CS634 Data Mining **Name:** Piyusha Sayal **UCID:** ps332 **Email:** ps332@njit.edu

FINAL-TERM PROJECT

Supervised Data Mining (Classification)

(OPTION 1)

Algorithms Chosen:

1. Support Vector Machine (Linear kernel, Sigmoid kernel, and Radial Basis Function kernel)
2. Naïve Bayes (Gaussian Naïve Bayes)
3. Also, implemented decision tree and random forest classification

Overview

Classification is one of the most common and basic data mining tasks to examine data where the classification is not known and will occur in the future. The goal is to predict where the outcome of certain tasks. The data where the result is known for classification is used to generate rules and train the model to predict the data where classification is unknown.

The supervised algorithm learns from the labelled data which refers to known inputs corresponding to known outputs, here the classes are known before training the model which is used to further predict future data. This technique helps identify the relationship between dependent and independent variables. After training, the model can then be applied to data for which the target value is not known. This helps identify casual relationships between variables, isolate their degree of correlation for each set of variables and develop a model showing a web of dependencies. There are referred to as predictive models.

Supervised training models can handle large data sets and are highly scalable. Supervised functions include classification, regression, and anomaly detection. In classification, the data items are divided into different categories or classes.

The goal of predictive classification is to predict the target class for each new data record that is not present in the historical data used to build the predictive classification model.

Classification begins with training data (data from which target values or class assignments are defined). Different algorithms/ techniques are used to find the relationship between the predictor attribute values and target values in training data. The model once built can be applied to new data values with unknown target classes to predict target values, this is referred to as test data. This step is known as testing a model and helps evaluate the model's accuracy.

Classification is used in customer segmentation, a recommender system to understand customer behavior and segment them into different categories based on predictor attribute variables.

A loan-issuing company divides customers applying for a loan into two classes: those who default and those who do not default. The predictor attribute values consist of various customer characteristics such as purchasing patterns, credit scores, income, etc. A model is built which predicts whether the new loan-issuing customer is likely to default or not.

Classification can have binary or multiclass targets. There exist different classification algorithms for supervised learning.

a) **Support Vector Machine:** Support Vector Machine is a machine learning approach that finds a hyperplane in N-dimensional space (N-the number of characteristics) that unambiguously classifies the input points. There are various possible hyperplanes for dividing the two sets of data points.

Our objective is to find a plane that has a maximum margin i.e. maximum distance between the data points of both classes. Increasing the margin adds in more confidence so that future data points can be classified more accurately.

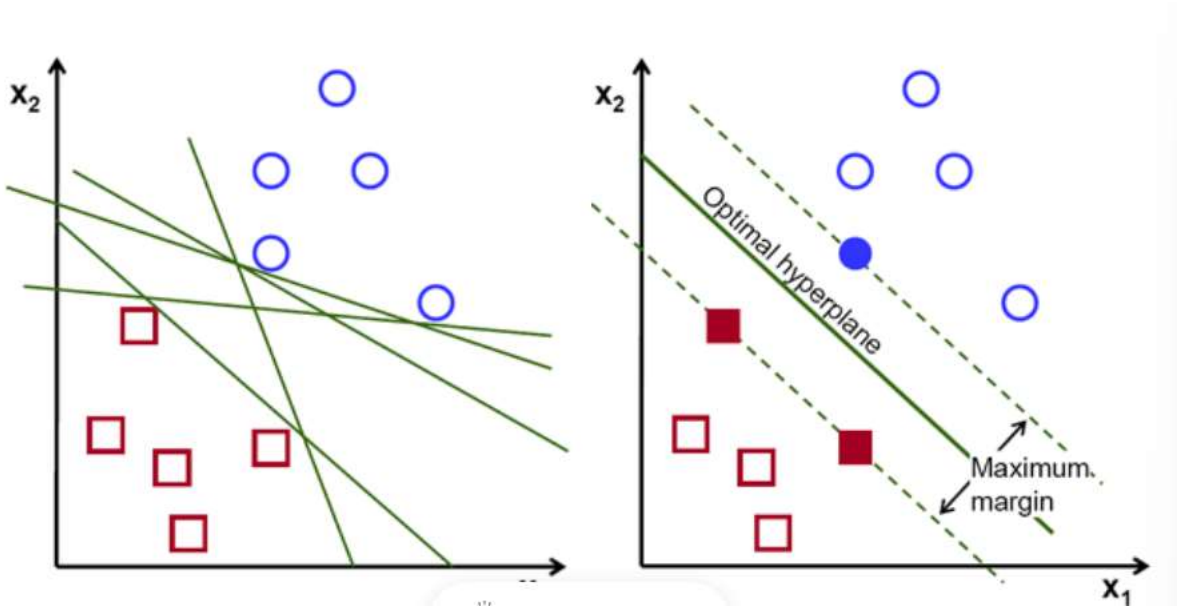

Figure 1: Support Vector Machine

The dimension of the hyperplane depends upon the number of input features. For example, a hyperplane is a line for 2 input features and a hyperplane is a plane for 3 input features.

Support vectors are data points that are closer to the hyperplane and have an effect on its location and orientation. These support vectors help in the

maximization of the classifier's margin and the construction of a support vector machine.

Support Vector Machine is robust to outliers. SVM kernel is a function that transforms low dimensional space to higher dimensional space. A non-separable problem is converted into a separable problem.

b) **Decision Tree:** It is one of the most common and powerful tools for classification and prediction. It is a flowchart-like structure that maps down all the possible outcomes for a series of related choices. The internal nodes indicate a test on a particular attribute and the branch represents the outcome of the test. The terminal node represents a class label.

Recursive partitioning keeps on repeated until splitting no longer adds value to the prediction results provided. Gini Impurity determines the accuracy of a split made among the classified groups, value ranges between 0 and 1. Here, 1 refers to random distribution among classes, and 0 refers to one particular class. These metrics help us evaluate the model built.

The decision tree is flexible because of its non-linear structure and focuses on probability and data to classify things. Decision trees can be used for customer recommendation systems. It helps clarify risks, objectives, and benefits.

c) **Random Forest:** It is a collection of a large number of decision trees, where each tree individually spits out a class prediction and the class with the maximum votes becomes the model's prediction. It can be utilized for both classification and regression-based problems.

Hyperparameters can be used to enhance the performance and predictive power of models or to make the model faster. The most commonly used hyperparameters are node size, the number of trees, and a number of features sampled. It uses the bagging ensemble method and feature randomness to create an uncorrelated forest of decision trees. Random forest only utilizes a subset of features.

d) **Bayesian Network:** Bayesian Network is a probability-based graphical model for the representation of an uncertain domain where each node corresponds to a random variable and each edge represents the conditional probability for that random variable. It helps model conditional dependency and causation by the representation of a directed graph. It can be used for probabilistic representation between disease and symptoms.

e) **Naïve Bayes:** It is a classification algorithm based on Baye's theorem. One of the common features among all the families of algorithms is that every pair of features being classified is independent of each other.

Requirements

Software Specifications

- Language: Python
- IDE: Jupyter Lab

Implementation

Flowchart

Figure 2: Data Mining Problem Solving Steps

Input Data

Data is fetched from the UCI repository. Dataset refers to the Molecular Biology (Promoter gene sequence)

Dataset Source:

https://archive.ics.uci.edu/ml/datasets/Molecular+Biology+(Promoter+Gene+Sequences)

Number of rows: 106

Number of columns: 3

Column Names: Class, ID, and Sequence

- **Class:** Values {+/-} indicates ("+" = Promoter)
- **ID:** The instance name (non-promoters named by position in the 1500-long nucleotide sequence.
- **Sequence:** DNA Sequence, values are filled with {'t', 'a', 'g', 'c'}

Data Set Characteristics:	Sequential, Domain-Theory	**Number of Instances:**	106	**Area:**	Life
Attribute Characteristics:	Categorical	**Number of Attributes:**	58	**Date Donated**	1990-06-30
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	93329

Figure 3: Dataset Characteristics

1	Column1	Column2	Column3
2	+	S10	tactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt
3	+	AMPC	tgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaa
4	+	AROH	gtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcg
5	+	DEOP2	aattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactc
6	+	LEU1_TRNA	tcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtag
7	+	MALEFG	aggggcaaggaggatggaaagaggttgccgtataaagaaactagagtccgtttaggt
8	+	MALK	caggggtggaggatttaagccatctcctgatgacgcatagtcagcccatcatgaat
9	+	RECA	tttctacaaaacacttgatactgtatgagcatacagtataattgcttcaacagaaca
10	+	RPOB	cgacttaatatactgcgacaggacgtccgttctgtgtaaatcgcaatgaaatggttt
11	+	RRNAB_P1	ttttaaatttcctcttgtcaggccggaataactccctataatgcgccaccactgaca
12	+	RRNAB_P2	gcaaaaataaatgcttgactctgtagcgggaaggcgtattatgcacaccccgcgccg
13	+	RRNDEX_P2	cctgaaattcagggttgactctgaaagaggaaagcgtaatatacgccacctcgcgac
14	+	RRND_P1	gatcaaaaaaatacttgtgcaaaaaattgggatccctataatgcgcctccgttgaga
15	+	RRNE_P1	ctgcaatttttctattgcggcctgcggagaactccctataatgcgcctccatcgaca
16	+	RRNG_P1	tttatatttttcgcttgtcaggccggaataactccctataatgcgccaccactgaca
17	+	RRNG_P2	aagcaaagaaatgcttgactctgtagcgggaaggcgtattatgcacaccgccgcgcc
18	+	RRNX_P1	atgcatttttccgcttgtcttcctgagccgactccctataatgcgcctccatcgaca
19	+	TNAA	aaacaatttcagaatagacaaaaactctgagtgtaataatgtagcctcgtgtcttgc
20	+	TYRT	tctcaacgtaacactttacagcggcgcgtcatttgatatgatgcgccccgcttcccg
21	+	ARAC	gcaaataatcaatgtggacttttctgccgtgattatagacacttttgttacgcgttt
22	+	LACI	gacaccatcgaatggcgcaaaacctttcgcggtatggcatgatagcgcccggaagag

Figure 4: Dataset (a)

23	+	MALT	aaaaacgtcatcgcttgcattagaaaggtttctggccgaccttataaccattaatta
24	+	TRP	tctgaaatgagctgttgacaattaatcatcgaactagttaactagtacgcaagttca
25	+	TRPP2	accggaagaaaaccgtgacattttaacacgtttgttacaaggtaaaggcgacgccgc
26	+	THR	aaattaaaatttttattgacttaggtcactaaatactttaaccaatataggcatagcg
27	+	BIOB	ttgtcataatcgacttgtaaaccaaattgaaaagatttaggtttacaagtctacacc
28	+	FOL	catcctcgcaccagtcgacgacggtttacgctttacgtatagtggcgacaatttttt
29	+	UVRBP1	tccagtataatttgttggcataattaagtacgacgagtaaaattacatacctgcccg
30	+	UVRBP3	acagttatccactattcctgtggataaccatgtgtattagagttagaaaacacgagg
31	+	LEXA	tgtgcagtttatggttccaaaatcgccttttgctgtatatactcacagcataactgt
32	+	PORI-L	ctgttgttcagtttttgagttgtgtataacccctcattctgatcccagcttatacgg
33	+	SPOT42	attacaaaaagtgctttctgaactgaacaaaaaagagtaaagttagtcgcgtagggt
34	+	M1RNA	atgcgcaacgcggggtgacaagggcgcgcaaaccctctatactgcgcgccgaagctg
35	+	GLNS	taaaaaactaacagttgtcagcctgtcccgcttataagatcatacgccgttatacgt
36	+	TUFB	atgcaatttttttagttgcatgaactcgcatgtctccatagaatgcgcgctacttgat
37	+	SUBB-E	ccttgaaaaagaggttgacgctgcaaggctctatacgcataatgcgccccgcaacgc
38	+	STR	tcgttgtatatttcttgacacctttcggcatcgccctaaaattcggcgtcctcata
39	+	SPC	ccgtttatttttttctacccatatccttgaagcggtgttataatgccgcgccctcgat
40	+	RPOA	ttcgcatatttttcttgcaaagttgggttgagctggctagattagccagccaatctt
41	+	RPLJ	tgtaaactaatgcctttacgtgggcggtgattttgtctacaatcttacccccacgta
42	+	PORI-R	gatcgcacgatctgtatacttatttgagtaaattaacccacgatcccagccattctt
43	+	ALAS	aacgcatacggtattttaccttcccagtcaagaaaacttatcttattcccacttttc
44	+	ARABAD	ttagcggatcctacctgacgctttttatcgcaactctctactgtttctccatacccg

