INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

DEPARTMENT OF
INDUSTRIAL AND MANAGEMENT ENGINEERING

IME672A

DATA MINING AND KNOWLEDGE DISCOVERY

Course Instructor - Dr. Faiz Hamid

Deodorant Instant Liking Data

Group No. 10

Satyansha Dev	200897
Shreyansh Pachauri	200954
Varun Pant	201093
Ravi Prakash Kumar	20103095
Ravi Kumar Vishwakarma	21105087

Table of Contents

Problem Statement and Knowing the Data Set

- The data has survey results of 5 different deodorants with 63 attributes containing several information.
- The target variable used for classifying the data is ‘Instant.Liking’.
- We are provided with a training data set containing 2500 tuples.
- Further we have been provided with a new unseen data set to make predictions for 5105 unclassified tuples.
- We intend to develop the model on the training data set and then use it to make predictions for the unclassified data.
- In the Jupyter Notebook we have written the code to visualize the Basic Statistical Description values (min, max, quartiles, mean, median, SD), Histograms, Box Plots and Violin Plots for the data.

Data Preprocessing

Data Cleaning

Removing Duplicate Tuples

There are no identical rows/data objects in this data

Attributes with missing values of data

There are 8 attributes ['q8.8', 'q8.9', 'q8.10', 'q8.12', 'q8.13', 'q8.17', 'q8.18', 'q8.20'] with missing data values. It is observable from the data that for all such attributes the number of missing values is nearly 80% of the data. So the better option is that we drop the columns corresponding to these attributes.

Irrelevant Attributes

There are 10 attributes ['q8.1', 'q8.5', 'q8.6', 'q8.11', 'q7', 'q8.2', 'q8.7', 's13.2', 'q8.19', 's7.involved.in…'] which are present in the classified training data set but are absent in the unclassified unseen data set for which we have to make the predictions. In that case these attributes will have no use in classifying the unseen data, and should therefore be dropped while building the model.

Dealing with the Outliers

Trying to remove the outliers

There are 7 attributes with respect to which there are outliers in the data as seen from the box plots. The count of all such data objects that are outliers with respect to one or more attributes is 1055. Removing all of them will lead to huge data loss, and poor training in the model formed thereafter. So we cannot drop the outliers in this data set.

Clustering to replace the outliers

Clustering is not a good way of working with this data-set because all the attributes are nominal/categorical/ordinal so if we try to visualize the clusters in plots, we will only get discrete points. Although we can get the count of data objects at each point in the plot and can further claim that those points with a large number of data objects are clusters and the rest are outliers, this is not a smart way of doing this. Even a decision tree can build a rule for classifying this way. So we won't apply clustering for replacing the outliers

Inference based approach to replace the outliers using Logistic Regression

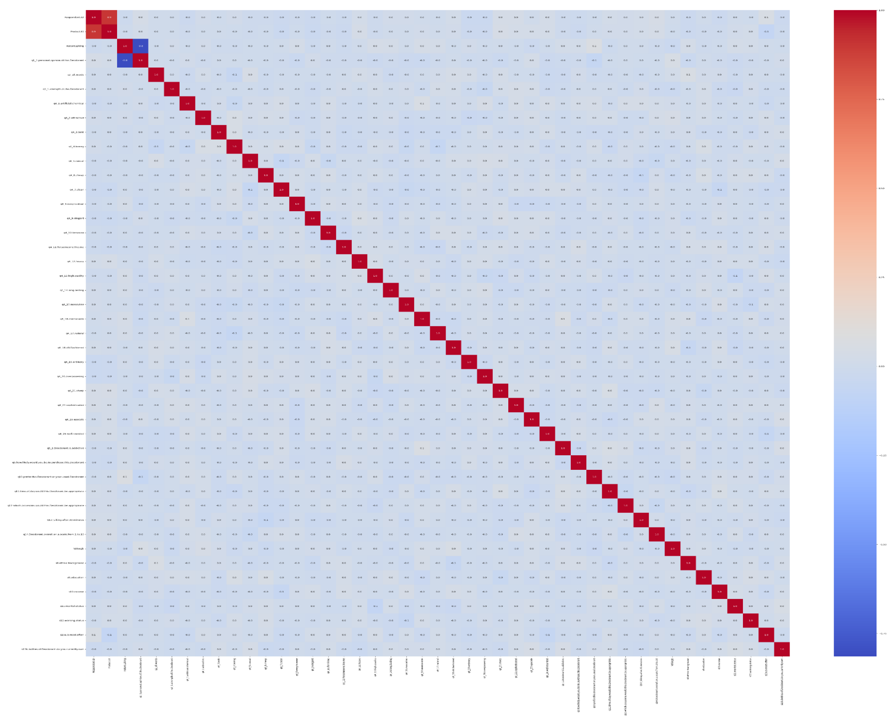
We try to create a logistic regression model on the part of the data set excluding the outliers taking the attribute with respect to which there are outliers as target class. We intend to thereafter use it to predict the values for the objects that are outliers. But while we are preparing the models we find that the accuracy of these models is quite low and due to high class imbalance in the attribute values, minority classes are heavily misclassified. Therefore we cannot even use this approach to remove the outliers.

