nlpimdb

April 24, 2025

0.0.1 Problem Statement

The primary objective of this project is to build a machine learning classification model that can predict the sentiment of IMDb movie reviews. The dataset contains a collection of movie reviews, and each review is labeled as either positive or negative. Using text preprocessing, feature extraction techniques (such as TF-IDF), and various classification algorithms, the project will aim to develop a model that can effectively classify the sentiment of movie reviews. The model's performance will be evaluated using standard classification metrics, such as accuracy, precision, recall, and F1-score.

[45]:
```python
## Necessary Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import randint
from sklearn.metrics import classification_report, confusion_matrix
import re
import nltk

nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Error loading stopwords: <urlopen error [Errno -3]
[nltk_data]     Temporary failure in name resolution>
[nltk_data] Error loading wordnet: <urlopen error [Errno -3] Temporary
[nltk_data]     failure in name resolution>
```

[45]: False

[46]:
```python
# Load dataset
df = pd.read_csv("/kaggle/input/imdbdataset/Imdb.csv")  # Assuming the dataset␣
↪has 'review' and 'sentiment' columns
print(df.head())
```

```
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive
```

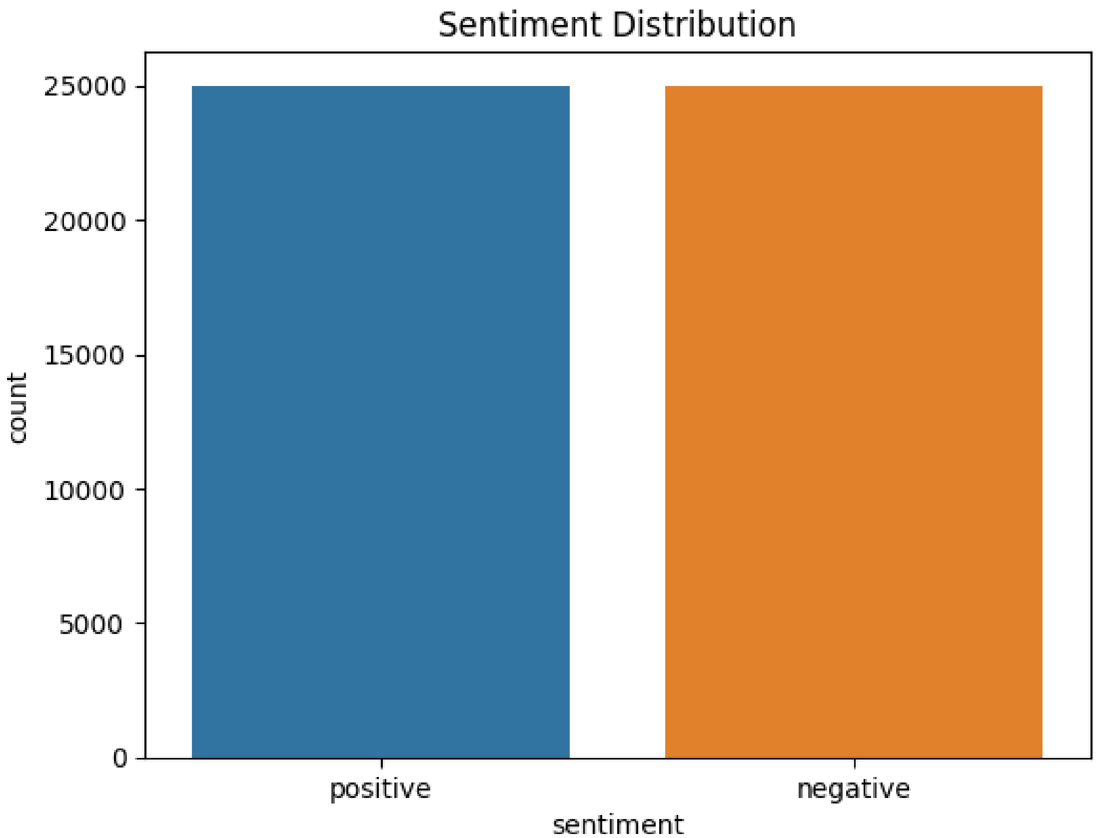
[3]:
```python
# Basic exploration
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
None
```

