CREDIT CARD FRAUD DETECTION

ADS PHASE 5

ABSTRACT

This paper is Phase 5 of Alok Mandal"Credit Card Fraud Detection" project, which he presented as a third-year Computer Science And Engineering student. Phase 5 is dubbed for "Clearly outline the problem statement, design thinking and phase development" and describe the dataset used pre-processing steps and model training process and explain the ML algorithm (KNN) , and evaluation metrics.

ALOK MANDAL
Applied Data Science

Problem Statement

The development of a model or system that can precisely identify and stop fraudulent transactions in real-time is the problem statement for credit card fraud detection using machine learning. This entails identifying authentic and fraudulent credit card transactions using multiple features linked to each transaction as well as historical transaction data. The following are the main goals:

1**. Classification**: Create a prediction model that can determine whether a credit card transaction is fraudulent or lawful. In binary classification, every transaction is given a label by the model.

2. **Real-time Detection**: As fraudulent transactions must be found and prevented before they are accepted, make sure the model is capable of making prompt and accurate choices in real-time.

3. **High Accuracy**: To prevent upsetting consumers, attain a high degree of accuracy in detecting fraudulent transactions while reducing false positives, or normal transactions that are mistakenly reported as fraudulent.

4**. Scalability**: Since credit card firms conduct millions of transactions per day, make sure the system can manage a high volume of transactions.

5**. Adaptability**: To be resilient to new threats, the system must be able to adjust to new and changing fraud patterns and approaches.

6**. Data Security**: In a financial context, it is imperative to safeguard sensitive transaction data and client information. Respect privacy laws and make sure your data is secure and encrypted.

7. **Efficiency**: Create a system that is as efficient as possible, reducing the amount of time and computing power needed for fraud detection, particularly for real-time processing.

Machine learning methods like deep learning (e.g., neural networks) and supervised learning (e.g., logistic regression, decision trees, random forests, support vector machines) are frequently employed to solve this issue. The transaction outcome (fraudulent or valid) and other parameters, including transaction amount, location, and time, are among the many features that these models are trained on using previous transaction data. The goal of the model is to forecast new, unseen transactions by drawing generalisations from the past data.

Overall, the goal of credit card fraud detection using machine learning is to protect customers and financial institutions from financial losses due to fraudulent transactions while ensuring a smooth and secure payment experience for legitimate cardholders.

Design Thinking Process

Machine learning can be used in conjunction with design thinking, a user-centered, creative approach to problem-solving, to create a credit card fraud detection system. It can be used in this context in a sequence of repetitive phases as follows:

1. **Empathize**:

Recognise the demands and concerns that credit card companies and financial institutions have when addressing fraud.

- Perform user research, surveys, and interviews to learn more about the difficulties and experiences of users.

2**. Define**:

- Clearly state the issue: "How might we effectively detect and prevent credit card fraud while minimising inconvenience to legitimate cardholders?"

. Determine the needs of the main players, such as clients, lenders, and authorities in charge of regulations.

3**. Ideate**:

- List possible fixes and strategies for detecting credit card fraud.

- Promote diversity of thought and inventiveness in a cross-functional team.

- Take into account both machine learning-based solutions and conventional rule-based fraud detection techniques.

4**. Prototype**:

- Construct mockups or low-fidelity prototypes of the fraud detection system to see how it might operate.

- Try out different machine learning algorithms and data pretreatment methods to find the ones that show the most promise.

5. **Test:**

- Use real or simulated data to test the prototypes and see how well they detect fraud while reducing false positives.

 - Get input from relevant parties, such as fraud analysts and customers.

 - Make revisions to the prototypes in light of user feedback.

6**. Iterate:**

- Make adjustments to the prototypes in light of the testing's findings.

 - If fresh ideas are needed, go back and revisit the ideation stage.

 - Keep trying and refining until a workable solution is found.

7**. Implement**:

- Building on the validated and improved prototype, create a credit card fraud detection system that is suitable for production.

 - Include data pipelines, real-time processing capabilities, and machine learning models.

8**. Evaluate**:

- Constantly track the system's functionality in an actual environment.

 - Track important performance metrics, like processing speed, false positive rate, and accuracy in detecting fraud.

- Get input from stakeholders and users to make continuous changes.

9**. Iterate and Improve**:

-The fraud detection system will be continuously improved with the help of user input and data-driven insights.

- To remain effective, adjust to new technologies and changing fraud tendencies.

