Movie Recommendation System and Sentiment Analysis

Overview

This paper introduces a movie recommendation system that leverages the Cosine Similarity algorithm to provide personalized movie recommendations. The system takes into account various factors, including:

- Genre: Recommends movies related to the user's input movie.
- Overview: Considers movie summaries.
- Cast: Takes into account the actors and actresses.
- Ratings: Incorporates user ratings.

Cosine Similarity Algorithm

The Cosine Similarity algorithm has proven effective in tests and accurately suggests relevant movies based on the user's preferences.

Sentiment Analysis

In addition to movie recommendations, the study explores sentiment analysis to classify reviews as either positive or negative. Two algorithms are employed for this task:

1. Naive Bayes (NB): A probabilistic classifier.
2. Support Vector Machine (SVM): Used for performance comparison.

The diversity of reviews requires careful consideration in choosing the right algorithm. Experimental results slightly favor SVM.

Python Libraries

Importing necessary libraries for data preprocessing, NLP, machine learning, and model evaluation.

In [ ]:

```
import numpy as np # Linear Algebra and lists
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import nltk #Used for NLP
from nltk.corpus import stopwords #MLP
```

```
from sklearn.feature_extraction.text import TfidfVectorizer #list to vector

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, c
import pickle
```

Read from CSV files into Pandas DataFrames.

In [ ]:
```
movies = pd.read_csv('tmdb_5000_movies.csv')
credits = pd.read_csv('tmdb_5000_credits.csv')
```

Before making anything like feature selection,feature extraction and classification.

firstly we start with basic data analysis.

Lets look at the first few rows of the first dataset.

In [ ]:
```
movies.head()
```

Showing the dimensions (number of rows and columns) of the 'movies' DataFrame.

In [ ]:
```
movies.shape
```

Lets look at the first few rows of the seconde dataset.

In [5]:
```
credits.head()
```

Out[5]:

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

merging the two datasets

In [6]:
```
movies = movies.merge(credits,on='title')
#merging the two datasets in movies, according to the title
```

In [ ]:
```
movies.shape
```

Display after merging

In [7]:
```
movies.head()
```

Out[7]:

	budget	genres	homepage	id	keywords	original_lang
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

In [8]:

```
#Printing the column names of the 'movies' DataFrame.
print("movies.columns")
```

```
movies.columns
```

Data Preprocessing

Selecting specific columns from the 'movies' DataFrame

In [9]:

```
movies = movies[['movie_id','title','overview','genres','keywords','cast','crew']]
#only kept essential coulums, dropped only the ones required
```

In [10]:
```
movies.head()
```

Out[10]:

	budget	genres	homepage	id	keywords	original_lang
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

Importing the 'ast' module (Abstract Syntax Trees) for literal_eval function.

In [11]:
```
import ast
#abstract syntax trees
```

In [12]:
```
def convert(text):
    L = []
    for i in ast.literal_eval(text):
        L.append(i['name'])
    return L
```

```
#The literal_eval safely evaluate an expression node or a string containing a Python l
#Reads input in the form of a dictionary and appends name only.
```

Removing rows with missing values (NaN) from the 'movies' DataFrame in-place.

In [13]:
```
movies.dropna(inplace=True)
```

In [14]:
```
movies['genres'] = movies['genres'].apply(convert)
movies.head()
#calling function convert and doing the function for genres
```

Out[14]:

	budget	genres	homepage	id	keywords	original_langu
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	
4	260000000	[Action, Adventure, Science Fiction]	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	

5 rows × 23 columns

In [15]:
```
movies['keywords'] = movies['keywords'].apply(convert)
movies.head()
#same thing for key words
```

Out[15]:

	budget	genres	homepage	id	keywords	original_langu
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	http://www.avatarmovie.com/	19995	[culture clash, future, space war, space colon...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drug abuse, exotic island, east india ...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[spy, based on novel, secret agent, sequel, mi...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[dc comics, crime fighter, terrorist, secret i...	
4	260000000	[Action, Adventure, Science Fiction]	http://movies.disney.com/john-carter	49529	[based on novel, mars, medallion, space travel...	

5 rows × 23 columns

Using ast.literal_eval to convert a string representation of a list of dictionaries to an actual list of dictionaries.

In [16]:
```
import ast
ast.literal_eval('[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id
```

Out[16]:
```
[{'id': 28, 'name': 'Action'},
 {'id': 12, 'name': 'Adventure'},
 {'id': 14, 'name': 'Fantasy'},
 {'id': 878, 'name': 'Science Fiction'}]
```

In [17]:
```
# Function 'convert3' extracts the names of the first three genres from a string repre
def convert3(text):
    L = []
    counter = 0
    for i in ast.literal_eval(text):
```

```
        if counter < 3:
            L.append(i['name'])
        counter+=1
    return L
```

In [18]:
```
movies['cast'] = movies['cast'].apply(convert)
movies.head()
#same thing for cast
```

Out[18]:

	budget	genres	homepage	id	keywords	original_langu
0	237000000	[Action, Adventure, Fantasy, Science Fiction]	http://www.avatarmovie.com/	19995	[culture clash, future, space war, space colon...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drug abuse, exotic island, east india ...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[spy, based on novel, secret agent, sequel, mi...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[dc comics, crime fighter, terrorist, secret i...	
4	260000000	[Action, Adventure, Science Fiction]	http://movies.disney.com/john-carter	49529	[based on novel, mars, medallion, space travel...	

5 rows × 23 columns

In [19]:
```
movies['cast'] = movies['cast'].apply(lambda x:x[0:3])
```

In [20]:
```
def fetch_director(text):
    L = []
    for i in ast.literal_eval(text):
        if i['job'] == 'Director':
            L.append(i['name'])
```

```
    return L

#only if job is director then append
```

In [21]:
```
movies['crew'] = movies['crew'].apply(fetch_director)
```

In [22]:
```
#movies['overview'] = movies['overview'].apply(lambda x:x.split())
movies.sample(5)
```

Out[22]:

	budget	genres	homepage	id	keywords	original_langu
3438	6500000	[Drama, Thriller]	http://www.theeastmovie.com/	87499	[secret organization, murder, environmentalism...	
3722	0	[Adventure, Action, Western]	http://www.blackthornmovie.com/	68818	[robbery, miner, treachery, sundance kid, nati...	
435	75000000	[Comedy, Fantasy, Family, Music, Animation]	http://www.munkyourself.com/	55301	[sequel, chipmunk, cruise ship, overboard]	
3404	6000000	[Action, Thriller, Crime]	http://www.theboondocksaints.com	8374	[arbitrary law, boston, twin brother, russian ...	
3114	10000000	[Drama, Thriller]	http://www.edmondthefilm.com/	18191	[new york, sex-shop, prostitute, sex, fortune ...	