[5]:
```python
print(df['sentiment'].value_counts())
```

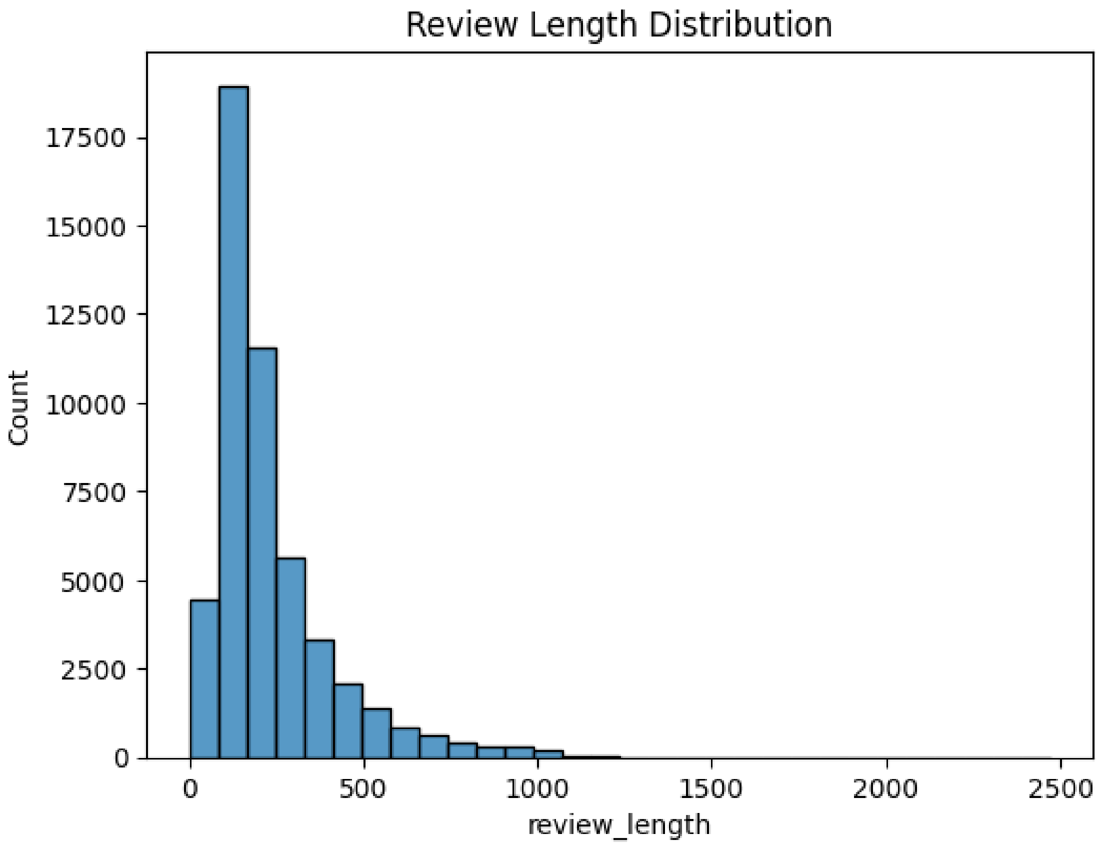
```
sentiment
positive    25000
negative    25000
Name: count, dtype: int64
```

[6]:
```python
# Visualize class distribution
sns.countplot(data=df, x='sentiment')
plt.title("Sentiment Distribution")
plt.show()
```



