DATA MINING TECHNIQUES

-Comprehensive analysis on Naïve Bayes, Decision tree and KNN algorithms

Introduction

Data mining is the process of deriving useful and actionable insights from very large sets of data which normally couldn't be processed by traditional data processing methodologies. It is the process by which organizations discover patterns and other useful information relevant to their business needs. There are numerous different kinds of data mining techniques that help make use of raw data into actionable insights, such as classification, regression, clustering, prediction and so on. In our research, we shall focus mainly on classification/prediction data mining techniques namely Naïve Bayes, Decision tree algorithm and K-Nearest Neighbour algorithm. Classification algorithms involve analyzing attributes related to different types of data and studying the class under which data falls into. Not only shall we analyze the working of these models on a given dataset, but also conduct a comparative analysis to discover which of these algorithms is best suited for the given dataset.

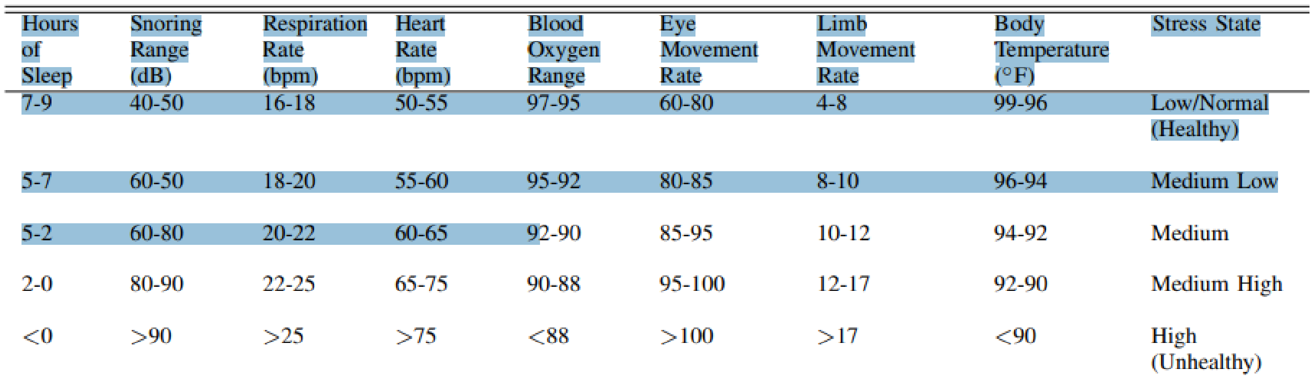
In our case study, we shall go through the following steps:

- Import all the required python libraries and dataset
- Text pre-processing of the dataset to be able to perform different operations
- Create an array for input features and an array for output feature (separate array for attributes and class labels)
- Checking whether the dataset is balanced or not
- Naïve Bayes implementation, confusion matrix and its evaluation scores
- Decision tree implementation, confusion matrix and its evaluation scores
- K-Nearest Neighbors implementation, confusion matrix and its evaluation scores
- Comparison between the three algorithms
- Visualization
- The data after processed

1. The Dataset

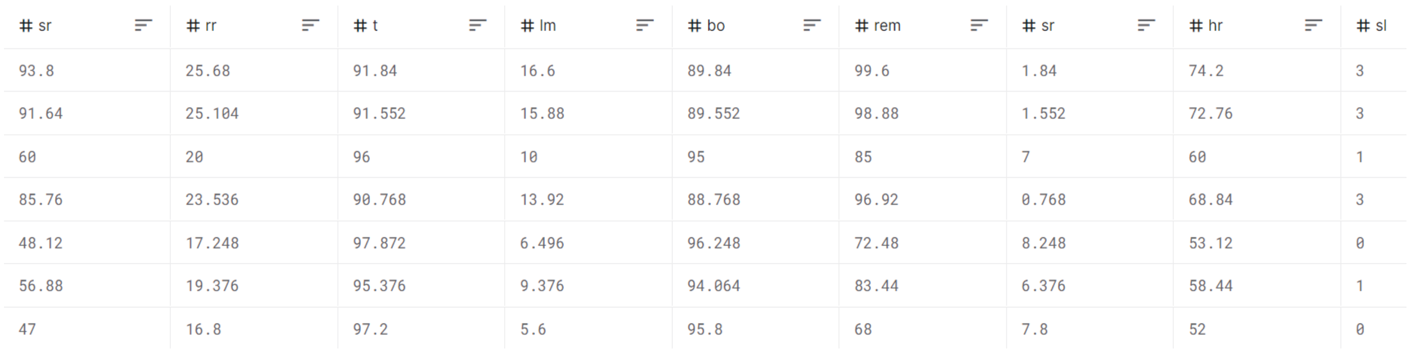
1.1 About the dataset:

Smart-Yoga Pillow or SaYoPillow is an edge device (a pillow) that monitors the relationship between stress and sleep. It analyzes the physiological changes occurring during one's sleep and uses it to predict the stress for the following day. The dataset we will process contains attributes and class labels in the csv format. In SayoPillow.csv, you will see the relationship between the parameters- snoring range of the user, respiration rate, body temperature, limb movement rate, blood oxygen levels, eye movement, number of hours of sleep, heart rate and Stress Levels (0- low/normal, 1 – medium low, 2- medium, 3-medium high, 4 -high). Therefore, we have nine data features with five class labels. The original dataset looks like the following:



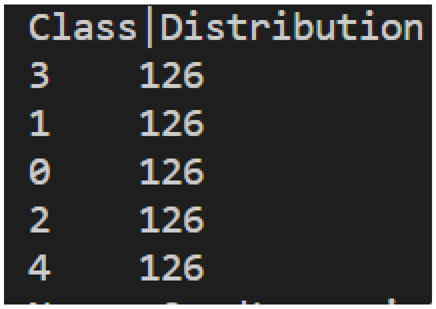
Hours of Sleep	Snoring Range (dB)	Respiration Rate (bpm)	Heart Rate (bpm)	Blood Oxygen Range	Eye Movement Rate	Limb Movement Rate	Body Temperature (°F)	Stress State
7-9	40-50	16-18	50-55	97-95	60-80	4-8	99-96	Low/Normal (Healthy)
5-7	60-50	18-20	55-60	95-92	80-85	8-10	96-94	Medium Low
5-2	60-80	20-22	60-65	92-90	85-95	10-12	94-92	Medium
2-0	80-90	22-25	65-75	90-88	95-100	12-17	92-90	Medium High
<0	>90	>25	>75	<88	>100	>17	<90	High (Unhealthy)

The corresponding csv format in the numerical data form looks like the following:



# sr	# rr	# t	# lm	# bo	# rem	# sr	# hr	# sl
93.8	25.68	91.84	16.6	89.84	99.6	1.84	74.2	3
91.64	25.104	91.552	15.88	89.552	98.88	1.552	72.76	3
60	20	96	10	95	85	7	60	1
85.76	23.536	90.768	13.92	88.768	96.92	0.768	68.84	3
48.12	17.248	97.872	6.496	96.248	72.48	8.248	53.12	0
56.88	19.376	95.376	9.376	94.064	83.44	6.376	58.44	1
47	16.8	97.2	5.6	95.8	68	7.8	52	0

The dataset has a size of 32.49 kB. We also need to check whether the dataset is balanced or not so that it will not produce wrong accuracy results. Using the Python library SMOTE, we can obtain the distribution of data in the five classes and the results are shown below:



```
Class|Distribution
3     126
1     126
0     126
2     126
4     126
```

Therefore, we find that the dataset is now perfectly balanced, has eight features namely #sr, #rr, #t, #lm, #bo, #rem, #sr, #hr along with the class feature #sl which has 5 class labels, is in csv format and also in the numeral data form. Now the data is ready for preprocessing.

1.2 Pre-processing of the dataset:

We use readily made python libraries instead of developing the code from scratch. The libraries used here are numpy, pandas, SMOTE, scikit-learn, matplotlib.pyplot and mlxtend.plotting by pip installing. The very first step of the pre-process is to import our dataset. We load the dataset as follows:

```
import pandas as pd

dataset = pd.read_csv('SaYoPillow.csv')

dataset.head()
```

We then separate the attributes and the class feature of the dataset into x and y arrays respectively:

```
x = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values
```

Next, we use SMOTE library to check if the classes are balanced or not:

```
X_smote, y_smote = SMOTE().fit_resample(x, y)

X_smote = pd.DataFrame(X_smote)

y_smote = pd.DataFrame(y_smote)

y_smote.iloc[:, 0].value_counts()
```

We find that the dataset is perfectly balanced. Split the dataset into training and testing data for both x and y:

```
X_train, X_test, y_train, y_test = train_test_split(X_smote, y_smote, test_size =
0.2, random_state = 0)
```

Here X_train is the training dataset for the array x (array of all feature values)

X_test is the testing dataset to test our models for the array x

y_train is the training dataset of array y (array of all class labels-0, 1, 2, 3, 4)

y_test is the testing dataset to test the models of array y.