Data Reduction

Correlation Analysis of Attributes using Chi-square test and Pearson's correlation coefficient

Both the tests indicate that there is no significant correlation between any two attributes in the data set.

The heat map of Pearson's Correlation coefficients for all pairs of attributes aside indicates the same.



Removing the Respondent ID

Respondent ID attribute is just like a serial number assigned to each data tuple. It takes all distinct values and has a role in the determination of the target variable's class. So we are dropping this.

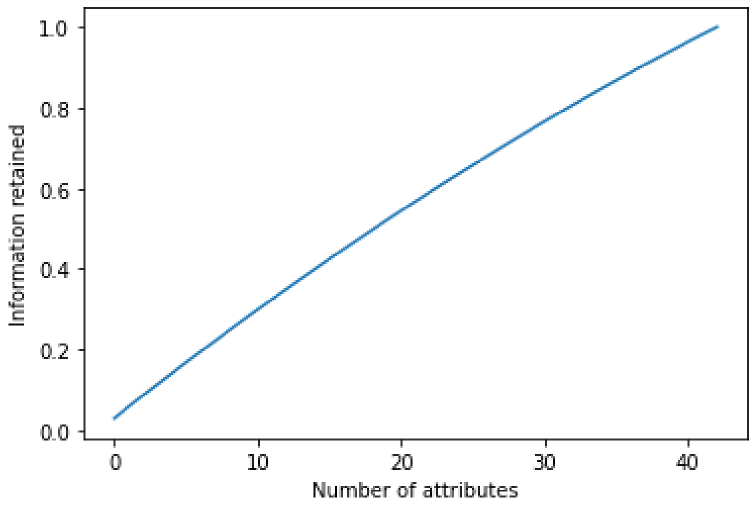
Dimensionality reduction using Principal Component Analysis

We tried applying Principal Component Analysis to determine new attributes from the combination of given attributes. But we are not able to find even a suitable combination of attributes to determine the

PC-1, which accounts for the most variance in the data and along which the data is most spread out. We visualize that for the given data all the PCs are of equal variance.

The diagram aside indicates that in order to retain high variance in the data set we need to retain more and more attributes. Reduction in attributes would result in loss of variance, hence loss of information.

Therefore attribute reduction is not an appropriate thing to be done.



Dealing with Class Imbalance

Imbalanced datasets are those where there is a severe skew in the class distribution in the minority class to the majority class. Here it can be seen the Class Imbalance in the Data with the ratio of 3:1 (Instant_dislike : Instant_like). Due to class imbalance problem our model will be trained better for the majority class as compared to the minority class. So the class imbalance problem needs to be balanced before proceeding to model building. Here we have to resample the data. Resampling involves creating a new transformed version of the training dataset in which the selected examples have a different class distribution. For this undersampling and oversampling methods are used here.