[7]:
```
# Check review lengths
df['review_length'] = df['review'].apply(lambda x: len(x.split()))
sns.histplot(df['review_length'], bins=30)
plt.title("Review Length Distribution")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version.
Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



0.1 PreProcessing

[47]:
```
# Preprocessing
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def clean_text(text):
    text = re.sub(r"<.*?>", "", text)  # Remove HTML tags
    text = re.sub(r"[^a-zA-Z]", " ", text)  # Remove non-letters
    tokens = text.lower().split()
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in␣
 ↪stop_words]
    return " ".join(tokens)

df['clean_review'] = df['review'].apply(clean_text)
```

1 FEATURE ENGINEERING

[5]:
```
# Basic features
df['word_count'] = df['clean_review'].apply(lambda x: len(x.split()))
df['char_count'] = df['clean_review'].apply(len)
df['avg_word_len'] = df['char_count'] / df['word_count']
```

[6]:
```
print(df['word_count'])
```

```
0        162
1         86
2         84
3         64
4        125
        …
49995     77
49996     57
49997    113
49998    112
49999     66
Name: word_count, Length: 50000, dtype: int64
```

[12]:
```
print(df['char_count'])
```

```
0        1070
1         641
2         565
3         425
4         840
         …
49995     493
49996     395
49997     783
49998     808
49999     412
Name: char_count, Length: 50000, dtype: int64
```

[13]:
```
print(df['avg_word_len'])
```

```
0        6.604938
1        7.453488
2        6.726190
3        6.640625
4        6.720000
           …
49995    6.402597
49996    6.929825
49997    6.929204
```

```
49998    7.214286
49999    6.242424
Name: avg_word_len, Length: 50000, dtype: float64
```

2 TFIDIF VECTORIZER PROCESSING

[7]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF vectorization
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['clean_review']).toarray()
```

[12]:
```
print(X)
```

```
[[0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 …
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]
 [0. 0. 0. … 0. 0. 0.]]
```

[13]:
```
# Encode target labels
df['label'] = df['sentiment'].map({'positive': 1, 'negative': 0})
y = df['label']
```

[14]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

3 Random Forest Classifier model training

[15]:
```
## hyperparamter
param_dist_rf = {
     'n_estimators': (60, 75),        # fewer trees = faster
    'max_depth': [10, 15],              # limit tree complexity
    'min_samples_split': (2, 5)
}
```

[16]:
```
rand_rf = RandomizedSearchCV(RandomForestClassifier(),␣
 ↪param_distributions=param_dist_rf,
                              n_iter=8, cv=5, scoring='accuracy',␣
 ↪random_state=42, n_jobs=-1)
rand_rf.fit(X_train, y_train)
```

[16]:
```
RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n_iter=8,
                   n_jobs=-1,
                   param_distributions={'max_depth': [10, 15],
                                        'min_samples_split': (2, 5),
                                        'n_estimators': (60, 75)},
                   random_state=42, scoring='accuracy')
```

[18]:
```python
rand_rf.best_params_
```

[18]:
```
{'n_estimators': 75, 'min_samples_split': 2, 'max_depth': 15}
```

[19]:
```python
ypred_rf=rand_rf.predict(X_test)
```

[20]:
```python
print(ypred_rf)
```

```
[0 1 0 … 1 0 1]
```

[21]:
```python
print("Trainin Score",rand_rf.score(X_train,y_train))
print("Testing Score",rand_rf.score(X_test,y_test))
```

```
Trainin Score 0.886025
Testing Score 0.8298
```

[22]:
```python
df=pd.DataFrame({'Actual':y_test,'Predicted':ypred_rf})
df
```

[22]:

	Actual	Predicted
33553	1	0
9427	1	1
199	0	0
12447	1	1
39489	0	0
…	…	…
28567	0	0
25079	1	1
18707	1	1
15200	0	0
5857	1	1

[10000 rows x 2 columns]

