Data Intensive Computing CSE 587

Project Phase 2

TITLE: FLIGHT SATISFACTION CLASSIFICATION

Name	UB id
Sriinitha Chinnapatlola	sriinith
Shravani Soma	ssoma

Problem Statement: We aim to consider various factors in predicting the passenger satisfaction for their flight. These factors include flight details like flight distance, inflight wifi services, food and drinks, seat comfort, departure and arrival delays, and many more. We also take customer's details into consideration for evaluating the model. These considerations include gender, age, and their loyalty. This results in a classification model which predicts if a customer would either be 'satisfied' or 'neutral or dissatisfied' based on all the above given factors.

1. Logistic Regression

Implementation code snippets:

Logistic Regression

```
In [33]: # LOGISTIC REGRESSION MODEL
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
y_pred = logistic_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy_percent_reg = accuracy*100
print('Accuracy achieved using Logistic Regression model:', accuracy_percent_reg)

Accuracy achieved using Logistic Regression model: 87.52717610113537
```

Fig: model fitting implementation code for Logistic Regression

Result Visualization code:

Result Visualizations for Logistic Regression Model

In [34]:
```
cm = confusion_matrix(y_test, y_pred)
binary_classes = ['satisfied', 'neutral or dissatisfied']
sns.heatmap(cm, annot=True, fmt='d', cmap='magma',xticklabels=binary_classes, yticklabels=binary_classes)
plt.title('Confusion Matrix for Logistic Regression')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
```

In [31]:
```
label_en = LabelEncoder()
y_test_transformed = label_en.fit_transform(y_test)
y_pred_transformed = label_en.transform(y_pred)
f_p, t_p, thresholds = roc_curve(y_test_transformed, y_pred_transformed); roc_auc = auc(f_p, t_p)
plt.plot(f_p, t_p, label=' AUC = %0.2f' % roc_auc)
plt.plot((0.0, 1.0), (0.0, 1.0), 'm-.')
plt.title('ROC Curve - Logistic Regression'); plt.xlabel('False Positive Rate (FPR)'); plt.ylabel('True Positive Rate (TPR)')
plt.xlim(0.0,1.0);plt.ylim(0.0, 1.0)
plt.legend(loc="lower right")
plt.show()
```

Fig: Result Visualization (confusion matrix and ROC) code for Logistic Regression

Reason for choosing the model:

The problem we choose is a binary classification problem (i.e. it predicts if the customer is “satisfied” or “neutral or dissatisfied” by their overall flight experience). The begin with, the simplest model to implement for a binary classification problem like our problem is “Logistic Regression”. Logistic Regression models are very easy to interpret understand and easy to implement.

Tuning/training the model:

Before the data is sent to the model for training, the data has been preprocessed. As the logistic regression model accepts only numeric data hence the categorical variables are converted to numeric and scaled before sending it to the model. After this step there are 21 columns (features) no features have been removed before training the model.

Model Effectiveness:

Accuracy is one of the metrics used to evaluate the performance of the model. For the chosen number of features and the preprocessing steps followed, the model achieved an accuracy of **87.52%.**

Another metric used to measure the effectiveness of the model on the dataset is the **confusion matrix.**

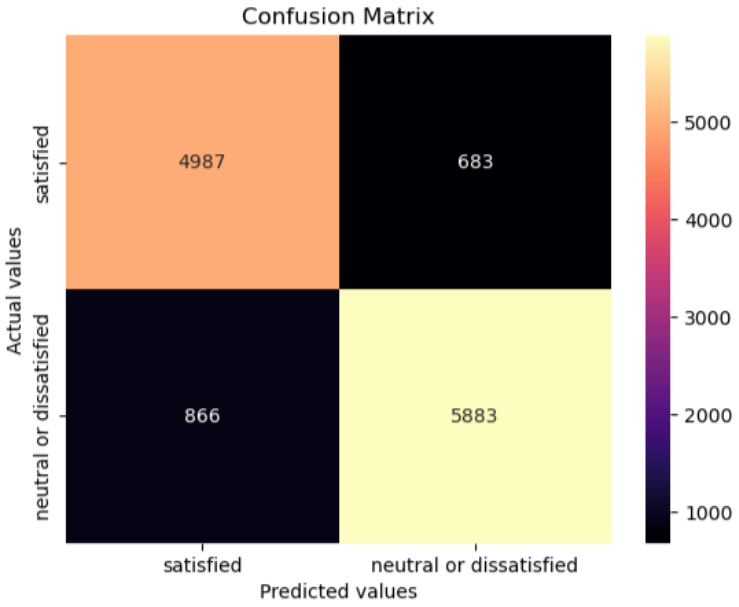


Fig: Heatmap of the confusion matrix for a logistic Regression model.

Analysis: The heat map of the confusion matrix is used to evaluate the performance of the Logistic Regression model. Since, our problem is a binary classification problem, there are 4 possible outcomes as mentioned above. From the above confusion matrix, we can conclude out of 5670 of the entries consisting of the target variable as "satisfied", 4987 values are correctly predicted as "satisfied" while the rest 683 are classified as "neutral or dissatisfied". And out of 6749 of the entries consisting of the target variable as "Neutral or Dissatisfied", 5883 values are correctly predicted as "Neutral or Dissatisfied" while the rest 866 are classified as "satisfied".

