Assignment No. 2

Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle

Step 1: Download the dataset from Kaggle

1. **Dataset link:** Email Spam Classification Dataset
2. Go to the Kaggle dataset page.
3. Download the file emails.csv.
4. Place the dataset in the same folder where you will run your Jupyter notebook. If the dataset is in another location, make sure to update the path in the code where pd.read_csv() is used.

Step 2: Open Jupyter Notebook

1. **Open Jupyter Notebook:**
    - Launch Jupyter Notebook (via **Anaconda** or command prompt with jupyter notebook).
    - Navigate to the directory where you have saved the emails.csv file.
    - Click **New** -> **Python 3** to create a new Python notebook.

Step 3: Import required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
```

- **Explanation**: The code imports necessary libraries:
    - pandas: for data manipulation and analysis.
    - numpy: for numerical operations.
    - train_test_split: to split the data into training and testing sets.

- SVC: to use Support Vector Classifier for classification.
- KNeighborsClassifier: for the K-Nearest Neighbors (KNN) classifier.
- metrics: to calculate accuracy and create a confusion matrix.

Step 4: Load and inspect the dataset

```
df = pd.read_csv('emails.csv')
df
```

- **Explanation**: The dataset is loaded using pd.read_csv() and stored in a DataFrame df. Running df displays the first few rows of the dataset for inspection.

```
df.shape
```

- **Explanation**: This displays the shape of the dataset (number of rows and columns).

Step 5: Check for missing values

```
df.isnull().any()
```

- **Explanation**: This checks for any missing values in the dataset. It returns True for columns that contain missing values.

Step 6: Drop the 'Email No.' column

```
df.drop(columns='Email No.', inplace=True)
df
```

- **Explanation**: Since the 'Email No.' column is irrelevant for classification, it's removed using df.drop().

Step 7: Check column names and unique values in 'Prediction' column

```
df.columns
```

- **Explanation**: This displays the column names in the dataset.

```
df.Prediction.unique()
```

- **Explanation**: This retrieves the unique values in the 'Prediction' column. It should contain two values: 0 (Not spam) and 1 (Spam).

Step 8: Replace 0/1 values in 'Prediction' with 'Not spam' and 'Spam'

```
df['Prediction'] = df['Prediction'].replace({0:'Not spam', 1:'Spam'})

df
```

- **Explanation**: The 'Prediction' column is updated to replace 0 with 'Not spam' and 1 with 'Spam' for better readability.

Step 9: Prepare features (X) and target (Y)

```
X = df.drop(columns='Prediction', axis=1)

Y = df['Prediction']
```

- **Explanation**:
  - X contains the features (all columns except 'Prediction').
  - Y contains the target (the 'Prediction' column), which we aim to classify.

Step 10: Split the data into training and testing sets

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=1)
```

- **Explanation**:
  - train_test_split() splits the data into training (80%) and testing (20%) sets.
  - random_state=1 ensures the split is reproducible.

Step 11: KNN Classifier

```
KN = KNeighborsClassifier

knn = KN(n_neighbors=7)

knn.fit(x_train, y_train)

y_pred = knn.predict(x_test)
```

- **Explanation**:
  - The K-Nearest Neighbors classifier is initialized with n_neighbors=7, meaning it will consider 7 nearest neighbors for classification.
  - The classifier is trained on x_train and y_train.
  - The model predicts labels for the test set (x_test), and the predictions are stored in y_pred.

```
print("Prediction: \n")

print(y_pred)
```

- **Explanation**: This prints the predicted labels for the test set.

Step 12: Evaluate KNN accuracy

```
M = metrics.accuracy_score(y_test, y_pred)

print("KNN accuracy: ", M)
```

- **Explanation**: The accuracy of the KNN model is calculated using accuracy_score() and printed. This shows how well the KNN model performed in predicting whether an email is spam or not.

Step 13: SVM Classifier

```
model = SVC(C=1)

model.fit(x_train, y_train)

y_pred = model.predict(x_test)
```

- **Explanation**:
  - The SVM classifier is initialized with a regularization parameter C=1, which controls the trade-off between maximizing the margin and minimizing the classification error.
  - The model is trained on x_train and y_train.
  - Predictions for the test set (x_test) are made, and results are stored in y_pred.

Step 14: Evaluate SVM accuracy and confusion matrix

```
kc = metrics.confusion_matrix(y_test, y_pred)

print("SVM accuracy: ", kc)
```

- **Explanation**: The confusion matrix is generated using confusion_matrix() and printed. The confusion matrix provides a detailed breakdown of the model's performance:
  - **True Negatives (TN)**: Emails correctly classified as "Not spam."
  - **False Positives (FP)**: Emails incorrectly classified as "Spam."
  - **False Negatives (FN)**: Emails incorrectly classified as "Not spam."
  - **True Positives (TP)**: Emails correctly classified as "Spam."