SENTIMENT ANALYSIS REPORT

1. INTRODUCTION

This report is an analysis of movie reviews dataset which contains two sub folders: positive and negative reviews folders (pos and neg). The aim of is to preprocess the text data, build classification models to conduct sentiment analysis on the given dataset. The analysis includes steps for preprocessing, model training using Support Vector Machine (SVM), decision Tree Classifiers and evaluation metrics such as confusion matrices.

2. DATA ANALYSIS DESCRIPTION

2.1. Data Loading

The data was loaded into a Pandas DataFrame, combining text content and corresponding sentiment labels (0 for positive, 1 for negative).

2.2. Pre processing Steps

- Implemented lemmatization using Spacy to reduce words to their root form
- Using Spacy, stop-words and non-alphabetic tokens were removed
- Converted all text into lowercase
- Used TF-IDF vectorization to limit vocabulary size to 1,000
- Used PCA to reduce TF-IDF feature space from 1,000 to 500, while retaining key information

3. DATASET SPLITTING

The dataset was then split into training and testing subsets

- Train-test split: 80% train size, 20% test size
- Random state : set to 42

4. CLASSIFICATION MODELS

Classification Models Implemented are;

- **Support Vector Machine (SVM)**: A linear kernel was used for effective binary classification.
- **Decision Tree Classifier**: This algorithm was applied to compare performance.

5. EVALUATION METRICS

The metrics below were used to evaluate model performance:

- **Confusion Matrix**: gives us insight into the true positives, true negatives, false positives, and false negatives.
- **Classification Report**: Includes precision, recall, F1-score, and accuracy for both classes.

6. VISUALIZATION

In order to better understand the output of the Decision Tree Classifier, Seaborn and matplotlib were implemented to plot a heatmap.

7. CODE IMPLEMENTATION

7.1. Data loading and preprocessing

```
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import spacy

# Load SpaCy's English model
nlp = spacy.load("en_core_web_sm")

# Dataset path
dataset = "/Users/damiojikutu/Desktop/Data analystics/python/Machine
Learning/Movie reviews"

if not os.path.exists(dataset):
    raise FileNotFoundError(f"Dataset folder not found: {dataset}")

# Prepare dataset
texts = []
labels = []
for label, folder in enumerate(['pos', 'neg']):
    folder_path = os.path.join(dataset, folder)
```

```
    if not os.path.exists(folder_path):
        raise FileNotFoundError(f"Folder not found: {folder_path}")
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)
        with open(file_path, 'r', encoding='utf-8') as file:
            texts.append(file.read())
            labels.append(label)

# Create a DataFrame
data = pd.DataFrame({'text': texts, 'label': labels})

# Define preprocessing function using SpaCy
def preprocess_with_spacy(text):

    doc = nlp(text.lower())  # Convert to lowercase
    tokens = [token.lemma_ for token in doc if token.is_alpha and not
token.is_stop]
    return " ".join(tokens)

# Apply preprocessing
data['processed_text'] = data['text'].apply(preprocess_with_spacy)
```

7.2. vectorization and feature reduction

```
Vectorize text using TF-IDF
vectorizer = TfidfVectorizer(max_features=1000)  # Limit features to 1000
X = vectorizer.fit_transform(data['processed_text']).toarray()
y = data['label']

# Feature reduction with PCA
pca = PCA(n_components=500)  # Reduce dimensions to 500
X_reduced = pca.fit_transform(X)
```

7.3. Train-Test Split

```
8. X_train, X_test, y_train, y_test = train_test_split(X_reduced, y,
   test_size=0.2, random_state=42)
```

7.4 Model Training and Evaluation.

```
# Support Vector Machine (SVM)
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

#  Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Evaluate SVM
svm_predictions = svm_model.predict(X_test)
svm_conf_matrix = confusion_matrix(y_test, svm_predictions)
print("SVM Confusion Matrix:\n", svm_conf_matrix)
```

```
print("\nSVM Classification Report:\n", classification_report(y_test,
svm_predictions))



# Evaluate Decision Tree
dt_predictions = dt_model.predict(X_test)
dt_conf_matrix = confusion_matrix(y_test, dt_predictions)
print("Decision Tree Confusion Matrix:\n", dt_conf_matrix)
print("\nDecision Tree Classification Report:\n",
classification_report(y_test, dt_predictions))
```

7.5 Visualisation.

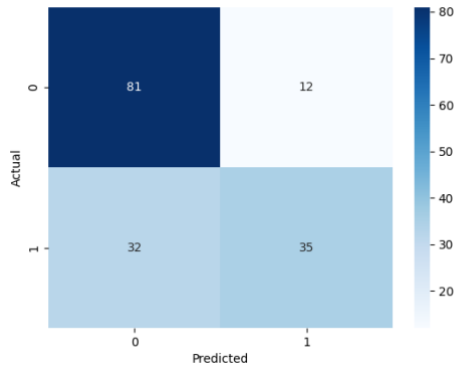
```
#visualise Decisition Tree
sns. heatmap(dt_conf_matrix , annot=True, cmap="Blues")
plt.title =("Decision Tree aConfusion Matrix:")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#visualise SVM
sns. heatmap(svm_conf_matrix , annot=True, cmap="Blues")
plt.title =("SVM Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

RESULTS

SVM confusion matrix :

[[81 12]
 [32 35]]

SVM confusion matrix visualized:



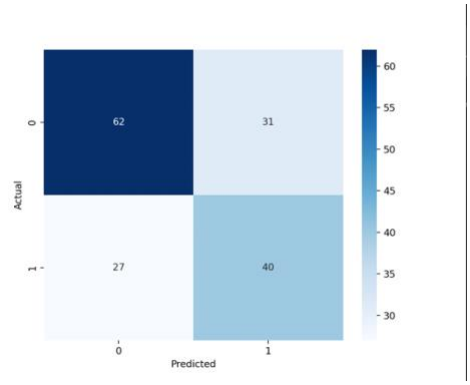
SVM classification report:

	Precision	Recall	F1-Score	Support
0	0.72	0.87	0.79	93
1	0.74	0.52	0.61	67
accuracy			0.72	160
macro avg	0.73	0.70	0.70	160
weighted avg	0.73	0.72	0.71	160

Decision Tree confusion matrix:

[[62 31]
 [27 40]]

Decision Tree confusion matrix visualized:



Decision Tree Classification Report:

	Precision	Recall	F1-Score	Support
0	0.70	0.67	0.68	93
1	0.56	0.60	0.58	67
Accuracy			0.64	160
macro avg	0.63	0.63	0.63	160
weighted avg	0.64	0.64	0.64	160