AI IN MARKETING PROJECT REPORT

SUBMITTED BY

Yaswanth Sai Valluru AP22110011098

SUBMITTED TO

Dr Mudassir Rafi

Assistant Professor

Department of Computer Science and Engineering

and

Dr Md Faiz Ahmad

Assistant Professor

Department of Management

"Comparative Analysis of Machine Learning

Models for Airline Reviews Classification"

Yaswanth Sai Valluru

Department of Computer Science and Engineering, SRM University-AP Andhra Pradesh

Abstract

The dynamic airline industry has experienced unprecedented growth in recent decades, intensifying the need for effective consumer feedback analysis. Robust data collection serves as the cornerstone for understanding consumer sentiments and driving informed decision-making. Sentiment analysis emerges as a pivotal tool, employing machine learning techniques to dissect textual data and unveil underlying attitudes and emotions. This paper delves into sentiment analysis applied to airline reviews, leveraging a diverse set of Machine Learning (ML) algorithms such as Naive Bayes, Support Vector Machine (SVM), k-Nearest Neighbours (KNN), Artificial Neural Networks (ANN), and Random Forest Classifiers.

Through meticulous evaluation, including metrics like accuracy, precision, recall, and F1-score, the performance of each algorithm is scrutinized. The results showcase the effectiveness of these models in discerning sentiments from airline reviews, providing insights for improving customer service and satisfaction in the airline industry.

Keywords: Machine Learning · Text Classification · Airline Reviews · Support Vector Machines · k-Nearest Neighbours · Naive Bayes · Artificial Neural Networks · Random Forest

1. Introduction

Airline reviews play a crucial role in shaping customer perceptions and driving business decisions in the aviation industry. With the advent of online platforms, the volume of such reviews has surged, necessitating automated analysis techniques. In this study, we employ machine learning algorithms to classify airline reviews based on textual content and associated features

1.1 Problem Description

The burgeoning airline industry faces a pressing need for efficient analysis of consumer feedback to inform strategic decision-making. With the proliferation of online platforms, the volume of airline reviews has surged, necessitating automated analysis techniques. This study aims to address this challenge by employing machine learning algorithms to classify airline reviews based on textual content and associated features. The goal is to develop robust models capable of discerning sentiments from diverse reviews, thereby providing airlines with actionable insights to enhance customer service and satisfaction. Through meticulous evaluation of various machine learning models, including Support Vector Machines, k-Nearest Neighbours, Naive Bayes, Artificial Neural Networks, and Random Forest, this research seeks to identify the most effective approach for sentiment analysis in the aviation industry.

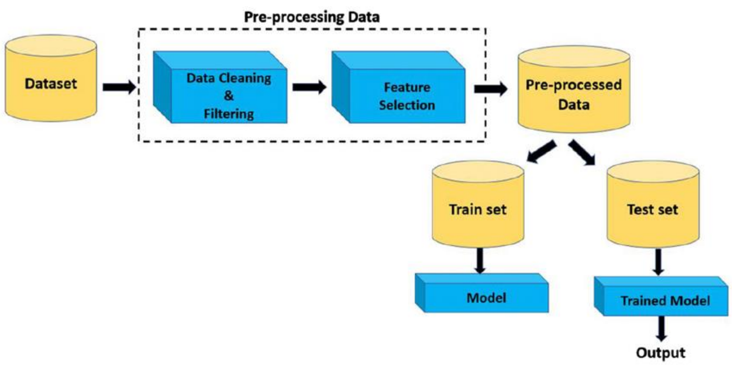
1.2 Author's Contribution

Summary of our contribution is as follows:

- Conceptualized and designed the research study.
- Collected and preprocessed the dataset, ensuring its quality and relevance to the research objectives.
- Implemented machine learning models for sentiment analysis of airline reviews.
- Conducted rigorous evaluation and analysis of model performance.

