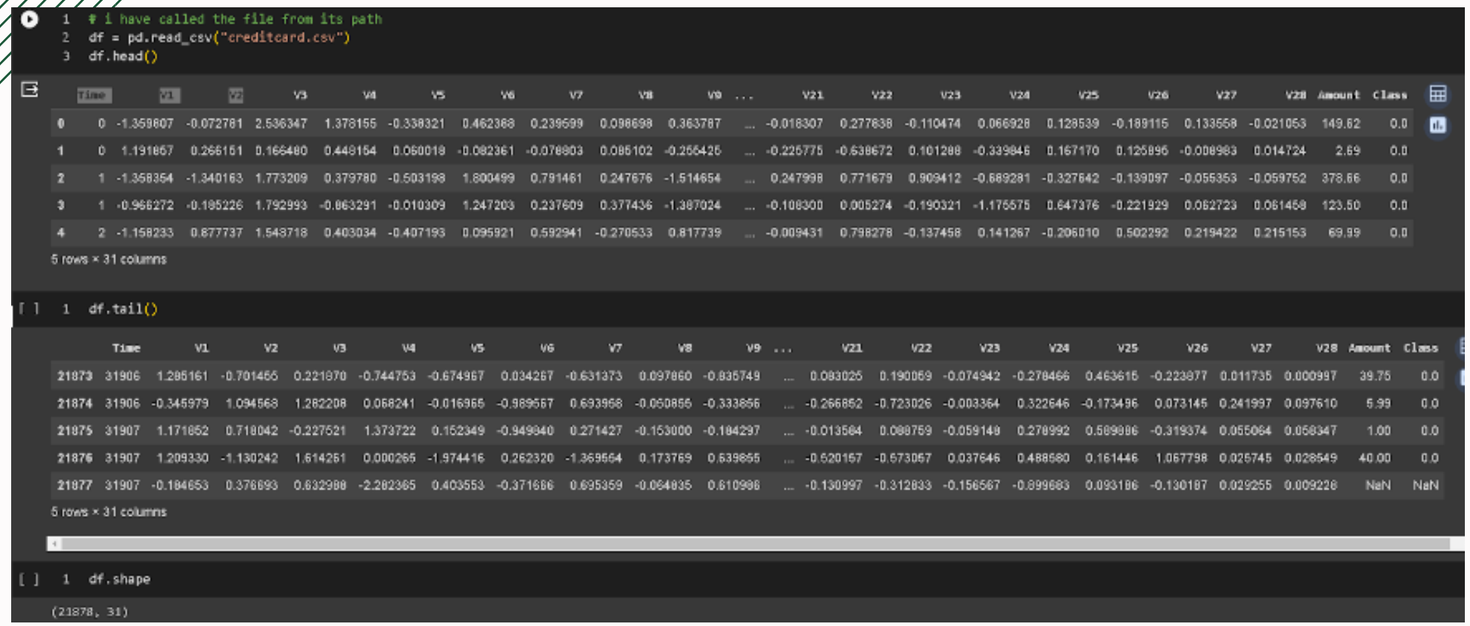
Data Source And Analytics

https://docs.google.com/document/d/1DVAPJuDUlO2clF4iwRvesunIJJM3mPhbUZBOE_vYiHK0/edit

It's important to check or analyse data so that you can remove any null value or unexpected values and errors. This is an important a step because the data we acquire could have these errors and it is important to remove it else the model will give errors. In this analyzation:

Data Analysis And Processing

- Checking the data set to know the form of data entries and the number of row and column:



```
# I have called the file from its path
df = pd.read_csv("creditcard.csv")
df.head()
```

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

5 rows × 31 columns

```
df.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
21873	31906	1.285161	-0.701455	0.221870	-0.744753	-0.674967	0.034267	-0.631373	0.097860	-0.835749	...	0.083025	0.190069	-0.074942	-0.270466	0.463615	-0.223877	0.011735	0.000997	39.75	0.0
21874	31906	-0.345979	1.094568	1.282208	0.068241	-0.016965	-0.989567	0.693968	-0.050855	-0.333856	...	-0.266852	-0.723026	-0.003364	0.322646	-0.173496	0.073145	0.241997	0.097610	5.99	0.0
21875	31907	1.171652	0.718042	-0.227521	1.373722	0.152349	-0.949940	0.271427	-0.153000	-0.184297	...	-0.013564	0.089759	-0.059149	0.270992	0.589986	-0.319374	0.055064	0.056347	1.00	0.0
21876	31907	1.209330	-1.130242	1.614251	0.000265	-1.974416	0.262320	-1.369564	0.173769	0.639855	...	-0.520157	-0.673067	0.037646	0.488580	0.161446	1.067798	0.026745	0.028549	40.00	0.0
21877	31907	-0.184653	0.376693	0.632998	-2.282365	0.403553	-0.371686	0.695369	-0.064035	0.610998	...	-0.130997	-0.312833	-0.156567	-0.899683	0.093186	-0.130187	0.029255	0.009228	NaN	NaN

5 rows × 31 columns