Figure 5: Dataset (b)

45	+	BIOA	gccttctccaaaacgtgttttttgttgttaattcggtgtagacttgtaaacctaaat
46	+	DEOP1	cagaaacgttttattcgaacatcgatctcgtcttgtgttagaattctaacatacggt
47	+	GALP2	cactaatttattccatgtcacacttttcgcatctttgttatgctatggttatttcat
48	+	HIS	atataaaaaagttcttgctttctaacgtgaaagtggtttaggttaaaagacatcagt
49	+	HISJ	caaggtagaatgctttgccttgtcggcctgattaatggcacgatagtcgcatcggat
50	+	ILVGEDA	ggccaaaaaatatcttgtactatttacaaaacctatggtaactctttaggcattcct
51	+	LACP1	taggcaccccaggctttacactttatgcttccggctcgtatgttgtgtggaattgtg
52	+	LPP	ccatcaaaaaaatattctcaacataaaaaactttgtgtaatacttgtaacgctacat
53	+	TRPR	tggggacgtcgttactgatccgcacgtttatgatatgctatcgtactctttagcgag
54	+	UVRB_P2	tcagaaatattatggtgatgaactgtttttttatccagtataatttgttggcataat
55	-	867	atatgaacgttgagactgccgctgagttatcagctgtgaacgacattctggcgtcta
56	-	1169	cgaacgagtcaatcagaccgctttgactctggtattactgtgaacattattcgtctc
57	-	802	caatggcctctaaacgggtcttgaggggtttttgctgaaaggaggaactatatgcg
58	-	521	ttgacctactacgccagcattttggcggtgtaagctaaccattccggttgactcaat
59	-	918	cgtctatcggtgaacctccggtatcaacgctggaaggtgacgctaacgcagatgcag
60	-	1481	gccaatcaatcaagaacttgaagggtggtatcagccaacagcctgacatccttcgtt
61	-	1024	tggatggacgttcaacattgaggaaggcataacgctactacctgatgtttactccaa
62	-	1149	gaggtggctatgtgtatgaccgaacgagtcaatcagaccgctttgactctggtatta
63	-	313	cgtagcgcatcagtgctttcttactgtgagtacgcaccagcgccagaggacgacgac
64	-	780	cgaccgaagcgagcctcgtcctcaatggcctctaaacgggtcttgaggggttttttg
65	-	1384	ctacggtgggtacaatatgctggatggagatgcgttcacttctggtctactgactcg
66	-	507	atagtctcagagtcttgacctactacgccagcattttggcggtgtaagctaaccatt

Figure 6: Dataset (c)

67	-	39	aactcaaggctgatacggcgagacttgcgagccttgtccttgcggtacacagcagcg
68	-	1203	ttactgtgaacattattcgtctccgcgactacgatgagatgcctgagtgcttccgtt
69	-	988	tattctcaacaagattaaccgacagattcaatctcgtggatggacgttcaacattga
70	-	1171	aacgagtcaatcagaccgctttgactctggtattactgtgaacattattcgtctccg
71	-	753	aagtgcttagcttcaaggtcacggatacgaccgaagcgagcctcgtcctcaatggcc
72	-	630	gaagaccacgcctcgccaccgagtagacccttagagagcatgtcagcctcgacaact
73	-	660	ttagagagcatgtcagcctcgacaacttgcataaatgctttcttgtagacgtgccct
74	-	1216	tattcgtctccgcgactacgatgagatgcctgagtgcttccgttactggattgtcac
75	-	835	tgctgaaaggaggaactatatgcgctcatacgatatgaacgttgagactgccgctga
76	-	35	catgaactcaaggctgatacggcgagacttgcgagccttgtccttgcggtacacagc
77	-	1218	ttcgtctccgcgactacgatgagatgcctgagtgcttccgttactggattgtcacca
78	-	668	catgtcagcctcgacaacttgcataaatgctttcttgtagacgtgccctacgcgctt
79	-	413	aggaggaactacgcaaggttggaacatcggagagatgccagccagcgcacctgcacg
80	-	991	tctcaacaagattaaccgacagattcaatctcgtggatggacgttcaacattgagga
81	-	751	tgaagtgcttagcttcaaggtcacggatacgaccgaagcgagcctcgtcctcaatgg
82	-	850	ctatatgcgctcatacgatatgaacgttgagactgccgctgagttatcagctgtgaa
83	-	93	gcggcagcacgtttccacgcggtgagagcctcaggattcatgtcgatgtcttccggt
84	-	1108	atccctaatgtctacttccggtcaatccatctacgttaaccgaggtggctatgtgta
85	-	915	tggcgtctatcggtgaacctccggtatcaacgctggaaggtgacgctaacgcagatg
86	-	1019	tctcgtggatggacgttcaacattgaggaaggcataacgctactacctgatgtttac
87	-	19	tattggcttgctcaagcatgaactcaaggctgatacggcgagacttgcgagccttgt
88	-	1320	tagagggtgtactccaagaagaggaagatgaggctagacgtctctgcatggagtatg

Figure 7: Dataset (d)

89	-	91	cagcggcagcacgtttccacgcggtgagagcctcaggattcatgtcgatgtcttccg
90	-	217	ttacgttggcgaccgctaggactttcttgttgattttccatgcggtgttttgcgcaa
91	-	957	acgctaacgcagatgcagcgaacgctcggcgtattctcaacaagattaaccgacaga
92	-	260	ggtgttttgcgcaatgttaatcgctttgtacacctcaggcatgtaaacgtcttcgta
93	-	557	aaccattccggttgactcaatgagcatctcgatgcagcgtactcctacatgaataga
94	-	1355	agacgtctctgcatggagtatgagatggactacggtgggtacaatatgctggatgga
95	-	244	tgttgattttccatgcggtgttttgcgcaatgttaatcgctttgtacacctcaggca
96	-	464	tgcacgggttgcgatagcctcagcgtattcaggtgcgagttcgatagtctcagagtc
97	-	296	aggcatgtaaacgtcttcgtagcgcatcagtgctttcttactgtgagtacgcaccag
98	-	648	ccgagtagacccttagagagcatgtcagcctcgacaacttgcataaatgctttcttg
99	-	230	cgctaggactttcttgttgattttccatgcggtgttttgcgcaatgttaatcgcttt
100	-	1163	tatgaccgaacgagtcaatcagaccgctttgactctggtattactgtgaacattatt
101	-	1321	agagggtgtactccaagaagaggaagatgaggctagacgtctctgcatggagtatga
102	-	663	gagagcatgtcagcctcgacaacttgcataaatgctttcttgtagacgtgccctacg
103	-	799	cctcaatggcctctaaacgggtcttgaggggttttttgctgaaaggaggaactatat
104	-	987	gtattctcaacaagattaaccgacagattcaatctcgtggatggacgttcaacattg
105	-	1226	cgcgactacgatgagatgcctgagtgcttccgttactggattgtcaccaaggcttcc
106	-	794	ctcgtcctcaatggcctctaaacgggtcttgaggggttttttgctgaaaggaggaac
107	-	1442	taacattaataaataaggaggctctaatggcactcattagccaatcaatcaagaact
108			
109			
110			

Figure 8: Dataset (e)

Source Code:

```
0]: #CSV Module is imported to access csv extension files with ease
    import csv
    #with open is used to read the Promoters csv file in read mode
    #This dataset is fetched from UCI Machine Learning Repository
    #having 106 DNA sequences, with 57 sequential nucleotides ("base-pairs") each.
    with open('Promoters.csv',mode='r') as file:
        data=csv.reader(file)
        for lines in data:
            print(lines)
```

Figure 9: Source Code (a)

CSV module is imported to easily work with CSV files. Promoters.csv file is read and the lines can be printed to analyze the data using csv.reader functionality.

```
[21]: #Pandas can be directly used to read the csv file
      import pandas as pd
      data=pd.read_csv('Promoters.csv')

[22]: data
```

Figure 10: Source Code (b)

CSV file can be directly read using the pandas library imported. Data can be printed directly to analyze and understand the problem definition.

```
: #The first column refers to the Class wheather it is '+' or '-'
  #The second column refers to the ID
  #The third column refers to the DNA sequence
  #Columns of the datset can be set using .columns
  data.columns=['Class', 'Id', 'Sequence']

: #The dataset consist of 106 rows and 3 columns
  data.shape

: (106, 3)

: #Data with desired set of columns can be printed
  data
```

Figure 11: Source Code (c)

Data loaded earlier using pandas doesn't consist of column names as per the dataset. We can add column names to the data using data.columns functionality. We can even check the shape and structure of the data using data. shape which gives us the number of (rows, columns) in the dataset. Data with the desired set of columns can be printed.

```
[26]: #Dtypes help us print the data types of the object
      #All the columns of dtypes are object
      data.dtypes
```

Figure 12: Source Code (d)

The data type of the columns present in the dataset can be printed using data.dtypes.

```
[27]: #value_counts can be used to check the number of '+' and '-' classes
      data['Class'].value_counts()

[28]: #class_values contains all values of the Class column
      class_values=data.loc[:, 'Class']

[29]: #head() helps us print the top 5 rows of class_values
      class_values.head()

[30]: #id_lues contains all values of the Id column
      id_values=data.loc[:, 'Id']

[31]: #head() helps us print the top 5 rows of id_values
      id_values.head()

[32]: #sequence_lues contains all values of the Sequence column
      sequence_values=data.loc[:, 'Sequence']

[33]: #head() helps us print the top 5 rows of id_values
      sequence_values.head()
```

Figure 13: Source Code (e)

We can count the number of rows/data with Class='+' i.e Promoter and ones with Class='-'. Separate analysis can be done on the different columns in the dataset. The values of a specific column can be selected using data.loc[:, (column_name)]. Data.loc helps us access rows and columns of the data frame by using labels. ':' is used to specify that we require all rows to be printed for a specific column given by column_name, both row, and column separated by a comma. Data.head() is used to print the top 5 rows of the dataset.

```
[34]: #A dictionary is created to store the value of DNA sequence
      sequence={}
      #The sequence_values is enumerated using the index and value
      for index,value in enumerate(sequence_values):
          dna_sequence=list(value)
          #Store each character in the DNA sequence, ignoring the tab character
          dna_sequence=[char for char in dna_sequence if char!='\t']
          #Add the class label to the DNA sequence
          dna_sequence.append(class_values[index])
          sequence[index]=dna_sequence
      #Check the very first DNA sequence value of the sequence dictionary
      sequence[0]
      ...

[35]: #DataFrame is  created to easily utilize dna sequence values
      sequence_df=pd.DataFrame(sequence)

[36]: sequence_df
      ...
```

Figure 14: Source Code (f)

A dictionary named sequence is created to store the value of DNA sequences. The values in the sequence_values variable are enumerated using the function by using tuple unpacking and fetching the index and value. A Variable named dna_sequence stores the value in a list. List transformation takes place by breaking the value into different characters consisting of { ‘t’,’ a’, ’g’, ’c’} and ignoring the tabs. The class value for this specific index is also appended to the dna_sequence. The values added in dna_sequence are stored in the sequence dictionary with its index value used as the key. This step involves the most important step for data preparation for classification. The sequence dictionary is converted to a data frame using pandas library.

```
[37]: #Sequence df consist of 58 rows and 106 columns
      sequence_df.shape

[38]: #Transpose the data
      sequence_df=sequence_df.transpose()

[39]: #Head lists the top 5 rows
      sequence_df.head()

[40]: #Displays the shape of the sequence_df column
      sequence_df.shape

[40]: (106, 58)

[41]: sequence_df.columns

[42]: sequence_df.rename(columns={57:'Class'},inplace=True)

[43]: sequence_df
```

Figure 15: Source Code (g)

We can check the shape and structure of the new data frame created i.e. sequence_df. The data frame can be transposed to get the appropriate structure of data as rows and columns. Head command is used to get the top 5 rows of the data frame. We can get the column names using columns and they can be renamed using rename function doing it inplace.