ROC curve: Another metric to evaluate the performance of the ROC curve. The greater the AUC, the better the performance of the model.

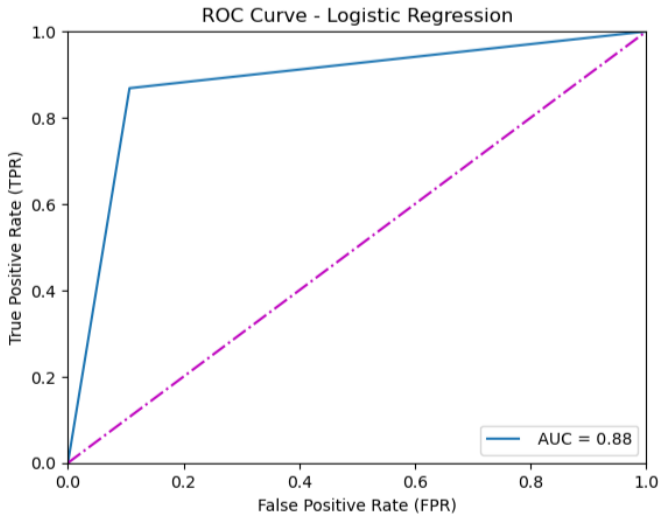


Fig: ROC curve for Logistic Regression

Analysis: The ROC plot is a plotted against the True positive rates(TPR) and False positive rates(FPR) at different threshold values. The threshold value is left at its default value. By varying the threshold value, we notice a change in the TPR and FPR rates. From this plot, we observe that the **AUC (Area under the curve) is 0.88.** The higher the AUC value the better the model is at predicting.

2. K-NN (K- nearest neighbor)

Implementation code snippets:

K- nearest Neighbour

In [43]:
```
import warnings
from sklearn.neighbors import KNeighborsClassifier
param_k = range(1,11)
accuracies_k = []
y_pred_highest = []
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=FutureWarning)
    for k in param_k:
        knn_model = KNeighborsClassifier(n_neighbors=k)
        knn_model.fit(X_train, y_train)
        y_pred = knn_model.predict(X_test)
        if k ==5:
            y_pred_highest = y_pred
        accuracy = accuracy_score(y_test, y_pred)
        accuracy_percent_knn = accuracy*100
        accuracies_k.append(accuracy_percent_knn)
        print('Accuracy achieved using K-nearest neighbour algorithm when k= {} is {}'.format(k,accuracy_percent_knn))
```

```
Accuracy achieved using K-nearest neighbour algorithm when k= 1 is 90.74804734680731
Accuracy achieved using K-nearest neighbour algorithm when k= 2 is 90.63531685320879
Accuracy achieved using K-nearest neighbour algorithm when k= 3 is 91.74651743296562
Accuracy achieved using K-nearest neighbour algorithm when k= 4 is 91.40027377405589
Accuracy achieved using K-nearest neighbour algorithm when k= 5 is 91.93171752959175
Accuracy achieved using K-nearest neighbour algorithm when k= 6 is 91.49689991142604
Accuracy achieved using K-nearest neighbour algorithm when k= 7 is 91.85924792656414
Accuracy achieved using K-nearest neighbour algorithm when k= 8 is 91.7545696110798
Accuracy achieved using K-nearest neighbour algorithm when k= 9 is 92.01223931073356
Accuracy achieved using K-nearest neighbour algorithm when k= 10 is 91.60963040502456
```

Fig: model fitting implementation code for K-NN

Result Visualization code:

In [46]:
```
plt.plot(param_k, accuracies_k)
plt.title('k-NN classifier accuracy for different various of k');plt.xlabel('k values'); plt.ylabel('Accuracy')
#plt.xlabel('k values'); plt.ylabel('Accuracy')
plt.ylim(80,100)
plt.show()
```

Fig: Code to plot accuracies achieved for various values of k

Result Visualizations for Logistic Regression Model

In [48]:
```
cm = confusion_matrix(y_test, y_pred_highest)
classes = ['satisfied', 'neutral or dissatisfied']
sns.heatmap(cm, annot=True, fmt='d', cmap='coolwarm',xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix for K-NN')
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.show()
```

```
In [35]: label_en = LabelEncoder()
         y_test_transformed = label_en.fit_transform(y_test)
         y_pred_transformed = label_en.transform(y_pred_highest)
         f_p, t_p, thresholds = roc_curve(y_test_transformed, y_pred_transformed); roc_auc = auc(f_p, t_p)
         plt.plot(f_p, t_p, label=' AUC = %0.2f' % roc_auc)
         plt.plot((0.0, 1.0), (0.0, 1.0), 'm-.')
         plt.title('ROC Curve - K-NN'); plt.xlabel('False Positive Rate (FPR)'); plt.ylabel('True Positive Rate (TPR)')
         plt.xlim(0.0,1.0);plt.ylim(0.0, 1.0)
         plt.legend(loc="lower right")
         plt.show()
```

Fig: Result Visualization(confusion matrix and ROC) code for K-NN

Reason for choosing the model:

K-nearest neighbor works by calculating the distance between each of the data points in the feature space and then selects the "k" datapoints closest to it and assigns the label that dominates the "K" datapoints to the datapoint that is to be classified. Out of all the algorithms we choose, comparatively K-NN achieved a higher accuracy. One of the reasons for this might be because K-NN is suitable for large datasets and higher number of features (dimensions). In our problem the number of features is 22 after the data preprocessing stage.

