Bahria University,

Karachi Campus



COURSE: DATA MINING

TERM: FALL 2021, CLASS: BSE- 6(B)

PROJECT REPORT

***PROJECT TITLE:***

***“MOVIE SUCCESS PREDICTION AND***

***RECOMMENDATION SYSTEM”***

Submitted By:

|  |  |  |
| --- | --- | --- |
| ***Sr. No.*** | ***Student Name*** | ***Registration Code*** |
| ***01*** | ***Aqsa Shamsher*** | ***02-131192-033*** |
| ***02*** | ***Iqra Afzaal*** | ***02-131192-048*** |
| ***03*** | ***Kinza Noor*** | ***02-131192-025*** |

Dr. Amina Jameel/ Engr. Ramsha Mashood

**Signed Remarks: Score: .**

**Table of Contents**

[1. INTRODUCTION AND PROBLEM 3](#_Toc108081001)

[2. TECHNOLOGY 3](#_Toc108081002)

[3. FUNCTIONALITIES 3](#_Toc108081003)

[4. MODULE DISTRIBUTION 4](#_Toc108081004)

[5. CODE 4](#_Toc108081005)

[6. INTERFACES 9](#_Toc108081006)

[7. CONCLUSION 10](#_Toc108081007)

# INTRODUCTION AND PROBLEM

Our system generates movie recommendation for its users, it lists out 05 movies similar to the name of the movie he/she has entered (recent watched movie name). Upon providing some details, the system predicts if the movie would be a hit or a flop. The system will analyze the comments on the movies made by users and will provide the count of positive and negative comments.

Choosing a movie to see is half the fun of viewing one. But how do you get started? The hardest part of a movie night can be choosing what to watch. How do you determine if a movie is worth seeing before investing money and time to buy a ticket and driving hours to see it? Would it succeed or fail?

# TECHNOLOGY

**Jupyter Notebook**

The Jupyter Notebook is an open-source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

# FUNCTIONALITIES

***The functionalities of project are following:***

* Recommend movies based upon search.
* Success prediction of movies based upon Directors, movie’s budget, actors and genre of upcoming release.
* Comment Analysis to analyze the negative and positive comments made by the users.

# MODULE DISTRIBUTION

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***S. No.*** | ***Modules*** | ***Student Name*** | | | |
| ***Kinza Noor*** | ***Aqsa Shamsher*** | ***Iqra Afzal*** |
| ***1*** | ***Success Prediction using NAÏVE BAYES*** | **✓** |  |  |
| ***2*** | ***Recommendation System using KNN*** |  | **✓** | **✓** |
| ***3*** | ***Comment Analysis using Sentiment Analysis*** |  | **✓** |  |

# CODE

from sklearn import metrics

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import ast

import nltk.classify.util

from nltk.classify import NaiveBayesClassifier

from nltk.corpus import movie\_reviews

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import pandas as pd

import nltk

nltk.download('movie\_reviews')

# This is how the Naive Bayes classifier expects the input

def create\_word\_features(words):

useful\_words = [word for word in words if word not in stopwords.words("english")]

my\_dict = dict([(word, True) for word in useful\_words])

return my\_dict

print("outside create word feature")

neg\_reviews = []

for fileid in movie\_reviews.fileids('neg'):

words = movie\_reviews.words(fileid)

neg\_reviews.append((create\_word\_features(words), "negative"))

print("neg\_reviews")

pos\_reviews = []

for fileid in movie\_reviews.fileids('pos'):

words = movie\_reviews.words(fileid)

pos\_reviews.append((create\_word\_features(words), "positive"))

print("filled review ")

train\_set = neg\_reviews[:750] + pos\_reviews[:750]

print("train set")

test\_set = neg\_reviews[750:] + pos\_reviews[750:]

print("test-set")

classifier = NaiveBayesClassifier.train(train\_set)

accuracy = nltk.classify.util.accuracy(classifier, test\_set)

df = pd.read\_csv("./IMDB Dataset.csv", sep=",")

reviews\_list = df['review']

i = 0

j =0

for review in reviews\_list:

print("inside loop")

words = word\_tokenize(review)

words = create\_word\_features(words)

print("inside above if")

res =classifier.classify(words)

if(res == 'positive'):

i=i+1

print("inside positive")

else:

j=j+1

def review\_res():

if( i > j):

root = Tk()

# specify size of window.

root.geometry("800x100")

label=Label(root,text="positive review = "+str(i))

label1=Label(root,text="as given movie has more positive review it can generate more profit reveniew in upcoming week")

label.pack()

label1.pack()

tk.mainloop()

elif(i<j):

root = Tk()

# specify size of window.

root.geometry("800x100")

label=Label(root,text="negative review = "+str(j))

label1=Label(root,text="as given movie has more negative review it's won't generate profit reveniew in upcoming week as expected")

label.pack()

label1.pack()

tk.mainloop()

dataset = pd.read\_csv("datasetfinal.csv")