5 rows × 23 columns

Whitespace Removal in List Elements

In [23]:
```
def collapse(L):
    L1 = []
    for i in L:
        L1.append(i.replace(" ",""))
```

```
    return L1
#replacing sapce with comma
```

Whitespace Removal in Movie Data Columns

In [24]:
```
movies['cast'] = movies['cast'].apply(collapse)
movies['crew'] = movies['crew'].apply(collapse)
movies['genres'] = movies['genres'].apply(collapse)
movies['keywords'] = movies['keywords'].apply(collapse)
```

In [25]:
```
movies.head()
```

Out[25]:

	budget	genres	homepage	id	keywords	origin
0	237000000	[Action, Adventure, Fantasy, ScienceFiction]	http://www.avatarmovie.com/	19995	[cultureclash, future, spacewar, spacecolony, ...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drugabuse, exoticisland, eastindiatrad...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[spy, basedonnovel, secretagent, sequel, mi6, ...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[dccomics, crimefighter, terrorist, secretiden...	
4	260000000	[Action, Adventure, ScienceFiction]	http://movies.disney.com/john-carter	49529	[basedonnovel, mars, medallion, spacetravel, p...	

5 rows × 23 columns

In [26]:
```
movies['overview'] = movies['overview'].apply(lambda x:x.split())
#coverting to a list
```

In [27]:
```
movies.head()
```

Out[27]:

	budget	genres	homepage	id	keywords	origin
0	237000000	[Action, Adventure, Fantasy, ScienceFiction]	http://www.avatarmovie.com/	19995	[cultureclash, future, spacewar, spacecolony, ...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drugabuse, exoticisland, eastindiatrad...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[spy, basedonnovel, secretagent, sequel, mi6, ...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[dccomics, crimefighter, terrorist, secretiden...	
4	260000000	[Action, Adventure, ScienceFiction]	http://movies.disney.com/john-carter	49529	[basedonnovel, mars, medallion, spacetravel, p...	

5 rows × 23 columns

In [28]:
```
movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['
#everything added to tags
```

In [29]:
```
movies.head()
```

Out[29]:

	budget	genres	homepage	id	keywords	origin
0	237000000	[Action, Adventure, Fantasy, ScienceFiction]	http://www.avatarmovie.com/	19995	[cultureclash, future, spacewar, spacecolony, ...	
1	300000000	[Adventure, Fantasy, Action]	http://disney.go.com/disneypictures/pirates/	285	[ocean, drugabuse, exoticisland, eastindiatrad...	
2	245000000	[Action, Adventure, Crime]	http://www.sonypictures.com/movies/spectre/	206647	[spy, basedonnovel, secretagent, sequel, mi6, ...	
3	250000000	[Action, Crime, Drama, Thriller]	http://www.thedarkknightrises.com/	49026	[dccomics, crimefighter, terrorist, secretiden...	
4	260000000	[Action, Adventure, ScienceFiction]	http://movies.disney.com/john-carter	49529	[basedonnovel, mars, medallion, spacetravel, p...	

5 rows × 24 columns

In [30]:

```
new = movies.drop(columns=['overview','genres','keywords','cast','crew'])
#new.head()
#dropping coums as everything is in
#dataset name --> new
```

In [31]:

```
new['tags'] = new['tags'].apply(lambda x: " ".join(x))
new.head()
#joining the lists in tags to a a string
```

Out[31]:

	budget	homepage	id	original_language	original_title	pop
0	237000000	http://www.avatarmovie.com/	19995	en	Avatar	150.
1	300000000	http://disney.go.com/disneypictures/pirates/	285	en	Pirates of the Caribbean: At World's End	139.
2	245000000	http://www.sonypictures.com/movies/spectre/	206647	en	Spectre	107.
3	250000000	http://www.thedarkknightrises.com/	49026	en	The Dark Knight Rises	112.
4	260000000	http://movies.disney.com/john-carter	49529	en	John Carter	43.

CountVectorizer In NLP

In [32]:
```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000,stop_words='english')
#stop words, keep main words
```

In [33]:
```
vector = cv.fit_transform(new['tags']).toarray()
```

In [34]:
```
vector
```

Out[34]:
```
array([[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]], dtype=int64)
```

In [35]:
```
vector.shape
```
Out[35]:
```
(1494, 5000)
```
In [36]:
```
from sklearn.metrics.pairwise import cosine_similarity
```
In [37]:
```
similarity = cosine_similarity(vector)
```
In [38]:
```
similarity
```
Out[38]:
```
array([[1.        , 0.08134892, 0.05423261, ..., 0.        , 0.05504819,
        0.02469324],
       [0.08134892, 1.        , 0.05882353, ..., 0.04428074, 0.        ,
        0.        ],
       [0.05423261, 0.05882353, 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.04428074, 0.        , ..., 1.        , 0.        ,
        0.04032389],
       [0.05504819, 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.02469324, 0.        , 0.        , ..., 0.04032389, 0.        ,
        1.        ]])
```
In [39]:
```
with open('similarity.pickle', 'wb') as efile:
    pickle.dump(similarity, efile, protocol=pickle.HIGHEST_PROTOCOL)
```
Movie Recommendation Top 5 Similar Titles

In [40]:
```
def recommend(movie):
    l=[]
    index = new[new['title'] == movie].index[0]
    distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x:
    for i in distances[1:6]:
        #print(new.iloc[i[0]].title)
        l.append(new.iloc[i[0]].title)

    return l
```
In [44]:
```
print("Enter a movie: ", end=" ")
movie=input()
recommend(movie)
```
```
Enter a movie:  Avatar
```
Out[44]:
```
['Aliens vs Predator: Requiem',
 'Battle: Los Angeles',
 "Ender's Game",
 'Apollo 18',
 'Edge of Tomorrow']
```
Sentiment Analysis