It's critical to include diverse teams in the design thinking process, such as data scientists, fraud analysts, software engineers, user experience (UX) designers, and financial services domain experts. In order to guarantee that the credit card fraud detection system not only successfully prevents fraud but also offers a great experience for legitimate cardholders and financial institutions, collaboration and empathy for the end users are essential.

Phases of Development

There are usually multiple stages involved in the creation of a machine learning-based credit card fraud detection system. Although the stages are arranged logically, they may be repeated and there may be overlap in the work done in each phase. The major stages of development are as follows:

1. **Data Collection and Preparation**:

- *Data Collection*: Compile transaction history for credit cards, including details about the amount, time, location, and cardholder.

- *Data Cleaning*: Remove missing numbers, handle outliers, and preprocess the data to make it consistent.

- *Feature Engineering*: To increase the model's capacity to identify fraud, add new features or modify current ones.

2. **Exploratory Data Analysis (EDA)**:

- Conduct data analysis and visualisation to learn more about the data and spot any patterns or abnormalities that might point to fraud.

3. **Data Splitting**:

- To train and assess the machine learning model, divide the dataset into test, validation, and training sets.

4**. Model Selection and Training**:

- Select the right machine learning techniques (e.g., logistic regression, decision trees, random forests, support vector machines, neural networks) to detect credit card fraud.

- Utilising the training set of data, train and optimise the chosen model. One can use methods such as hyperparameter tweaking.

5. **Model Evaluation**:

- Evaluate the model's effectiveness by utilising the validation dataset. The receiver operating characteristic (ROC) curve, accuracy, precision, recall, F1-score, and other metrics are frequently used in evaluations.

- Based on the evaluation results, optimise the model while taking the trade-off between false positives and false negatives into account.

6. **Ensemble Methods and Anomaly Detection (Optional)**:

- Take into account utilising group techniques, like bagging or stacking, to enhance model performance.

- Examine anomaly detection methods that can be used in addition to conventional classification strategies.

7. **Real-time Processing**:

- Use real-time processing capabilities to quickly decide how to handle incoming credit card transactions.

8. **Model Deployment**:

- Install the trained model in a live credit card processing environment to enable real-time processing.

- Assure the deployment's scalability, dependability, and security.

9. **Monitoring and Alerting**:

- Keep an eye on the model's functionality and alerting mechanisms to spot problems or evolving fraud trends.

- Put in place procedures for routinely retraining and updating the model so that it can adjust to changing fraud strategies.

10**. Post-Deployment Evaluation**:

- Assess the model's effectiveness over time in a production environment.

- To find areas for improvement, get input from stakeholders and fraud analysts.

11. **Regulatory Compliance and Security**:

- Assure adherence to data privacy laws (such as GDPR and PCI DSS) and put robust data security procedures in place to safeguard sensitive client data.

12**. Documentation and Reporting**:

- Maintain thorough documentation of the model, data, and processes for transparency and future reference.

- Generate regular reports on fraud detection performance and system health.

13**. Ongoing Improvement**:

- Constantly enhance the model by adding fresh information and understandings.

- Keep abreast of new developments in the fraud landscape and modify the system as necessary.

In order for a credit card fraud detection system to continue to be effective in detecting and stopping fraudulent transactions while reducing false positives, it must be continuously monitored, evaluated, and adjusted.

Credit Card Fraud Detection using

Introduction

In [1]:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
sns.set()

%matplotlib inline
```

```
(284807, 31)
```

Out[1]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0. 0	−1. 359807	−0. 072781	2. 536347	1. 378155	−0. 338321	0. 462388	0. 239599	0. 098698
1	0. 0	1. 191857	0. 266151	0. 166480	0. 448154	0. 060018	−0. 082361	−0. 078803	0. 085102
2	1. 0	−1. 358354	−1. 340163	1. 773209	0. 379780	−0. 503198	1. 800499	0. 791461	0. 247676
3	1. 0	−0. 966272	−0. 185226	1. 792993	−0. 863291	−0. 010309	1. 247203	0. 237609	0. 377436
4	2. 0	−1. 158233	0. 877737	1. 548718	0. 403034	−0. 407193	0. 095921	0. 592941	−0. 270533

```
5 rows × 31 columns
```

In [2]:

```
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
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

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

Out[3]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e-
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e-
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e-
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e-

8 rows × 31 columns

How many are fraud and how many are not fraud ?