2. Dataset

2.1 Pre-processing



2.2 Initial Dataset

0	Flight was amazing	Alison Soetantyo	01-03-2024	Singapore Airlines	TRUE	Flight was amazing. The crew onboard this fl...	Solo Leisure	Dec-23	Jakarta to Singapore	Business Class	4	4	4	4	4	9	yes
1	seats on this aircraft are dreadful	Robert Watson	21-02-2024	Singapore Airlines	TRUE	Â Â Booking an emergency exit seat still meant...	Solo Leisure	Feb-24	Phuket to Singapore	Economy Class	5	3	4	4	1	3	no
2	Food was plentiful and tasty	S Han	20-02-2024	Singapore Airlines	TRUE	Excellent performance on all fronts. I would...	Family Leisure	Feb-24	Siem Reap to Singapore	Economy Class	1	5	2	1	5	10	yes
3	â��how much food was available	D Laynes	19-02-2024	Singapore Airlines	TRUE	Pretty comfortable flight considering I was f...	Solo Leisure	Feb-24	Singapore to London Heathrow	Economy Class	5	5	5	5	5	10	yes
4	â��service was consistently goodâ��	A Othman	19-02-2024	Singapore Airlines	TRUE	The service was consistently good from start ...	Family Leisure	Feb-24	Singapore to Phnom Penh	Economy Class	5	5	5	5	5	10	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8095	an uneventful flight	N Vickers	20-06-2016	Korean Air	TRUE	KE124, Brisbane to Incheon (A330) and KE867,...	Business	Jun-16	BNE to ULN via ICN	Economy Class	5	4	5	3	4	7	yes
8096	Korean Air always impresses	Kim Holloway	12-06-2016	Korean Air	FALSE	Our recent flight was our fourth trip to the...	Couple Leisure	Jun-16	SYD to LHR via ICN	Economy Class	3	5	5	4	5	10	yes
8097	didnâ��t offer anything	C Clark	06-06-2016	Korean Air	TRUE	I flew Korean Air from Bali to Seoul in Pres...	Business	Apr-16	DPS to ICN	Business Class	4	5	5	5	1	2	no
8098	appreciated the service onboard	E Petan	21-04-2016	Korean Air	FALSE	Seoul to Paris with Korean Air. I am traveli...	Business	Apr-16	ICN to CDG	Business Class	5	1	3	4	5	10	yes
8099	genuinely friendly staff	D Lanor	12-04-2016	Korean Air	FALSE	The 13 hour flight in Business class from Se...	Business	Apr-16	ICN to YYZ	Business Class	3	5	3	3	5	10	yes

2.3 Feature vector and their types

```
Title                        object
Name                         object
Review Date                  object
Airline                      object
Verified                     object
Reviews                      object
Type of Traveller            object
Month Flown                  object
Route                        object
Class                        object
Seat Comfort                  int64
Staff Service                 int64
Food & Beverages              int64
Inflight Entertainment        int64
Value For Money               int64
Overall Rating                int64
Recommended                  object
dtype: object
```

3. Machine Learning Models Used

3.1 Support Vector Machine

The goal of the Support Vector Machine (SVM) classification algorithm is to optimize the margin between classes while determining the best hyperplane to divide various classes in a feature space. In order to define the hyperplane, it first finds support vectors, or the data points that are closest to the decision boundary.

The Support Vector Classifier (SVC) is a classification implementation of the Support Vector Machine (SVM) technique that is specifically developed for issues with non-linearly separable classes. The goal of SVC is to identify the ideal hyperplane that efficiently divides classes while maximizing the margin. Even when the data is not linearly separable in the original feature space, it uses kernel functions to transform the data into a higher-dimensional space, enabling linear separation. Because of its versatility, SVC is used extensively in a variety of fields, including image recognition, text classification, and bioinformatics.

3.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a nonparametric approach used in classification and regression problems. A new data point is classified by allocating it to the majority class in the feature space among its K closest neighbors. KNN depends on how similar new instances are to preexisting data points. Its success is highly dependent on the choice of K, which also affects decision boundaries and the trade-off between bias and variance.