One Hot Encoding

```
[44]: #One Hot Encoding is a process by which categorical variables are
      #converted to a form that could be provided to Machine learning
      #algorithms for better job prediction
      numerical_sequence=pd.get_dummies(sequence_df)

[45]: numerical_sequence.head()
```

```
[16]: #Shape of the numerical_sequence is 106 rows and 230 columns
      numerical_sequence.shape

      •••

[17]: #Only one Class is important for prediction, hence Class_- is
      #dropped from the numerical_sequence dataframe
      numerical_sequence.drop('Class_-', axis=1, inplace=True)

[18]: numerical_sequence.head()

      •••

[19]: #The Class_+ is renamed as Class for ease of running machine learning algorithms
      numerical_sequence.rename(columns = {'Class_+':'Class'}, inplace = True)
```

Figure 16: Source Code (h)

One Hot Encoding is a type of data transformation for converting categorical variables, and mapping to integers. Each categorical value will be converted new categorical column with values 1 or 0 assigned to those columns. This helps the machine learning algorithm to make better predictions. There exists a function named get_dummies in pandas to enable one hot encoding. The new created data frame is visualized by printing the top 5 rows using head command. The shape of numerical_sequence can be checked using shape. We will drop the “Class_-“ from the dataset using drop because we need only column for class as target value to evaluate the model performance. Also, rename the column name from “Class_+” to “Class” to simplify the data preparation and data representation.

Training and Testing Classification

```
[51]: #The required set of libraries and packages are downloaded and installed from sklearn
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.naive_bayes import GaussianNB
      from sklearn import svm
      from sklearn.metrics import classification_report, accuracy_score
```

```
]: from sklearn.model_selection import train_test_split
#The data and class label are seperated from the numerical_sequence dataframe
X = numerical_sequence.drop(['Class'], axis = 1).values
y = numerical_sequence['Class'].values

#define a seed for reproducibility. random_state simply sets a seed to the random generator,
#so that your train-test splits are always deterministic. If you don't set a seed, it is different each time.
seed = 1

# Splitting data into training and testing data keeping 75% as training data and 25% as test_data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = seed)
```

```
[53]: X_train.shape

[53]: (79, 228)

[54]: X_test.shape

[54]: (27, 228)

[55]: y_train.shape

[55]: (79,)

[56]: y_test.shape

[56]: (27,)
```

Figure 17: Source Code (i)

After the data preparation and pre-processing are done, training and testing classification are carried out. The required set of libraries to carry out different analyses for different classification algorithms is carried out and loaded from the sklearn package. Various metrics parameters are also important and are loaded such as classification report and accuracy. The numerical_sequence data is stored in X and y values. The data is split into training and testing data using 75% data as training data and 25% as testing data. The random_seed=1, where the seed is defined for reproducibility. If this value is not set, every time the split results are different. We can analyze the shape and structure of the training and test data for X_train, X_test, y_train, and y_test.

Decision Tree Classifier:

Decision Tree Classifier

```
57]: scoring='accuracy'
     #Maximum depth of the decision tree is kept as 5,
     #the more splits it has the more information it captures
     decision_tree=DecisionTreeClassifier(max_depth=5)

55]: #KFold is a class that lets you split your data into K non overlapping folds
     #cross_val_score evaluates the data and returns the score. The score is evaluated
     #by randomly splitting training sets into distinct subsets called folds then the
     #model is trained and evlauted on the folds
     #picking a different fold each time for evaluation and training on other folds
     from sklearn.model_selection import KFold, cross_val_score
```

Figure 18: Source Code (j)

Accuracy is used as a scoring parameter. Decision Tree Classifier is used to built a classification machine learning algorithm, the tree will have a maximum depth of 5 as specified in the parameter. To check the accuracy of the model, two evaluation metrics are loaded kFold and cross_val_score. kFolds is used to split dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by cross-validation. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, cross-validation generator/ splitting stratergy is also supplied keeping the scoring parameter as ‘accuracy’.

```
[]: #The data is split into 10 non overlapping folds
   kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
   #Cross validation score is evaluated for the decision tree model,
   #using 10 folds of data and accuracy as the scoring model
   cv_results = cross_val_score(decision_tree, X_train, y_train, cv = kfold, scoring = 'accuracy')
   decision_tree_result=cv_results
   #The evaluated results for 10 as cross validation paraamter is displayed using mean and standard deviation
   print('Decision Tree Results', 'Mean:', round(cv_results.mean(),5),
         'Standard Deviation:',  round(cv_results.std(),5))
```

Figure 19: Source Code (k)

KFold is used to split the dataset into 10 non-overlapping sets keeping shuffle=True so the dataset is shuffled before splitting into batches. If the shuffle is True, random state affects the ordering of the indices controlling the randomness of each fold. Cross Validation score is evaluated for the decision tree built using X_train, y_train, keeping cross-validation as kfolds and scoring as accuracy.

```
1]: #Decision Tree model is fitted with training data and predictions
    #are made using the test data
    decision_tree.fit(X_train,y_train)
    y_pred=decision_tree.predict(X_test)
    #The Accuracy and classification report which consists on
    #precision, recall, f1 score are printed
    print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
    print('Classification Report')
    print(classification_report(y_test,y_pred))
```

Figure 20: Source Code (I)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

Figure 21: Source Code (m)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[78]: from sklearn.metrics import confusion_matrix
      #Confusion matrix determines the performance of the classification model
      confusion_matrix = confusion_matrix(y_test, y_pred)
      import seaborn as sns
      sns.heatmap(confusion_matrix, annot=True)
```

Figure 22: Source Code (n)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

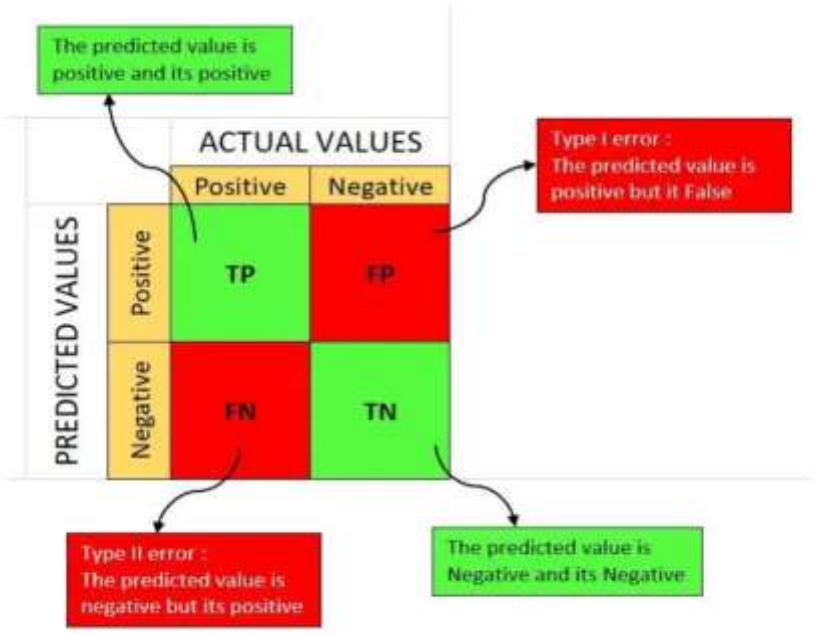

Figure 23: Confusion Matrix

Random Forest Classifier:

```
79]: #Random forest Classifier model is built using 10 trees with max depth as 5
     #and maximum 1 feature is used when considering best split
     random_forest= RandomForestClassifier(max_depth = 5, n_estimators = 10, max_features = 1 )
```

```
1]: #The data is split into 10 non overlapping folds
    kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
    #Cross validation score is evaluated for the decision tree model,
    #using 10 folds of data and accuracy as the scoring model
    cv_results = cross_val_score(random_forest, X_train, y_train, cv = kfold, scoring = 'accuracy')
    random_forest_result=cv_results
    #The evaluated results for 10 as cross validation paroamter is displayed using mean and standard deviation
    print('Random Forest Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',  round(cv_results.std(),5)
```

Figure 24: Source Code (o)

Accuracy is used as a scoring parameter. Random Forest Classifier is used to built a classification machine learning algorithm, the tree will have a maximum depth of 5, taking 10 decision trees and a maximum of 1 feature selection when looking for the best split. as specified in the parameter. To check the accuracy and performance of the model, two evaluation metrics are loaded kFold and cross_val_score are used.

kFolds is used to split dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by using a cross-validation evaluation metric. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, cross-validation generator/ splitting strategy is also supplied keeping the scoring parameter as 'accuracy'.

```
]: #Random Forest model is fitted with training data and predictions are made using the test data
   random_forest.fit(X_train,y_train)
   y_pred=random_forest.predict(X_test)
   #The Accuracy and classification report which consists on precision, recall, f1 score are printed
   print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
   print('Classification Report')
   print(classification_report(y_test,y_pred))
```

Figure 25: Source Code (p)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
[83]: import matplotlib.pyplot as plt
      from sklearn import metrics
      #The false positive rate, true positive rate
      false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
      #The roc_auc_score returns the area under curve score value
      area_under_curve = metrics.roc_auc_score(y_test,y_pred)
      area_under_curve=round(area_under_curve,5)

      plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
      plt.ylabel("True Positive Rate")
      plt.xlabel("False Positive Rate")
      plt.legend()

      plt.show()
      ...
```

Figure 26: Source Code (q)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[84]: from sklearn.metrics import confusion_matrix
      #Confusion matrix determines the performance of the classification model
      confusion_matrix = confusion_matrix(y_test, y_pred)
      import seaborn as sns
      sns.heatmap(confusion_matrix, annot=True)
      ...
```

Figure 27: Source Code (r)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

Gaussian Naïve Bayes:

Gaussian Naive Bayes

```
5]: #Gaussian Nive Bayes model is instantiated
    gaussian_nb=GaussianNB()
```

```
]: #The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(gaussian_nb, X_train, y_train, cv = kfold, scoring = 'accuracy')
gaussian_nb_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Gaussian Naive Bayes Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

Figure 28: Source Code (s)

Accuracy is used as a scoring parameter. Random Forest Classifier is built without passing any specific parameter. To check the accuracy and performance of the model, two evaluation metrics are loaded kFold and cross_val_score are used. kFolds is used to split dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by using a cross-validation evaluation metric. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, cross-validation generator/ splitting strategy is also supplied keeping the scoring parameter as ‘accuracy’.

```
#Gaussian Naive Bayes model is fitted with training data and predictions are made using the test data
gaussian_nb.fit(X_train,y_train)
y_pred=gaussian_nb.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

Figure 29: Source Code (t)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

Figure 30: Source Code (u)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds

metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[89]: from sklearn.metrics import confusion_matrix
      #Confusion matrix determines the performance of the classification model
      confusion_matrix = confusion_matrix(y_test, y_pred)
      import seaborn as sns
      sns.heatmap(confusion_matrix, annot=True)
```

Figure 31: Source Code (v)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

Support Vector Machine:

```
Support Vector Machine

[90]: #Support Vector machine models with different kernels: linear, radial basis function and sigmoid
      linear_svm=svm.SVC(kernel = 'linear')
      radial_basis_func_svm=svm.SVC(kernel = 'rbf')
      sigmoid_svm=svm.SVC(kernel = 'sigmoid')
```

Figure 32: Source Code (w)

The support Vector machine classification model is used to built a classification machine learning algorithm , by specifying the type of kernel as the parameter involved. The kernel's function is to receive data as input and transform it into the desired form. Different SVM algorithms employ various sorts of kernel functions.