```
df.shape
```

```
(21878, 31)
```

Process Flow

Data Analysis
Processing

- Checking whether the entries are empty or null

```
# cleaning the dataset
df.isnull().sum()# this line of code tell how many null values are there in a column
```

```
# here we are removing the null values
df = df.dropna()
df.isnull().sum()

'''notes
the random forest classifier can be used for both classification and regression modeling
thus we have used it here well the special thing about the rfc model is that it doesnt require and kind of data cleaning
still i have clean the data becoz to show it doen in this way.
'''
```

```
'notes\nthe random forest classifier can be used for both classification and regression modeling\nthus we have used it here well the special thing about the rfc model is that it doesnt require and kind
 f data cleaning\nstill i have clean the data becoz to show it doen in this way.\n'
```

Process Flow

- Using describe function we will obtain the basic description about the data:

```
df.describe()#this line of code gives the basic details
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
count	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	21877.000000	...	21877.000000	21877.000000	21877.000000	21877.000000	21877
mean	16933.586689	-0.241062	0.222117	0.737919	0.262269	-0.175253	0.087517	-0.141050	0.019343	0.662077	...	-0.043034	-0.140148	-0.037820	0.010748	0
std	11000.194861	1.880944	1.618145	1.740987	1.461003	1.414899	1.322632	1.323942	1.383809	1.277302	...	0.849630	0.644928	0.612908	0.600788	0
min	0.000000	-30.552380	-40.978852	-31.103685	-5.172595	-32.092129	-23.496714	-26.548144	-41.484823	-7.175097	...	-20.262054	-8.887017	-26.751181	-2.836627	-7
25%	5495.000000	-0.980944	-0.350080	0.297985	-0.648258	-0.756737	-0.690693	-0.597986	-0.169177	-0.273410	...	-0.257011	-0.564594	-0.175535	-0.331512	-0
50%	10769.000000	-0.298324	0.206459	0.880765	0.222377	-0.210154	-0.176575	-0.068815	0.026059	0.515114	...	-0.109865	-0.111890	-0.040570	0.062135	0
75%	27778.000000	1.164879	0.858520	1.520392	1.132696	0.332763	0.481050	0.447518	0.281486	1.330402	...	0.059945	0.267695	0.074160	0.398021	0
max	31907.000000	1.960497	16.713389	4.101716	11.927512	34.099309	21.393069	34.303177	20.007208	10.392889	...	22.614889	5.805795	13.876221	3.695503	9

0 rows × 31 columns

- Getting the count of fraud count and non fraud count in the Class column:

```
# getting the total of all the non-fraud entries in the dtas
non_fraud = len(df[df.Class == 0])
print(non_fraud)
```

```
21411
```

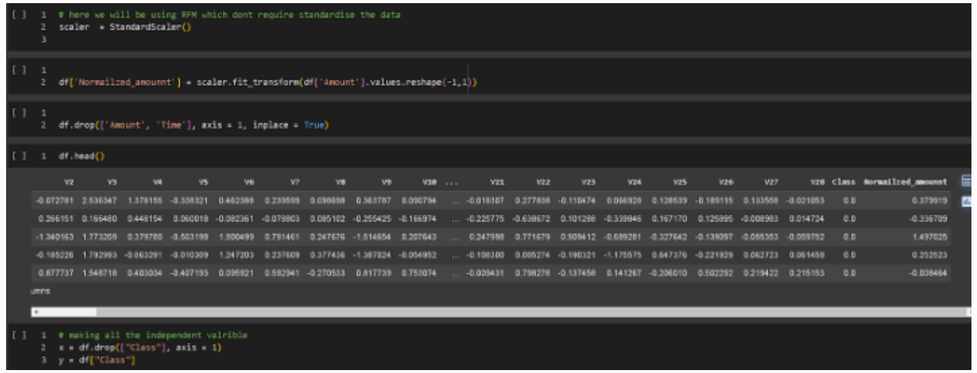
```
# getting the total fraud form the complete entries
fraud = len(df[df.Class == 1])
print(fraud)
```

```
86
```

Process Flow

Data Source

- Normalizing the values which are varied from the range other columns are present.



```
[ ] 1 # here we will be using RFM which dont require standardise the data
    2 scaler  = StandardScaler()
    3

[ ] 1
    2 df['Normalized_amount'] = scaler.fit_transform(df['Amount'].values.reshape(-1,1))

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

[ ] 1 df.head()
```