KNeighborsClassifier is a classification implementation of the K-Nearest Neighbors (KNN) approach available in the Python scikit-learn module. It works well with both small and large datasets since it is flexible and easy to use. Users can select distance metrics, the number of neighbors (K) for predictions, and weighting methods based on closeness with KNeighborsClassifier. KNeighborsClassifier is a popular choice for real-world classification jobs because of its ease of use, effectiveness, and versatility in handling decision boundaries.

3.3 ANN

Artificial Neural Networks (ANN) are computational models inspired by the structure and function of biological neural networks, such as the human brain. Comprising interconnected nodes organized into layers, each node, or neuron, processes input data through an activation function to produce an output. ANNs typically consist of an input layer, hidden layers (which perform computations), and an output layer. Through connections with associated weights and biases, neurons communicate information between layers, adjusting these parameters during training via backpropagation to minimize prediction errors. Activation functions, like ReLU or sigmoid, introduce non-linearity, enabling ANNs to learn complex patterns. During training, data is propagated forward through the network, and errors are iteratively corrected. ANNs have shown versatility in tasks ranging from classification to regression, image recognition, and natural language processing. However, their effectiveness depends on extensive data and computational resources, with architectural and hyperparameter tuning being key challenges. Despite these complexities, ANNs remain a cornerstone of modern machine learning, continuously advancing our ability to understand and model complex data relationships.

3.4 Random Forest

Random Forest is a popular ensemble learning method for classification and regression. During training, it builds several decision trees, averaging the predictions of each tree by calculating the mean (for regression) or the mode (for classification). Random Forest improves generalization performance by adding randomness to feature selection and data sampling, which reduces overfitting. When the forecasts of several decision trees are combined, Random Forest produces predictions that are more accurate and dependable than those of a single tree.

RandomForestClassifier, a component of Python's scikit-learn, uses Random Forest for classification. During training, it builds several trees and integrates their predictions. It inherits the versatility of Random Forest, with hyperparameters governing tree number, depth, and feature considerations that may be adjusted. It is well suited for a variety of classification tasks, less prone to overfitting, and widely utilized for handling high-dimensional data and capturing complicated relationships.

3.5 Naïve Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem with a strong assumption of independence between features. Despite its simplistic nature, Naive Bayes often performs well in practice, especially with text classification tasks such as sentiment analysis or spam detection. It calculates the probability of each class given a set of features by assuming that each feature contributes independently to the probability of the class. Naive Bayes is computationally efficient and works well with high-dimensional data. However, its assumption of feature independence may not hold true in many real-world scenarios, which can affect its accuracy. Despite this limitation, Naive Bayes remains a popular choice for its simplicity, speed, and effectiveness in certain contexts, particularly when dealing with text data. Additionally, it serves as a useful baseline model for comparison with more complex algorithms.