ANKUR

In [47]:
```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saif\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[47]:
```
True
```

Reading Tab-Separated Data into a DataFrame

In [48]:
```
dataset = pd.read_csv('reviews.txt',sep = '\t', names =['Reviews','Comments'])
```

In [49]:
```
dataset.head()
```

Out[49]:

	Reviews	Comments
0	1	The Da Vinci Code book is just awesome.
1	1	this was the first clive cussler i've ever rea...
2	1	i liked the Da Vinci Code a lot.
3	1	i liked the Da Vinci Code a lot.
4	1	I liked the Da Vinci Code but it ultimatly did...

Analyzing Review

In [50]:
```
a=dataset['Reviews'].value_counts()
```

In [51]:
```
stopset = set(stopwords.words('english'))
```

In [52]:
```
vectorizer = TfidfVectorizer(use_idf = True,lowercase = True, strip_accents='ascii',st
```

Transformation with TfidfTransformer

In [53]:
```
from sklearn.feature_extraction.text import TfidfTransformer
```

In [54]:
```
transformer = TfidfTransformer()
```

In [55]:
```
# Initialize TfidfVectorizer with 'english' stopwords
vectorizer = TfidfVectorizer(stop_words='english')

X = vectorizer.fit_transform(dataset.Comments)
y = dataset.Reviews
X = transformer.fit_transform(X)
```

Data Splitting

In [56]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
```

Naive Bayes Classifier Training and Modeling

In [57]:
```
from sklearn import naive_bayes
clf = naive_bayes.MultinomialNB()
clf.fit(X_train,y_train)
```

Out[57]:
```
▾ MultinomialNB
MultinomialNB()
```

In [58]:
```
accuracy_score(y_test,clf.predict(X_test))*100 #testing accuracy
```

Out[58]:
```
97.32658959537572
```

In [59]:
```
accuracy_score(y_train,clf.predict(X_train))*100 #training accuracy
```

Out[59]:
```
99.51210697506325
```

In [60]:
```
clf = naive_bayes.MultinomialNB()
clf.fit(X_train,y_train)
```

Out[60]:
```
▾ MultinomialNB
MultinomialNB()
```

In [61]:
```
accuracy_score(y_test,clf.predict(X_test))*100 #ALWAYS ACCURACY SCORE
```

Out[61]:
```
97.32658959537572
```

In [62]:
```
from sklearn.pipeline import Pipeline
```

In [63]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# defining parameter range
# param_grid = {'C': [0.1, 1, 10, 100, 1000],
#               'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
#               'kernel': ['rbf']}
param_grid = {
'alpha': (1, 0.1, 0.01, 0.001, 0.0001, 0.00001)
}
grid = GridSearchCV(naive_bayes.MultinomialNB(), param_grid, refit = True, verbose = 3

# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV 1/5] END ...........................alpha=1;, score=0.980 total time=   0.0s
[CV 2/5] END ...........................alpha=1;, score=0.979 total time=   0.0s
[CV 3/5] END ...........................alpha=1;, score=0.986 total time=   0.0s
[CV 4/5] END ...........................alpha=1;, score=0.985 total time=   0.0s
[CV 5/5] END ...........................alpha=1;, score=0.976 total time=   0.0s
[CV 1/5] END .........................alpha=0.1;, score=0.974 total time=   0.0s
[CV 2/5] END .........................alpha=0.1;, score=0.973 total time=   0.0s
[CV 3/5] END .........................alpha=0.1;, score=0.980 total time=   0.0s
[CV 4/5] END .........................alpha=0.1;, score=0.982 total time=   0.0s
[CV 5/5] END .........................alpha=0.1;, score=0.971 total time=   0.0s
[CV 1/5] END ........................alpha=0.01;, score=0.975 total time=   0.0s
[CV 2/5] END ........................alpha=0.01;, score=0.970 total time=   0.0s
[CV 3/5] END ........................alpha=0.01;, score=0.978 total time=   0.0s
[CV 4/5] END ........................alpha=0.01;, score=0.976 total time=   0.0s
[CV 5/5] END ........................alpha=0.01;, score=0.967 total time=   0.0s
[CV 1/5] END .......................alpha=0.001;, score=0.975 total time=   0.0s
[CV 2/5] END .......................alpha=0.001;, score=0.969 total time=   0.0s
[CV 3/5] END .......................alpha=0.001;, score=0.979 total time=   0.0s
[CV 4/5] END .......................alpha=0.001;, score=0.974 total time=   0.0s
[CV 5/5] END .......................alpha=0.001;, score=0.966 total time=   0.0s
[CV 1/5] END ......................alpha=0.0001;, score=0.976 total time=   0.0s
[CV 2/5] END ......................alpha=0.0001;, score=0.967 total time=   0.0s
[CV 3/5] END ......................alpha=0.0001;, score=0.979 total time=   0.0s
[CV 4/5] END ......................alpha=0.0001;, score=0.972 total time=   0.0s
[CV 5/5] END ......................alpha=0.0001;, score=0.966 total time=   0.0s
[CV 1/5] END .......................alpha=1e-05;, score=0.977 total time=   0.0s
[CV 2/5] END .......................alpha=1e-05;, score=0.967 total time=   0.0s
[CV 3/5] END .......................alpha=1e-05;, score=0.979 total time=   0.0s
[CV 4/5] END .......................alpha=1e-05;, score=0.972 total time=   0.0s
[CV 5/5] END .......................alpha=1e-05;, score=0.966 total time=   0.0s
```

Out[63]:

GridSearchCV

estimator: MultinomialNB

MultinomialNB

In [64]:

```
print(grid.best_score_)
print(grid.best_estimator_)
```

```
0.9813869000655047
MultinomialNB(alpha=1)
```

In [65]:

```
weighted_prediction = grid.predict(X_test)
```

In [66]:

```
print('Accuracy:', accuracy_score(y_test, weighted_prediction))
print('F1 score:', f1_score(y_test, weighted_prediction,average='weighted'))
print('Recall:', recall_score(y_test, weighted_prediction, average='weighted'))
print('Precision:', precision_score(y_test, weighted_prediction,average='weighted'))
```

```
Accuracy: 0.9732658959537572
F1 score: 0.973221595222043
Recall: 0.9732658959537572
Precision: 0.9733545264035355
```

In [67]:

```
print ('Clasification report:\n', classification_report(y_test, weighted_prediction))
```