This completes the data pre-processing stage. We can now implement our different models and evaluate them.

2. Naïve Bayes Implementation

Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that a given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class and is thus classified.

2.1 Fitting the dataset to the model

To implement the algorithm, we import the GaussianNB built-in class from the scikit-learn library. The GaussianNB method performs the required Naïve Bayes implementation that we seek. In order to fit the training sets to this method, we assign a variable called classifier to GaussianNB. Then we fit the training sets (x and y) as follows:

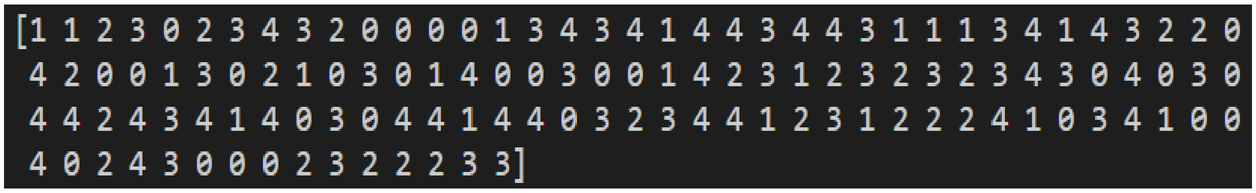
```
classifier = GaussianNB()

classifier.fit(X_train, y_train)
```

2.2 Predicting the Output

Now that we have fit the sets, we shall predict the output (y) for the given data (x). Here, we have stored the predicted output in the array called y_pred_NB:

```
y_pred_NB = classifier.predict(X_test)
```

The predicted output values for the given data-X_test, will be as follows:


```
[1 1 2 3 0 2 3 4 3 2 0 0 0 0 1 3 4 3 4 1 4 4 3 4 4 3 1 1 1 3 4 1 4 3 2 2 0
 4 2 0 0 1 3 0 2 1 0 3 0 1 4 0 0 3 0 0 1 4 2 3 1 2 3 2 3 2 3 4 3 0 4 0 3 0
 4 4 2 4 3 4 1 4 0 3 0 4 4 1 4 4 0 3 2 3 4 4 1 2 3 1 2 2 2 4 1 0 3 4 1 0 0
 4 0 2 4 3 0 0 0 2 3 2 2 2 3 3]
```

2.3 Confusion Matrix - Naïve Bayes

So, we have trained the algorithm using X_train and y_train datasets. Then we produced output (y_pred_NB) using the X-test dataset. Next, we have to compare the produced results with the actual results we have which is y_test, to analyze how efficient was the algorithm. This is achieved through the use confusion matrix. It is one of the common ways to evaluate how efficient a given model/algorithm is. We import confusion_matrix from the scikit-learn library and pass the actual output (y_test) and predicted output (y_pred_NB) and store the value in cm_NB.

```
cm_NB = confusion_matrix(y_test,y_pred_NB)
```

To plot the confusion matrix, we `import matplotlib.pyplot as plt`

```
fig, ax = plot_confusion_matrix(conf_mat=cm_NB)
plt.show()
```

The confusion matrix for the Naïve Bayes model is as follows:

The confusion matrix is used to evaluate models by calculating accuracy, precision, recall and f-scores of the model.

2.4 Evaluation Scores

Before we calculate evaluation values, we obtain TP – True positives, FP – False Positives, TN – True Negatives, FN – False Negatives from the matrix:

```
TP_NB=29
```

```
TN_NB=19+22+28+28

FP_NB=0

FN_NB=0
```

We can mathematically calculate accuracy, precision, recall and f-scores using the following formulas:

Accuracy_NB = (TP_NB + TN_NB) / (TP_NB + TN_NB + FP_NB + FN_NB)