4. Code for model training, testing and printing the accuracy

```
# Define the path to the dataset file
path = '/ar.csv'

# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, zero_one_loss
from scipy.sparse import hstack
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.ensemble import RandomForestClassifier

# Read the dataset into a pandas DataFrame
data = pd.read_csv(path, encoding='latin1')

# Separate features (X) and target variable (y)
X = data.drop(columns=["Recommended"])
y = data["Recommended"]

# Define categorical columns
categorical_columns = ["Title", "Airline", "Type of Traveller", "Route", "Class", "Recommended"]

# Extract categorical data
categorical_data = data[categorical_columns]

# One-hot encode categorical variables
encoder = OneHotEncoder()
categorical_encoded = encoder.fit_transform(categorical_data)

# Convert 'Month Flown' to datetime format and drop rows with missing values
data['Month Flown'] = pd.to_datetime(data['Month Flown'], format='%b-%y', errors='coerce')
data = data.dropna(subset=['Month Flown'])

# Define numerical columns
numerical_columns = ["Seat Comfort", "Staff Service", "Food & Beverages",
                     "Inflight Entertainment", "Value For Money", "Overall Rating"]

# Extract numerical features
X_numeric = data[numerical_columns]

# Extract text data for sentiment analysis
text_data = data["Reviews"]

# Transform text data into TF-IDF vectors
tfidf_vectorizer = TfidfVectorizer()
text_tfidf = tfidf_vectorizer.fit_transform(text_data)
```

```python
# Define the path to the dataset file
path = '/ar.csv'

# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, mean_absolute_error, zero_one_loss
from scipy.sparse import hstack
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.ensemble import RandomForestClassifier

# Read the dataset into a pandas DataFrame
data = pd.read_csv(path, encoding='latin1')

# Separate features (X) and target variable (y)
X = data.drop(columns=["Recommended"])
y = data["Recommended"]

# Define categorical columns
categorical_columns = ["Title", "Airline", "Type of Traveller", "Route", "Class", "Recommended"]

# Extract categorical data
categorical_data = data[categorical_columns]

# One-hot encode categorical variables
encoder = OneHotEncoder()
categorical_encoded = encoder.fit_transform(categorical_data)

# Convert 'Month Flown' to datetime format and drop rows with missing values
data['Month Flown'] = pd.to_datetime(data['Month Flown'], format='%b-%y', errors='coerce')
data = data.dropna(subset=['Month Flown'])

# Define numerical columns
numerical_columns = ["Seat Comfort", "Staff Service", "Food & Beverages",
                     "Inflight Entertainment", "Value For Money", "Overall Rating"]

# Extract numerical features
X_numeric = data[numerical_columns]

# Extract text data for sentiment analysis
text_data = data["Reviews"]

# Transform text data into TF-IDF vectors
tfidf_vectorizer = TfidfVectorizer()
text_tfidf = tfidf_vectorizer.fit_transform(text_data)
```

```python
# Combine numerical features, one-hot encoded categorical features, and TF-IDF vectors
X_combined_sparse = hstack((X_numeric, categorical_encoded, text_tfidf))

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_combined_sparse, y, test_size=0.3, random_state=42)

# Support Vector Machine classifier
clf = SVC(kernel='rbf', C=1.0)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

# Evaluate SVM classifier
accuracy_SVM = accuracy_score(y_test, y_pred)
precision_SVM = precision_score(y_test, y_pred)
recall_SVM = recall_score(y_test, y_pred)
f1_SVM = f1_score(y_test, y_pred)
misclassification_rate_SVM = 1 - accuracy_SVM
zero_one_loss_value_SVM = zero_one_loss(y_test, y_pred)
mae_SVM = mean_absolute_error(y_test, y_pred)

# Print SVM classifier performance metrics
print("For SVM     ")
print("Accuracy:", accuracy_SVM)
print("Precision:", precision_SVM)
print("Recall:", recall_SVM)
print("F1 Score:", f1_SVM)
print("Misclassification Rate:", misclassification_rate_SVM)
print("Zero-One Loss:", zero_one_loss_value_SVM)
print("Mean Absolute Error:", mae_SVM)
print("    ******  ")

# K-Nearest Neighbors classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluate KNN classifier
accuracy_knn = accuracy_score(y_test, y_pred)
precision_knn = precision_score(y_test, y_pred)
recall_knn = recall_score(y_test, y_pred)
f1_knn = f1_score(y_test, y_pred)
misclassification_rate_knn = 1 - accuracy_knn
zero_one_loss_knn = zero_one_loss(y_test, y_pred)
mae_knn = mean_absolute_error(y_test, y_pred)
```

```python
# Print KNN classifier performance metrics
print("Accuracy (KNN):", accuracy_knn)
print("Precision (KNN):", precision_knn)
print("Recall (KNN):", recall_knn)
print("F1 Score (KNN):", f1_knn)
print("Misclassification Rate (KNN):", misclassification_rate_knn)
print("Zero-One Loss (KNN):", zero_one_loss_knn)
print("Mean Absolute Error (KNN):", mae_knn)
print("    ******  ")


# Multinomial Naive Bayes classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
y_pred = naive_bayes.predict(X_test)

# Evaluate Naive Bayes classifier
accuracy_nb = accuracy_score(y_test, y_pred)
precision_nb = precision_score(y_test, y_pred)
recall_nb = recall_score(y_test, y_pred)
f1_nb = f1_score(y_test, y_pred)
misclassification_rate_nb = 1 - accuracy_nb
zero_one_loss_nb = zero_one_loss(y_test, y_pred)
mae_nb = mean_absolute_error(y_test, y_pred)

# Print Naive Bayes classifier performance metrics
print("For Naive Bayes:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("F1 Score:", f1_nb)
print("Misclassification Rate:", misclassification_rate_nb)
print("Zero-One Loss:", zero_one_loss_nb)
print("Mean Absolute Error:", mae_nb)
print("    ******  ")