```
Clasification report:
               precision    recall  f1-score   support

           0       0.98      0.96      0.97       580
           1       0.97      0.99      0.98       804

    accuracy                           0.97      1384
   macro avg       0.97      0.97      0.97      1384
weighted avg       0.97      0.97      0.97      1384
```

In [68]:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, weighted_prediction)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

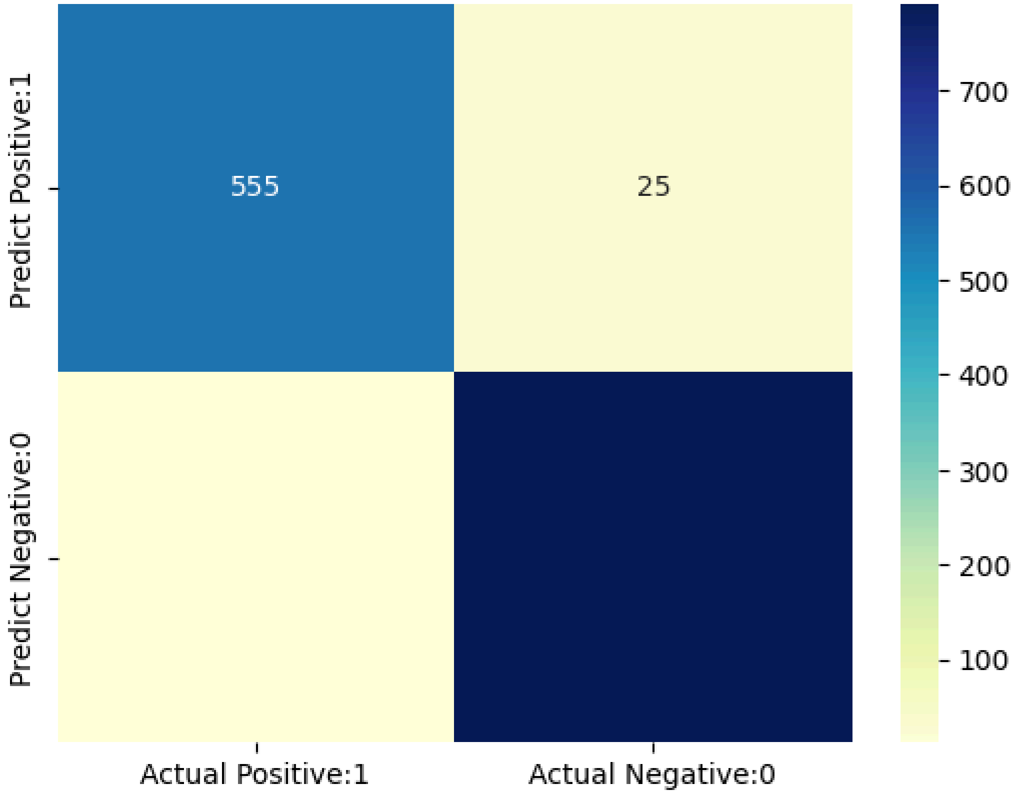
```
Confusion matrix

 [[555  25]
 [ 12 792]]

True Positives(TP) =  555

True Negatives(TN) =  792

False Positives(FP) =  25

False Negatives(FN) =  12
```

In [69]:

```
import seaborn as sns
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[69]:

```
<Axes: >
```



In [70]:
```
from sklearn.metrics import roc_curve
weighted_prediction = grid.predict(X_test)
fpr1, tpr1, thresh1 = roc_curve(y_test, weighted_prediction, pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

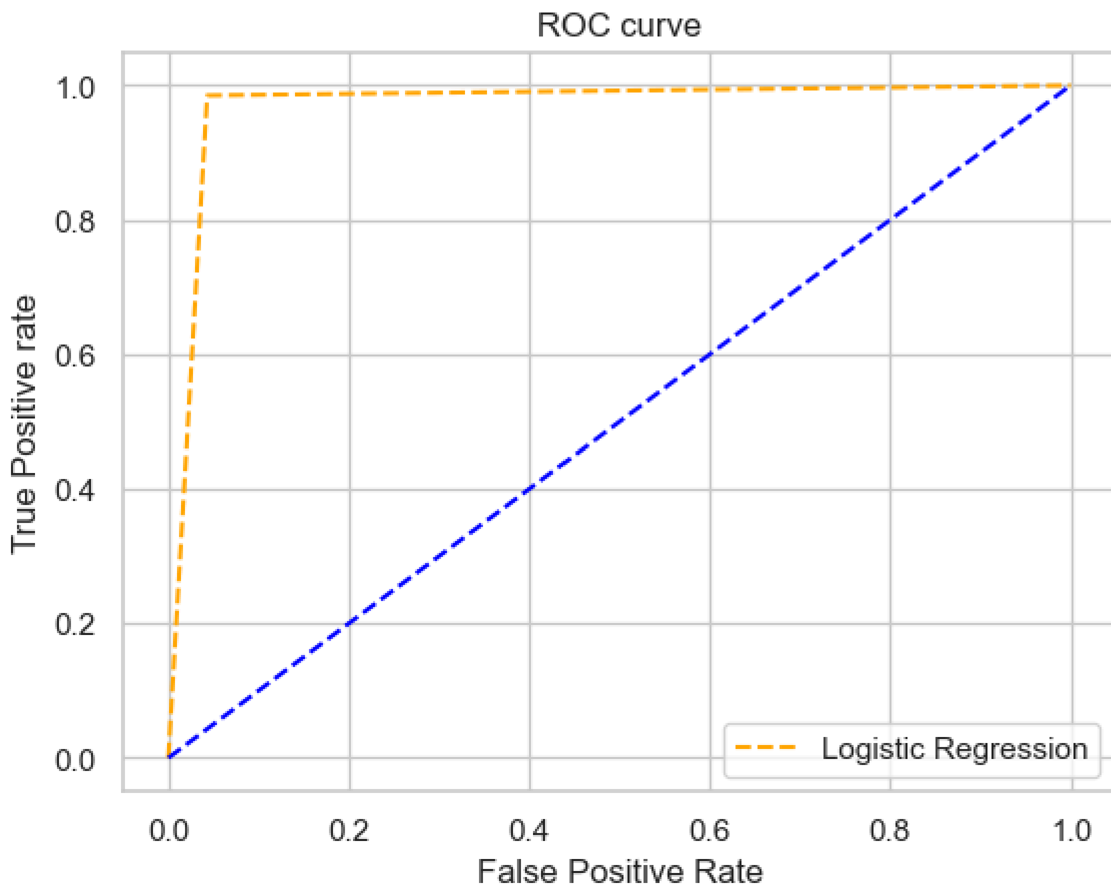
In [71]:
```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, weighted_prediction)


print(auc_score1)
```

```
0.9709855892949047
```

Visualizing Using ROC Curves:

In [72]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the seaborn style
sns.set(style='whitegrid')

# Create ROC curve plot
sns.lineplot(x=fpr1, y=tpr1, linestyle='--', color='orange', label='Logistic Regressic
sns.lineplot(x=p_fpr, y=p_tpr, linestyle='--', color='blue')

# Title
plt.title('ROC curve')
```

```
# X label
plt.xlabel('False Positive Rate')

# Y label
plt.ylabel('True Positive rate')

# Show plot
plt.show()
```



