---

title: "Customer Satisfaction Prediction Report"

author: "Group 13- Harshini Balam & Neetu Aavula"

date: "2024-05-02"

output:

html\_document:

toc: true

toc\_depth: 2

word\_document:

toc: true

toc\_depth: '2'

pdf\_document:

toc: true

toc\_depth: '2'

---

```{r setup, include=FALSE}

knitr::opts\_chunk$set(echo = TRUE)

```

# Customer Satisfaction Prediction Report

## Data Understanding and Preprocessing

```{r,message=FALSE}

# Load necessary libraries

library(tidyverse)

# Load the dataset

data <- read.csv("C:/Users/Harshini/OneDrive - Kent State University/Fundementals of Machine Learning/Final Project/Customer Satisfaction dataset.csv")

# Explore dataset structure and summary

str(data)

summary(data)

# Check for missing values

missing\_values <- colSums(is.na(data))

missing\_values

# Handle missing values (if any)

# For demonstration, assuming 'NA' values in 'satisfaction' are unknown

#below line is not required as there are no unknowns

#data$satisfaction[is.na(data$satisfaction)] <- "unknown"

# Remove rows with any missing values

data <- na.omit(data)

# Check for missing values

missing\_values <- colSums(is.na(data))

missing\_values

# Clean the data

data$Age <- as.numeric(data$Age)

# Visualize features

# For example, distribution of 'Flight Distance'

ggplot(data, aes(x = Flight.Distance)) +

geom\_histogram(bins = 30, fill = "skyblue", color = "black") +

labs(x = "Flight Distance", y = "Frequency", title = "Distribution of Flight Distance")

```

## Data Visualization

#### 1. Histogram of Numeric Features

```{r}

# Histogram of 'Flight Distance'

hist(data$Flight.Distance,

main = "Distribution of Flight Distance",

xlab = "Flight Distance",

col = "skyblue",

border = "black")

```

#### 2. Bar Plot of Categorical Features

```{r}

# Bar plot of 'Type of Travel'

barplot(table(data$Type.of.Travel),

main = "Type of Travel Distribution",

xlab = "Type of Travel",

ylab = "Count",

col = "lightblue")

```

#### 3. Boxplot for Comparing Features by Satisfaction

```{r}

# Boxplot of 'Seat Comfort' by 'Satisfaction'

boxplot(data$Seat.comfort ~ data$satisfaction,

main = "Seat Comfort by Satisfaction",

xlab = "Satisfaction",

ylab = "Seat Comfort",

col = c("skyblue", "lightgreen"),

names = c("Neutral/Dissatisfied", "Satisfied"))

```

#### 4. Bar Plot of Satisfaction Distribution

```{r}

# Bar plot of 'Satisfaction'

barplot(table(data$satisfaction),

main = "Distribution of Satisfaction",

xlab = "Satisfaction",

ylab = "Count",

col = "lightblue")

```

## Data Transformation and feature engineering

```{r}

# Load necessary libraries

library(dplyr)

# Define the required columns based on confirmed column names

required\_columns <- c("Type.of.Travel", "Class", "Flight.Distance",

"Inflight.wifi.service", "Departure.Arrival.time.convenient",

"Ease.of.Online.booking", "Food.and.drink", "Online.boarding",

"Seat.comfort", "Inflight.entertainment", "On.board.service",

"Leg.room.service", "Baggage.handling", "Checkin.service",

"Inflight.service", "Cleanliness", "satisfaction")

# Subset data to include only required columns

data\_subset <- data %>%

select(one\_of(required\_columns))

# View the structure of the transformed dataset

str(data\_subset)

```

```{r}

# Convert 'satisfaction' to binary (0 or 1)

data\_subset$satisfaction <- ifelse(data\_subset$satisfaction == "satisfied", 1, 0)

# One-hot encoding for categorical variables (Type.of.Travel and Class)

data\_subset <- data\_subset %>%

mutate(Type.of.Travel = ifelse(Type.of.Travel == "Business travel", 1, 0)) %>%

mutate(Class\_Eco = ifelse(Class == "Eco", 1, 0)) %>%

mutate(Class\_Eco\_Plus = ifelse(Class == "Eco Plus", 1, 0)) %>%

mutate(Class\_Business = ifelse(Class == "Business", 1, 0)) %>%

select(-Class) # Drop original 'Class' column after encoding

# Normalize numeric features using Min-Max Scaling

data\_subset$Flight.Distance <- (data\_subset$Flight.Distance - min(data\_subset$Flight.Distance)) /

(max(data\_subset$Flight.Distance) - min(data\_subset$Flight.Distance))

# Check for missing values (if any) and handle them

if (anyNA(data\_subset)) {

data\_subset <- na.omit(data\_subset)

}

head(data\_subset)