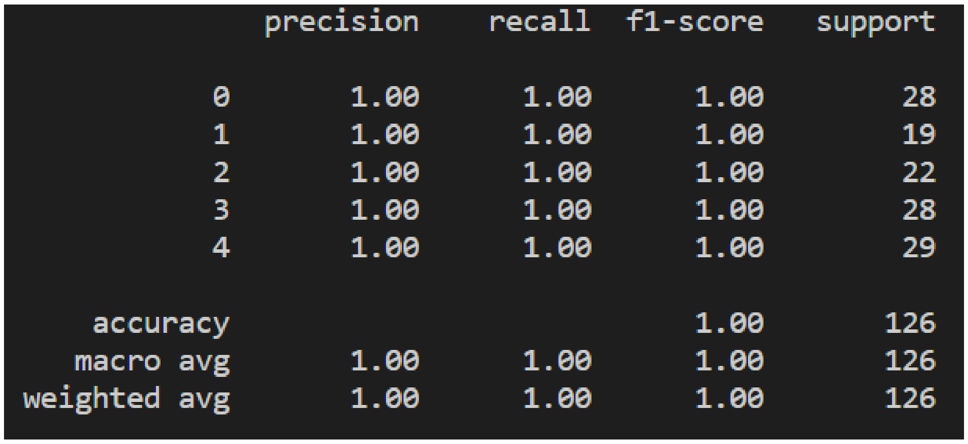
Precision_NB = TP_NB / (TP_NB + FP_NB)

Recall_NB = TP_NB / (TP_NB + FN_NB)

F1_Score_NB = 2 * Precision_NB * Recall_NB / (Precision_NB + Recall_NB)

However, we can also use python libraries to output a matrix that shows all values. The actual python code attatched with the project demonstrates both ways of calculating these measures. Following is the classification report for the Naïve Bayes by importing classification_matrix from scikit-learn.

```
print(classification_report(y_test, y_pred_NB))
```



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        19
           2       1.00      1.00      1.00        22
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        29

    accuracy                           1.00       126
   macro avg       1.00      1.00      1.00       126
weighted avg       1.00      1.00      1.00       126
```

Accuracy=1   Precision=1   Recall=1   f1-score=1

As we can see, all measures hold a value of 1, indicating a perfect score for the model. i.e. The model is 100% efficient.

3. Decision Tree Implementation

The next algorithm used is Decision Tree which utilizes certain parameters to continuously split the data so that it can be better classified. Just like how we implemented Naive Bayes, we can simply change some values in the imported values and implement the model easily.

3.1 Fitting the Dataset

We will use another type of classifier to implement this model by importing DecisionTreeClassifier.

```
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
```

After defining the classifier, we fit the data using:

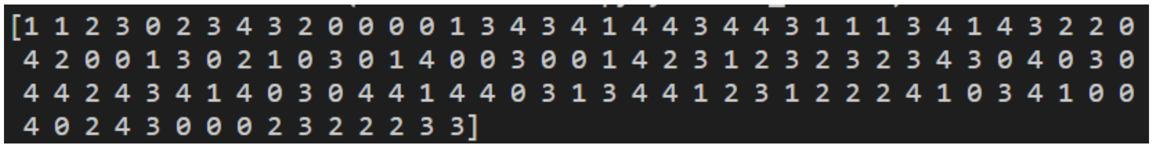
```
classifier.fit(X_train, y_train)
```

3.2 Predict the Output

We then predict the output of the decision tree model using:

```
y_pred_DT = classifier.predict(X_test)
```

The predicted output is as follows:



```
[1 1 2 3 0 2 3 4 3 2 0 0 0 0 1 3 4 3 4 1 4 4 3 4 4 3 1 1 1 3 4 1 4 3 2 2 0
 4 2 0 0 1 3 0 2 1 0 3 0 1 4 0 0 3 0 0 1 4 2 3 1 2 3 2 3 2 3 4 3 0 4 0 3 0
 4 4 2 4 3 4 1 4 0 3 0 4 4 1 4 4 0 3 1 3 4 4 1 2 3 1 2 2 2 4 1 0 3 4 1 0 0
 4 0 2 4 3 0 0 0 2 3 2 2 2 3 3]
```

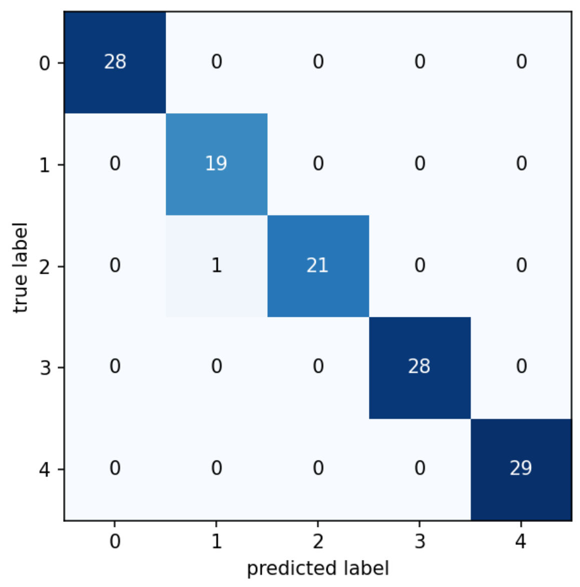
3.3 Confusion Matrix

We have seen the confusion matrix for Naïve Bayes Algorithm and found that the model is completely efficient. We will now see how the confusion matrix for the Decision Tree Algorithm