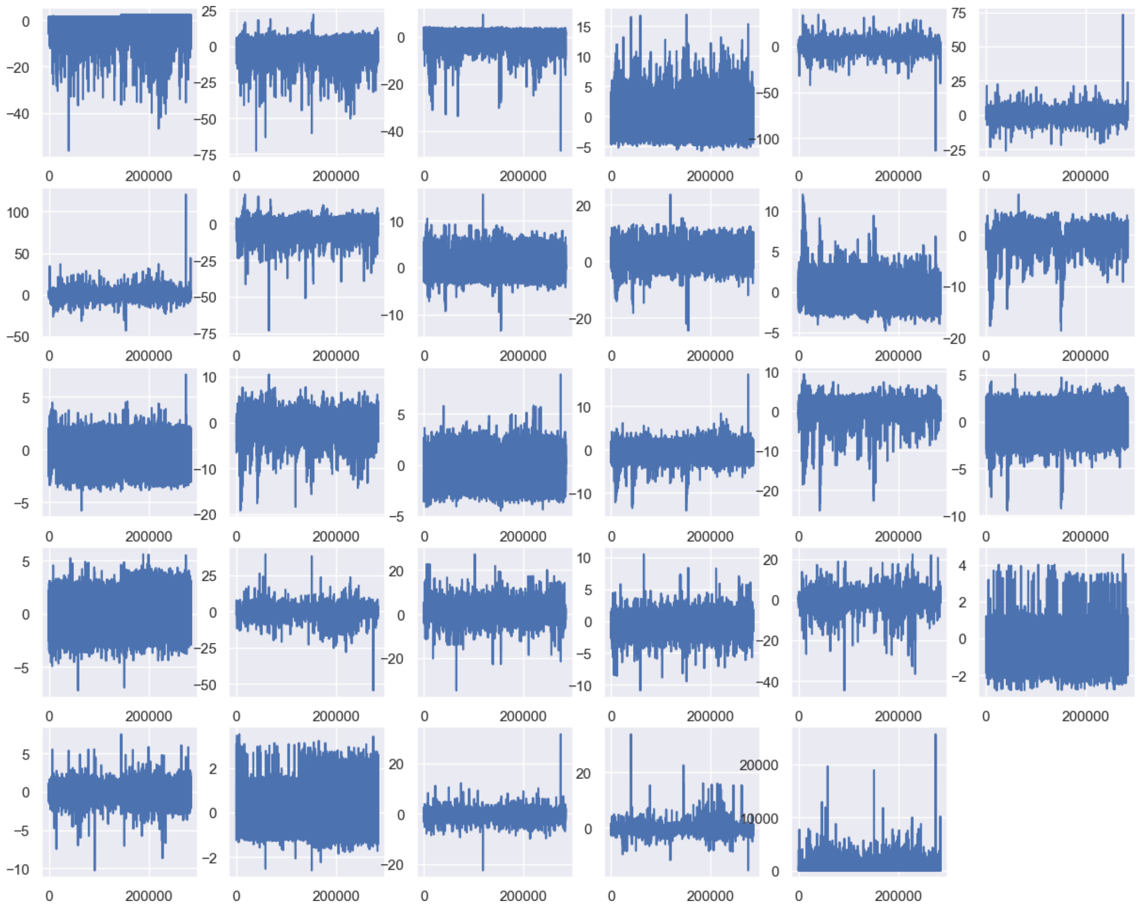
In [4]:
```
class_names = {0:'Not Fraud', 1:'Fraud'}

print(df.Class.value_counts().rename(index = class_names))
```