Tuning/training the model:

The hyper parameter that we can tune in K-NN is the value of k. We performed hyper parameter tuning on the preprocessed data by varying the value of "k" and choosing the value that gave the highest accuracy out of all. We have run the loop for different values of k starting from 1 until 10 and recording the accuracies for each value of k.

Model Effectiveness:

Here as we performed hyper parameter tuning, here is the list of accuracies achieved.

Fig: Table representing the accuracies for different values of k.

Hyper parameter (k)	Accuracy (%)
1	**90.74%**
2	**90.63%**
3	**91.74%**
4	**91.40%**
5	**91.93%**
6	**91.49%**

7	**91.85%**
8	**91.75%**
9	**92.01%**
10	**91.60%**

Fig: Table representing the accuracies for different values of k.

From the above table, we can conclude that, for the dataset that we have chosen, hyper parameter tuning has no much of an effect on the accuracy of the model as the model achieved almost the same accuracy for three different values of k. The highest **accuracy** achieved out of all is **91.93%.**

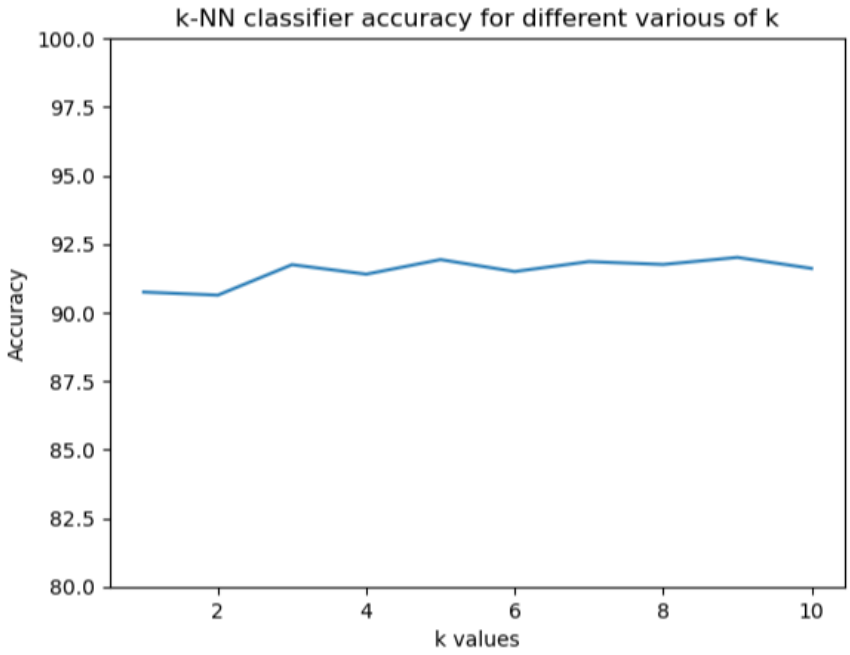


Fig: Plot representing accuracies for different values of k

Another metric to compare the effectiveness of the model with other models is the **confusion matrix.**

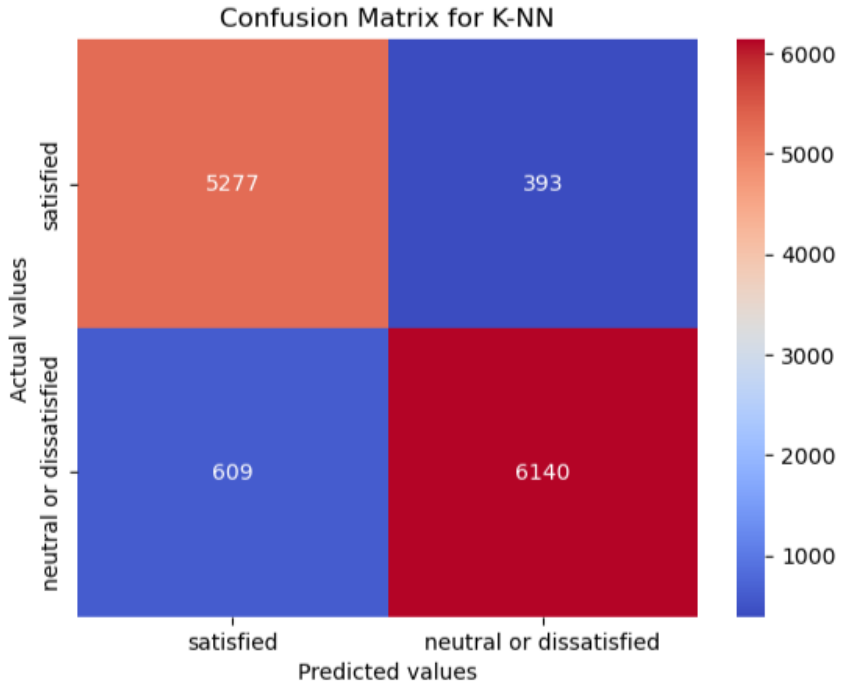


Fig: Heatmap of the confusion matrix for K-NN model.

Analysis: The above heat map of the confusion matrix is used to evaluate the performance of the K-NN algorithm. From the above confusion matrix, we can infer out of 5670 of the entries consisting of the target variable as "satisfied", 5277 values are correctly predicted as "satisfied" while the rest 393 are classified as "neutral or dissatisfied". And out of 6749 of the entries consisting of the target variable as "Neutral or Dissatisfied", 6140 values are correctly predicted as "Neutral or Dissatisfied" while the rest 609 are classified as "satisfied".