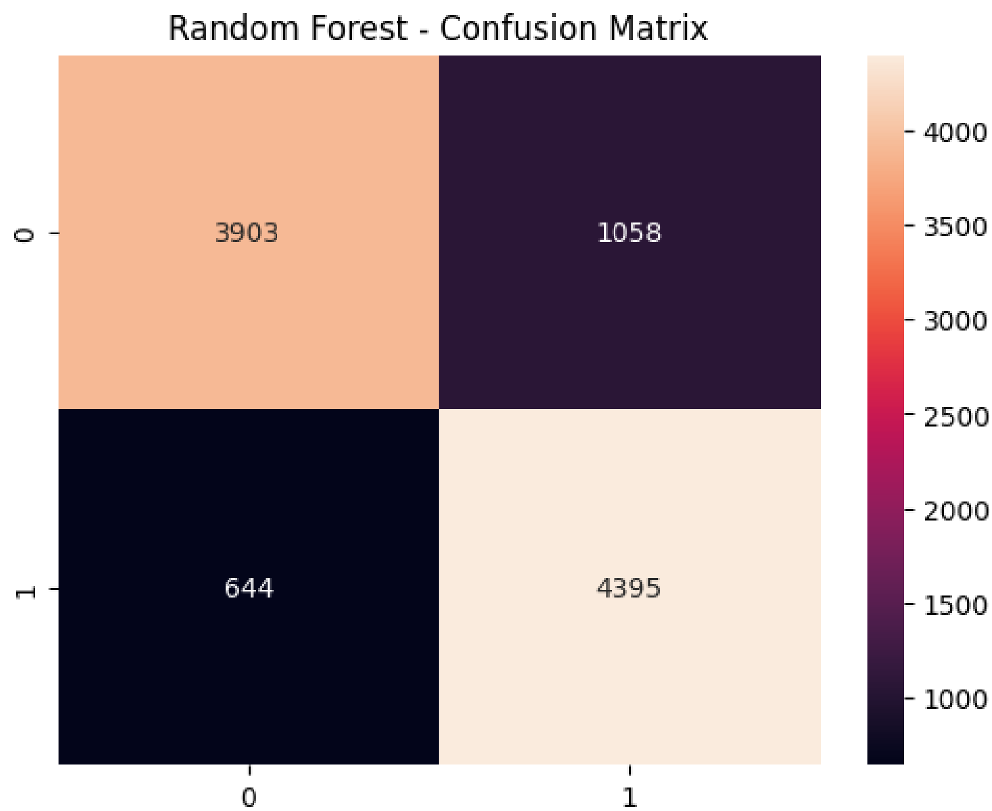
[23]:
```python
cm_rf=confusion_matrix(y_test,ypred_rf)
print(cm_rf)
print(classification_report(y_test,ypred_rf))
```

```
[[3903 1058]
 [ 644 4395]]
              precision    recall  f1-score   support
```

```
              0       0.86      0.79      0.82      4961
              1       0.81      0.87      0.84      5039

       accuracy                           0.83     10000
      macro avg       0.83      0.83      0.83     10000
   weighted avg       0.83      0.83      0.83     10000
```

[24]:
```
sns.heatmap(cm_rf, annot=True, fmt='d')
plt.title(f"Random Forest - Confusion Matrix")
plt.show()
```



The Accuracy for Random forest is 83%, which is the least among all the other model trainings

Logistic Regression Model

[25]:
```
param_dist_lr = {
    'C': loguniform(0.001, 10),
    'solver': ['liblinear']
}
```

[26]:
```
model_lr=LogisticRegression()
```

[27]:
```
rand_lr = RandomizedSearchCV(LogisticRegression(),␣
↪param_distributions=param_dist_lr,
                                n_iter=10, cv=5, scoring='accuracy',␣
↪random_state=42)
rand_lr.fit(X_train, y_train)
```

[27]:
```
RandomizedSearchCV(cv=5, estimator=LogisticRegression(),
                   param_distributions={'C':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at
0x7c289fcf6dd0>,
                                        'solver': ['liblinear']},
                   random_state=42, scoring='accuracy')
```

[28]:
```
print("Trainin Score",rand_lr.score(X_train,y_train))
print("Testing Score",rand_lr.score(X_test,y_test))
```

```
Trainin Score 0.92285
Testing Score 0.887
```