```

### Model Building and Evaluation

```{r}

set.seed(123) # Set seed for reproducibility

train\_indices <- sample(1:nrow(data\_subset), 0.8 \* nrow(data\_subset)) # 80% for training

train\_data <- data\_subset[train\_indices, ]

test\_data <- data\_subset[-train\_indices, ]

```

```{r}

# Perform PCA

pca\_result <- prcomp(train\_data, center = TRUE, scale. = TRUE)

# Load the factoextra package if not already loaded

library(factoextra)

# Visualize variables in the principal component space (biplot)

fviz\_pca\_var(pca\_result, col.var = "cos2",

gradient.cols = c("blue", "orange", "green"),

repel=TRUE)

```

```{r}

# Identify the features you want to keep

features\_to\_keep <- c("Class\_Eco", "Inflight.entertainment", "Online.boarding", "Class\_Business", "satisfaction")

features\_to\_keep

# Subset the dataset to keep only the selected features

if (length(features\_to\_keep) > 0) {

train\_data <- train\_data[, features\_to\_keep]

}

# Perform PCA

pca\_result <- prcomp(train\_data, center = TRUE, scale. = TRUE)

# Load the factoextra package

library(factoextra)

# Visualize variables in the principal component space (biplot)

fviz\_pca\_var(pca\_result, col.var = "cos2",

gradient.cols = c("blue", "orange", "green"),

repel = TRUE)

```

### Pair Metrics (Scatterplot Matrix)

```{r}

library(psych)

# Assuming pca\_result$x contains the transformed data from PCA

pairs.panels(train\_data[, c("satisfaction", "Class\_Business", "Online.boarding", "Inflight.entertainment", "Class\_Eco")],

gap = 0,

bg = c("red", "yellow", "blue")[pca\_result$satisfaction],

pch = 21)

```

```{r}

new\_data <- train\_data[, c("satisfaction", "Class\_Business", "Online.boarding", "Inflight.entertainment", "Class\_Eco")]

str(new\_data)

dim(new\_data)

```

#### Performing the Machine Learning Model K-Means Classifer

```{r}

library(tidyverse) # data manipulation

library(factoextra) # clustering algorithms & visualization

library(ISLR)

library(caret)

set.seed(123)

train.index <- sample(row.names(new\_data),0.8\*dim(new\_data)[1])

test.index <- setdiff(row.names(new\_data),train.index)

train.df <- new\_data[train.index,]

test.df <- new\_data[test.index,]

```

```{r}

norm.values <- preProcess(train.df, method=c("center", "scale"))

train.df <- predict(norm.values, train.df)

test.df <- predict(norm.values, test.df)

```

```{r}

# Assuming test data is stored in test\_data dataframe

# Normalize the test data

test\_norm <- scale(test\_data)

# Convert the normalized test data back to a dataframe

test.norm.df <- as.data.frame(test\_norm)

# Check the dimensions

dim(test.norm.df)

```

```{r}

library(kknn)

library(caret)

# Convert satisfaction to a factor with two levels

train.df$satisfaction <- factor(train.df$satisfaction)

# Define training control

train.control <- trainControl(method = "cv", number = 5)

# Sample 1000 observations from the training data frame

train\_sample <- train.df[sample(nrow(train.df), 10000), ]

# Train the kknn model using the sampled data

fit <- train(satisfaction ~ ., data = train\_sample,

method = "kknn",

trControl = train.control,

preProcess = "scale",

tuneLength = 5)

# Print the trained model

print(fit)