RBF is the most often utilized type of kernel function. Because it has a confined and finite response along the whole x-axis. The kernel functions return the inner product of two points in a suitable feature space.

Linear Support Vector Machine:

Linear Support Vector Machine

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(linear_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
linear_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Linear SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

Figure 33: Source Code (x)

Accuracy is used as a scoring parameter. To check the accuracy and performance of the model, two evaluation metrics are loaded kFold and cross_val_score are used. kFolds is used to split the dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by using a cross-validation evaluation metric. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, a cross-validation generator/ splitting strategy is also supplied keeping the scoring parameter as ‘accuracy’.

```
[92]: #Linear SVM model is fitted with training data and predictions are made using the test data
linear_svm.fit(X_train,y_train)
y_pred=linear_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

Figure 34: Source Code (y)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
[93]: import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

Figure 35: Source Code (z)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds

metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[94]: from sklearn.metrics import confusion_matrix
      #Confusion matrix determines the performance of the classification model
      confusion_matrix = confusion_matrix(y_test, y_pred)
      import seaborn as sns
      sns.heatmap(confusion_matrix, annot=True)
```

Figure 36: Source Code (aa)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

Radial Basis Function Support Vector Machine:

Radial Basis Function Support Vector Machine

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(radial_basis_func_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
radial_basis_func_svm_result=cv_results
#The evaluated results for 10 as cross validation paramter
#is displayed using mean and standard deviation
print('Radial Basis Function SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

Figure 37: Source Code (ab)

Accuracy is used as a scoring parameter. To check the accuracy and performance of the model, two evaluation metrics are loaded kFold and cross_val_score are used. kFolds is used to split the dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by using a cross-validation evaluation metric. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, a cross-validation generator/ splitting strategy is also supplied keeping the scoring parameter as 'accuracy'.

```
96]: #Radial Basis Function SVM model is fitted with
     #training data and predictions are made using the test data
     radial_basis_func_svm.fit(X_train,y_train)
     y_pred=radial_basis_func_svm.predict(X_test)
     #The Accuracy and classification report
     #which consists on precision, recall, f1 score are printed
     print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
     print('Classification Report')
     print(classification_report(y_test,y_pred))
```

Figure 38: Source Code (ac)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the

parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
97]: import matplotlib.pyplot as plt
     from sklearn import metrics
     #The false positive rate, true positive rate
     false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
     #The roc_auc_score returns the area under curve score value
     area_under_curve = metrics.roc_auc_score(y_test,y_pred)
     area_under_curve=round(area_under_curve,5)

     plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
     plt.ylabel("True Positive Rate")
     plt.xlabel("False Positive Rate")
     plt.legend()

     plt.show()
```

Figure 39: Source Code (ad)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[98]: from sklearn.metrics import confusion_matrix
      #Confusion matrix determines the performance of the classification model
      confusion_matrix = confusion_matrix(y_test, y_pred)
      import seaborn as sns
      sns.heatmap(confusion_matrix, annot=True)
      ...
```

Figure 40: Source Code (ae)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

Sigmoid Support Vector Machine:

Sigmoid Support Vector Machine

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(sigmoid_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
sigmoid_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Sigmoid SVM Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',
      round(cv_results.std(),5))
```

Figure 41: Source Code (af)

Accuracy is used as a scoring parameter. To check the accuracy and performance of the model, two evaluation metrics are loaded kFold and cross_val_score are used. kFolds is used to split the dataset into k consecutive folds without shuffling by default. Each fold is used for validation keeping the rest k-1 for the training dataset. The cross_val_score evaluates a score by using a cross-validation evaluation metric. There exist different parameters to choose from: an estimator is basically the object used to fit the data, the data to fit and the target variable to try to predict. Additionally, a cross-validation generator/ splitting strategy is also supplied keeping the scoring parameter as ‘accuracy’.

```
[100]: #Sigmoid SVM model is fitted with training data and predictions are made using the test data
       sigmoid_svm.fit(X_train,y_train)
       y_pred=sigmoid_svm.predict(X_test)
       #The Accuracy and classification report which consists on precision, recall, f1 score are printed
       print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
       print('Classification Report')
       print(classification_report(y_test,y_pred))
```

Figure 42: Source Code (ag)

Model fitting is the process of adjusting its parameters in order to improve accuracy. The fit function is used to take the X_train and y_train data to train the decision tree classifier model. Fitting the model refers to finding the most optimal solution for the coordinates/parameters of the equation to further predict results for test data with high accuracy. Prediction is made for the X_test data. Then we can check the accuracy score evaluated using y_test and y_predicted result. The classification report shows the main classification metrics, taking y_test and y_pred as the parameters. The report provided as output consists of a precision, recall (sensitivity), f1-score and support value.

```
[1]: import matplotlib.pyplot as plt
     from sklearn import metrics
     #The false positive rate, true positive rate
     false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
     #The roc_auc_score returns the area under curve score value
     area_under_curve = metrics.roc_auc_score(y_test,y_pred)
     area_under_curve=round(area_under_curve,5)

     plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
     plt.ylabel("True Positive Rate")
     plt.xlabel("False Positive Rate")
     plt.legend()

     plt.show()
```

Figure 43: Source Code (ah)

Matplotlib library is imported to visualize the results in the graph which provides effective and easy analysis. The false positive rate, true positive rate and thresholds metrics are provided using y_test and y_pred. The roc_auc_score i.e. Receiver Operating Characteristic Curve from prediction scores is calculated to visualize the area under the curve. Area Under Curve can be plotted using false positive rate and true positive rate.

```
[102]: from sklearn.metrics import confusion_matrix
       #Confusion matrix determines the performance of the classification model
       confusion_matrix = confusion_matrix(y_test, y_pred)
       import seaborn as sns
       sns.heatmap(confusion_matrix, annot=True)
```

Figure 44: Source Code (ai)

The confusion matrix is NXN matrix imported for evaluation of the performance of a classification model. Even the seaborn library is imported to plot and visualize the confusion matrix as a heatmap. A good model is one with high true positives and true negative rates. It is a tabular representation of correct and incorrect predictions made by the classifier.

Complete Code

```
]: #CSV Module is imported to access csv extension files with ease
   import csv
   #with open is used to read the Promoters csv file in read mode
   #This dataset is fetched from UCI Machine Learning Repository
   #having 106 DNA sequences, with 57 sequential nucleotides (“base-pairs”) each.
   with open('Promoters.csv',mode='r') as file:
       data=csv.reader(file)
       for lines in data:
           print(lines)
```

```
1]: #Pandas can be directly used to read the csv file
    import pandas as pd
    data=pd.read_csv('Promoters.csv')

2]: data
```

```
[23]: #The first column refers to the Class wheather it is '+' or '-'
      #The second column refers to the ID
      #The third column refers to the DNA sequence
      #Columns of the datset can be set using .columns
      data.columns=['Class', 'Id', 'Sequence']

[24]: #The dataset consist of 106 rows and 3 columns
      data.shape

      •••

[25]: #Data with desired set of columns can be printed
      data

      •••

[26]: #Dtypes help us print the data types of the object
      #All the columns of dtypes are object
      data.dtypes

      •••
```

Figure 45: Complete Code (a)

```
[27]: #value_counts can be used to check the number of '+' and '-' classes
      data['Class'].value_counts()

      •••

[28]: #class_values contains all values of the Class column
      class_values=data.loc[:, 'Class']

[29]: #head() helps us print the top 5 rows of class_values
      class_values.head()

      •••

[30]: #id_lues contains all values of the Id column
      id_values=data.loc[:, 'Id']

[31]: #head() helps us print the top 5 rows of id_values
      id_values.head()

      •••
```

```
[32]: #sequence_lues contains all values of the Sequence column
      sequence_values=data.loc[:, 'Sequence']

[33]: #head() helps us print the top 5 rows of id_values
      sequence_values.head()
      •••

[34]: #A dictionary is created to store the value of DNA sequence
      sequence={}
      #The sequence_values is enumerated using the index and value
      for index,value in enumerate(sequence_values):
          dna_sequence=list(value)
          #Store each character in the DNA sequence, ignoring the tab character
          dna_sequence=[char for char in dna_sequence if char!='\t']
          #Add the class label to the DNA sequence
          dna_sequence.append(class_values[index])
          sequence[index]=dna_sequence
      #Check the very first DNA sequence value of the sequence dictionary
      sequence[0]
      •••
```

Figure 46: Complete Code (b)

```
[35]: #DataFrame is  created to easily utilize dna sequence values
      sequence_df=pd.DataFrame(sequence)

[36]: sequence_df
      •••

[37]: #Sequence df consist of 58 rows and 106 columns
      sequence_df.shape

[37]: (58, 106)

[38]: #Transpose the data
      sequence_df=sequence_df.transpose()

[39]: #Head lists the top 5 rows
      sequence_df.head()
      •••

[40]: #Displays the shape of the sequence_df column
      sequence_df.shape

[40]: (106, 58)

[41]: sequence_df.columns

[41]: RangeIndex(start=0, stop=58, step=1)

[42]: sequence_df.rename(columns={57:'Class'},inplace=True)

[43]: sequence_df
      •••
```

Figure 47: Complete Code (c)

```
4]: #One Hot Encoding is a process by which categorical variables are
    #converted to a form that could be provided to Machine learning
    #algorithms for better job prediction
    numerical_sequence=pd.get_dummies(sequence_df)
```

```
5]: numerical_sequence.head()
```

```
[46]: #Shape of the numerical_sequence is 106 rows and 230 columns
      numerical_sequence.shape
```

```
[47]: #Only one Class is important for prediction, hence Class_- is
      #dropped from the numerical_sequence dataframe
      numerical_sequence.drop('Class_-', axis=1, inplace=True)
```

```
[48]: numerical_sequence.head()
```

```
[49]: #The Class_+ is renamed as Class for ease of running machine learning algorithms
      numerical_sequence.rename(columns = {'Class_+':'Class'}, inplace = True)
```


```
[47]: #Only one Class is important for prediction, hence Class_- is dropped from the numerical_sequence dataframe
numerical_sequence.drop('Class_-', axis=1, inplace=True)
```

```
[48]: numerical_sequence.head()
```

```
[49]: #The Class_+ is renamed as Class for ease of running machine learning algorithms
numerical_sequence.rename(columns = {'Class_+':'Class'}, inplace = True)
```


Training and Testing Classification

```
[51]: #The required set of libraries and packages are downloaded and installed from sklearn
      from sklearn.tree import DecisionTreeClassifier
      from sklearn.ensemble import RandomForestClassifier
      from sklearn.naive_bayes import GaussianNB
      from sklearn import svm
      from sklearn.metrics import classification_report, accuracy_score
```

```
2]: from sklearn.model_selection import train_test_split
    #The data and class label are seperated from the numerical_sequence dataframe
    X = numerical_sequence.drop(['Class'], axis = 1).values
    y = numerical_sequence['Class'].values

    #define a seed for reproducibility. random_state simply sets a seed to the random generator,
    #so that your train-test splits are always deterministic. If you don't set a seed, it is different each time.
    seed = 1

    # Splitting data into training and testing data keeping 75% as training data and 25% as test_data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = seed)
```

Figure 48: Complete Code (d)

```
[53]: X_train.shape
...
[54]: X_test.shape
...
[55]: y_train.shape
...
[56]: y_test.shape
...
```

Decision Tree Classifier

```
scoring='accuracy'
#Maximum depth of the decision tree is kept as 5,
#the more splits it has the more information it captures
decision_tree=DecisionTreeClassifier(max_depth=5)
```

```
#KFold is a class that lets you split your data into K non overlapping folds
#cross_val_score evaluates the data and returns the score. The score is evaluated
#by randomly splitting training sets into distinct subsets called folds then the
#model is trained and evlauted on the folds
#picking a different fold each time for evaluation and training on other folds
from sklearn.model_selection import KFold, cross_val_score
```

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model, using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(decision_tree, X_train, y_train, cv = kfold, scoring = 'accuracy')
decision_tree_result=cv_results
#The evaluated results for 10 as cross validation paraamter is displayed using mean and standard deviation
print('Decision Tree Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',  round(cv_results.std(),5))
...
```