Prediction for Movie Reviews

In [73]:
```
movie_review_list=['Bad movie, wouldnt recommend']
movie_vector=vectorizer.transform(movie_review_list)
pred = grid.predict(movie_vector)
```

In [74]:
```
pred
```

Out[74]:
```
array([0], dtype=int64)
```

ROSHITA

In [75]:
```
from sklearn.svm import SVC
model=SVC(probability=True)
model.fit(X_train,y_train)
```

Out[75]:

SVC

```
SVC(probability=True)
```

In [76]:

```
model.score(X_test,y_test)
```

Out[76]:

```
0.9739884393063584
```

In [77]:

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}
grid_SVC = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)
grid_SVC.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.955 total time=   0.2s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.962 total time=   0.2s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.973 total time=   0.2s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.967 total time=   0.2s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.967 total time=   0.2s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.967 total time=   0.7s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.969 total time=   0.7s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.982 total time=   0.7s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.982 total time=   0.7s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.977 total time=   0.7s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.567 total time=   1.0s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.1s
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.975 total time=   0.1s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.977 total time=   0.1s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.983 total time=   0.1s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.981 total time=   0.1s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.978 total time=   0.1s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.972 total time=   0.1s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.976 total time=   0.1s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.982 total time=   0.1s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.980 total time=   0.1s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.976 total time=   0.1s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.968 total time=   0.6s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.973 total time=   0.7s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.988 total time=   0.6s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.982 total time=   0.6s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.980 total time=   0.7s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.982 total time=   0.1s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.979 total time=   0.1s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.988 total time=   0.1s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.983 total time=   0.1s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.979 total time=   0.1s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.988 total time=   0.1s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.990 total time=   0.1s
```

```
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.987 total time=   0.1s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.973 total time=   0.1s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.976 total time=   0.1s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.984 total time=   0.1s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.984 total time=   0.1s
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.980 total time=   0.1s
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.968 total time=   0.6s
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.973 total time=   0.6s
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.988 total time=   0.7s
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.982 total time=   0.6s
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.980 total time=   0.6s
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.567 total time=   1.0s
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.982 total time=   0.1s
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.979 total time=   0.1s
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.988 total time=   0.1s
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.994 total time=   0.1s
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.990 total time=   0.1s
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.990 total time=   0.1s
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.984 total time=   0.1s
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.982 total time=   0.1s
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.987 total time=   0.1s
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.990 total time=   0.1s
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.986 total time=   0.1s
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.973 total time=   0.1s
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.976 total time=   0.1s
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.985 total time=   0.1s
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.984 total time=   0.1s
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.980 total time=   0.1s
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.968 total time=   0.6s
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.973 total time=   0.6s
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.988 total time=   0.6s
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.982 total time=   0.6s
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.980 total time=   0.6s
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.982 total time=   0.1s
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.979 total time=   0.1s
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.988 total time=   0.1s
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.986 total time=   0.1s
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.996 total time=   0.1s
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.990 total time=   0.1s
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.990 total time=   0.1s
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.986 total time=   0.0s
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.986 total time=   0.1s
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.994 total time=   0.1s
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.990 total time=   0.1s
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.992 total time=   0.1s
[CV 1/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.984 total time=   0.1s
[CV 2/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.982 total time=   0.1s
[CV 3/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.987 total time=   0.1s
[CV 4/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.990 total time=   0.1s
```

```
[CV 5/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.986 total time=   0.1s
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.973 total time=   0.1s
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.976 total time=   0.1s
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.985 total time=   0.1s
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.984 total time=   0.1s
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.980 total time=   0.1s
```

Out[77]:

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [78]:

```
weighted_prediction_SVC = grid_SVC.predict(X_test)
```

Performance

In [79]:

```
print('Accuracy:', accuracy_score(y_test, weighted_prediction_SVC))
print('F1 score:', f1_score(y_test, weighted_prediction_SVC,average='weighted'))
print('Recall:', recall_score(y_test, weighted_prediction_SVC, average='weighted'))
print('Precision:', precision_score(y_test, weighted_prediction_SVC,average='weighted'
```

```
Accuracy: 0.986271676300578
F1 score: 0.9862597549434926
Recall: 0.986271676300578
Precision: 0.9862997895392668
```

In [80]:

```
print ('Clasification report:\n', classification_report(y_test, weighted_prediction_SV
```

```
Clasification report:
               precision    recall  f1-score   support

           0       0.99      0.98      0.98       580
           1       0.98      0.99      0.99       804

    accuracy                           0.99      1384
   macro avg       0.99      0.99      0.99      1384
weighted avg       0.99      0.99      0.99      1384
```

In [81]:

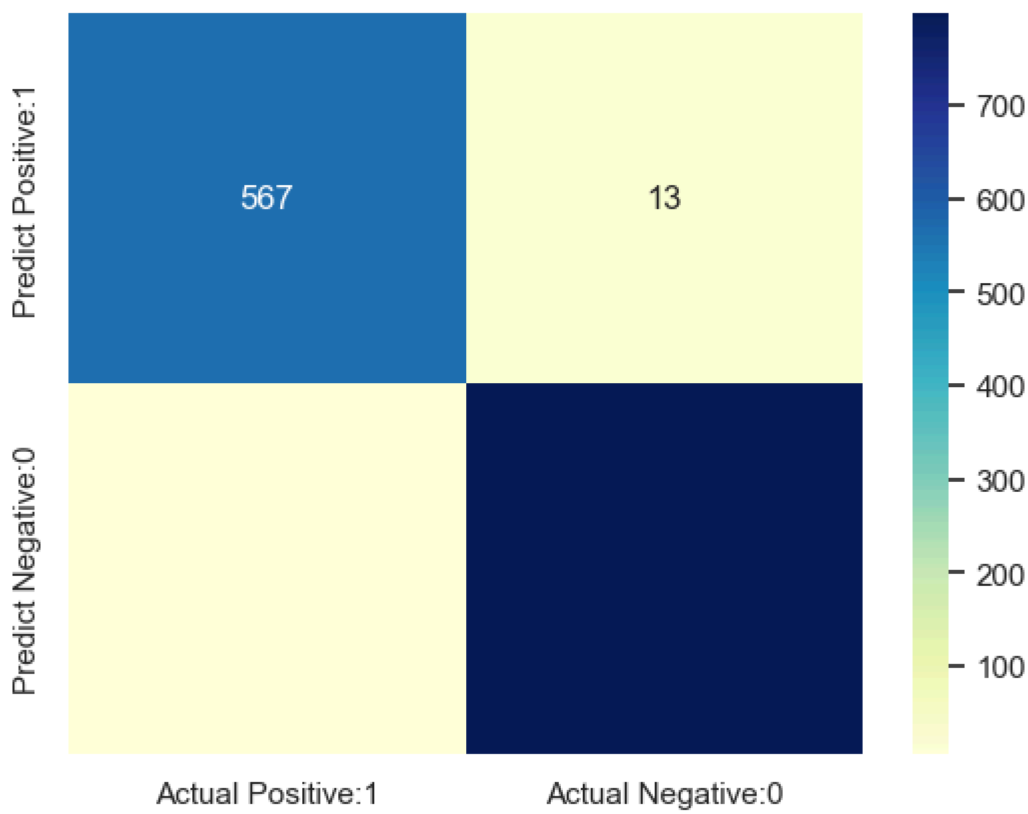
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, weighted_prediction_SVC)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[567  13]
 [  6 798]]

True Positives(TP) =  567

True Negatives(TN) =  798

False Positives(FP) =  13

False Negatives(FN) =  6
```