# Artificial Neural Network classifier
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
y_pred = y_pred.flatten()
```

```python
# Print KNN classifier performance metrics
print("Accuracy (KNN):", accuracy_knn)
print("Precision (KNN):", precision_knn)
print("Recall (KNN):", recall_knn)
print("F1 Score (KNN):", f1_knn)
print("Misclassification Rate (KNN):", misclassification_rate_knn)
print("Zero-One Loss (KNN):", zero_one_loss_knn)
print("Mean Absolute Error (KNN):", mae_knn)
print("    ******  ")


# Multinomial Naive Bayes classifier
naive_bayes = MultinomialNB()
naive_bayes.fit(X_train, y_train)
y_pred = naive_bayes.predict(X_test)

# Evaluate Naive Bayes classifier
accuracy_nb = accuracy_score(y_test, y_pred)
precision_nb = precision_score(y_test, y_pred)
recall_nb = recall_score(y_test, y_pred)
f1_nb = f1_score(y_test, y_pred)
misclassification_rate_nb = 1 - accuracy_nb
zero_one_loss_nb = zero_one_loss(y_test, y_pred)
mae_nb = mean_absolute_error(y_test, y_pred)

# Print Naive Bayes classifier performance metrics
print("For Naive Bayes:")
print("Accuracy:", accuracy_nb)
print("Precision:", precision_nb)
print("Recall:", recall_nb)
print("F1 Score:", f1_nb)
print("Misclassification Rate:", misclassification_rate_nb)
print("Zero-One Loss:", zero_one_loss_nb)
print("Mean Absolute Error:", mae_nb)
print("    ******  ")


# Artificial Neural Network classifier
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
y_pred = y_pred.flatten()
```

```python
# Artificial Neural Network classifier
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
y_pred = y_pred.flatten()

# Evaluate ANN classifier
accuracy_ann = accuracy_score(y_test, y_pred)
precision_ann = precision_score(y_test, y_pred)
recall_ann = recall_score(y_test, y_pred)
f1_ann = f1_score(y_test, y_pred)
misclassification_rate_ann = 1 - accuracy_ann
zero_one_loss_ann = zero_one_loss(y_test, y_pred)
mae_ann = mean_absolute_error(y_test, y_pred)

# Print ANN classifier performance metrics
print("For ANN:")
print("Accuracy:", accuracy_ann)
print("Precision:", precision_ann)
print("Recall:", recall_ann)
print("F1 Score:", f1_ann)
print("Misclassification Rate:", misclassification_rate_ann)
print("Zero-One Loss:", zero_one_loss_ann)
print("Mean Absolute Error:", mae_ann)
print("    ******  ")


# Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)
y_pred = random_forest.predict(X_test)

# Evaluate Random Forest classifier
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)
misclassification_rate_rf = 1 - accuracy_rf
zero_one_loss_rf = zero_one_loss(y_test, y_pred)
mae_rf = mean_absolute_error(y_test, y_pred)

# Print Random Forest classifier performance metrics
```

```
# Artificial Neural Network classifier
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=1)
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)
y_pred = y_pred.flatten()

# Evaluate ANN classifier
accuracy_ann = accuracy_score(y_test, y_pred)
precision_ann = precision_score(y_test, y_pred)
recall_ann = recall_score(y_test, y_pred)
f1_ann = f1_score(y_test, y_pred)
misclassification_rate_ann = 1 - accuracy_ann
zero_one_loss_ann = zero_one_loss(y_test, y_pred)
mae_ann = mean_absolute_error(y_test, y_pred)

# Print ANN classifier performance metrics
print("For ANN:")
print("Accuracy:", accuracy_ann)
print("Precision:", precision_ann)
print("Recall:", recall_ann)
print("F1 Score:", f1_ann)
print("Misclassification Rate:", misclassification_rate_ann)
print("Zero-One Loss:", zero_one_loss_ann)
print("Mean Absolute Error:", mae_ann)
print("    ******  ")


# Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)
random_forest.fit(X_train, y_train)
y_pred = random_forest.predict(X_test)

# Evaluate Random Forest classifier
accuracy_rf = accuracy_score(y_test, y_pred)
precision_rf = precision_score(y_test, y_pred)
recall_rf = recall_score(y_test, y_pred)
f1_rf = f1_score(y_test, y_pred)
misclassification_rate_rf = 1 - accuracy_rf
zero_one_loss_rf = zero_one_loss(y_test, y_pred)
mae_rf = mean_absolute_error(y_test, y_pred)

# Print Random Forest classifier performance metrics
```

```
# Print Random Forest classifier performance metrics
print("For Random Forest:")
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
print("Misclassification Rate:", misclassification_rate_rf)
print("Zero-One Loss:", zero_one_loss_rf)
print("Mean Absolute Error:", mae_rf)
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

model_names = ["SVM", "KNN", "NB", "ANN", "RFC"]
accuracy_scores = [accuracy_SVM, accuracy_knn, accuracy_nb, accuracy_ann, accuracy_rf]
precision_scores = [precision_SVM, precision_knn, precision_nb, precision_ann, precision_rf]
Recall_scores = [recall_SVM, recall_knn, recall_nb, recall_ann, recall_rf]
F_measure_scores = [f1_SVM, f1_knn, f1_nb, f1_ann, f1_rf]
Mae_scores = [mae_SVM, mae_knn, mae_nb, mae_ann, mae_rf]

model_metrics = {
    "Model": model_names,
    "Accuracy": accuracy_scores,
    "precision": precision_scores,
    "Recall": Recall_scores,
    "F_measure": F_measure_scores,
    "Mean Absolute Error": Mae_scores
}

df = pd.DataFrame(model_metrics)
plt.figure(figsize=(12, 6))

bar_width = 0.15
index = np.arange(len(model_names))

plt.bar(index - 2 * bar_width, df["Accuracy"], bar_width, label="Accuracy")
plt.bar(index - bar_width, df["precision"], bar_width, label="Precision")
plt.bar(index, df["Recall"], bar_width, label="Recall")
plt.bar(index + bar_width, df["F_measure"], bar_width, label="F_measure")
plt.bar(index + 2 * bar_width, df["Mean Absolute Error"], bar_width, label="MAE")

plt.title("Performance metrics comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.legend()
plt.grid(True)
plt.xticks(index, model_names, rotation=45)
plt.tight_layout()
plt.show()
```