Undersampling

Here we randomly eliminate the tuples from the majority class until there are an equal number of Instant like and dislike tuples. So our target here is to reduce the dataset from the majority class to make equal samples of data corresponding to liking and disliking both. For this we have used groupby and RandomUnderSampler methods.
Drawback : Sample size is reduced. Random undersampling deletes examples from the majority class and can result in losing information invaluable to a model. So this is not a good way of handling the class imbalanced data.

Oversampling

Resamples the minority tuples so that the resulting training set contains an equal number of Instant like and dislike tuples. Random oversampling duplicates the minority class. So here we will be using the oversampled data.

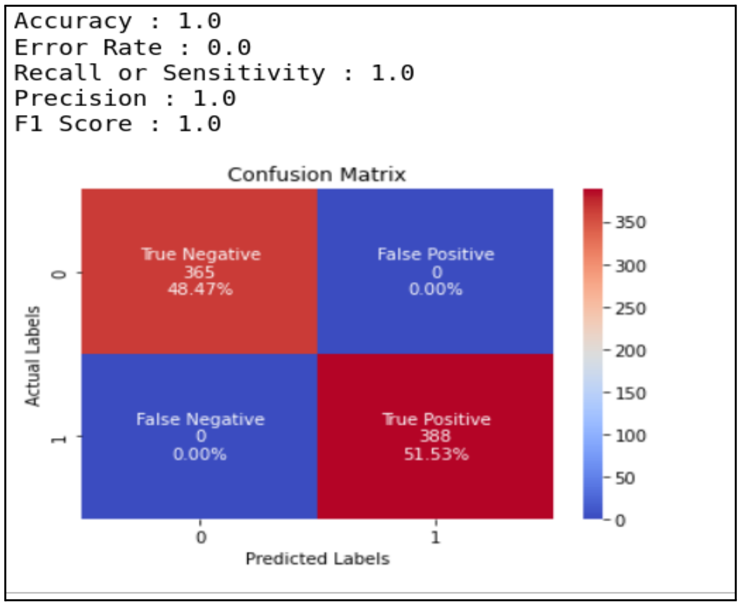
Hyper-Parameter Tuning

Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a model argument whose value is set before the learning process begins. The key to machine learning algorithms is hyperparameter tuning, which selects the optimal value for the hyperparameter based on the training data set so that the learning algorithm works at its best in building a model over the data.

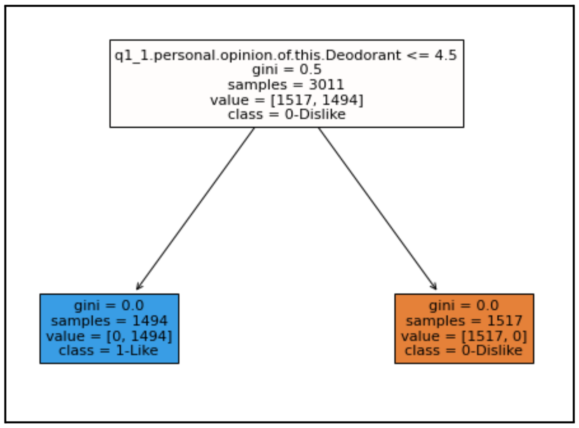
Basic Classification Models

Decision Tree

After hyperparameter tuning we got the best hyperparameter as **'max_depth': 2, 'min_samples_leaf': 1, 'min_samples_split': 2** .
The tree we obtained and is plotted below. The confusion matrix and the accuracy results we obtained for the model on test ata are as aside.