Figure 49: Complete Code (e)

```
[71]: #Decision Tree model is fitted with training data and predictions are made using the test data
      decision_tree.fit(X_train,y_train)
      y_pred=decision_tree.predict(X_test)
      #The Accuracy and classification report which consists on precision, recall, f1 score are printed
      print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
      print('Classification Report')
      print(classification_report(y_test,y_pred))
      ...
```

---

[75]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

•••

[78]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 50: Complete Code (f)

Random Forest Classifier

```
#Random forest Classifier model is built using 10 trees with max depth as 5
#and maximum 1 feature is used when considering best split
random_forest= RandomForestClassifier(max_depth = 5, n_estimators = 10, max_features = 1 )
```

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(random_forest, X_train, y_train, cv = kfold, scoring = 'accuracy')
random_forest_result=cv_results
#The evaluated results for 10 as cross validation paraamter is displayed using mean and standard deviation
print('Random Forest Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',  round(cv_results.std(),5))
```

[82]:
```
#Random Forest model is fitted with training data and predictions are made using the test data
random_forest.fit(X_train,y_train)
y_pred=random_forest.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

[83]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

[84]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 51: Complete Code (g)

Gaussian Naive Bayes

```
#Gaussian Nive Bayes model is instantiated
gaussian_nb=GaussianNB()
```

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(gaussian_nb, X_train, y_train, cv = kfold, scoring = 'accuracy')
gaussian_nb_result=cv_results
#The evaluated results for 10 as cross validation parameter
#is displayed using mean and standard deviation
print('Gaussian Naive Bayes Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',  round(cv_results.std(),5))
```

[87]:
```
#Gaussian Naive Bayes model is fitted with training data and predictions are made using the test data
gaussian_nb.fit(X_train,y_train)
y_pred=gaussian_nb.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

[88]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

[89]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 52: Complete Code (h)

Support Vector Machine

[90]:
```
#Support Vector machine models with different kernels: linear, radial basis function and sigmoid
linear_svm=svm.SVC(kernel = 'linear')
radial_basis_func_svm=svm.SVC(kernel = 'rbf')
sigmoid_svm=svm.SVC(kernel = 'sigmoid')
```

Figure 53: Complete Code (i)

]:
```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(linear_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
linear_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Linear SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

[92]:
```
#Linear SVM model is fitted with training data and predictions are made using the test data
linear_svm.fit(X_train,y_train)
y_pred=linear_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

***

[93]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

[94]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 54: Complete Code (j)

5]:
```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(radial_basis_func_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
radial_basis_func_svm_result=cv_results
#The evaluated results for 10 as cross validation paramter
#is displayed using mean and standard deviation
print('Radial Basis Function SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

[96]:
```
#Radial Basis Function SVM model is fitted with training data and predictions are made using the test data
radial_basis_func_svm.fit(X_train,y_train)
y_pred=radial_basis_func_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

...

Figure 55: Complete Code (k)

[97]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

[98]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 56: Complete Code (I)

Sigmoid Support Vector Machine

]:
```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(sigmoid_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
sigmoid_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Sigmoid SVM Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',
      round(cv_results.std(),5))
```

[100]:
```
#Sigmoid SVM model is fitted with training data and predictions are made using the test data
sigmoid_svm.fit(X_train,y_train)
y_pred=sigmoid_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

•••

[101]:
```
import matplotlib.pyplot as plt
from sklearn import metrics
#The false positive rate, true positive rate
false_positive_rate,true_positive_rate, thresholds = metrics.roc_curve(y_test,y_pred)
#The roc_auc_score returns the area under curve score value
area_under_curve = metrics.roc_auc_score(y_test,y_pred)
area_under_curve=round(area_under_curve,5)

plt.plot(false_positive_rate,true_positive_rate, label="Area Under Curve" + str(area_under_curve))
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.legend()

plt.show()
```

[102]:
```
from sklearn.metrics import confusion_matrix
#Confusion matrix determines the performance of the classification model
confusion_matrix = confusion_matrix(y_test, y_pred)
import seaborn as sns
sns.heatmap(confusion_matrix, annot=True)
```

Figure 57: Complete Code (m)

Output Data

```
#CSV Module is imported to access csv extension files with ease
import csv
#with open is used to read the Promoters csv file in read mode
#This dataset is fetched from UCI Machine Learning Repository
#having 106 DNA sequences, with 57 sequential nucleotides ("base-pairs") each.
with open('Promoters.csv',mode='r') as file:
    data=csv.reader(file)
    for lines in data:
        print(lines)
```

```
['ï»¿Column1', 'Column2', 'Column3']
['+', 'S10', '\t\ttactagcaatacgcttgcgttcggtggttaagtatgtataatgcgcgggcttgtcgt']
['+', 'AMPC', '\t\ttgctatcctgacagttgtcacgctgattggtgtcgttacaatctaacgcatcgccaa']
['+', 'AROH', '\t\tgtactagagaactagtgcattagcttatttttttgttatcatgctaaccacccggcg']
['+', 'DEOP2', '\taattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaatactaacaaactc']
['+', 'LEU1_TRNA', '\ttcgataattaactattgacgaaaagctgaaaaccactagaatgcgcctccgtggtag']
['+', 'MALEFG', '\taggggcaaggaggatggaaagaggttgccgtataaagaaactagagtccgtttaggt']
['+', 'MALK', '\t\tcaggggtggaggatttaagccatctcctgatgacgcatagtcagcccatcatgaat']
['+', 'RECA', '\t\ttttctacaaaacacttgatactgtatgagcatacagtataattgcttcaacagaaca']
['+', 'RPOB', '\t\tcgacttaatatactgcgacaggacgtccgttctgtgtaaatcgcaatgaaatggttt']
['+', 'RRNAB_P1', '\tttttaaatttcctcttgtcaggccggaataactccctataatgcgccaccactgaca']
['+', 'RRNAB_P2', '\tgcaaaaataaatgcttgactctgtagcgggaaggcgtattatgcacaccccgcgccg']
['+', 'RRNDEX_P2', '\tcctgaaattcagggttgactctgaaagaggaaagcgtaatatacgccacctcgcgac']
['+', 'RRND_P1', '\tgatcaaaaaaatacttgtgcaaaaaattgggatccctataatgcgcctccgttgaga']
```

Figure 58: Output Data (a)

The lines in csv file (dataset) can be printed using the csv.reader option by reading the csv file and printing lines.

```
[21]: #Pandas can be directly used to read the csv file
      import pandas as pd
      data=pd.read_csv('Promoters.csv')

[22]: data
```

[22]:

	Column1	Column2	Column3
0	+	S10	\t\ttactagcaatacgcttgcgttcggtggttaagtatgtataat...
1	+	AMPC	\t\ttgctatcctgacagttgtcacgctgattggtgtcgttacaat...
2	+	AROH	\t\tgtactagagaactagtgcattagcttatttttttgttatcat...
3	+	DEOP2	\taattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaa...
4	+	LEU1_TRNA	\ttcgataattaactattgacgaaaagctgaaaaccactagaatgc...
...	...	...	...
101	-	799	\t\tcctcaatggcctctaaacgggtcttgaggggttttttgctga...
102	-	987	\t\tgtattctcaacaagattaaccgacagattcaatctcgtggat...
103	-	1226	\t\tcgcgactacgatgagatgcctgagtgcttccgttactggatt...
104	-	794	\t\tctcgtcctcaatggcctctaaacgggtcttgaggggttttttt...
105	-	1442	\t\ttaacattaataaataaggaggctctaatggcactcattagcc...

106 rows × 3 columns

Figure 59: Output Data (b)

The data read using pandas can be directly printed and analyzed

```
[26]: #Dtypes help us print the data types of the object
      #All the columns of dtypes are object
      data.dtypes

[26]: Class       object
      Id          object
      Sequence    object
      dtype: object

[27]: #value_counts can be used to check the number of '+' and '-' classes
      data['Class'].value_counts()

[27]: +    53
      -    53
      Name: Class, dtype: int64

[28]: #class_values contains all values of the Class column
      class_values=data.loc[:, 'Class']

[29]: #head() helps us print the top 5 rows of class_values
      class_values.head()

[29]: 0    +
      1    +
      2    +
      3    +
      4    +
      Name: Class, dtype: object
```

Figure 60: Output Data (c)

This shows the data type of the columns in the dataset and the count for Class ={+/-}

```
[30]: #id_lues contains all values of the Id column
      id_values=data.loc[:, 'Id']

[31]: #head() helps us print the top 5 rows of id_values
      id_values.head()

[31]: 0          S10
      1         AMPC
      2         AROH
      3        DEOP2
      4    LEU1_TRNA
      Name: Id, dtype: object

[32]: #sequence_lues contains all values of the Sequence column
      sequence_values=data.loc[:, 'Sequence']

[33]: #head() helps us print the top 5 rows of id_values
      sequence_values.head()

[33]: 0    \t\ttactagcaatacgcttgcgttcggtggttaagtatgtataat...
      1    \t\ttgctatcctgacagttgtcacgctgattggtgtcgttacaat...
      2    \t\tgtactagagaactagtgcattagcttatttttttgttatcat...
      3    \taattgtgatgtgtatcgaagtgtgttgcggagtagatgttagaa...
      4    \ttcgataattaactattgacgaaaagctgaaaaccactagaatgc...
      Name: Sequence, dtype: object
```

Figure 61: Output Data (d)

Similarly, values can be checked for all the columns of the dataset.

```
[34]: #A dictionary is created to store the value of DNA sequence
      sequence={}
      #The sequence_values is enumerated using the index and value
      for index,value in enumerate(sequence_values):
          dna_sequence=list(value)
          #Store each character in the DNA sequence, ignoring the tab character
          dna_sequence=[char for char in dna_sequence if char!='\t']
          #Add the class label to the DNA sequence
          dna_sequence.append(class_values[index])
          sequence[index]=dna_sequence
      #Check the very first DNA sequence value of the sequence dictionary
      sequence[0]

[34]: ['t',
       'a',
       'c',
       't',
       'a',
       'g',
       'c',
       'a',
       'a',
       't',
       'a',
       'c',
```

Figure 62: Output Data (e)

This output shows the DNA sequence data transformed and stored in sequence dictionary at index 0.

```
[35]: #DataFrame is  created to easily utilize dna sequence values
      sequence_df=pd.DataFrame(sequence)

[36]: sequence_df
```

[36]:

	0	1	2	3	4	5	6	7	8	9	...	96	97	98	99	100	101	102	103	104	105
0	t	t	g	a	t	a	c	t	c	t	...	c	c	t	a	g	c	g	c	c	t
1	a	g	t	a	c	g	a	t	g	t	...	c	g	a	g	a	c	t	g	t	a
2	c	c	a	t	g	g	g	t	a	t	...	g	c	t	a	g	t	a	c	c	a
3	t	t	c	t	a	g	g	c	c	t	...	a	t	g	g	a	c	t	g	g	c
4	a	a	t	g	t	g	g	t	t	a	...	g	a	a	g	g	a	t	a	t	a
5	g	t	a	t	a	c	g	a	t	a	...	t	g	c	g	c	a	c	c	c	t
6	c	c	g	g	a	a	g	c	a	a	...	a	g	c	t	a	t	t	t	c	t
7	a	c	a	a	t	a	t	a	a	t	...	g	a	g	g	t	g	c	a	t	a
8	a	t	g	t	t	g	g	a	t	t	...	a	c	a	t	g	g	a	c	c	a
9	t	g	a	g	a	g	g	a	a	t	...	c	t	a	a	t	c	a	g	a	t
10	a	a	a	t	a	a	a	a	t	c	...	c	t	c	c	c	c	c	a	a	a

Figure 63: Output Data (f)

This output shows the sequence data frame created for all the DNA sequence values in the dataset.