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

model_names = ["SVM", "KNN", "NB", "ANN", "RFC"]
accuracy_scores = [accuracy_SVM, accuracy_knn, accuracy_nb, accuracy_ann, accuracy_rf]
precision_scores = [precision_SVM, precision_knn, precision_nb, precision_ann, precision_rf]
Recall_scores = [recall_SVM, recall_knn, recall_nb, recall_ann, recall_rf]
F_measure_scores = [f1_SVM, f1_knn, f1_nb, f1_ann, f1_rf]
Mae_scores = [mae_SVM, mae_knn, mae_nb, mae_ann, mae_rf]

model_metrics = {
    "Model": model_names,
    "Accuracy": accuracy_scores,
    "precision": precision_scores,
    "Recall": Recall_scores,
    "F_measure": F_measure_scores,
    "Mean Absolute Error": Mae_scores
}

df = pd.DataFrame(model_metrics)
plt.figure(figsize=(12, 6))

bar_width = 0.15
index = np.arange(len(model_names))

plt.bar(index - 2 * bar_width, df["Accuracy"], bar_width, label="Accuracy")
plt.bar(index - bar_width, df["precision"], bar_width, label="Precision")
plt.bar(index, df["Recall"], bar_width, label="Recall")
plt.bar(index + bar_width, df["F_measure"], bar_width, label="F_measure")
plt.bar(index + 2 * bar_width, df["Mean Absolute Error"], bar_width, label="MAE")