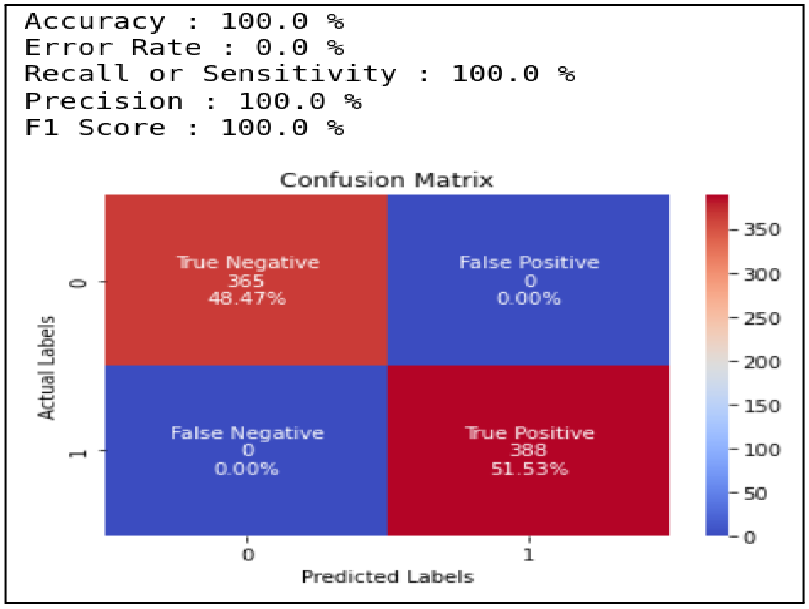
```
Accuracy : 1.0
Error Rate : 0.0
Recall or Sensitivity : 1.0
Precision : 1.0
F1 Score : 1.0
```



Random Forest

After hyperparameter tuning we got the best hyperparameter as **{'bootstrap': True, 'max_depth': 80, 'max_features': 2, 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 200}**

The confusion matrix and the accuracy results we obtained for the model on test data are as aside:

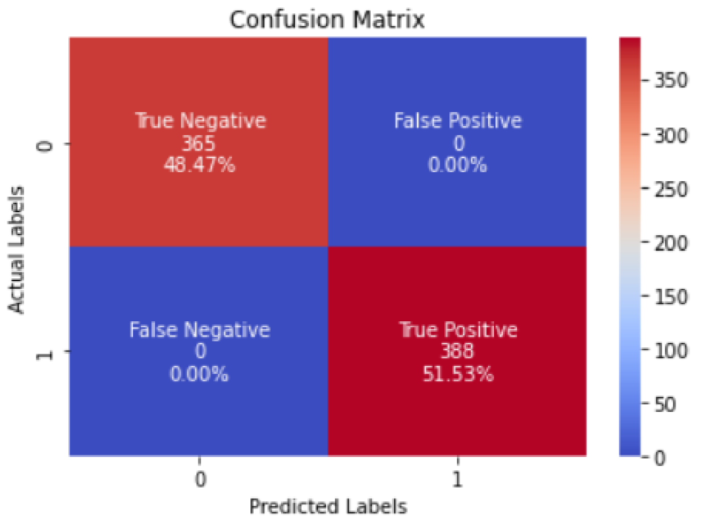


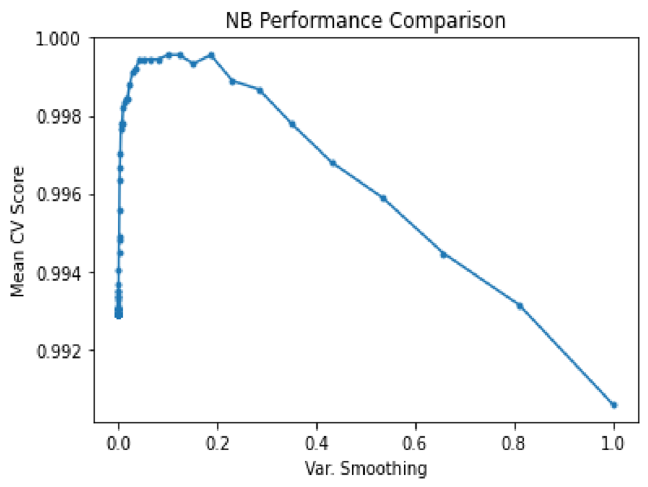
```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %
```

