**Assignment No.: ML 2**

**Title:** 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.

**Theory:**

Email Spam Classification using K-Nearest Neighbors and Support Vector Machine. To classify emails as spam or not spam using binary classification, we can employ two popular machine learning algorithms: K-Nearest Neighbors (KNN) and Support Vector Machine (SVM). Here's how to implement and analyze their performance using the Email Spam Classification Dataset CSV on Kaggle:

**1) Data Preprocessing**

Load the emails.csv dataset from Kaggle into a pandas DataFrame.

Split the data into features (email text) and labels (spam or not spam).

Preprocess the email text by removing stopwords, punctuation, and converting to lowercase.

Vectorize the preprocessed text using techniques like CountVectorizer or TfidfVectorizer.

Split the data into training and testing sets.

**2) K-Nearest Neighbors (KNN) Classification**

KNN algorithm :

Step 1: Choose the number of K neighbors

K is a hyperparameter that specifies the number of nearest neighbors to consider when making a prediction.

Step 2: Calculate the distance

The algorithm calculates the distance between the new data point and all points in the training data. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.

Step 3: Find the K nearest neighbors

The algorithm selects the K data points with the shortest distance to the new data point.

Step 4: Make a prediction

For classification, the algorithm assigns the class label that is most prevalent among the K neighbors to the new data point.

It is important to preprocess the data (e.g., scaling features) and tune the hyperparameters (e.g., K value) to improve the performance of the KNN algorithm.

KNN can be computationally expensive, especially with large datasets, as it calculates distances to all data points during prediction. It is also sensitive to the choice of the number of neighbors (K) and the distance metric. Additionally, it may not perform well with high-dimensional data or imbalanced class distributions.

**3) Support Vector Machine (SVM) Classification**

The SVM algorithm aims to find the hyperplane that best separates the classes in the feature space. For non-linear data, SVM uses the kernel trick to map the data into a higher-dimensional space, where a hyperplane can be located to separate the classes.

SVM is sensitive to the choice of hyperparameters, particularly the kernel selection and its parameters. Proper tuning and validation are crucial for obtaining the best performance from an SVM model.

Import the SVC from sklearn.svm.

Instantiate the SVM model with appropriate parameters like kernel, C, and gamma.

Fit the model on the training data.

Evaluate the model's performance on the test set using metrics like accuracy, precision, recall, and F1-score.

**4) Performance Comparison:**

Compare the accuracy, precision, recall, and F1-score of both KNN and SVM models.

Analyze the confusion matrices to understand the true positives, true negatives, false positives, and false negatives for each model.

Determine which model performs better for your specific email spam detection task based on the evaluation metrics.

By following these steps and using the provided dataset, you can implement and compare the performance of KNN and SVM for email spam classification. The KNN algorithm classifies an email based on the majority class of its k nearest neighbors, while SVM finds the optimal hyperplane that separates spam and non-spam emails in the feature space

**5) Explanation of the Code:**

**Loading the Dataset:** The dataset is loaded into a pandas DataFrame. Ensure that the CSV file is in the correct location.

**Preprocessing:** The labels are converted from categorical (spam/ham) to binary (1/0).

**Data Splitting:** The dataset is split into training and testing sets, with 80% of the data used for training and 20% for testing.

**Vectorization:** The email text is transformed into numerical format using TF-IDF vectorization, which helps in converting text data into a format suitable for machine learning models.

**Model Training:**

* KNN: A K-Nearest Neighbors classifier is instantiated and trained on the training data.
* SVM: A Support Vector Machine classifier is instantiated with a linear kernel and trained similarly.

**Performance Evaluation:** The performance of both models is evaluated using classification reports and confusion matrices. Metrics such as precision, recall, and F1-score are provided for each model.

**Conclusion:**

* K-Nearest Neighbors (KNN): This model is straightforward to implement and can perform well but may be affected by the choice of k and the distance metric used. It can also struggle with large datasets due to its lazy learning nature.
* Support Vector Machine (SVM): Typically offers high accuracy and precision, especially with well-separated classes. However, it may require careful tuning of hyperparameters and can be sensitive to the choice of kernel.
* import pandas as pd
* from sklearn.model\_selection import train\_test\_split
* from sklearn.preprocessing import StandardScaler
* from sklearn.neighbors import KNeighborsClassifier
* from sklearn.svm import SVC
* from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix
* # Load the dataset
* url = 'D:/Sem-I 2024-25/LP-III/emails.csv'  # Replace with your actual file path or URL
* data = pd.read\_csv(url)
* # Data preprocessing
* # Separate features and labels
* X = data.iloc[:, 1:-1].values  # All columns except the first (email name) and the last (label)
* y = data.iloc[:, -1].values    # Last column is the label
* # Split the data into training and testing sets
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)
* # Standardize the data (important for SVM)
* scaler = StandardScaler()
* X\_train = scaler.fit\_transform(X\_train)
* X\_test = scaler.transform(X\_test)
* # K-Nearest Neighbors
* knn = KNeighborsClassifier(n\_neighbors=5)
* knn.fit(X\_train, y\_train)
* y\_pred\_knn = knn.predict(X\_test)
* # Support Vector Machine
* svm = SVC(kernel='linear', random\_state=42)
* svm.fit(X\_train, y\_train)
* y\_pred\_svm = svm.predict(X\_test)
* # Evaluate the models
* print("K-Nearest Neighbors (KNN) Performance:")
* print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_knn)}")
* print("Classification Report:\n", classification\_report(y\_test, y\_pred\_knn))
* print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_knn))
* print("\nSupport Vector Machine (SVM) Performance:")
* print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_svm)}")
* print("Classification Report:\n", classification\_report(y\_test, y\_pred\_svm))
* print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_svm))
* # Compare the models
* if accuracy\_score(y\_test, y\_pred\_knn) > accuracy\_score(y\_test, y\_pred\_svm):
* print("\nKNN performed better.")
* else:
* print("\nSVM performed better.")