```
[37]: #Sequence df consist of 58 rows and 106 columns
      sequence_df.shape

[37]: (58, 106)

[38]: #Transpose the data
      sequence_df=sequence_df.transpose()

[39]: #Head lists the top 5 rows
      sequence_df.head()
```

[39]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	57
0	t	a	c	t	a	g	c	a	a	t	...	g	c	t	t	g	t	c	g	t	+
1	t	g	c	t	a	t	c	c	t	g	...	c	a	t	c	g	c	c	a	a	+
2	g	t	a	c	t	a	g	a	g	a	...	c	a	c	c	c	g	g	c	g	+
3	a	a	t	t	g	t	g	a	t	g	...	a	a	c	a	a	a	c	t	c	+
4	t	c	g	a	t	a	a	t	t	a	...	c	c	g	t	g	g	t	a	g	+

5 rows × 58 columns

```
[40]: #Displays the shape of the sequence_df column
      sequence_df.shape

[40]: (106, 58)

[41]: sequence_df.columns

[41]: RangeIndex(start=0, stop=58, step=1)

[42]: sequence_df.rename(columns={57:'Class'},inplace=True)

[43]: sequence_df
```

Figure 64: Output Data (g)

The sequence_df structure consists of 58 rows and 106 columns, this data frame needs to be transposed and then we can get the correct structure and format of the data frame.

```
[42]: sequence_df.rename(columns={57:'Class'},inplace=True)

[43]: sequence_df
```

[43]:

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	Class
0	t	a	c	t	a	g	c	a	a	t	...	g	c	t	t	g	t	c	g	t	+
1	t	g	c	t	a	t	c	c	t	g	...	c	a	t	c	g	c	c	a	a	+
2	g	t	a	c	t	a	g	a	g	a	...	c	a	c	c	c	g	g	c	g	+
3	a	a	t	t	g	t	g	a	t	g	...	a	a	c	a	a	a	c	t	c	+
4	t	c	g	a	t	a	a	t	t	a	...	c	c	g	t	g	g	t	a	g	+
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
101	c	c	t	c	a	a	t	g	g	c	...	g	a	a	c	t	a	t	a	t	-
102	g	t	a	t	t	c	t	c	a	a	...	t	c	a	a	c	a	t	t	g	-
103	c	g	c	g	a	c	t	a	c	g	...	a	a	g	g	c	t	t	c	c	-
104	c	t	c	g	t	c	c	t	c	a	...	a	g	g	a	g	g	a	a	c	-
105	t	a	a	c	a	t	t	a	a	t	...	t	c	a	a	g	a	a	c	t	-

106 rows × 58 columns

Figure 65: Output Data (h)

We can print the data frame to visualize the data that is used further for classification problems.

One Hot Encoding

```
]: #One Hot Encoding is a process by which categorical variables are
   #converted to a form that could be provided to Machine Learning
   #algorithms for better job prediction
   numerical_sequence=pd.get_dummies(sequence_df)

]: numerical_sequence.head()
```

]:

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	55_a	55_c	55_g	55_t	56_a	56_c	56_g	56_t	Class_+	Class_-
0	0	0	0	1	1	0	0	0	0	1	...	0	0	1	0	0	0	0	1	1	0
1	0	0	0	1	0	0	1	0	0	1	...	1	0	0	0	1	0	0	0	1	0
2	0	0	1	0	0	0	0	1	1	0	...	0	1	0	0	0	0	1	0	1	0
3	1	0	0	0	1	0	0	0	0	0	...	0	0	0	1	0	1	0	0	1	0
4	0	0	0	1	0	1	0	0	0	0	...	1	0	0	0	0	0	1	0	1	0

5 rows × 230 columns

```
[6]: #Shape of the numerical_sequence is 106 rows and 230 columns
     numerical_sequence.shape

[6]: (106, 230)

[7]: #Only one Class is important for prediction, hence Class_- is
     #dropped from the numerical_sequence dataframe
     numerical_sequence.drop('Class_-', axis=1, inplace=True)

[8]: numerical_sequence.head()
```

[8]:

	0_a	0_c	0_g	0_t	1_a	1_c	1_g	1_t	2_a	2_c	...	54_t	55_a	55_c	55_g	55_t	56_a	56_c	56_g	56_t	Class_+
0	0	0	0	1	1	0	0	0	0	1	...	0	0	0	1	0	0	0	0	1	1
1	0	0	0	1	0	0	1	0	0	1	...	0	1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	1	1	0	...	0	0	1	0	0	0	0	1	0	1
3	1	0	0	0	1	0	0	0	0	0	...	0	0	0	0	1	0	1	0	0	1
4	0	0	0	1	0	1	0	0	0	0	...	1	1	0	0	0	0	0	1	0	1

5 rows × 229 columns

```
[9]: #The Class_+ is renamed as Class for ease of running machine learning algorithms
     numerical_sequence.rename(columns = {'Class_+':'Class'}, inplace = True)
```

Figure 66: Output Data (i)

The sequence_df is converted to numerical_df by performing one-hot encoding for the dataset formation for the classification problem.

Training and Testing Classification

```
]: #The required set of libraries and packages are downloaded and installed from sklearn
   from sklearn.tree import DecisionTreeClassifier
   from sklearn.ensemble import RandomForestClassifier
   from sklearn.naive_bayes import GaussianNB
   from sklearn import svm
   from sklearn.metrics import classification_report, accuracy_score
```

```
]: from sklearn.model_selection import train_test_split
   #The data and class label are seperated from the numerical_sequence dataframe
   X = numerical_sequence.drop(['Class'], axis = 1).values
   y = numerical_sequence['Class'].values

   #define a seed for reproducibility. random_state simply sets a seed to the random generator,
   #so that your train-test splits are always deterministic. If you don't set a seed, it is different each time
   seed = 1

   # Splitting data into training and testing data keeping 75% as training data and 25% as test_data
   X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = seed)
```

```
3]: X_train.shape

3]: (79, 228)

4]: X_test.shape

4]: (27, 228)

5]: y_train.shape

5]: (79,)

6]: y_test.shape

6]: (27,)
```

Figure 67: Output Data (j)

The training data is 75% and the test data is 25 %. The X_train shape is 79 X 228, X_test shape is 27 x 228. For the output data, y_train is (79,) and y_test is (27,)

Decision Tree Classifier:

```
70]: #The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(decision_tree, X_train, y_train, cv = kfold, scoring = 'accuracy')
decision_tree_result=cv_results
#The evaluated results for 10 as cross validation paraamter is displayed using mean and standard deviation
print('Decision Tree Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))

Decision Tree Results Mean: 0.60714 Standard Deviation: 0.13098
```

```
[71]: #Decision Tree model is fitted with training data and predictions are made using the test data
decision_tree.fit(X_train,y_train)
y_pred=decision_tree.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.852
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.76      0.87        17
           1       0.71      1.00      0.83        10

    accuracy                           0.85        27
   macro avg       0.86      0.88      0.85        27
weighted avg       0.89      0.85      0.85        27
```

Figure 68: Output Data (k)

The Decision Tree Classifier model gives a mean accuracy of 0.60714 with a standard deviation of 0.13098 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 85.2%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one for a perfect classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is quite close for both values.

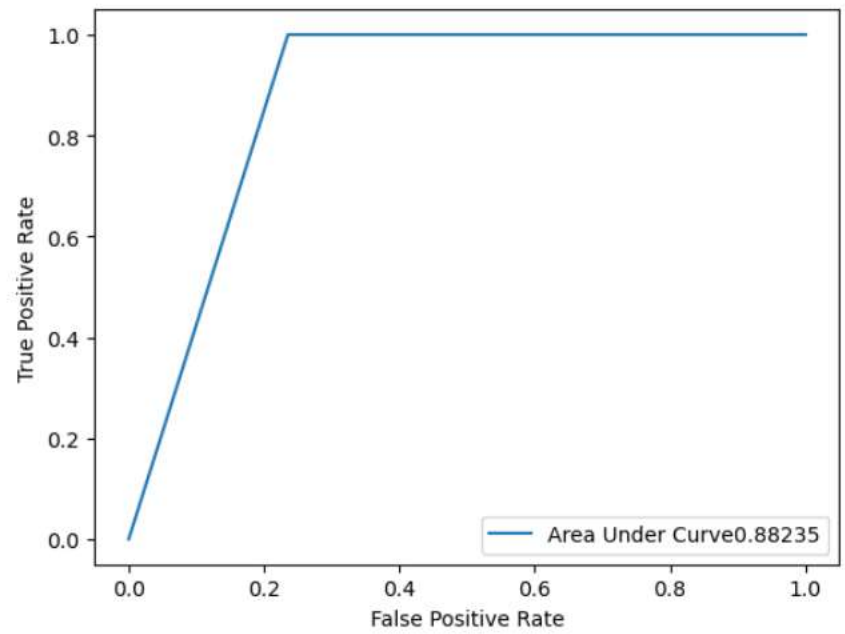

Figure 69: Output Data (l)

The receiver operating characteristics curve gives an area under the curve of about 0.88235. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

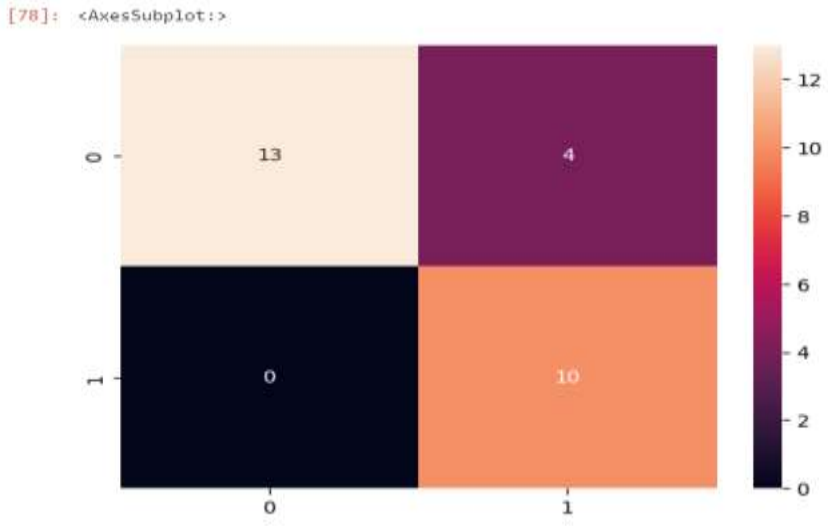

Figure 70: Output Data (m)

True Positive: 13

True Negative: 10

False Positive: 0

False Negative: 4

The True positive and true negative should be high, while the false positive and false negative should be low.

Random Forest Classifier:

```
]: #The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(random_forest, X_train, y_train, cv = kfold, scoring = 'accuracy')
random_forest_result=cv_results
#The evaluated results for 10 as cross validation paraamter is displayed using mean and standard deviation
print('Random Forest Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

```
Random Forest Results Mean: 0.61429 Standard Deviation: 0.2158
```

```
]: #Random Forest model is fitted with training data and predictions are made using the test data
random_forest.fit(X_train,y_train)
y_pred=random_forest.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.519
Classification Report
              precision    recall  f1-score   support

           0       0.75      0.35      0.48        17
           1       0.42      0.80      0.55        10

    accuracy                           0.52        27
   macro avg       0.59      0.58      0.52        27
weighted avg       0.63      0.52      0.51        27
```

Figure 71: Output Data (n)

The Random Forest Classifier model gives a mean accuracy of 0.6142 with a standard deviation of 0.2158 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 51.9%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one for a perfect

classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is a bit higher for binary value 1 than for binary value 0.