```
cm_DT = confusion_matrix(y_test, y_pred_DT)
```

We again use the plot_confusion_matrix library to produce the confusion matrix

```
fig, ax = plot_confusion_matrix(conf_mat=cm_DT)

plt.show()
```



If we carefully compare all the values of this matrix to the matrix of Naïve Bayes, we find that just one value is different. It is likely that the Decision Tree Algorithm is less efficient than the previous one, as it had a perfect score.

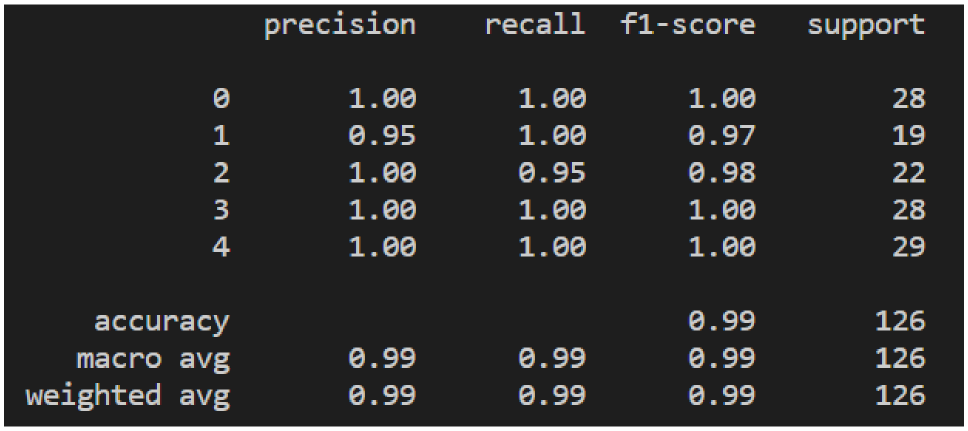
2.4 Evaluation Scores

From the confusion matrix generated, we obtain the TP, FP, FP, FN values

```
TP_DT = 29 #True Positives (Decision Tree)

TN_DT = 19+1+21+28+28 #True Negatives (Decision Tree)

FP_DT = 0 #False Positives (Decision Tree)

FN_DT = 0 #False Negatives (Decision Tree)
```

The formulas for finding accuracy, precision, recall and f-scores remain the same for all models. Here, the evaluation scores can be demonstrated using:

```
print(classification_report(y_test, y_pred_DT))
```


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       0.95      1.00      0.97        19
           2       1.00      0.95      0.98        22
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        29

    accuracy                           0.99       126
   macro avg       0.99      0.99      0.99       126
weighted avg       0.99      0.99      0.99       126
```

Therefore, for the Decision Tree Algorithm, we find that

Accuracy=0.99 Precision=0.99 Recall=0.99 f1-score=0.99

4. K-Nearest Neighbor Implementation

Also known as KNN Algorithm is a supervised learning algorithm which is also very common and basic. This model is also referred to as lazy algorithm due to its lack of looking solutions beyond a certain threshold. This model can be used for both classification and regression. KNN works by calculating the distances between a query and all of the instances in the data, picking the K closest examples to the query, and then voting for the most frequent label for classification or average the labels for regression.