Visualizing Confusion

In [82]:
```python
import seaborn as sns
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[82]: <Axes: >



ROC Curve for Binary Classification And Visualizing Model Performance

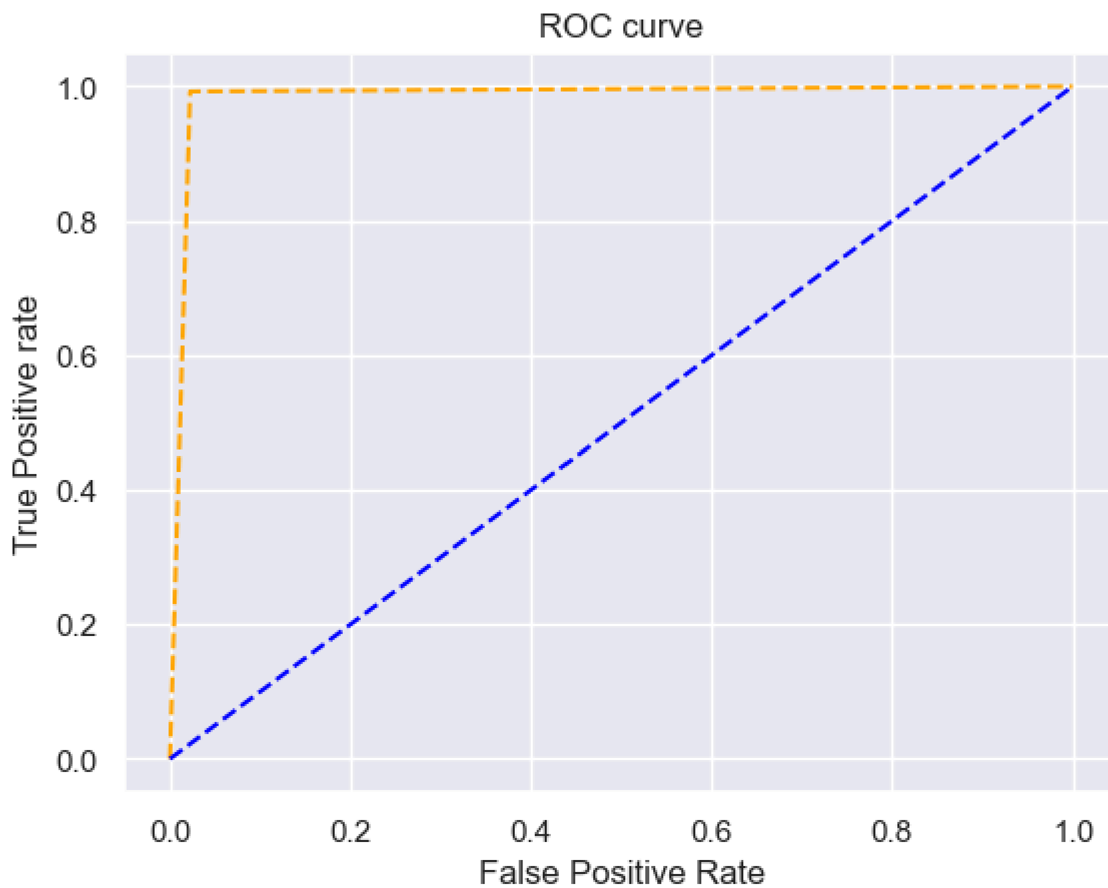
In [83]:
```python
from sklearn.metrics import roc_curve
```

```python
weighted_prediction2 = model.predict(X_test)
fprSVC, tprSVC, thresh1 = roc_curve(y_test, weighted_prediction_SVC, pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

In [84]:
```python
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test, weighted_prediction_SVC)


print(auc_score1)
```

```
0.9850617601646937
```

In [85]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the seaborn style
sns.set_style('darkgrid')

# plot roc curves
plt.plot(fprSVC, tprSVC, linestyle='--', color='orange')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# Title
plt.title('ROC curve')

