SVM on Non Vectorial Data (DNA Sequence)

Aim:

To implement SVM classification on non vectorial data

SVM Classification:

Support Vector Machine (SVM) is a supervised learning technique used for binary classification. It finds the optimal hyperplane that best separates data points into two classes. SVM aims to maximize the margin between the two classes, ensuring better generalization. The classification process involves the following steps:

1. **Initialization**: Define the dataset and choose a kernel for classification.
2. **Hyperplane Selection**: Identify the optimal decision boundary that maximizes the margin between the two classes.
3. **Support Vectors Identification**: Determine the critical data points (support vectors) that define the margin and influence the hyperplane's position.
4. **Optimization**: Use an optimization algorithm, such as Sequential Minimal Optimization (SMO), to find the best hyperplane.
5. **Prediction**: Classify new data points based on their position relative to the hyperplane.

Algorithm for SVM on Non-Vectorial Data (Without Visualization)

1. **Load the dataset** from the given URL.
2. **Preprocess the data**: Extract labels and DNA sequences and convert into numerical format.
3. **Encode class labels** using `LabelEncoder`.
4. **Split the dataset** into training and testing sets.
5. **Train an SVM classifier** with a linear kernel.
6. **Make predictions** on the test data.
7. **Evaluate the model** using accuracy score and a confusion matrix.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/molecular-biology/promoter-gene-sequences/promoters.data',
names=['Class', 'id', 'Sequence'])
data
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 106,\n  \"fields\":
[\n    {\n      \"column\": \"Class\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"-\",\n          \"+\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"id\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 106,\n
\"samples\": [\n          \" 663\",\n          \"RRNAB_P2\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sequence\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 106,\n        \"samples\":
[\n          \"\\t\\
tgagagcatgtcagcctcgacaacttgcataaatgctttcttgtagacgtgccctacg\",\n
\"\\tgcaaaaataaatgcttgactctgtagcgggaaggcgtattatgcacaccccgcgccg\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"data"}
```


```
# generate list of DNA sequences
classes = data.loc[:, 'Class']
sequences = list(data.loc[:, 'Sequence'])
d = {}

# loop through sequences and split into individual nucleotides
for i, seq in enumerate(sequences):
  # split into nucleotides, remove tab characters
  nucleotides = list(seq)
  nucleotides = [x for x in nucleotides if x != '\t']

  # append class assignment
  nucleotides.append(classes[i])

  # add to dataset
  d[i] = nucleotides

df = pd.DataFrame(d)
df
```


```
{"type":"dataframe","variable_name":"df"}
```

```
df = df.transpose()
df
```

```
{"type":"dataframe","variable_name":"df"}
```

```
x = df.iloc[:, :-1]
y = df.iloc[:, -1]

# Encode DNA sequence (a, t, c, g -> 0, 1, 2, 3)
encoding = {'a': 0, 't': 1, 'c': 2, 'g': 3}
x_encoded = x.applymap(lambda i: encoding[i])
```

```
<ipython-input-42-3bc61d6d9d4b>:3: FutureWarning: DataFrame.applymap
has been deprecated. Use DataFrame.map instead.
  x_encoded = x.applymap(lambda i: encoding[i])
```

```
# Encode labels ('+' -> 1, '-' -> 0)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x_encoded,
y_encoded, test_size=0.2, random_state=42)

# Create an SVM classifier (using a linear kernel for simplicity)
svm_classifier = SVC(kernel='linear')
svm_classifier.fit(x_train, y_train)
```

```
SVC(kernel='linear')
```

```
y_pred = svm_classifier.predict(x_test)
acc = accuracy_score(y_test, y_pred)
acc*100
```

```
81.81818181818183
```

```
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        11
           1       0.82      0.82      0.82        11

    accuracy                           0.82        22
   macro avg       0.82      0.82      0.82        22
weighted avg       0.82      0.82      0.82        22
```