4.1 Fitting the Dataset

We use the imported python class **KNeighborsClassifier** from sklearn.neighbors library. The classifier is defined as:

```
classifier = KNeighborsClassifier(n_neighbors=5)
```

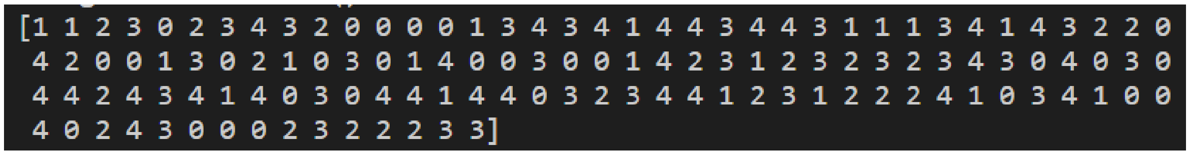
This class has only one argument, which is n_neighbors. This is essentially K's value. Although there is no optimal value for K, it is chosen after testing and assessment, 5 appears to be the most generally utilized value for the KNN algorithm to begin with. Then we fit the dataset:

```
classifier.fit(X_train, y_train)
```

4.2 Predict the Output

The output for the test data is predicted using:

```
y_pred_KNN = classifier.predict(X_test)
```



```
[1 1 2 3 0 2 3 4 3 2 0 0 0 0 1 3 4 3 4 1 4 4 3 4 4 3 1 1 1 3 4 1 4 3 2 2 0
 4 2 0 0 1 3 0 2 1 0 3 0 1 4 0 0 3 0 0 1 4 2 3 1 2 3 2 3 2 3 4 3 0 4 0 3 0
 4 4 2 4 3 4 1 4 0 3 0 4 4 1 4 4 0 3 2 3 4 4 1 2 3 1 2 2 2 4 1 0 3 4 1 0 0
 4 0 2 4 3 0 0 0 2 3 2 2 2 3 3]
```

4.3 Confusion Matrix

Using the following lines of code, we generate Confusion matrix for KNN

```
cm_KNN=confusion_matrix(y_test, y_pred_KNN)

fig, ax = plot_confusion_matrix(conf_mat=cm_KNN)

plt.show()
```

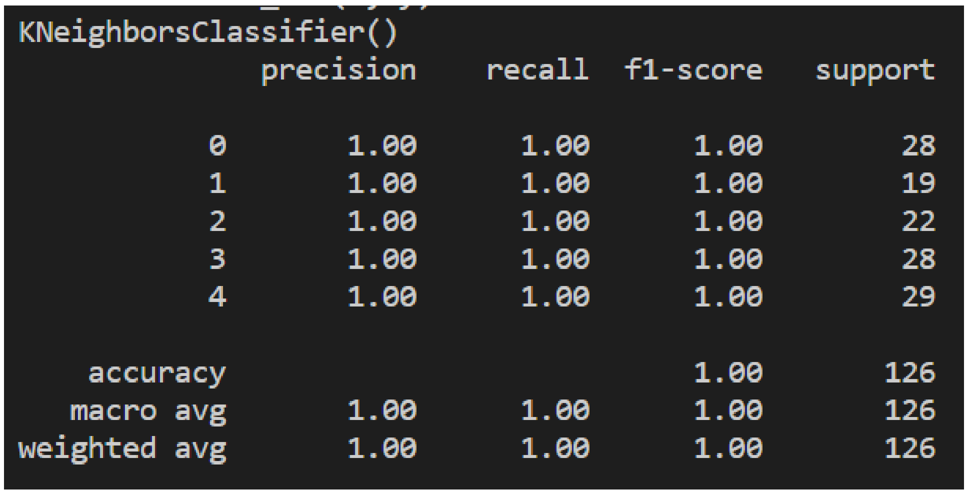
predicted label

Just like Naïve Bayes, KNN also have a similar confusion matrix, which means all the evaluation scores will also be equal.

4.4 Evaluation Scores

```
TP_KNN=29

TN_KNN=19+22+28+28

FP_KNN=0

FN_KNN=0

print(classification_report(y_test, y_pred_KNN))
```

This line of code will generate all evaluation scores



```
KNeighborsClassifier()
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        19
           2       1.00      1.00      1.00        22
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        29

    accuracy                           1.00       126
   macro avg       1.00      1.00      1.00       126
weighted avg       1.00      1.00      1.00       126
```

Therefore, Accuracy=1 Precision=1 Recall=1 F-scores=1

By this, we have implemented all three models. Now we shall visualize the results and compare which model suits the dataset the best.

