Exercise 9 Date: 19/03/2025

SVM - Non Vectorial Dataset

Aim: To build a Support Vector Machine (SVM) to perform binary classification on a non-vectorial dataset using a string kernel.

Algorithm:

The String Kernel is a specialized kernel function used in Support Vector Machines (SVMs) for analyzing sequence-based data, such as text, DNA sequences, or protein structures. Instead of relying on traditional vector-based representations, the String Kernel directly computes similarities between string sequences by measuring the number of common substrings or subsequences.

Mathematically, the String Kernel computes the similarity $K(x, y)$ between two strings x and y by counting shared subsequences, weighted by their importance:

$$K(x, y) = \sum_{s \in \Sigma^*} \lambda^{|s|} C_s(x) C_s(y)$$

Where:

- Σ represents the alphabet of characters in the dataset.
- s is a subsequence common to both x and y.
- λ is a decay factor ($0 < \lambda \leq 1$) that assigns higher weight to shorter subsequences.
- $C_s(x)$ and $C_s(y)$ denote the number of times s appears in x and y, respectively.

The String Kernel is particularly useful in text classification, spam filtering, and bioinformatics applications, where feature extraction from raw sequences is challenging. It allows SVMs to work directly with sequences, preserving structural information without explicit feature engineering.

Step 1: Import Libraries

- Import necessary Python libraries such as pandas, numpy, scikit-learn modules, and tqdm for progress tracking.

Step 2: Load the Dataset

- Load the dataset into a pandas DataFrame and extract the first 2000 samples.

Step 3: Prepare the Data

- Select the 'review' column as the feature (X) and the 'sentiment' column as the target (y).
- Split the dataset into training and testing sets using an 80-20 split.

Step 4: Define the String Kernel Function

- Implement a custom string kernel that computes similarity between text samples based on character n-grams.
- Convert each text sequence into a set of character n-grams.
- Compute the kernel matrix by counting common n-grams between pairs of samples.

Step 5: Compute Kernel Matrices

- Compute the training kernel matrix using the training data.
- Compute the test kernel matrix using test data compared to training data.

Step 6: Train the SVM Model

- Use scikit-learn's SVM classifier with a precomputed kernel.
- Train the model using the computed kernel matrices.

Step 7: Make Predictions

- Use the trained model to predict sentiment labels on the test dataset.

Step 8: Evaluate the Model

- Evaluate the model's performance using the classification report, which includes precision, recall, and F1-score.

Import the libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn.kernel_approximation import Nystroem
```

Import and load the Dataset

In [2]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("lakshmi25npathi/imdb-dataset-of-50k-movie-reviews")

print("Path to dataset files:", path)
```

```
Path to dataset files: /home/ai-b1/.cache/kagglehub/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews/versions/1
```

In [3]:
```
df = pd.read_csv(path+'/IMDB Dataset.csv')
```

In [4]:
```
df
```

Out[4]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

Split features and target

In [5]:
```
X = df['review'][:2000]
```

In [6]:
```
y=df['sentiment'][:2000]
```

Split dataset into train and test data

In [7]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Create string kernel function

In [8]:
```
from tqdm import tqdm

def string_kernel(X, Y, ngram=3):
    """
    Simple string kernel based on character n-grams (substrings).
    This function computes the overlap between n-grams of two text sequences.
    """
    def n_grams(sequence, n):
        return [sequence[i:i+n] for i in range(len(sequence)-n+1)]

    X_ngrams = [n_grams(doc, ngram) for doc in X]
    Y_ngrams = [n_grams(doc, ngram) for doc in Y]

    kernel_matrix = np.zeros((len(X), len(Y)))

    for i, x_ngrams in tqdm(enumerate(X_ngrams), total=len(X), desc="Computing Kernel"):
        for j, y_ngrams in enumerate(Y_ngrams):
            common_ngrams = len(set(x_ngrams) & set(y_ngrams))
            kernel_matrix[i, j] = common_ngrams

    return kernel_matrix
```

Train the kernel

In [9]:
```
kernel_train = string_kernel(X_train, X_train)
```

```
Computing Kernel: 100%|██████████| 1600/1600 [02:49<00:00,  9.42it/s]
```

In [10]:
```
kernel_test = string_kernel(X_test, X_train)
```

```
Computing Kernel: 100%|██████████| 400/400 [00:42<00:00,  9.45it/s]
```

Apply SVM with the trained kernel

In [11]:
```
svm_model = svm.SVC(kernel='precomputed')
svm_model.fit(kernel_train, y_train)
```

Out[11]:
```
SVC(kernel='precomputed')
```

In [12]:
```
y_pred = svm_model.predict(kernel_test)
```

Performance Metrics

In [13]:
```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.79      0.82      0.81       195
    positive       0.82      0.80      0.81       205

    accuracy                           0.81       400
   macro avg       0.81      0.81      0.81       400
weighted avg       0.81      0.81      0.81       400
```

Result

A Support Vector Machine (SVM) classifier was successfully implemented using the String Kernel to analyze text-based data.