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

Plotting the variables using subplots

In [5]:

```
fig = plt.figure(figsize = (15, 12))

plt.subplot(5, 6, 1) ; plt.plot(df.V1) ; plt.subplot(5, 6, 15) ; plt.plot(d
plt.subplot(5, 6, 2) ; plt.plot(df.V2) ; plt.subplot(5, 6, 16) ; plt.plot(d
plt.subplot(5, 6, 3) ; plt.plot(df.V3) ; plt.subplot(5, 6, 17) ; plt.plot(d
plt.subplot(5, 6, 4) ; plt.plot(df.V4) ; plt.subplot(5, 6, 18) ; plt.plot(d
plt.subplot(5, 6, 5) ; plt.plot(df.V5) ; plt.subplot(5, 6, 19) ; plt.plot(d
plt.subplot(5, 6, 6) ; plt.plot(df.V6) ; plt.subplot(5, 6, 20) ; plt.plot(d
plt.subplot(5, 6, 7) ; plt.plot(df.V7) ; plt.subplot(5, 6, 21) ; plt.plot(d
plt.subplot(5, 6, 8) ; plt.plot(df.V8) ; plt.subplot(5, 6, 22) ; plt.plot(d
plt.subplot(5, 6, 9) ; plt.plot(df.V9) ; plt.subplot(5, 6, 23) ; plt.plot(d
plt.subplot(5, 6, 10) ; plt.plot(df.V10) ; plt.subplot(5, 6, 24) ; plt.plot
plt.subplot(5, 6, 11) ; plt.plot(df.V11) ; plt.subplot(5, 6, 25) ; plt.plot
plt.subplot(5, 6, 12) ; plt.plot(df.V12) ; plt.subplot(5, 6, 26) ; plt.plot
plt.subplot(5, 6, 13) ; plt.plot(df.V13) ; plt.subplot(5, 6, 27) ; plt.plot
plt.subplot(5, 6, 14) ; plt.plot(df.V14) ; plt.subplot(5, 6, 28) ; plt.plot
plt.subplot(5, 6, 29) ; plt.plot(df.Amount)
plt.show()
```



In [6]:

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

In [7]:

```
feature_names = df.iloc[:, 1:30].columnstarget =
df.iloc[:1, 30: ].columns

print(feature_names)
print(target)
```

```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V2
1',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
Index(['Class'], dtype='object')
```

In [8]:

```
data_features = df[feature_names]
data_target = df[target]
```

In [9]:

```
X_train, X_test, y_train, y_test = train_test_split(data_features, data_targprint("Length of X_train is:
{X_train}".format(X_train = len(X_train)))

print("Length of X_test is: {X_test}".format(X_test = len(X_test)))

print("Length of y_train is: {y_train}".format(y_train = len(y_train)))print("Length of y_test is:
```

```
Length of X_train is: 199364
Length of X_test is: 85443
Length of y_train is: 199364
Length of y_test is: 85443
```

In [10]:

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix
```

In [11]:

```
model = LogisticRegression()

model.fit(X_train, y_train.values.ravel())
```

```
C:\Users\DSCET\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sci
kit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession (https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression)
  n_iter_i = _check_optimize_result(
```

Out[11]:

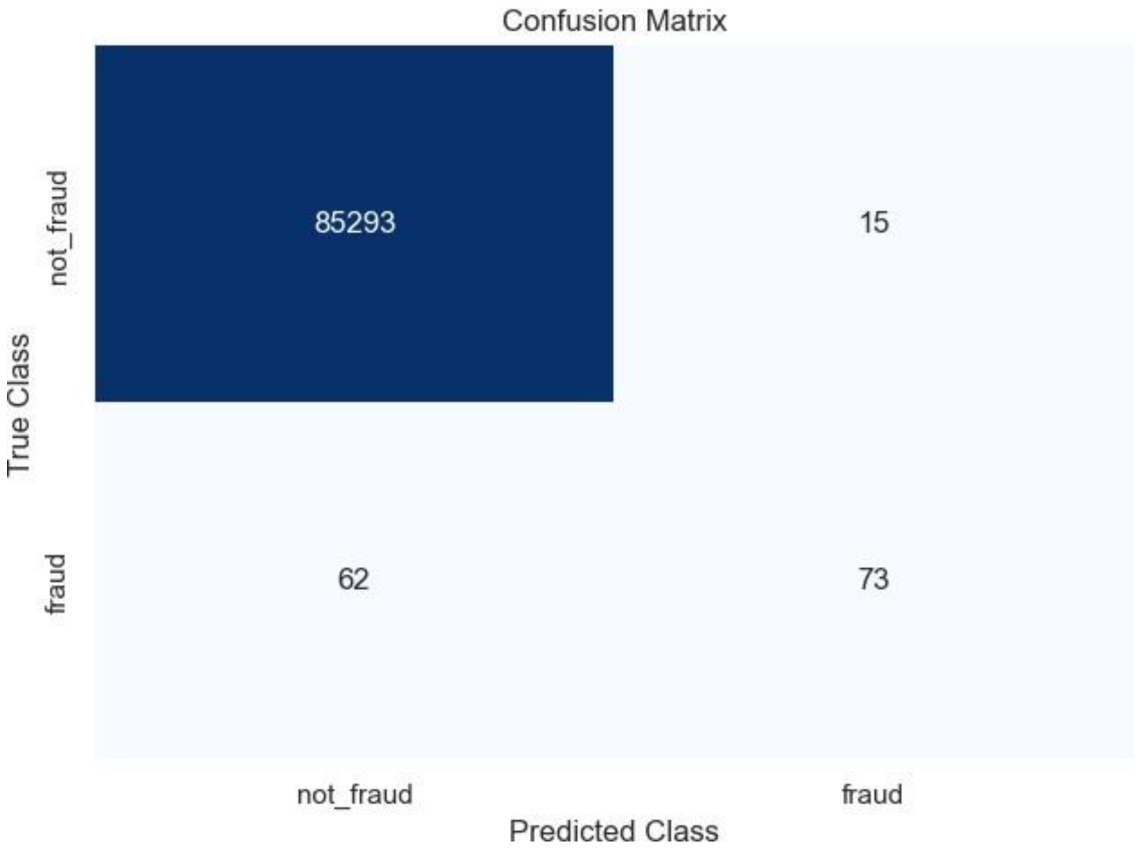
```
▾LogisticRegression