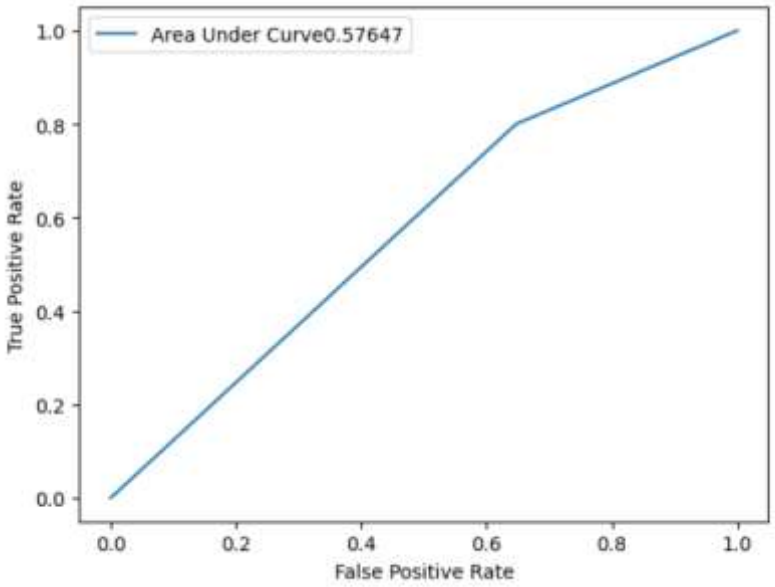

Figure 72: Output Data (o)

The receiver operating characteristics curve gives an area under the curve of about 0.57647. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

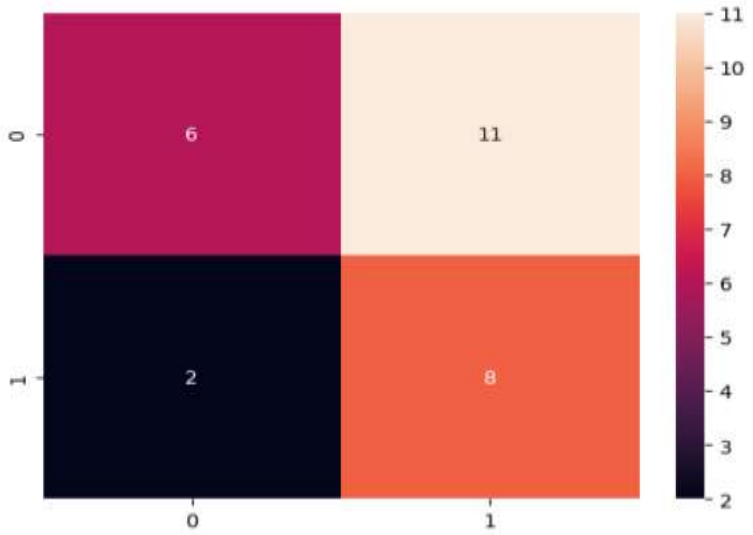

Figure 73: Output Data (p)

True Positive: 6

True Negative: 8

False Positive: 2

False Negative: 11

The True positive and true negative should be high, while the false positive and false negative should be low.

Gaussian Naïve Bayes:

```
86]: #The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(gaussian_nb, X_train, y_train, cv = kfold, scoring = 'accuracy')
gaussian_nb_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Gaussian Naive Bayes Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

```
Gaussian Naive Bayes Results Mean: 0.8375 Standard Deviation: 0.1125
```

```
[87]: #Gaussian Naive Bayes model is fitted with training data and predictions are made using the test data
gaussian_nb.fit(X_train,y_train)
y_pred=gaussian_nb.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.926
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.88      0.94        17
           1       0.83      1.00      0.91        10

    accuracy                           0.93        27
   macro avg       0.92      0.94      0.92        27
weighted avg       0.94      0.93      0.93        27
```

Figure 74: Output Data (q)

The Gaussian Naïve Bayes Classifier model gives a mean accuracy of 0.8375 with a standard deviation of 0.1125 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 92.6%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among

all relevant instances. Precision and recall are both equal to one for a perfect classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is quite close for binary value 1 and binary value 0.

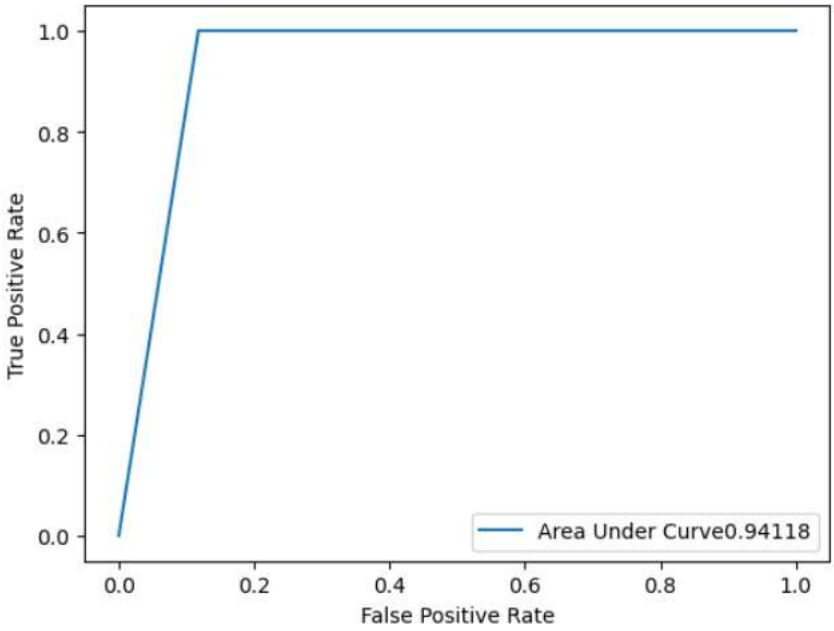

Figure 75: Output Data (r)

The receiver operating characteristics curve gives an area under the curve of about 0.94118. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

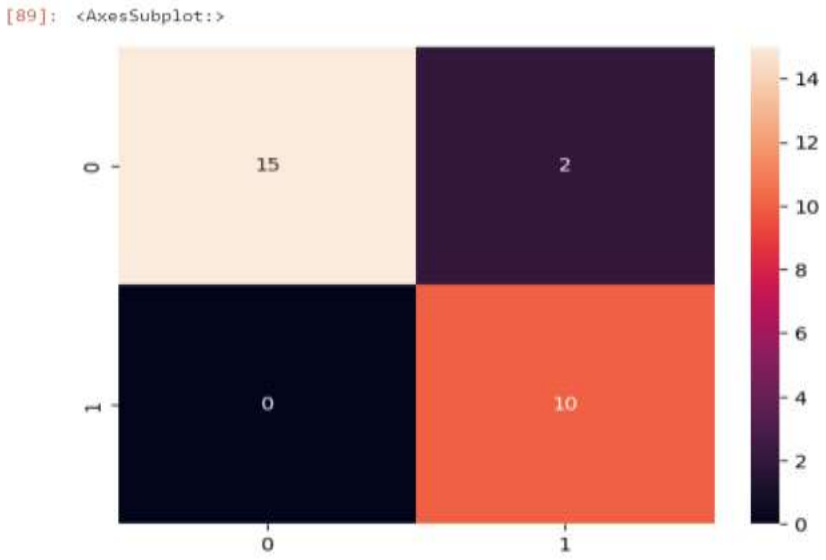

Figure 76: Output Data (s)

True Positive: 15

True Negative: 10

False Positive: 0

False Negative: 2

The True positive and true negative should be high, while the false positive and false negative should be low.

Linear Support Vector Machine:

```
1]: #The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(linear_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
linear_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Linear SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))

Linear SVM Results Mean: 0.9125 Standard Deviation: 0.09763
```

```
[92]: #Linear SVM model is fitted with training data and predictions are made using the test data
linear_svm.fit(X_train,y_train)
y_pred=linear_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.963
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.91      1.00      0.95        10

    accuracy                           0.96        27
   macro avg       0.95      0.97      0.96        27
weighted avg       0.97      0.96      0.96        27
```

Figure 77: Output Data (t)

The Linear Support Vector Machine Classifier model gives a mean accuracy of 0.9125 with a standard deviation of 0.09763 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 96.3%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one for a perfect classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is quite close for binary value 1 and binary value 0.

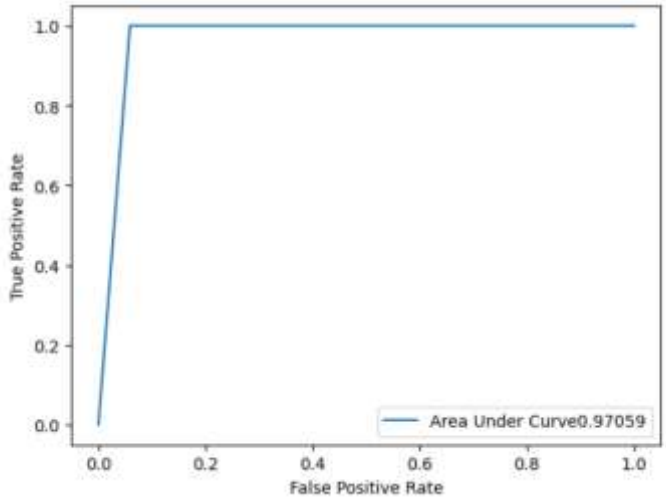

Figure 78: Output Data (u)

The receiver operating characteristics curve gives an area under the curve of about 0.97059. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

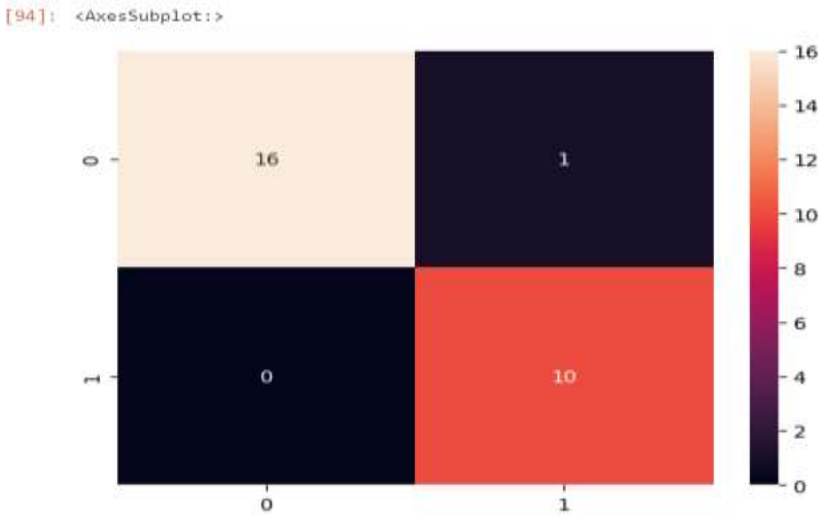

Figure 79: Output Data (v)

True Positive: 16

True Negative: 10

False Positive: 0

False Negative: 1

The True positive and true negative should be high, while the false positive and false negative should be low.

Radial Basis Function Support Vector Machine:

Radial Basis Function Support Vector Machine

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(radial_basis_func_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
radial_basis_func_svm_result=cv_results
#The evaluated results for 10 as cross validation paramter
#is displayed using mean and standard deviation
print('Radial Basis Function SVM Results', 'Mean:', round(cv_results.mean(),5),
      'Standard Deviation:',  round(cv_results.std(),5))
```

```
Radial Basis Function SVM Results Mean: 0.875 Standard Deviation: 0.1118
```

```
[96]: #Radial Basis Function SVM model is fitted with training data and predictions are made using the test data
radial_basis_func_svm.fit(X_train,y_train)
y_pred=radial_basis_func_svm.predict(X_test)
#The Accuracy and classification report which consists on precision, recall, f1 score are printed
print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
print('Classification Report')
print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.926
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.88      0.94        17
           1       0.83      1.00      0.91        10

    accuracy                           0.93        27
   macro avg       0.92      0.94      0.92        27
weighted avg       0.94      0.93      0.93        27
```

Figure 80: Output Data (w)

The Radial Basis Function Support Vector Machine Classifier model gives a mean accuracy of 0.875 with a standard deviation of 0.1118 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 92.6%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one

for a perfect classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is quite close for binary value 1 and binary value 0.