2. ROC Curve: From the ROC curve, we can infer that the AUC value for K-NN is **0.92** which means that the model is performing well. The curve starts at (0,0) and ends at (1,1).

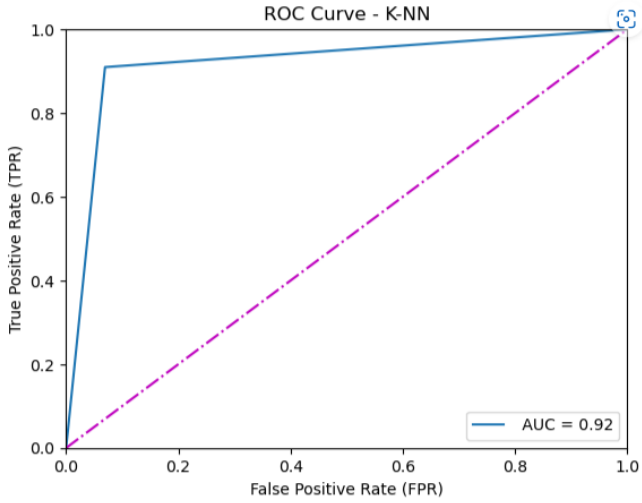


Fig: ROC curve for K-NN

3. Naïve Bayes' Algorithm

Implementation code snippets:

Naive Baye's Algorithm

```
In [49]: # NAIVE BAYE'S ALGORITHM
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred = nb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy_percent_nb = accuracy*100
print('Accuracy using Naive Bayes algorithm:', accuracy_percent_nb)

Accuracy using Naive Bayes algorithm: 86.23882760286658
```

Fig: model fitting implementation code for Naïve Bayes'

Result Visualization code:

Result Visualizations for Naive Baye's

```
In [51]: cm = confusion_matrix(y_test, y_pred)
classes = ['satisfied', 'neutral or dissatisfied']
sns.heatmap(cm, annot=True, fmt='d', cmap='viridis',xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix for Naive Bayes');plt.xlabel('Predicted values');plt.ylabel('Actual values')
plt.show()
```

```
In [38]: label_en = LabelEncoder()
y_test_transformed = label_en.fit_transform(y_test)
y_pred_transformed = label_en.transform(y_pred)
f_p, t_p, thresholds = roc_curve(y_test_transformed, y_pred_transformed); roc_auc = auc(f_p, t_p)
plt.plot(f_p, t_p, label=' AUC = %0.2f' % roc_auc)
plt.plot((0.0, 1.0), (0.0, 1.0), 'm-.')
plt.title('ROC Curve - Naive Bayes'); plt.xlabel('False Positive Rate (FPR)'); plt.ylabel('True Positive Rate (TPR)')
plt.xlim(0.0,1.0);plt.ylim(0.0, 1.0)
plt.legend(loc="lower right")
plt.show()
```

Fig: Result Visualization (confusion matrix and ROC) code for Naïve Bayes'

Reason for choosing the model:

Naïve Bayes' works on the assumption that the features in the dataset are independent of each other. Naïve Bayes' calculates the probability of a datapoint belonging to each of the labels and it assigns the label with the highest probability. As naïve Bayes' assumes that the features are independent of each other the computations are more efficient and simpler. One of the reasons for choosing this model is that Naïve Bayes' works well with categorical data and our dataset involves considerable number of categorical values.

Tuning/training the model:

Though naïve Bayes' works well with categorical values, it cannot directly train on categorical values. Hence, as we did for other models, the categorical values have been converted to numerical values before sending the data for training.

Model Effectiveness:

The model achieved an accuracy of **86.2%.** To get a better understanding of the performance of the model, we have the heatmap of the confusion matrix which helps us in getting better insights.