	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V21	V22	V23	V24	V25	V26	V27	V28	Class	Normalized_amount
	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0.0	0.379919
	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0.0	-0.336709
	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0.0	1.497025
	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0.0	0.252623
	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0.0	-0.039664

umns

```
[ ] 1 # making all the independent valrible
    2 x = df.drop(["Class"], axis = 1)
    3 y = df["Class"]
```

Process Flow

02 Data Modelling

- Data will be divided in dependable and independent parts namely “y” and “x” respectively. After this we will use these “x”(independent data) and “y”(dependent data) data.

```
# making all the independent valrible
x = df.drop(["Class"], axis = 1)
y = df["Class"]
```

```
from sklearn.model_selection import train_test_split
```

```
# making the trian and test data by dividing the data with the testsize of 0.3
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.3, random_state = 1)

```

Process Flow

- Here we will be making a hybrid model using the random forest method and CNN model, as these have the best accuracy when performed individually.

```
1  cnn_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_1 (Conv1D)           (None, 27, 32)            128

 max_pooling1d_1 (MaxPooli   (None, 13, 32)            0
 ng1D)

 flatten_1 (Flatten)         (None, 416)               0

 dense_2 (Dense)             (None, 128)               53376

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 53633 (209.50 kB)
Trainable params: 53633 (209.50 kB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Accuracy

```python
# Create the CNN model
cnn_model = Sequential()
# cnn_model = Sequential()
cnn_model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=(x_train.shape[
cnn_model.add(MaxPooling1D(pool_size=2))
cnn_model.add(Flatten())
cnn_model.add(Dense(128, activation='relu'))
cnn_model.add(Dense(1, activation='sigmoid'))
# Create the random forest model
rf_model = RandomForestClassifier(n_estimators=100)

# Train the CNN model
cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
cnn_model.fit(x_train, y_train, epochs=10)

# Extract the features from the CNN model
cnn_features = cnn_model.predict(x_test)

# Train the random forest model on the CNN features
rf_model.fit(cnn_features, y_test)

# Make predictions on the test set
y_pred = rf_model.predict(cnn_model.predict(x_test))
accuracy = np.sum(y_pred == y_test) / len(y_test)
print('Accuracy:', accuracy*100)
```

```
Epoch 1/10
479/479 [==============================] - 2s 3ms/step - loss: 0.0258 - accuracy: 0.9957
Epoch 2/10
479/479 [==============================] - 2s 3ms/step - loss: 0.0076 - accuracy: 0.9978
Epoch 3/10
479/479 [==============================] - 2s 5ms/step - loss: 0.0049 - accuracy: 0.9982
Epoch 4/10
479/479 [==============================] - 2s 4ms/step - loss: 0.0050 - accuracy: 0.9981
Epoch 5/10
479/479 [==============================] - 2s 4ms/step - loss: 0.0039 - accuracy: 0.9987
Epoch 6/10
479/479 [==============================] - 2s 4ms/step - loss: 0.0036 - accuracy: 0.9990
Epoch 7/10
479/479 [==============================] - 2s 3ms/step - loss: 0.0033 - accuracy: 0.9991
Epoch 8/10
479/479 [==============================] - 2s 3ms/step - loss: 0.0030 - accuracy: 0.9993
Epoch 9/10
479/479 [==============================] - 2s 4ms/step - loss: 0.0030 - accuracy: 0.9990
Epoch 10/10
479/479 [==============================] - 2s 5ms/step - loss: 0.0029 - accuracy: 0.9990
206/206 [==============================] - 1s 2ms/step
206/206 [==============================] - 0s 2ms/step
Accuracy: 99.96953077391835
```

99.969%

Accuracy of Hybrid Model

By utilizing random forest and gradient boosting algorithms for credit card fraud detection, we achieved an **accuracy rate of 99.969%.**

99.969%

Conclusion

In conclusion we can say that, the **hybrid model, with an accuracy of 99.97%,** outperforms other algorithms that had an accuracy of up to 99.908% at detecting credit card fraud.

We have combined two powerful algorithms, Random forest and Gradient Boosting Algorithm, which resulted as **hybrid model being the best solution for detecting credit card fraud.**

Moreover ,We demonstrated that this hybrid model isn't just suitable for one dataset; it's an excellent solution for this type of problem since it can manage the variety of data types commonly present in fraud detection.

Thank you


entry point
Double Bottom
Target
Neckline
Stop
Entry
Target
Stop
Exhibit 3: US economic growth expected
as of May 5, 2021
US QoQ annualized
GDP growth
Consensus
2021