LogisticRegression()
```

In [12]:

```
pred = model.predict(X_test)
```

In [13]:

```
class_names = ['not_fraud', 'fraud']

matrix = confusion_matrix(y_test, pred)

# Create pandas dataframe

dataframe = pd.DataFrame(matrix, index=class_names, columns=class_names)

# Create heatmap

sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues", fmt = 'g')plt.title("Confusion
```



Evaluation

For a financial institution dealing with identifying fraud, Sensitivity and F1 - Score might be more important metrics. F1- Score reprsents a more balanced result as it is the harmonic mean between Precision and Recall. Sensitivity is more important in the sense that we are more interested in identifying fraud than than identifying legitimate customers. (Assumption)

In [14]:

```
from sklearn.metrics import f1_score, recall_scoref1_score =
round(f1_score(y_test, pred), 2)

recall_score = round(recall_score(y_test, pred), 2)

print("Sensitivity/Recall for Logistic Regression Model 1 : {recall_score}".print("F1 Score for Logistic
```

```
Sensitivity/Recall for Logistic Regression Model 1 : 0.54
F1 Score for Logistic Regression Model 1 : 0.65
```

to be continued ....

In [ ]:

KNN(K-Nearest Neighbours)

K-Nearest Neighbours (KNN) is a simple and intuitive machine learning algorithm used for both classification and regression tasks. It's a non-parametric, instance-based learning algorithm, which means it doesn't make any underlying assumptions about the data distribution. Instead, it makes predictions based on the similarity between the data points. Here's a detailed explanation of the K-Nearest Neighbours algorithm:

Algorithm Overview:

1**. Training Phase**:

- Since KNN doesn't learn any parameters from the data, it doesn't engage in any explicit "training" during this phase; instead, it just stores the full dataset in memory.

2. **Prediction Phase**:

- KNN finds its K-nearest neighbours from the training dataset when a new data point is submitted for prediction. Based on a selected distance metric—typically the Euclidean distance—these neighbours are identified.

3. **Classification (KNN for Classification)**:

- In classification tasks, KNN uses the majority class among its K-nearest neighbours to determine the class label to be assigned to a new data point. A majority vote system is employed.

- For instance, the new data point is categorised as class A if K=5 and three of the five closest neighbours are in class A and two are in class B.

4. **Regression (KNN for Regression)**:

- In regression tasks, KNN averages the target values (or other aggregation functions, such as median) of its K-nearest neighbours to forecast a continuous value for the incoming data point.

- The average of these values, or 20, is the prediction for the new data point if, for example, K=5 and the goal values of the five nearest neighbours are [10, 15, 20, 25, 30].

Parameters in KNN:

- *K*: The quantity of closest neighbours to take into account. Selecting a suitable value for K is essential. The model may become more susceptible to noise if K is smaller, and it may become unduly biassed if K is greater.

- *Distance Metric*: The measurement of distance that is employed to determine how similar two data points are. The most widely used measure is Euclidean distance, however depending on the type of data, alternative metrics like Manhattan distance, cosine similarity, or custom distance functions may be utilised.

Key Considerations:

- *Scaling*: Since KNN is sensitive to feature scale, it is imperative to scale the dataset's features. Larger scale features may predominate in calculations of distance.

- *Outliers*: When K is small, outliers can have a major impact on KNN's predictions. It can be required to handle outliers or perform data preparation.

- *Choice of K*: To determine the ideal value for a given problem, one should use techniques like as cross-validation to fine-tune the hyperparameter.

- *Curse by Dimensionality*: As the feature space's dimensionality rises, KNN performance may suffer. This phenomenon is referred to as the "curse of dimensionality." Significant neighbours are hard to locate in high-dimensional areas because of the sparse nature of the data in these regions.