# X label
plt.xlabel('False Positive Rate')

# Y label
plt.ylabel('True Positive rate')

# Show plot
plt.show()
```



Prediction

In [86]:
```
movie_review_list=['Bad movie, wouldnt recommend']
movie_vector=vectorizer.transform(movie_review_list)
pred = model.predict(movie_vector)
```

In [87]:
```
pred
```

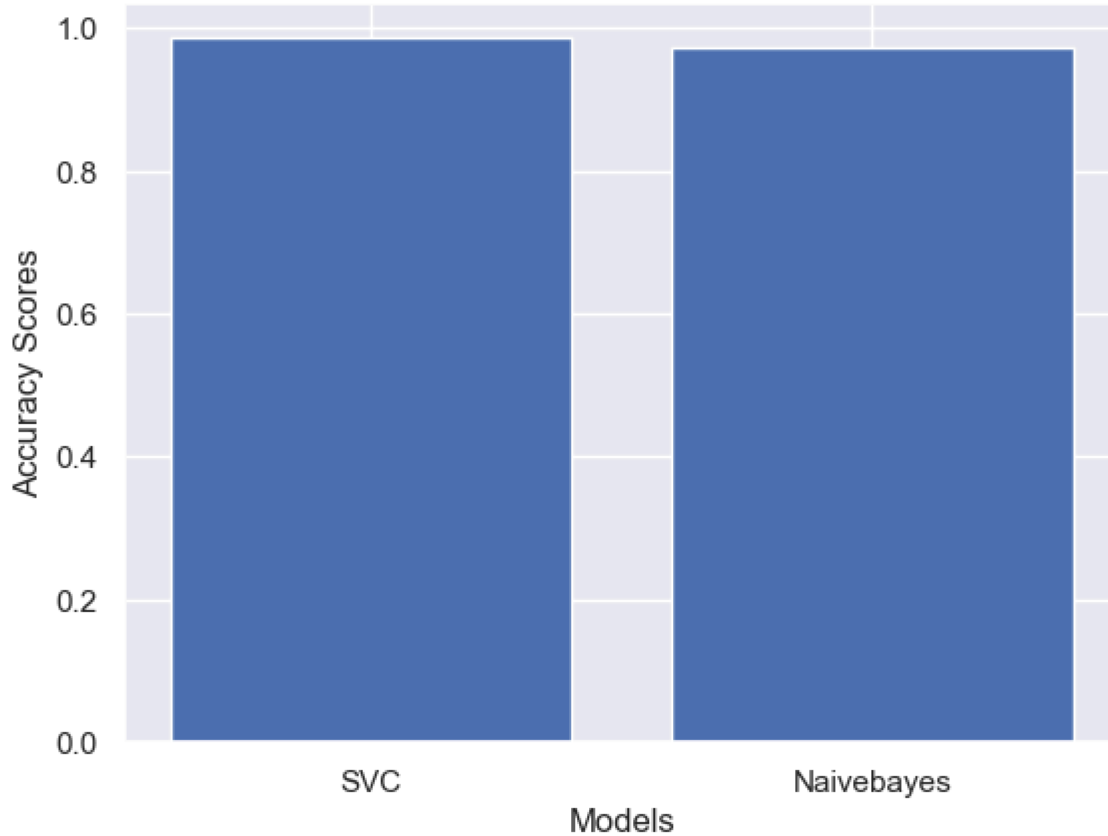
Out[87]:
```
array([0], dtype=int64)
```

AAYUSHI

Comparing Model Accuracy

In [88]:
```
x=np.array(["SVC","Naivebayes"])
y=np.array([accuracy_score(y_test, weighted_prediction_SVC),accuracy_score(y_test, wei
plt.xlabel("Models")
plt.ylabel("Accuracy Scores")
plt.bar(x,y)
```

Out[88]:
```
<BarContainer object of 2 artists>
```



In [89]:
```
MLA = [
    grid,
    grid_SVC,
    ]
```

In [90]:
```
MLA_columns = []
MLA_compare = pd.DataFrame(columns = MLA_columns)

row_index = 0
for alg in MLA:

    predicted = alg.predict(X_test)
    fp, tp, th = roc_curve(y_test, predicted)
    MLA_name = alg
    MLA_compare.loc[row_index, 'Algorithm'] = MLA_name
    MLA_compare.loc[row_index, 'Accuracy'] = round(alg.score(X_test, y_test), 4)
    MLA_compare.loc[row_index, 'Precision'] = precision_score(y_test, predicted)
    MLA_compare.loc[row_index, 'Recall'] = recall_score(y_test, predicted)
    MLA_compare.loc[row_index, 'AUC'] = auc(fp, tp)

    row_index+=1


MLA_compare
```

Out[90]:

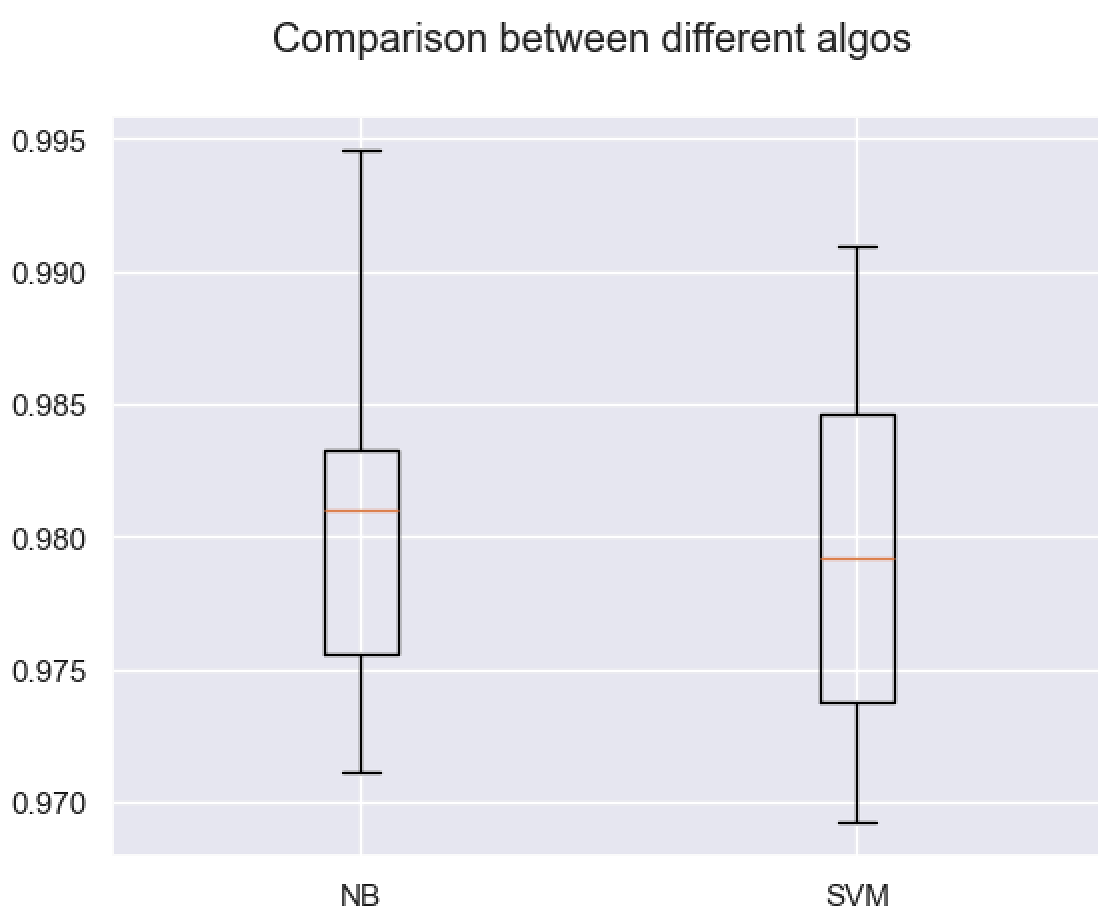
	Algorithm	Accuracy	Precision	Recall	AUC
0	GridSearchCV(estimator=MultinomialNB(),\n ...	0.9733	0.96940	0.985075	0.970986
1	GridSearchCV(estimator=SVC(),\n pa...	0.9863	0.98397	0.992537	0.985062

In [91]:
```
models = []