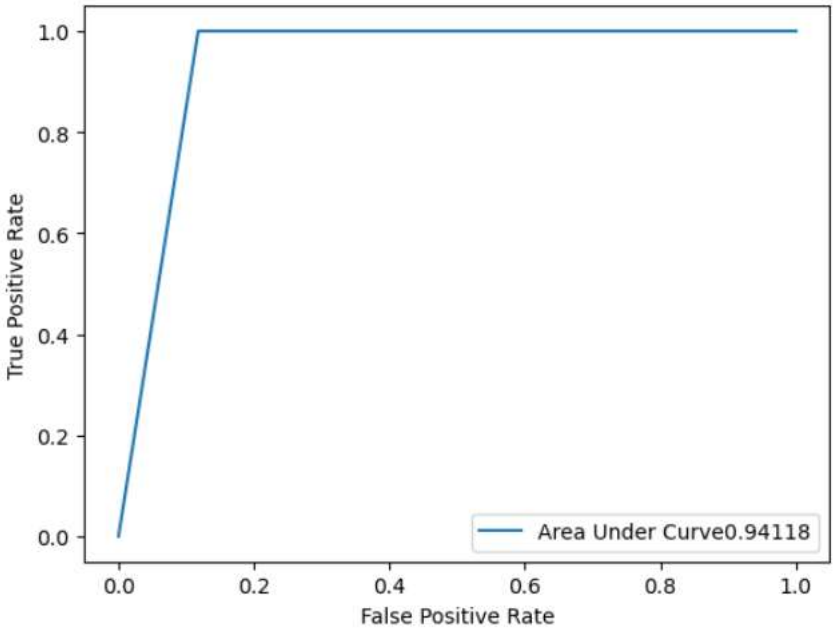

Figure 81: Output Data (x)

The receiver operating characteristics curve gives an area under the curve of about 0.94118. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

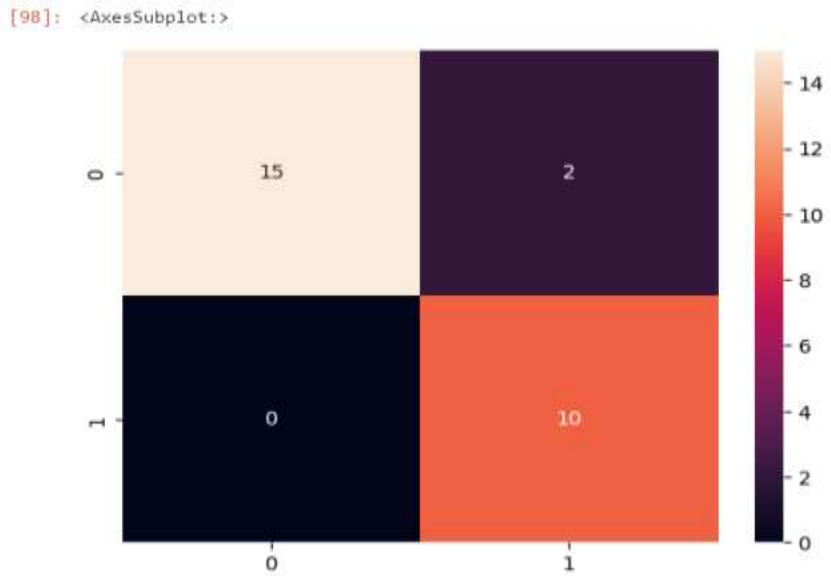

Figure 82: Output Data (y)

True Positive: 15

True Negative: 10

False Positive: 0

False Negative: 2

The True positive and true negative should be high, while the false positive and false negative should be low.

Sigmoid Support Vector Machine:

Sigmoid Support Vector Machine

```
#The data is split into 10 non overlapping folds
kfold = KFold(n_splits = 10, shuffle=True, random_state=True)
#Cross validation score is evaluated for the decision tree model,
#using 10 folds of data and accuracy as the scoring model
cv_results = cross_val_score(sigmoid_svm, X_train, y_train, cv = kfold, scoring = 'accuracy')
sigmoid_svm_result=cv_results
#The evaluated results for 10 as cross validation paraamter
#is displayed using mean and standard deviation
print('Sigmoid SVM Results', 'Mean:', round(cv_results.mean(),5),'Standard Deviation:',
      round(cv_results.std(),5))
```

```
Sigmoid SVM Results Mean: 0.925 Standard Deviation: 0.1
```

```
[100]: #Sigmoid SVM model is fitted with training data and predictions are made using the test data
       sigmoid_svm.fit(X_train,y_train)
       y_pred=sigmoid_svm.predict(X_test)
       #The Accuracy and classification report which consists on precision, recall, f1 score are printed
       print('Accuracy Score:', round(accuracy_score(y_test,y_pred),3))
       print('Classification Report')
       print(classification_report(y_test,y_pred))
```

```
Accuracy Score: 0.926
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.88      0.94        17
           1       0.83      1.00      0.91        10

    accuracy                           0.93        27
   macro avg       0.92      0.94      0.92        27
weighted avg       0.94      0.93      0.93        27
```

Figure 83: Output Data (z)

The Sigmoid Support Vector Machine Classifier model gives a mean accuracy of 0.925 with a standard deviation of 0.1 after 10-fold cross-validation. The accuracy of the model based on the test output values and the predicted values is 92.6%. Precision is defined as the proportion of relevant occurrences among all examples retrieved. Recall, also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one for a perfect classifier. There are 17 instances of binary value 0 and 10 instances of binary value 1. The f1-score is quite close for binary value 1 and binary value 0.

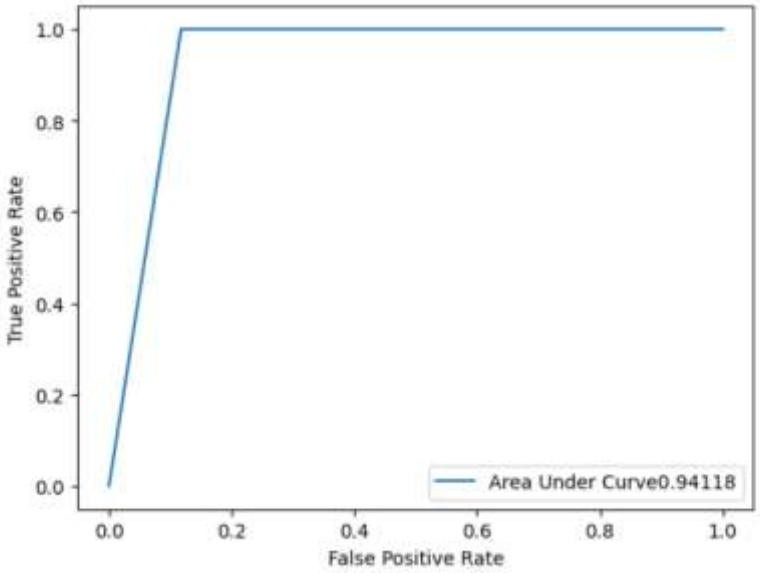

Figure 84: Output Data (aa)

The receiver operating characteristics curve gives an area under the curve of about 0.94118. This graph is plotted using the True positive rate and the False positive rate. The higher the area under the curve the better the model is.

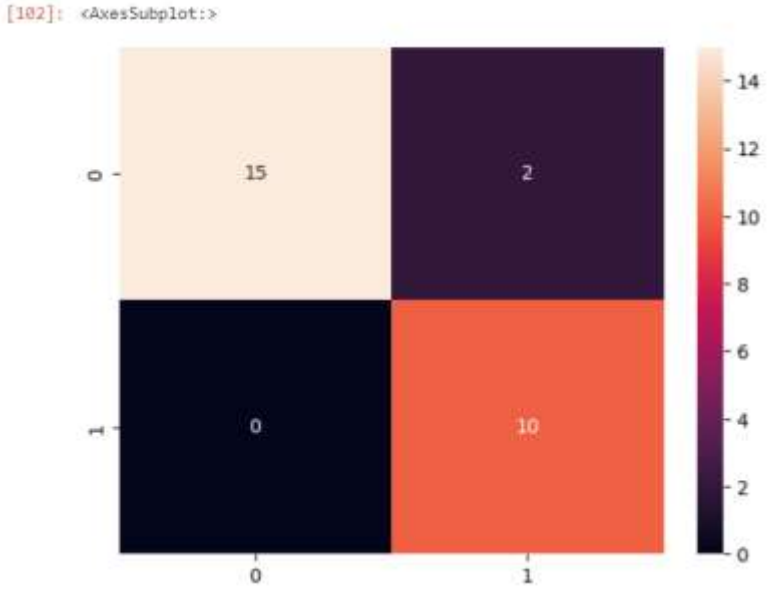

Figure 85: Output Data (ab)

True Positive: 15

True Negative: 10

False Positive: 0

False Negative: 2

The True positive and true negative should be high, while the false positive and false negative should be low.

Result

Machine Learning Model	Cross Validation Results Mean (Std)	Accuracy Score	Area Under the curve	True Positive Rate	False Positive Rate	True Negative Rate	False Negative Rate
Decision Tree Classifier	0.60714 (0.13098)	85.2 %	0.88235	13	0	10	4
Random Forest Classifier Gaussian Naïve Bayes	0.61429 (0.2158)	51.9 %	0.57647	6	2	8	11
Gaussian Naïve Bayes	0.8375 (0.1125)	92.6 %	0.94118	15	0	10	2
Linear Support Vector Machine	0.9125 (0.09763)	96.3%	0.97059	16	0	10	1
Radial Basis Function Support Vector Machine	0.875 (0.1118)	92.6%	0.94118	15	0	10	2
Sigmoid Support Vector Machine	0.925 (0.1)	92.6%	0.94118	15	0	10	2

Analysis:

In order to avoid overfitting, we need to test how well model will perform with new data. The dataset is initially split into training and test dataset. The dataset partitioned into number of subsets, holding onto one set and training the model on the rest. Then we can test model on the hold out set. **K Fold cross validation** has a single parameter k, the value of k is chosen such that each train/test sample is large enough to statistically represent the broader dataset.

Precision is defined as the proportion of relevant occurrences among all examples retrieved. **Recall,** also known as sensitivity, is the proportion of recovered occurrences among all relevant instances. Precision and recall are both equal to one for a perfect classifier.

The **Area Under the Curve (AUC)** is a measure of a classifier's ability to discriminate between classes and is used to summarize the ROC curve. The greater the AUC, the better the model's ability in differentiating between positive and negative classifications.

A **confusion matrix** is a tabular breakdown of a classifier's accurate and wrong predictions. It is used to assess the effectiveness of a categorization model.

a) Accuracy: Accuracy simply evaluates how frequently the classifier predicts correctly. It is the number of right guesses divided by the total number of forecasts.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

b) Precision: The ratio of the total number of successfully categorized positive classes divided by the total number of anticipated positive classes is defined as precision. Or, how much we predicted properly out of all the predictive positive classifications. Precision is essential (ideally 1).

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{\text{Predictions Actually Positive}}{\text{Total Predicted positive}}$$

c) Recall: The ratio of the total number of correctly categorized positive classes divided by the total number of positive classes is known as recall. Or, how much of the positive classes have we successfully predicted? The recall rate should be high (ideally 1).

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{\text{Predictions Actually Positive}}{\text{Total Actual positive}}$$

d) F1-Score: The F1 score, which ranges from 0 to 1, is the harmonic mean of accuracy and recall. The F1 score attempts to strike a balance between precision and recall for your classifier. If your accuracy is poor, your F1 score is low, and if your recall is low, your F1 score is low again.

$$\text{F1-Score} = 2^* \frac{(\text{Recall}^*\text{Precision})}{(\text{Recall} + \text{Precision})}$$

Conclusion

Based on different supervised classification algorithms implemented and various evaluation metrics calculated to measure the performance, we can make the following observations:

1. Accuracy score seems to be high for the support vector machine and the Gaussian Naïve Bayes.
2. The area under the curve is also high for the support vector machine and the Gaussian Naïve Bayes
3. The Random Forest Classifier performs poorly with the least accuracy score and the least area under the curve.
4. Even the false positives and false negatives are quite high for the random forest classifier, which leads to poor performance.