5. Visualization/ Results

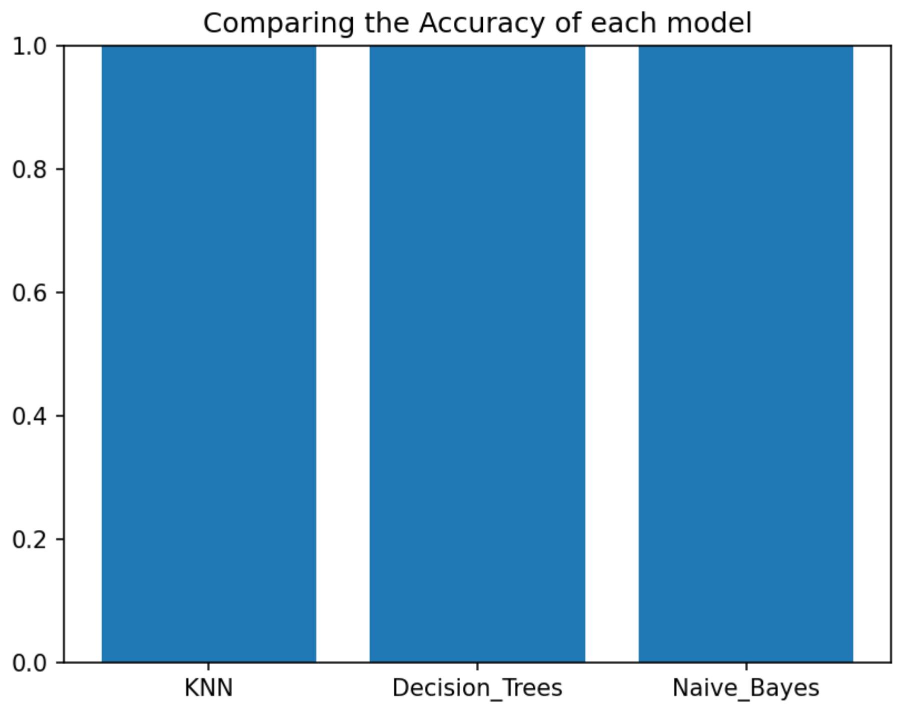
To be able to better understand and analyze the different scores of each algorithm used, we shall generate the results in bar-chart representations and compare altogether. Here, I have potted using python library *matplotlib.pyplot* as plt.

5.1 Accuracy Comparison

First, comparing the accuracies of all models, the following code is used:

```
Accuracy = [Accuracy_KNN, Accuracy_DT, Accuracy_NB]

Methods_acc = ['KNN', 'Decision_Trees', 'Naive_Bayes']

Accuracy_pos = np.arange(len(Methods_acc))

plt.ylim(0*.5,2*.5)

plt.bar(Accuracy_pos, Accuracy)

plt.xticks(Accuracy_pos, Methods_acc)

plt.title('Comparing the accuracy of each model')

plt.show()
```

The result is as follows:



KNN- 100%   Decision Tree – 100%   Naïve Bayes – 100%

In terms of Accuracy, all three models have perfect scores, so we don't rely on accuracy alone for the evaluation.

5.2 Comparing Precision

```
Precision = [Precision_DT, Precision_KNN, Precision_NB]

Methods_prec = [ 'Decision_Trees','KNN', 'Naive_Bayes']

precision_position = np.arange(len(Methods_prec))

plt.ylim(0*5,2*.5)

plt.bar(precision_position, Precision)

plt.xticks(precision_position, Methods_prec)

plt.title('Comparing the Precision of each model')

plt.show()
```

This generates the bar-chart for Precision as follows:

Decision Tree- 99%   KNN – 100%   Naïve Bayes – 100%

Here, we find that not all values are the same. KNN and Naïve Bayes have perfect scores while Decision tree has a lesser score. This indicates both KNN and Naïve Bayes are perfectly precise for the database.

5.3 Comparing Recall

```
Recall = [Recall_DT, Recall_KNN, Recall_NB]

Methods_rec = [ 'Decision_Trees','KNN', 'Naive_Bayes']

recall_position = np.arange(len(Methods_rec))

plt.ylim(0*.5,2*.5)

plt.bar(recall_position, Recall)

plt.xticks(recall_position, Methods_rec)

plt.title('Comparing the Recall of each model')

plt.show()
```