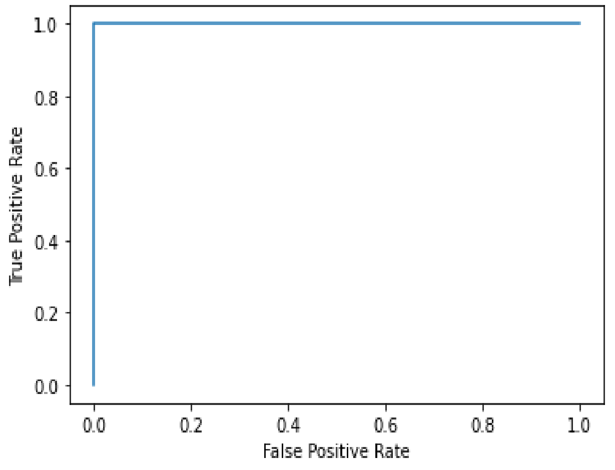
Naive Bayes Classifier

Applying hyper parameter tuning we find that the best obtained hyper parameter is **{'var_smoothing': 0.123}** corresponding to the **mean CV score of 0.99955**. The trade-off between mean CV score and the Var. Smoothening can be seen in the figure aside, obtained during Hyper Parameter Tuning. We can view confusion matrix, accuracy results and the ROC curve as below.

```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %
```





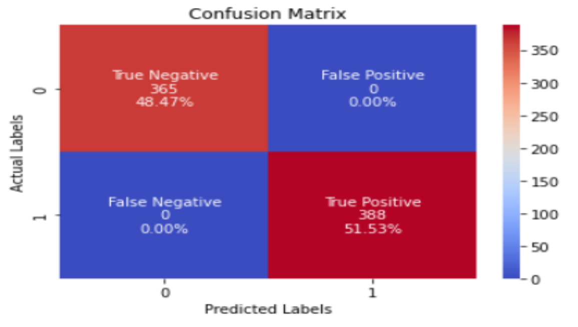


Support Vector Machines

After hyperparameter tuning we got the best hyperparameter as **{'C': 0.1, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 1, 'kernel': 'linear', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}**

The confusion matrix and the accuracy results we obtained for the model on test data are as aside:

```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       365
           1       1.00      1.00      1.00       388

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753
```

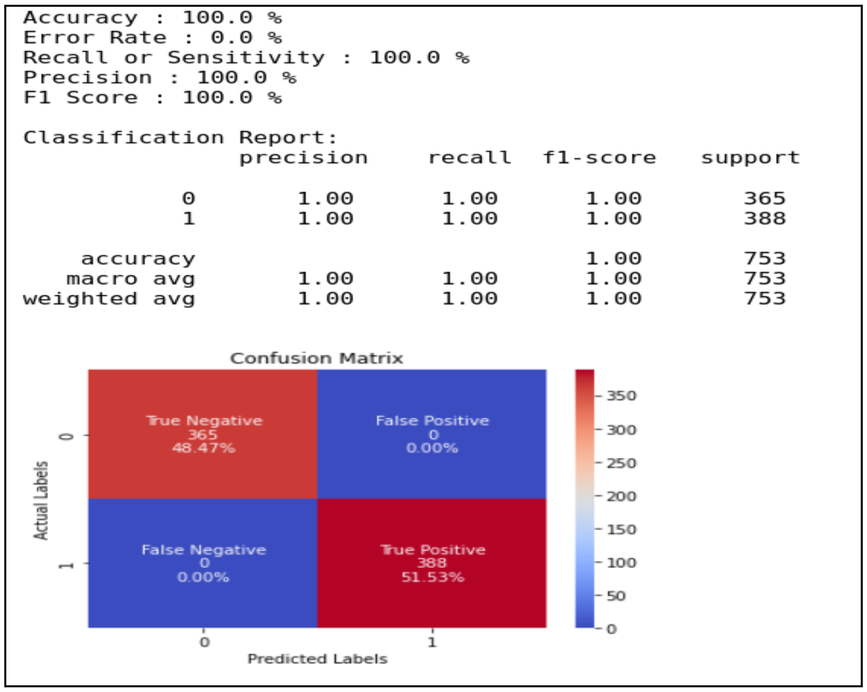


Logistic Regression

After hyperparameter tuning we got the best hyperparameter as

C=0.0018329807108324356,
max_iter=10,
penalty='l1',
solver='saga'