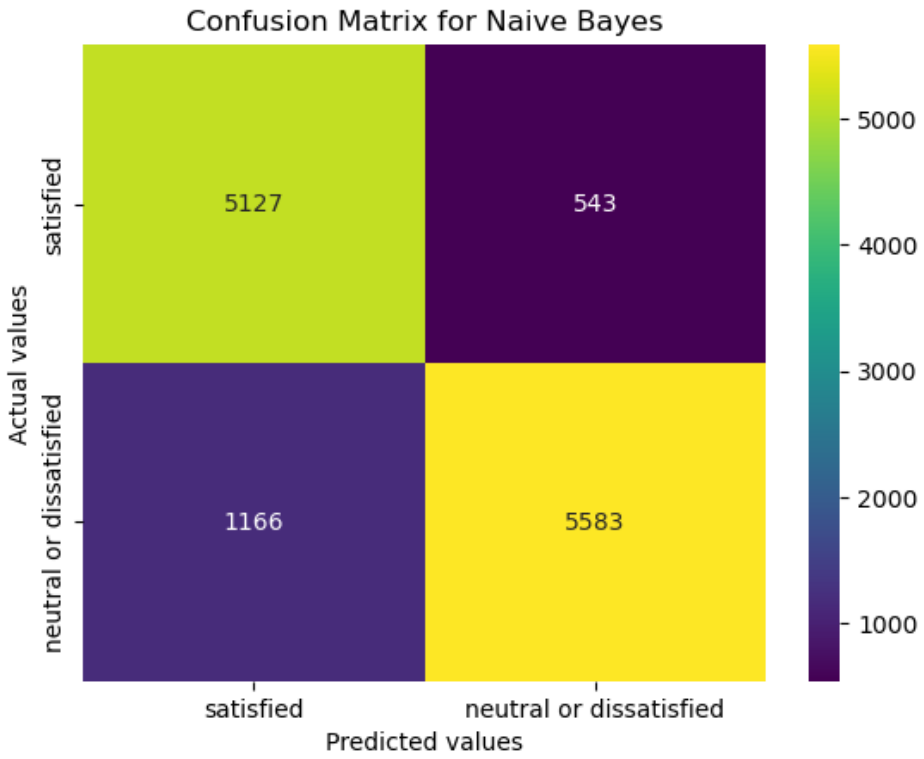


Fig: Heatmap of the confusion matrix for Naïve Bayes' algorithm.

Analysis: The above heat map of the confusion matrix is used to evaluate the performance of the Naive Bayes. Since, our problem is a binary classification problem, there are 4 possible outcomes as mentioned above. From the above confusion matrix, we can conclude out of 5670 of the entries consisting of the target variable as "satisfied", 5127 values are correctly predicted as "satisfied" while the rest 543 are classified as "neutral or dissatisfied". And out of 6749 of the entries consisting of the target variable as "Neutral or Dissatisfied", 5583 values are correctly predicted as "Neutral or Dissatisfied" while the rest 1166 are classified as "satisfied".

2. ROC curve: A ROC curve has been drawn to the model to understand the performance of the model in terms of sensitivity. The observed AUC for Naïve Bayes' is **0.87**.

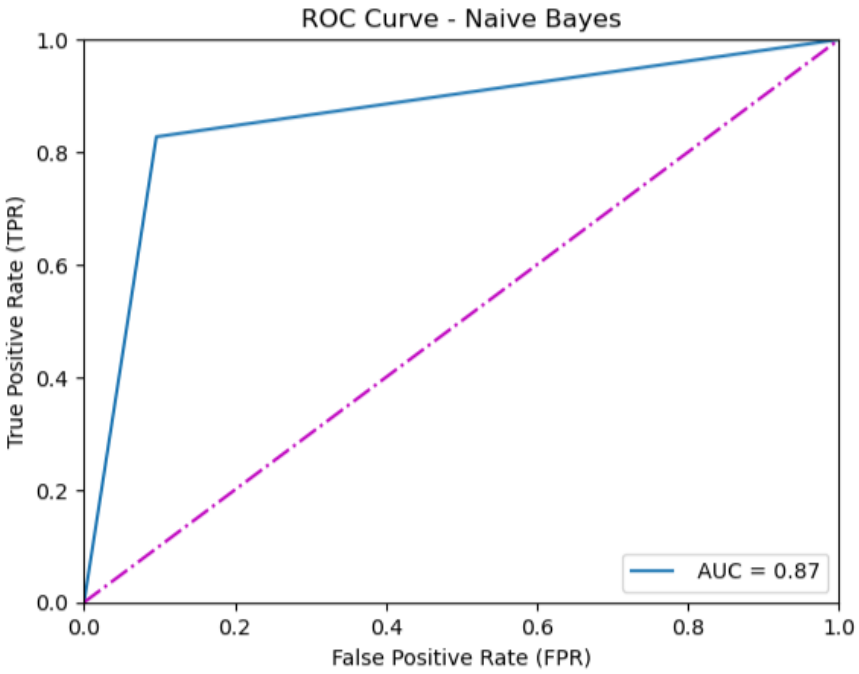


Fig: ROC curve for Naïve Bayes'

4. Support Vector Machine (SVM):

Implementation code snippets:

Support Vector Machine (SVM)

```
In [69]: # SVM ALGORITHM IMPLEMENTATION
from sklearn import svm
svm_model = svm.SVC(kernel='linear')
svm_model.fit(X_train, y_train)
y_pred = svm_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy_percent_svm = accuracy*100
print('Accuracy using Support Vector Machine:', accuracy_percent_svm)

Accuracy using Support Vector Machine: 87.98615025364361
```

Fig: model fitting implementation code for SVM

Result Visualization code:

Result Visualizations for SVM

```
In [71]: matrix = confusion_matrix(y_test, y_pred)
classes = ['satisfied', 'neutral or dissatisfied']
sns.heatmap(matrix, annot=True, fmt='d', cmap='inferno',xticklabels=classes, yticklabels=classes)
plt.title('Confusion Matrix for SVM');plt.xlabel('Predicted values');plt.ylabel('Actual values')
plt.show()
```

```
In [41]: label_en = LabelEncoder()
y_test_transformed = label_en.fit_transform(y_test)
y_pred_transformed = label_en.transform(y_pred)
f_p, t_p, thresholds = roc_curve(y_test_transformed, y_pred_transformed); roc_auc = auc(f_p, t_p)
plt.plot(f_p, t_p, label=' AUC = %0.2f' % roc_auc)
plt.plot((0.0, 1.0), (0.0, 1.0), 'm-.')
plt.title('ROC Curve - SVM'); plt.xlabel('False Positive Rate (FPR)'); plt.ylabel('True Positive Rate (TPR)')
plt.xlim(0.0,1.0);plt.ylim(0.0, 1.0)
plt.legend(loc="lower right")
plt.show()
```

Fig: Result Visualization (confusion matrix and ROC) code for SVM

Reason for choosing the model:

Support vector machine (SVM) works on both linearly separable and linearly in-separable data. It works by finding the best decision boundary that separates each of the classes. When the datasets are huge, SVM uses the technique of kernel approximation. Our dataset consists of ninety thousand rows which is not very huge hence in this case using SVM is not computationally difficult.