plt.title("Performance metrics comparison")
plt.xlabel("Model")
plt.ylabel("Score")
plt.legend()
plt.grid(True)
plt.xticks(index, model_names, rotation=45)
plt.tight_layout()
plt.show()
```

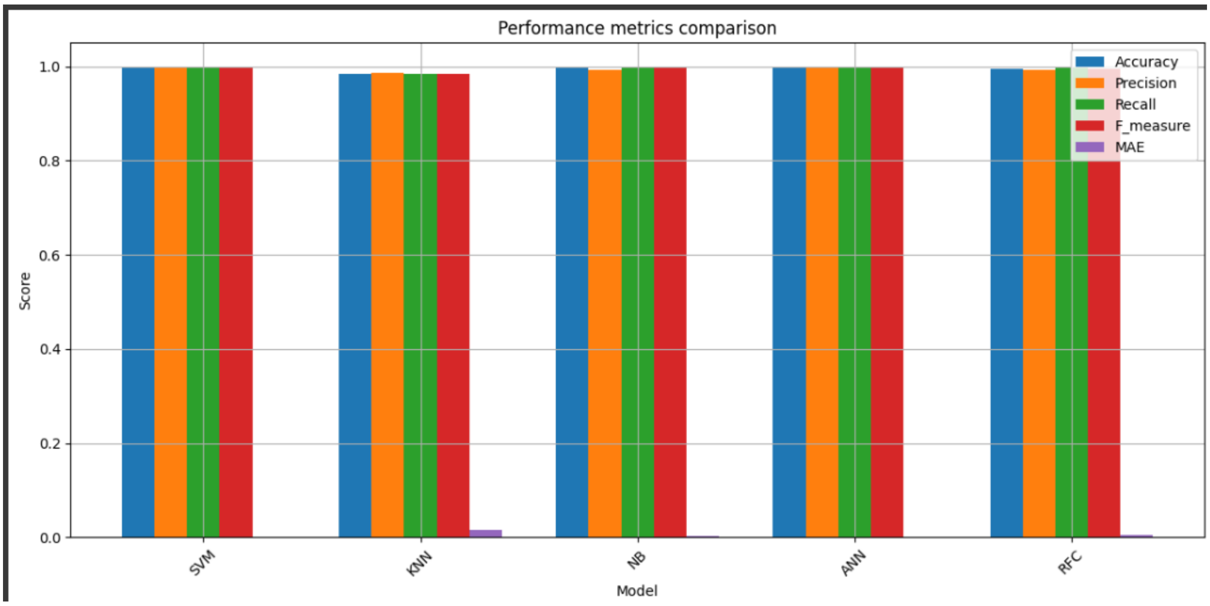
5. Performance analysis

The performance metrics obtained for each model are summarized as follows:

Table 1. Performance Metrics of Machine Learning Models

Model	Accuracy	Precision	Recall	F1 Score	Mean Absolute Error
SVM	1.0	1.0	1.0	1.0	0.0
KNN	0.984	0.985	0.984	0.985	0.016
Naive Bayes	0.996	0.993	1.0	0.997	0.004
Artificial NN	1.0	1.0	1.0	1.0	0.0
Random Forest	0.995	0.993	0.998	0.995	0.005

6. Testing the models



7. Conclusion and future findings

The comparative analysis underscores the significance of machine learning in discerning sentiments from airline reviews. Among the evaluated models, Random Forest emerges as the most accurate, achieving an impressive 99.5% accuracy rate. These findings offer actionable insights for airlines to enhance customer service and satisfaction by leveraging sentiment analysis. By implementing proactive strategies informed by machine learning, airlines can foster stronger customer relationships and improve business performance. Looking ahead, future research could explore advanced techniques to further enhance sentiment analysis accuracy and continuously optimize customer relationship management practices in the aviation industry.

8. References and Datasets

Sentimental Analysis of Customer Feedback and Reviews for Airline Services using Language Representation Model.

9. Link to colab notebook

Notebook link.