[29]:
```
print("Best Parameters for Logistic Regression:", rand_lr.best_params_)
print("Best Score (Train CV):", rand_lr.best_score_)
```

```
Best Parameters for Logistic Regression: {'C': 2.9154431891537547, 'solver':
'liblinear'}
Best Score (Train CV): 0.8863749999999999
```

[30]:
```
y_pred_lr = rand_lr.predict(X_test)
```

[31]:
```
print(y_pred_lr)
```

```
[0 1 0 … 1 0 1]
```

[32]:
```
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred_lr})
df
```

[32]:

	Actual	Predicted
33553	1	0
9427	1	1
199	0	0
12447	1	1
39489	0	0
…	…	…
28567	0	0
25079	1	1
18707	1	1

```
15200       0          0
5857        1          1

[10000 rows x 2 columns]
```

[33]:

```
cm_lr=confusion_matrix(y_test,y_pred_lr)
print(cm_lr)
print("Classification Report:\n", classification_report(y_test, y_pred_lr))
```

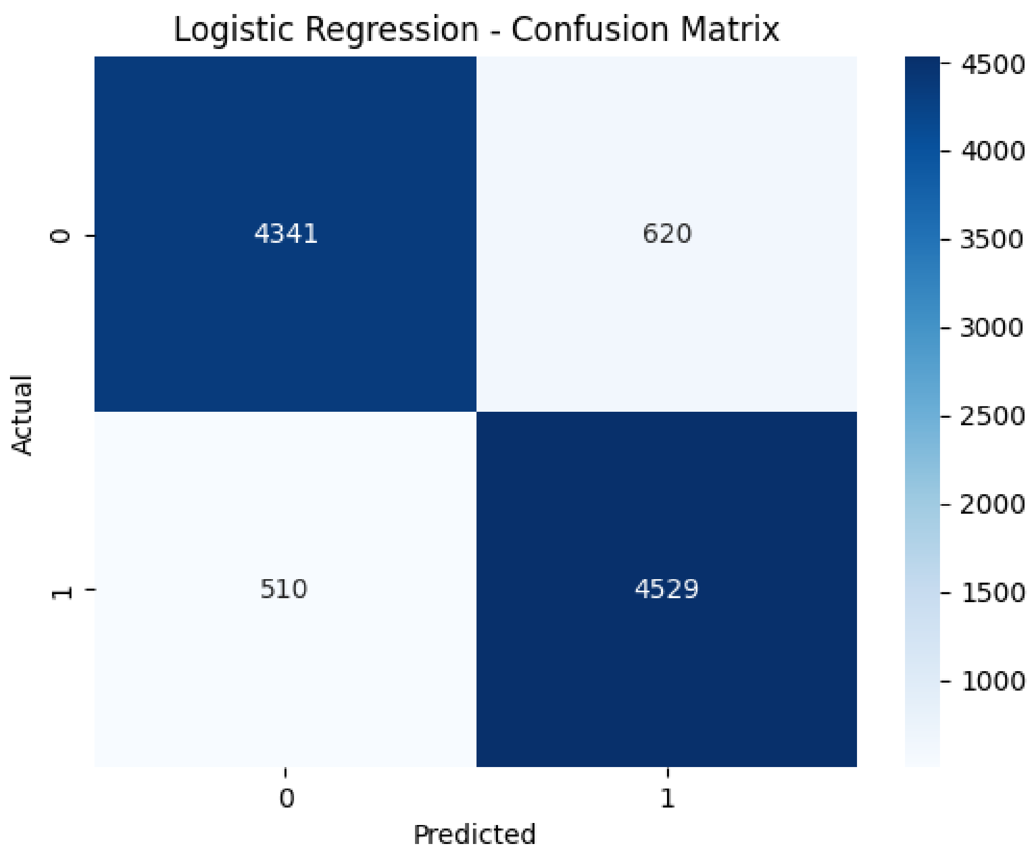
```
[[4341  620]
 [ 510 4529]]
Classification Report:
               precision    recall  f1-score   support

           0       0.89      0.88      0.88      4961
           1       0.88      0.90      0.89      5039

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

[34]:

```
sns.heatmap(cm_lr, annot=True, fmt='d',cmap='Blues')
plt.title(f"Logistic Regression - Confusion Matrix")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Logistic Regression Accuracy is 89%,which is a great model predictions compared to Random forest

4 Support Vector Machine

[35]:
```
svm=LinearSVC()
```

[36]:
```
svm.fit(X_train,y_train)
```

[36]:
```
LinearSVC()
```

[37]:
```
y_pred_svm=svm.predict(X_test)
```

[38]:
```
print(y_pred_svm)
```

```
[0 1 0 … 1 0 1]
```

[39]:
```
print("Training Score",svm.score(X_train,y_train))
print("Testing Score",svm.score(X_test,y_test))
```

```
Training Score 0.92985
Testing Score 0.8819
```

[40]:
```
df=pd.DataFrame({'Actual':y_test,'Predicted':y_pred_svm})
df
```

[40]:

	Actual	Predicted
33553	1	0
9427	1	1
199	0	0
12447	1	1
39489	0	0
…	…	…
28567	0	0
25079	1	1
18707	1	1
15200	0	0
5857	1	1

[10000 rows x 2 columns]

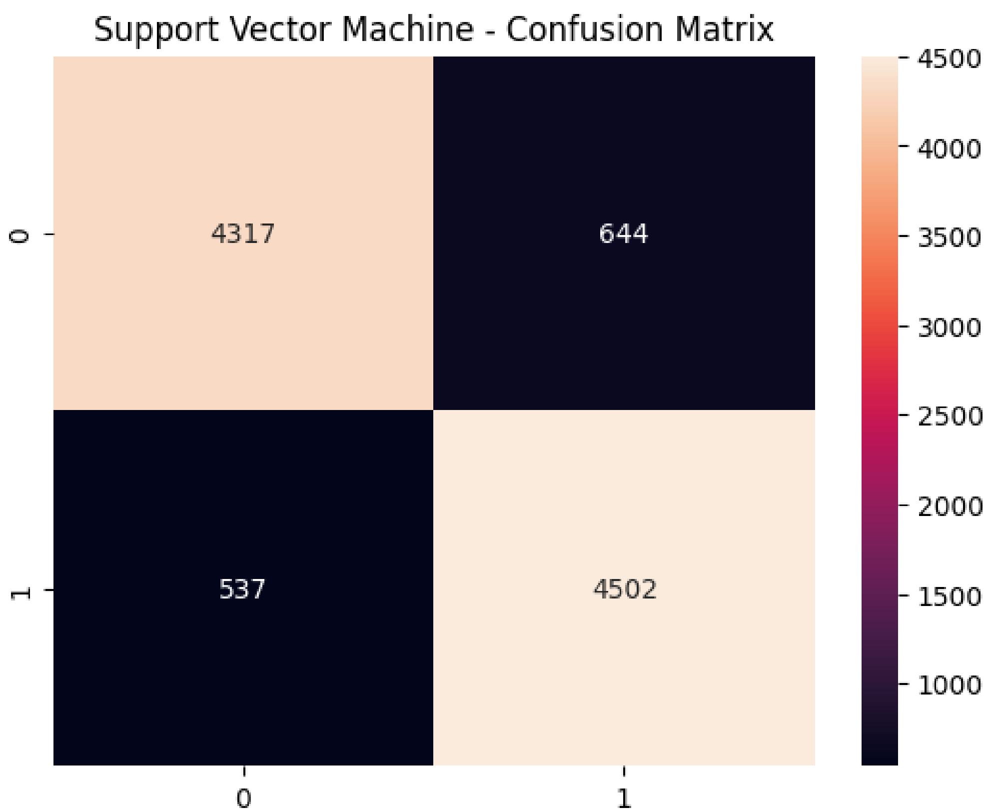
[41]:
```
cm_rf=confusion_matrix(y_test,y_pred_svm)
print(cm_rf)
print(classification_report(y_test,y_pred_svm))
```

```
[[4317  644]
 [ 537 4502]]
              precision    recall  f1-score   support

           0       0.89      0.87      0.88      4961
           1       0.87      0.89      0.88      5039

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

[42]:
```
sns.heatmap(cm_rf, annot=True, fmt='d')
plt.title(f"Support Vector Machine - Confusion Matrix")
plt.show()
```



The Acuuracy for SVC is 88% and is the second best model prediction for IMDB Predictions

5 NOTE:

The **LSTM** AND **BERT** models **were neither covered** in the recorded nor in the live sections. So, Im not sure how you expect us to implement somethiing which were not taught by you but I have done LSTM based on my understanding from Youtube videos.

6 LSTM

[48]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Tokenize
tokenizer = Tokenizer(num_words=10000)
tokenizer.fit_on_texts(df['clean_review'])
X_seq = tokenizer.texts_to_sequences(df['clean_review'])
```

```
X_pad = pad_sequences(X_seq, maxlen=200)
```