Pros:

- Easy to comprehend and put into practise.
- Effective in problems involving both regression and classification.
- Able to adjust to non-linear decision limits.
- There is no training period because the model learns to memorise the data.

Cons:

- Because it needs to calculate the distances to every data point, it can be computationally expensive for huge datasets.
- Dependent on the distance metric and K selection.

Vulnerable to the dimensionality curse in spaces with high dimensions.

- May be biased towards the dominant class, hence it is not appropriate for datasets that are unbalanced.

When working with small to medium-sized datasets and when interpretability and simplicity are more important than computing efficiency, KNN is a useful approach. It is frequently used in conjunction with other algorithms and techniques for enhanced performance, and it can function as a baseline model for a variety of machine learning applications.

When evaluating the performance of a K-Nearest Neighbors (KNN) algorithm, you can use a variety of evaluation metrics, depending on the nature of the problem (classification or regression) and the specific goals of your analysis. Here are the most common evaluation metrics for KNN:

KNN for Classification:

1. Accuracy: Accuracy is a simple and commonly used metric for classification tasks. It measures the percentage of correctly classified instances in the dataset.

Accuracy=Total Number of Predictions\Number of Correct Predictions

2. Precision: Precision is the ratio of correctly predicted positive instances to the total instances predicted as positive. It's useful when the cost of false positives is high.

Precision=True Positives\(True Positives + False Positives)

3. Recall (Sensitivity): Recall measures the ability of the model to identify all relevant instances. It's the ratio of correctly predicted positive instances to the total positive instances in the dataset.

Recall=True Positives\(True Positives + False Negatives)

4. F1-Score: The F1-Score is the harmonic mean of precision and recall. It provides a balance between precision and recall.

F1-Score=2·Precision·Recall \(Precision + Recall)

5. Confusion Matrix: A confusion matrix provides a more detailed view of the model's performance. It shows the number of true positives, true negatives, false positives, and false negatives.

KNN for Regression:

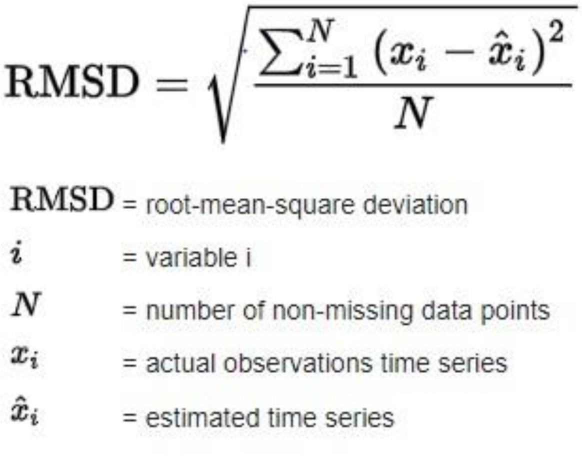
1. Mean Absolute Error (MAE): MAE measures the average absolute difference between the predicted and actual values. It's easy to understand but not robust to outliers.

$$\mathrm{MAE} = \sum_{i=1}^{n} |\ \mathrm{yi} - y^{i}\ |$$

2. Mean Squared Error (MSE): MSE calculates the average of the squared differences between the predicted and actual values. It gives higher weight to larger errors and is sensitive to outliers.

$$\mathrm{MSE{=}1n{-}2n{\sum}i{=}1(yi} - y^{i})^{2}$$

3. Root Mean Squared Error (RMSE): RMSE is the square root of the MSE and is preferred when you want the error metric to be in the same unit as the target variable.



$$\mathrm{RMSD} = \sqrt{\frac{\sum_{i=1}^{N}\left(x_i - \hat{x}_i\right)^2}{N}}$$

RMSD	= root-mean-square deviation
i	= variable i
N	= number of non-missing data points
x_i	= actual observations time series
$\hat{x}_i$	= estimated time series

4. R-squared (R^2): R-squared measures the proportion of the variance in the target variable that is explained by the model. It ranges from 0 to 1, with higher values indicating a better fit.

$$\text{R-Squared} = 1-\left(\frac{SSR}{SST}\right)=1-\frac{\sum_{i=1}^{n}(yi-\widehat{yi})^2}{\sum_{i=1}^{n}(yi-\overline{yi})^2}$$

Where MSE(model) is the mean squared error of the model's predictions, and MSE(mean) is the mean squared error of the mean of the target variable.

For both classification and regression tasks, it's essential to choose evaluation metrics that align with the specific objectives and requirements of your project. The choice of metric may depend on factors such as the class distribution, the cost of false positives and false negatives, and the nature of the target variable.

Usage/Example Algorithm:

```
# Naive Bayes Classification
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns;

#Importing the dataset
dataset = pd.read_csv('creditcard.csv')
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, 30]

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3,
random_state = 2)


# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting K-NN to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
```

```
classifier.fit(X_train, y_train)

# Predicting the Training set results
y_pred = classifier.predict(X_train.values)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_train, y_pred)

#Heat map of a confusion matrix
import seaborn as sns
sns.heatmap(cm,fmt=".0f",xticklabels=['CreditcardFraud_No','CreditcardFraud_Yes'],yticklabels=['CreditcardFraud_No','CreditcardFraud_Yes'],annot=True)
#sns.heatmap(cm,fmt=".0f",annot=True)

#Calculating Performance Metrics for Training Set
FP = cm.sum(axis=0) - np.diag(cm)
FN = cm.sum(axis=1) - np.diag(cm)
TP = np.diag(cm)
TN = cm.sum() - (FP + FN + TP)
FP = FP.astype(float)
FN = FN.astype(float)
TP = TP.astype(float)
TN = TN.astype(float)
# Sensitivity, hit rate, recall, or true positive rate
TPR = TP/(TP+FN)
print("Recall",TPR)
# Specificity or true negative rate
TNR = TN/(TN+FP)
print("Specificity",TNR)
# Precision or positive predictive value
PPV = TP/(TP+FP)
print("Precision",PPV)
# Negative predictive value
NPV = TN/(TN+FN)
print("Negative Predictive Value",NPV)
# Fall out or false positive rate
FPR = FP/(FP+TN)
print("False Positive Rate",FPR)
# False negative rate
FNR = FN/(TP+FN)
print("False Negative Rate",FNR)
# False discovery rate
FDR = FP/(TP+FP)
print("False Discovery Rate",FDR)
# Overall accuracy for each class
ACC = (TP+TN)/(TP+FP+FN+TN)
print("Accuracry",ACC)
```

```
# Fitting K-NN to the Testing set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p =
2)
classifier.fit(X_test, y_test)

# Predicting the Testing set results
y_pred1 = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_pred1)

#create an empty data frame that we have to predict
variety=pd.DataFrame()
variety['Time']=[0]
variety['V1']=[-1.3598071336738]
variety['V2']=[-0.0727811733098497]
variety['V3']=[2.53634673796914]
variety['V4']=[1.37815522427443]
variety['V5']=[-0.338320769942518]
variety['V6']=[0.462387777762292]
variety['V7']=[0.239598554061257]
variety['V8']=[0.0986979012610507]
variety['V9']=[0.363786969611213]
variety['V10']=[0.0907941719789316]
variety['V11']=[-0.551599533260813]
variety['V12']=[-0.617800855762348]
variety['V13']=[-0.991389847235408]
variety['V14']=[-0.311169353699879]
variety['V15']=[1.46817697209427]
variety['V16']=[-0.470400525259478]
variety['V17']=[0.207971241929242]
variety['V18']=[0.0257905801985591]
variety['V19']=[0.403992960255733]
variety['V20']=[0.251412098239705]
variety['V21']=[-0.018306777944153]
variety['V22']=[0.277837575558899]
variety['V23']=[-0.110473910188767]
variety['V24']=[0.0669280749146731]
variety['V25']=[0.128539358273528]
variety['V26']=[-0.189114843888824]
variety['V27']=[0.133558376740387]
variety['V28']=[-0.0210530534538215]
variety['Amount']=[149.62]
print(variety)
```

```
y_pred1=classifier.predict(variety.values)
print("Did this transcation is Fraud:")
print(y_pred1)
```