models.append(('NB', naive_bayes.MultinomialNB()))
models.append(('SVM', SVC()))
```

Algorithm Performance Comparison

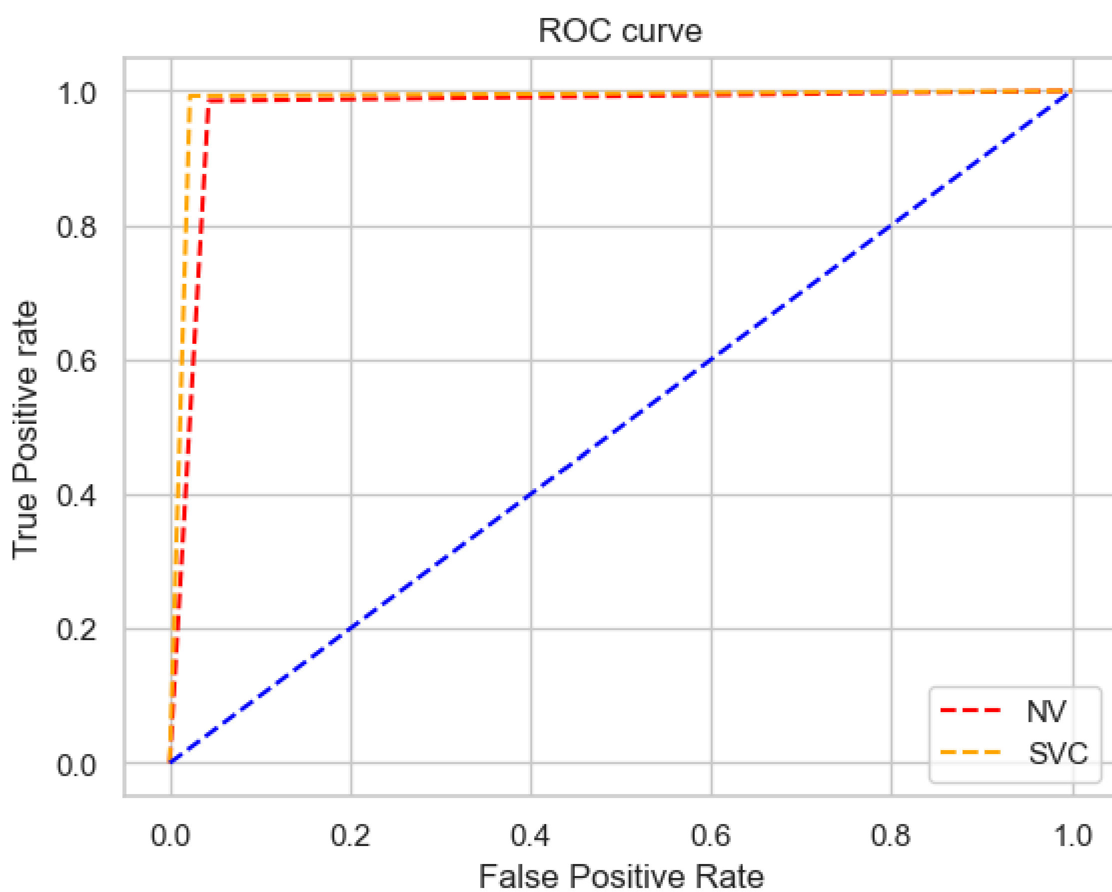
In [92]:
```
from sklearn import svm,model_selection
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10)
        cv_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfolc
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Comparison between different algos')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

```
NB: 0.980847 (0.006865)
SVM: 0.979401 (0.006816)
```



In [93]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the seaborn style
sns.set_style('whitegrid')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--', color='red')
plt.plot(fprSVC, tprSVC, linestyle='--', color='orange')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')

# Title
plt.title('ROC curve')

# X label
plt.xlabel('False Positive Rate')

# Y label
plt.ylabel('True Positive rate')

# Add legend
plt.legend(['NV', 'SVC'])

# Show plot
plt.show()
```



ATHARVA

recommend movie by user input

In [ ]:
```
print("Enter a movie:", end=" ")
movie=input()
```

```
Enter a movie:
```

In [ ]:
```
recommend(movie)
```

In [ ]:
```
import requests
```

In [ ]:
```
pip install IMDbPY
```

Fetching IMDb

In [ ]:
```
import imdb
ia = imdb.IMDb()
search = ia.search_movie(movie)
id = search[0].movieID
```

In [ ]:
```
from bs4 import BeautifulSoup
```

In [ ]:
```
page = requests.get('https://www.imdb.com/title/tt{}/reviews?ref_=tt_urv'.format(id))
soup = BeautifulSoup(page.content, 'html.parser')
```

In [ ]:
```
print('https://www.imdb.com/title/tt{}/reviews?ref_=tt_urv'.format(id))
```

In [ ]:
```
reviews=[]
```

In [ ]:
```
movie_data=soup.find_all('div',attrs= {'class': 'lister-item-content'})
```

In [ ]:
```
for store in movie_data:
    review = store.find('a', class_ = 'title').text.replace('\n', '')
    reviews.append(review)
```

In [ ]:
```
reviews
```

Sentiment Analysis Predictions for Movie Reviews

In [ ]:
```
for i in reviews:
  movie_vector=vectorizer.transform([i])
  pred = grid_SVC.predict(movie_vector)
  print(i, pred)

#0- negative
#1- positive
```

In [ ]:
```
#Website uses SVC
```

Conclusion

This paper presents a comprehensive exploration into two major components: Movie Recommendation System and Sentiment Analysis. The Movie Recommendation System utilizes the Cosine Similarity algorithm to provide accurate movie suggestions based on various factors such as genre, overview, cast, and ratings. The algorithm demonstrates consistent and reliable results through multiple tests, affirming its effectiveness.

In the realm of sentiment analysis, two algorithms, Naïve Bayes (NB) and Support Vector Machine (SVM) Classifier, are employed to classify reviews as positive or negative. The goal is to identify the most suitable algorithm for diverse reviews. Experimental results reveal that SVM outperforms NB, albeit by a small margin, emphasizing its superiority in sentiment analysis.

The study identifies potential areas for future improvement, including enhancing the accuracy of sentiment analysis for classifying sarcastic or ironic reviews, extending sentiment analysis to languages beyond English, and refining movie recommendations based on users' preferences.

Despite the system's high accuracy, certain limitations exist. The system may falter when the user inputs a movie not present in the dataset or enters the name differently. Additionally, the linguistic barrier in sentiment analysis is acknowledged, as only English reviews are currently analyzed, and sarcasm or irony can lead to misclassification.