[49]:
```
X_train_lstm, X_test_lstm, y_train_lstm, y_test_lstm = train_test_split(X_pad,␣
↪y, test_size=0.2, random_state=42)
```

[50]:
```
model_lstm = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=200),
    LSTM(64),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
2025-04-24 04:07:28.871150: E
external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
```

[24]:
```
model_lstm.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
model_lstm.fit(X_train_lstm, y_train_lstm, epochs=3, batch_size=128,␣
↪validation_split=0.1)
```

```
Epoch 1/3
282/282              66s 217ms/step -
accuracy: 0.7593 - loss: 0.4704 - val_accuracy: 0.8790 - val_loss: 0.2873
Epoch 2/3
282/282              60s 212ms/step -
accuracy: 0.9169 - loss: 0.2218 - val_accuracy: 0.8640 - val_loss: 0.3100
Epoch 3/3
282/282              60s 214ms/step -
accuracy: 0.9407 - loss: 0.1659 - val_accuracy: 0.8832 - val_loss: 0.3386
```

[24]:
```
<keras.src.callbacks.history.History at 0x7eb8793c98d0>
```

[25]:
```
loss, acc = model_lstm.evaluate(X_test_lstm, y_test_lstm)
print(f"LSTM Test Accuracy: {acc:.4f}")
```

```
313/313              9s 29ms/step -
accuracy: 0.8849 - loss: 0.3178
LSTM Test Accuracy: 0.8838
```

LSTM model also predicted 88% accuracy which can be considered as the third good model predictions

Conclusion: The Logistic Regression model training was the best compared to all other models because it gave a accurate score of 89%.

7 WORD CLOUDS VISUALIZATION

[13]:
```
# Combining all positive and negative reviews into two large text blobs
positive_text = " ".join(df[df['sentiment'] == 'positive']['review'].
 ↪astype(str))
negative_text = " ".join(df[df['sentiment'] == 'negative']['review'].
 ↪astype(str))
```

[15]:
```
# Generating word clouds
wordcloud_pos = WordCloud(width=800, height=400, background_color='white').
 ↪generate(positive_text)
wordcloud_neg = WordCloud(width=800, height=400, background_color='black',␣
 ↪colormap='Pastel1').generate(negative_text)
```

[17]:
```
# Plotting
fig, axs = plt.subplots(1, 2, figsize=(18, 8))
axs[0].imshow(wordcloud_pos, interpolation='bilinear')
axs[0].set_title('Positive Reviews Word Cloud', fontsize=18)
axs[0].axis('off')

axs[1].imshow(wordcloud_neg, interpolation='bilinear')
axs[1].set_title('Negative Reviews Word Cloud', fontsize=18)
axs[1].axis('off')

plt.tight_layout()
plt.show()
```

8 Predicting future sentiments based on new reviews

For predicting future sentiments based on new reviews, I have choosen LogisticRegression model for training because it gave the best accurate accuracy i.e 89% based on all other models.