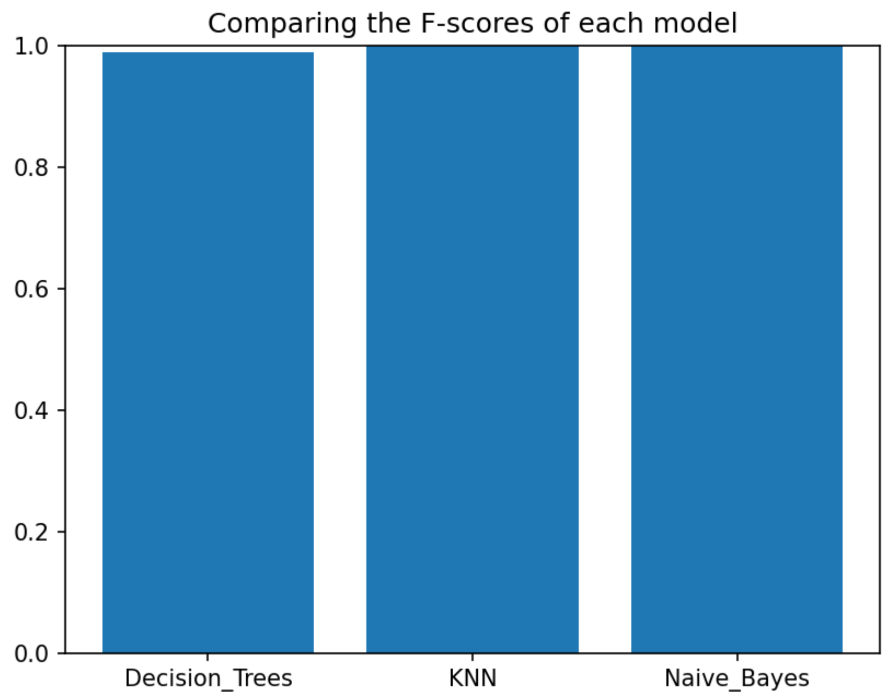
The results are as follows:

Decision Tree – 99% KNN – 100% Naïve Bayes – 100%
Similar to the precision values, Recall also has similar scores for all models.

5.4 Comparing F-Scores

```
F1_Score = [F1_Score_DT, F1_Score_KNN, F1_Score_NB]

Methods_f1 = [ 'Decision_Trees','KNN', 'Naive_Bayes']

f1_score_position = np.arange(len(Methods_f1))

plt.ylim(0*.5,2*.5)

plt.bar(f1_score_position, F1_Score)

plt.xticks(f1_score_position, Methods_f1)

plt.title('Comparing the F-scores of each model')

plt.show()
```

The f-scores are the scores which we rely on to decide the overall evaluation of a model. It is actually the best predictor of which model is the best amongst the chosen three. It defines a relationship between precision and recall of a particular model. The results are as follows:



We see that except for Decision tree, the models have perfect scores showing that they are 100% efficient.

This concludes the prediction of the three models and their visualization.

6. CONCLUSION

Although the accuracies of the three models had a perfect 100% evaluation, this factor alone cannot identify the overall evaluation of a particular model. We see how precise a model can be through the precision scores, and take this also into consideration for the evaluation. Then we calculate the Recall score, define the relationship between recall and precision through f-score and decide which model is the best

Therefore, in our project, for the given dataset, both KNN and Naïve Bayes model fit the best and has an evaluation score as 100%

REFERENCES

1. L. Rachakonda, A. K. Bapatla, S. P. Mohanty, and E. Kougianos, “SaYoPillow: Blockchain-Integrated Privacy-Assured IoMT Framework for Stress Management Considering Sleeping Habits”, IEEE Transactions on Consumer Electronics (TCE), Vol. 67, No. 1, Feb 2021, pp. 20-29.

2. L. Rachakonda, S. P. Mohanty, E. Kougianos, K. Karunakaran, and M. Ganapathiraju, “Smart-Pillow: An IoT based Device for Stress Detection Considering Sleeping Habits”, in Proceedings of the 4th IEEE International Symposium on Smart Electronic Systems (iSES), 2018, pp. 161--166.

3. Rohan Gupta. Nov 12, 2019.Towards Data Science. Towardsdatascience.com. https://towardsdatascience.com/

4. Scott Robinson. K-Nearest Neighbors Algorithm in Python and Scikit-Learn .StackAbuse. https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/

5. Dr. Emmanuel Tsukerman.June-17,2020. Explore Python Libraries: Imbalanced-learn.pluralsight. https://www.pluralsight.com/guides/explore-python-libraries:-imbalanced-learn