The confusion matrix and the accuracy results we obtained for the model on test data are as aside:



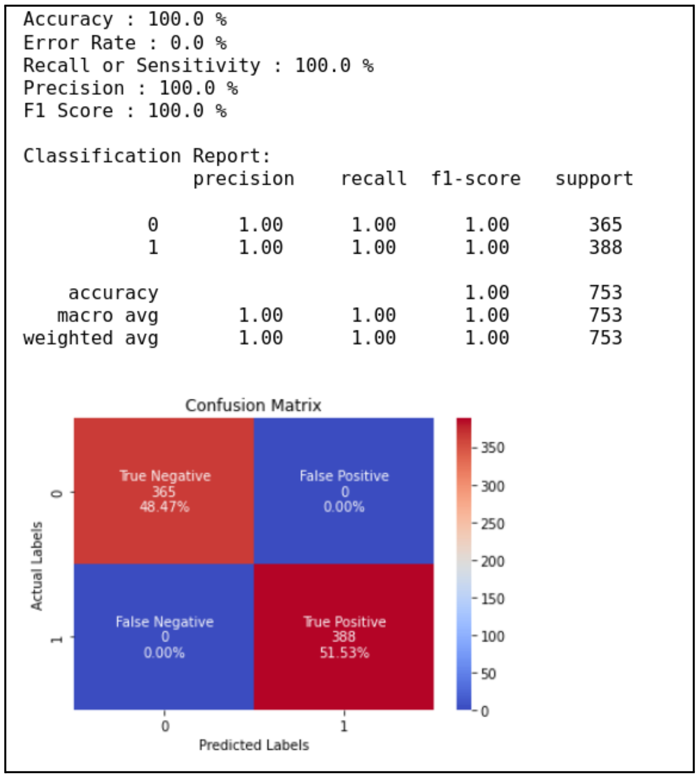
```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       365
           1       1.00      1.00      1.00       388

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753
```

Ensemble Methods

Maximum Voting Classifier

Hard Voting

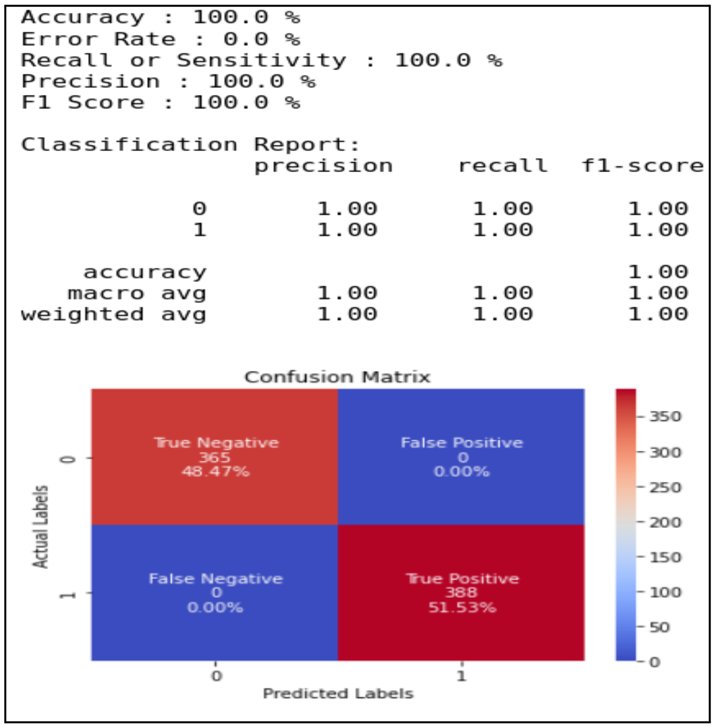
In hard voting, the predicted output class is a class with the highest majority of votes i.e the class which had the highest probability of being predicted by each of the classifiers.
The confusion matrix and the accuracy results we obtained for the model on test data are as aside:



```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       365
           1       1.00      1.00      1.00       388

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753
```