Tuning/training the model:

As there are no hyper parameters to tune for an SVM model, the model has been trained on the data after the feature extraction step.

Model Effectiveness:

The model achieved an accuracy of **87.98%.** Another important metrics to evaluate the performance of our model is confusion matrix.

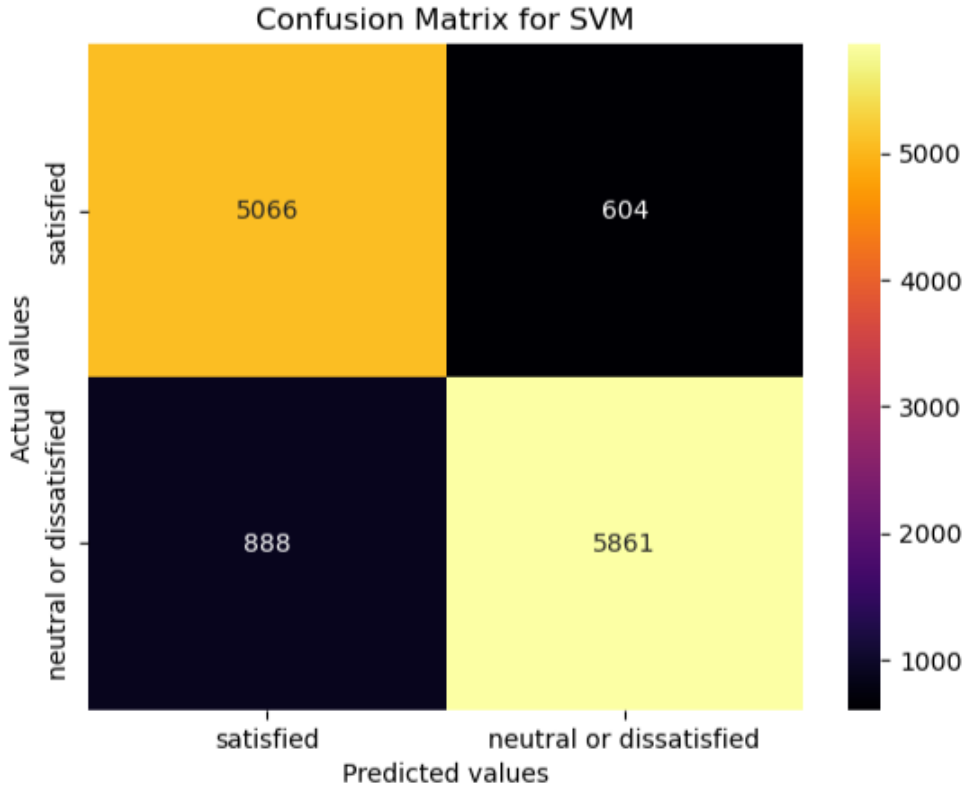


Fig: Heatmap of the confusion matrix for SVM

Analysis: Analysis: The above heat map of the confusion matrix is used to evaluate the performance of SVM. From the above confusion matrix, we can conclude out of 5670 of the entries consisting of the target variable as "satisfied", 5066 values are correctly predicted as "satisfied" while the rest 604 are classified as "neutral or dissatisfied". And out of 6749 of the entries consisting of the target variable as "Neutral or Dissatisfied", 5861 values are correctly predicted as "Neutral or Dissatisfied" while the rest 888 are classified as "satisfied".

2. ROC Curve: The observed AUC for SVM is **0.88** which is high and concludes the same about the model's performance like the other metrics like accuracy and Confusion Matrix.

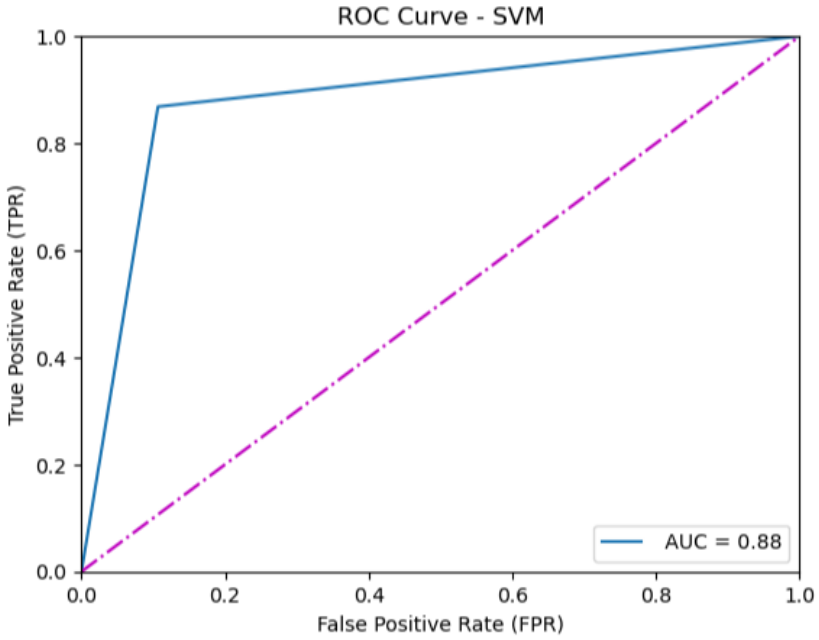


Fig: ROC curve for SVM

5. Random Forest:

Implementation code:

Random Forest

```
In [57]: from sklearn.ensemble import ExtraTreesClassifier
         ex=ExtraTreesClassifier()
         ex.fit(X,y)
         colss = X.columns
```

```
In [59]: import matplotlib.pyplot as plt
         plt.figure(figsize=(20,10))
         plt.title('Feature importances')
         feat=pd.Series(ex.feature_importances_,index=colss)
         feat.nlargest(15).plot(kind='bar', color="g", align="center")
         plt.tight_layout()
         plt.grid(False)
         plt.show()
```

Fig: Feature selection code for Random Forest

```
In [62]: from sklearn.ensemble import RandomForestClassifier
         random_forest_model = RandomForestClassifier()
         random_forest_model.fit(X_train, y_train)
         y_pred = random_forest_model.predict(X_test)
```

```
In [65]: y_pred = random_forest_model.predict(X_test)
         accuracy = accuracy_score(y_test, y_pred)
         accuracy_percent_rfc = accuracy*100
         print('Accuracy achieved using Random forest:', accuracy_percent_rfc)
```

```
Accuracy achieved using Random forest: 95.65987599645705
```

Fig: Model fitting implementation code for Random Forest

Result Visualization code:

```
In [66]: cm = confusion_matrix(y_test, y_pred)
         classes = ['satisfied', 'neutral or dissatisfied']
         sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',xticklabels=classes, yticklabels=classes)
         plt.title('Confusion Matrix for Random Forest');plt.xlabel('Predicted values');plt.ylabel('Actual values')
         plt.show()
```

```
In [49]: label_en = LabelEncoder()
         y_test_transformed = label_en.fit_transform(y_test)
         y_pred_transformed = label_en.transform(y_pred)
         f_p, t_p, thresholds = roc_curve(y_test_transformed, y_pred_transformed); roc_auc = auc(f_p, t_p)
         plt.plot(f_p, t_p, label=' AUC = %0.2f' % roc_auc)
         plt.plot((0.0, 1.0), (0.0, 1.0), 'm-.')
         plt.title('ROC Curve - Random Forest'); plt.xlabel('False Positive Rate (FPR)'); plt.ylabel('True Positive Rate (TPR)')
         plt.xlim(0.0,1.0);plt.ylim(0.0, 1.0)
         plt.legend(loc="lower right")
         plt.show()
```

Fig: Result Visualization ((confusion matrix and ROC) code for Random Forest

Reason for choosing the model:

Random forest is said to perform better when there are many features in the dataset. One of the biggest advantages of this algorithm is that is can model non-linear relationships between the target and the feature variables. As our dataset consists of large number of rows even after the pre-processing stage, we have chosen to implement this algorithm as it performs better with large dataset.

Tuning/training the model:

Though, random forest works well for large number of features, we have tried identifying 15 important features and training the model using those features and dropping the rest. This helped us to achieve a better accuracy.