[53]:
```
## defining function for new texts
def predict_sentiment(new_reviews):
```

```
    preprocessed_reviews=[
            ' '.join([word for word in word_tokenize(text.lower())
               if word.isalpha() and word not in stop_words]) ### removing␣
↪stopwords
            for text in new_reviews]

    texts_tfidf = vectorizer.transform(preprocessed_reviews) ### converting to␣
↪numerical values
    predictions = rand_lr.predict(texts_tfidf) ### model prediction
    return predictions
```

[54]:

```
print("Please enter a text to classify its sentiment(type 'exit' to stop):")
while True:
    user_input=input("Enter") ### user-input defined
    if user_input.lower()=="exit":
        print("exiting program")
        break
    predicted_sentiment = predict_sentiment([user_input])
    print(f"Review : {user_input}\nPredicted Sentiment :␣
↪{predicted_sentiment[0]}\n")
```

```
Please enter a text to classify its sentiment(type 'exit' to stop):

Enter The pakage implies that Warren Beatty and Goldie Hawn are pulling off a
huge bank robbery, but that's not what I got out of it! I didn't get anything!
In the first half there's a new character (without introduction) in every other
scene. The first half-hour is completely incomprehensible, the rest is just one
long, annoying, underlit chase scene. There's always an irritating sound in the
background whether it's a loud watch ticking, a blaring siren, a train
whistling, or even the horrible score by Quincy Jones. There are a lot of parts
that are laughably bad, too. Like, the bad guys chasing Beatty on thin ice with
a CAR! Or, the police arriving at the scene roughly fifteen times. I really
hated this movie!

Review : The pakage implies that Warren Beatty and Goldie Hawn are pulling off a
huge bank robbery, but that's not what I got out of it! I didn't get anything!
In the first half there's a new character (without introduction) in every other
scene. The first half-hour is completely incomprehensible, the rest is just one
long, annoying, underlit chase scene. There's always an irritating sound in the
background whether it's a loud watch ticking, a blaring siren, a train
whistling, or even the horrible score by Quincy Jones. There are a lot of parts
that are laughably bad, too. Like, the bad guys chasing Beatty on thin ice with
a CAR! Or, the police arriving at the scene roughly fifteen times. I really
hated this movie!
Predicted Sentiment : 0


Enter "Autumn Spring" tells of the misadventures of a dapper, walrus faced, 78
(approx) year old Czech man who haplessly befuddles and bemuses all who know him
```

```
with his mischievous ways while his wife meticulously plans her funeral.
Centerpiece Hana (BrodskÃ½) shows us how to get babes to kiss you when your 78
and how to cop a feel in an elevator and get thanked for it as he pranks his way
from day to day in this warm and glowing look at old age and one man's creative,
amusing, but socially unacceptable ways of enjoying life while refusing to be
relegated to the old folk's home. "Autumn Spring" is a plodding, subtle comedy
with messages for all ages which will have the greatest appeal with more mature
foreign film buffs. (B+)

Review : "Autumn Spring" tells of the misadventures of a dapper, walrus faced,
78 (approx) year old Czech man who haplessly befuddles and bemuses all who know
him with his mischievous ways while his wife meticulously plans her funeral.
Centerpiece Hana (BrodskÃ½) shows us how to get babes to kiss you when your 78
and how to cop a feel in an elevator and get thanked for it as he pranks his way
from day to day in this warm and glowing look at old age and one man's creative,
amusing, but socially unacceptable ways of enjoying life while refusing to be
relegated to the old folk's home. "Autumn Spring" is a plodding, subtle comedy
with messages for all ages which will have the greatest appeal with more mature
foreign film buffs. (B+)
Predicted Sentiment : 1


Enter exit

exiting program
```

Here, 0 : Negative 1 : Positive