```

#####After the test it means k=9

```{r}

# Load necessary libraries

library(kknn)

library(caret)

# Assuming your normalized data and the satisfaction columns are ready

# Fit the kknn model

fit <- kknn(satisfaction ~ ., train = train.df, test = test.df, k = 9, distance = 2)

# Predict class labels using the default settings

pred <- predict(fit)

# Ensure that both pred and the actual satisfaction column are factors and have the same levels

pred <- factor(pred, levels = levels(train.df$satisfaction))

actual <- factor(test.df$satisfaction, levels = levels(train.df$satisfaction))

# Evaluate the predictions

conf.mat <- confusionMatrix(data = pred, reference = actual)

print(conf.mat)

```

```{r}

# Load necessary libraries

library(kknn)

library(gmodels)

# Assuming your normalized data and the satisfaction columns are ready

# Fit the kknn model

fit <- kknn(satisfaction ~ ., train = train.df, test = test.df, k = 9, distance = 2)

# Predict class labels using the default settings

pred <- predict(fit)

# Ensure that both pred and the actual satisfaction column are factors and have the same levels

pred <- factor(pred, levels = levels(train.df$satisfaction))

actual <- factor(test.df$satisfaction, levels = levels(train.df$satisfaction))

# Create the confusion matrix using gmodels

conf.mat <- CrossTable(x = pred, y = actual, prop.chisq = FALSE)

print(conf.mat)

```

#####Naive Bayes Classification Model

```{r}

#performing Naives bayes Classifier

library(caret)

library(e1071)

#lets use NB model

#Clean the data, and divide into training and test

#train\_indices <- sample(1:nrow(new\_data), 0.8 \* nrow(new\_data)) # 80% for training

train\_indices <-createDataPartition(new\_data$satisfaction, p=0.8, list=FALSE)

train\_data <- new\_data[train\_indices, ]

test\_data <- new\_data[-train\_indices, ]

#Building a Naive Bayes Classifier

nb\_model <- naiveBayes(satisfaction~Class\_Business+Online.boarding+Inflight.entertainment+Class\_Eco,data = train\_data)

nb\_model

```

```{r}

# Now, use the model on the test set

library("gmodels")

# Predict the default status of test dataset

Predicted\_Test\_labels <-predict(nb\_model,test\_data)

# Show the confusion matrix of the classifier

CrossTable(x=test\_data$satisfaction,y=Predicted\_Test\_labels, prop.chisq = FALSE)

```

- True Negatives (TN): 7612

- False Positives (FP): 1924

- False Negatives (FN): 1666

- True Positives (TP): 5373

```{r}

#Make predictions and return probability of each class

Predicted\_Test\_labels <-predict(nb\_model,test\_data, type = "raw")

#show the first few values

head(Predicted\_Test\_labels)

```

###Comapring both the models KNN and Naive Bayes to know the best Fit

```{r}

# Load necessary libraries

library(ggplot2)

# Define the accuracy, precision, recall, and F1 score for KNN and Naive Bayes

Accuracy\_knn <- 0.85 # Replace with your actual KNN accuracy

Accuracy\_nb <- 0.78 # Replace with your actual Naive Bayes accuracy

precision\_knn <- 0.82 # Replace with your actual KNN precision

precision\_nb <- 0.76 # Replace with your actual Naive Bayes precision

recall\_knn <- 0.88 # Replace with your actual KNN recall

recall\_nb <- 0.72 # Replace with your actual Naive Bayes recall

f1\_score\_knn <- 0.85 # Replace with your actual KNN F1 score

f1\_score\_nb <- 0.74 # Replace with your actual Naive Bayes F1 score

# Create a comparison table

comparison\_table <- data.frame(

Model = c("KNN", "Naive Bayes"),

Accuracy = c(Accuracy\_knn, Accuracy\_nb),

Precision = c(precision\_knn, precision\_nb),

Recall = c(recall\_knn, recall\_nb),

F1\_Score = c(f1\_score\_knn, f1\_score\_nb)

)

print(comparison\_table)

# Create a bar plot to compare accuracy

accuracy\_data <- data.frame(Model = c("KNN", "Naive Bayes"), Accuracy = c(Accuracy\_knn, Accuracy\_nb))

ggplot(accuracy\_data, aes(x = Model, y = Accuracy, fill = Model)) +

geom\_bar(stat = "identity") +

labs(title = "Comparison of Accuracy between KNN and Naive Bayes") +

theme\_minimal()

```