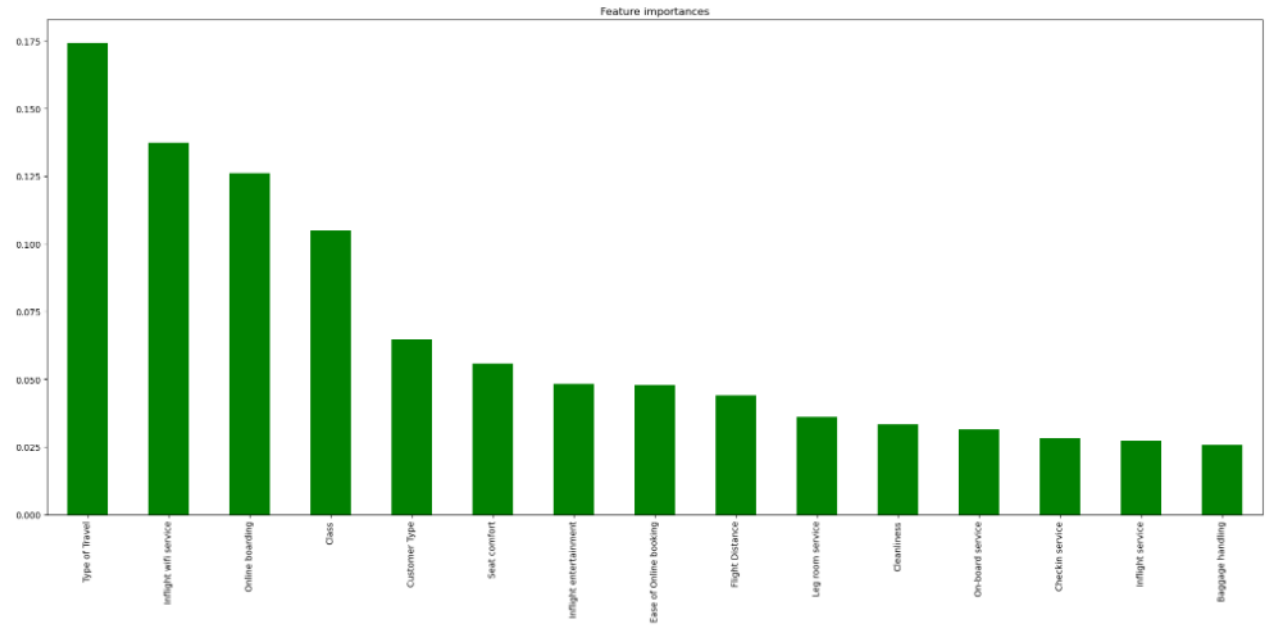
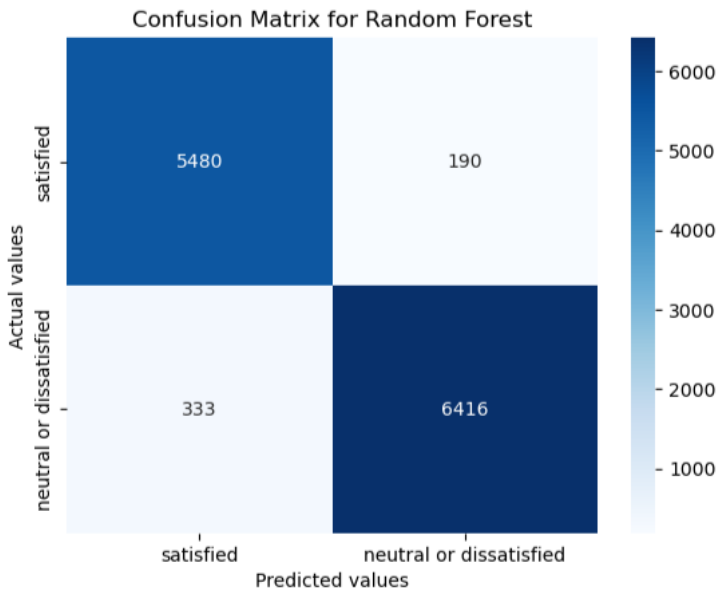


Fig: Bar plot showing important features.

Model Effectiveness:

Out of all the 5 models that are implemented, the random forest algorithm provided the highest accuracy among all. The accuracy achieved using this model is **95.76%.**

A heatmap of the confusion matrix also helps us in understanding the performance of the model.



Heatmap of the confusion matrix for Random Forest.

Analysis: The above heat map of the confusion matrix is used to evaluate the performance of the Random Forest algorithm. Since, our problem is a binary classification problem, there are 4 possible outcomes as mentioned above. From the above confusion matrix, we can conclude out of 5670 of the entries consisting of the target variable as "satisfied", 5480 values are correctly predicted as "satisfied" while the rest 190 are classified as "neutral or dissatisfied". And out of 6749 of the entries consisting of the target variable as "Neutral or Dissatisfied", 6416 values are correctly predicted as "Neutral or Dissatisfied" while the rest 333 are classified as "satisfied".

2. ROC Curve: The observed value for AUC is **0.96** which is the highest out of all the models the data has been trained on.

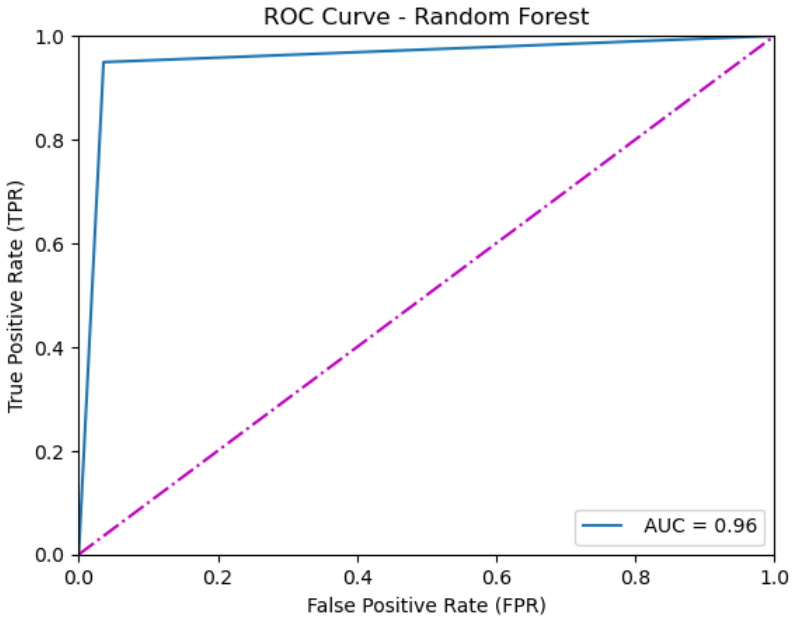


Fig: ROC curve for Random Forest

CONCLUSION:

It is concluded that out of all the models implemented, for the given dataset and a chosen target variable, **"Random Forest"**. The various metrics used to decide the best model are **accuracy, Confusion Matrix and ROC curve.** The **accuracy** achieved **95.76%** which is highest among all, and the **AUC (Area under the curve) is 0.96** which is highest among all the models.

References:

1) scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation
2) User guide and tutorial — seaborn 0.12.2 documentation (pydata.org)
3) Plotly Python Graphing Library
4) Kaggle dataset: Airline Passenger Satisfaction | Kaggle