Soft Voting

The output class is the prediction based on the average of probability given to that class.
The confusion matrix and the accuracy results we obtained for the model on test data are as aside:



```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %

Classification Report:
              precision    recall  f1-score

           0       1.00      1.00      1.00
           1       1.00      1.00      1.00

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

Averaging Methods

Applying this method we were able to visualize that increasing the number of models would result in decreasing in the variance of the predictions on the test data, thereby indicating that the overfitting gets reduced.

Number of models used	Average RMSE Score on 5 fold cross validation
4	0.1410607398995643
3	0.2465098829570221
2	0.39441581273123544

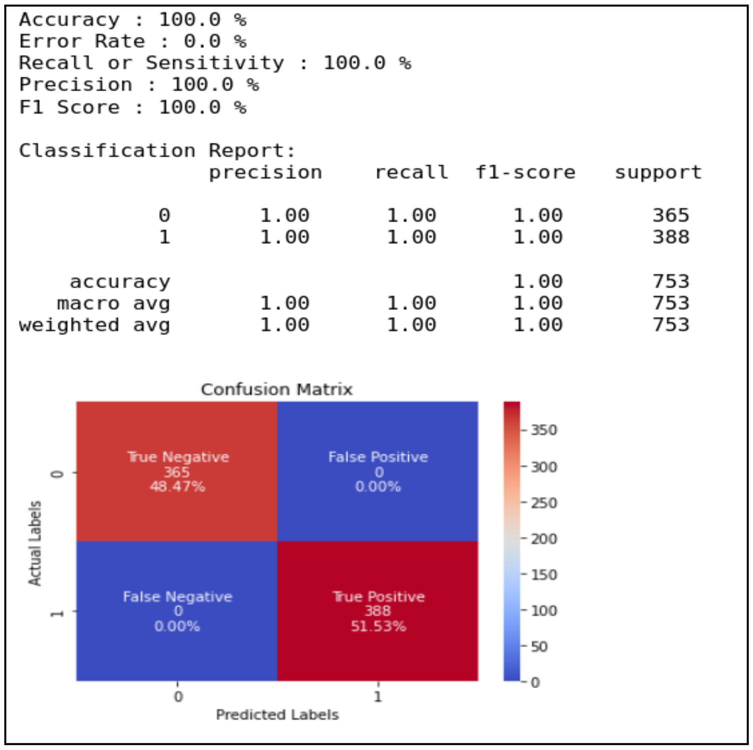
Bagging

Bagging technique uses these subsets, created from the original dataset using bootstrap to train models using the same learning algorithm. The final predictions are determined by combining the predictions from all the models.
Applying this method we observe that there is a decrease in the variance of the predictions on the test data, thereby indicating that the overfitting gets reduced. The code output below indicates the same.

```
Mean of: 1.000, std: (+/-) 0.000 [DecisionTreeClassifier]
Mean of: 0.753, std: (+/-) 0.002 [Bagging DecisionTreeClassifier]

Mean of: 0.903, std: (+/-) 0.013 [RandomForestClassifier]
Mean of: 0.753, std: (+/-) 0.002 [Bagging RandomForestClassifier]
```

Boosting

AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly. Final prediction of class attribute is made by weighing over all the k-classifier's votes. The confusion matrix and the accuracy results we obtained for the model on test data are as aside:



```
Accuracy : 100.0 %
Error Rate : 0.0 %
Recall or Sensitivity : 100.0 %
Precision : 100.0 %
F1 Score : 100.0 %

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       365
           1       1.00      1.00      1.00       388

    accuracy                           1.00       753
   macro avg       1.00      1.00      1.00       753
weighted avg       1.00      1.00      1.00       753
```

Why are we getting 100% accuracy on all our models?

The reason why we are getting this is because the data set we have been provided with, has a direct correlation between the target variable and the attribute **‘q1_1.personal.opinion.of.this.Deodorant’.**

- Whenever the target variable Instant.Liking is 1, q1_1.personal.opinion.of.this.Deodorant takes the values 1, 2, 3 or 4.
- Whenever the target variable Instant.Liking is 0, q1_1.personal.opinion.of.this.Deodorant takes the values 5, 6 or 7.