# Split dataset into X (independent variable ) and y (dependent variable)

X = dataset.iloc[:, :-1].values #loc/iloc

y = dataset.iloc[:, 11].values

# Encoding categorical data

# Encoding the Independent Variable

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

number = LabelEncoder()

nameencoder=LabelEncoder()

actor1encoder=LabelEncoder()

actor2encoder=LabelEncoder()

actor3encoder=LabelEncoder()

genresencoder=LabelEncoder()

imdbscoreencoder=LabelEncoder()

budgetencoder=LabelEncoder()

grossencoder=LabelEncoder()

profitencoder =LabelEncoder()

#Encoding each categorical features.

dataset['director\_name'] = nameencoder.fit\_transform(dataset['director\_name'])

dataset['actor\_1\_name'] = actor1encoder.fit\_transform(dataset['actor\_1\_name'])

dataset['actor\_2\_name'] = actor2encoder.fit\_transform(dataset['actor\_2\_name'].astype(str))

dataset['actor\_3\_name'] = actor3encoder.fit\_transform(dataset['actor\_3\_name'].astype(str))

dataset['genres'] = genresencoder.fit\_transform(dataset['genres'])

# dataset['imdb\_score'] = imdbscoreencoder.fit\_transform(dataset['imdb\_score'])

# dataset['budget'] = budgetencoder.fit\_transform(dataset['budget'])

# dataset['gross'] = grossencoder.fit\_transform(dataset['gross'])

# dataset['profit\_percent'] = profitencoder.fit\_transform(dataset['profit\_percent'])

# Deal with all features. You skipped actor\_2\_name, actor\_3\_name. Take care of missing datas in actor\_2 and actor\_3.

features = ["director\_name", "actor\_1\_name", "genres","imdb\_score","budget","gross","profit\_percent"]

# Encoding the Dependent Variable

labelencoder\_y = LabelEncoder()

y = labelencoder\_y.fit\_transform(y)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset[features], y, test\_size = 0.2, random\_state = 0)

# test\_size = 0.2 means 20% data is in test set

# random\_state = 0 means it will generate same test set and train set

from sklearn.preprocessing import StandardScaler

# Scalind the data

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

#Create a Gaussian Classifier

model = GaussianNB()

# Train the model using the training sets

model.fit(X\_train, y\_train)

# predict values using the training data

nb\_predict\_train = model.predict(X\_test)

# import the performance metrics library

movies = pd.read\_csv('C:/Users/adil/Downloads/dm project part 1movie recomendation/tmdb\_5000\_movies.csv')

credits = pd.read\_csv('C:/Users/adil/Downloads/dm project part 1movie recomendation/tmdb\_5000\_credits.csv')

movies = movies.merge(credits,on='title')

movies = movies[['movie\_id','title','overview','genres','keywords','cast','crew']]

def convert(text):

L = []

for i in ast.literal\_eval(text):

L.append(i['name'])

return L

def convert3(text):

L = []

counter = 0

for i in ast.literal\_eval(text):

if counter < 3:

L.append(i['name'])

counter+=1

return L

def fetch\_director(text):

L = []

for i in ast.literal\_eval(text):

if i['job'] == 'Director':

L.append(i['name'])

return L

def collapse(L):

L1 = []

for i in L:

L1.append(i.replace(" ",""))

return L1

movies.dropna(inplace=True)

movies['genres'] = movies['genres'].apply(convert)

movies['keywords'] = movies['keywords'].apply(convert)

movies['cast'] = movies['cast'].apply(convert)

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

movies['crew'] = movies['crew'].apply(fetch\_director)

movies['cast'] = movies['cast'].apply(collapse)

movies['crew'] = movies['crew'].apply(collapse)

movies['genres'] = movies['genres'].apply(collapse)

movies['keywords'] = movies['keywords'].apply(collapse)

movies['overview'] = movies['overview'].apply(lambda x:x.split())

movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']

new = movies.drop(columns=['overview','genres','keywords','cast','crew'])

new['tags'] = new['tags'].apply(lambda x: " ".join(x))

cv = CountVectorizer(max\_features=5000,stop\_words='english')

vector = cv.fit\_transform(new['tags']).toarray()

similarity = cosine\_similarity(vector)

List = []

def recommend():

index = new[new['title'] == entry10.get()].index[0]

distances = sorted(list(enumerate(similarity[index])),reverse=True,key = lambda x: x[1])

for i in distances[1:6]:

print(new.iloc[i[0]].title)

List.append(new.iloc[i[0]].title)

root = Tk()

t = Text(root)

for x in List:

t.insert(END, x + '\n')

t.pack()

root.mainloop()

def movie\_Predict():

predict=[entry1.get(),entry2.get(),variable.get(),entry3.get(),entry4.get(),entry6.get(),entry7.get()]

predict[0]=nameencoder.transform([predict[0]])

predict[1]=actor1encoder.transform([predict[1]])

predict[2]= genresencoder.transform([predict[2]])

# Scale or normalize the datas.

predict = scaler.transform([predict])

prediction = model.predict(predict)

if prediction == 1:

print("HIT")

class Window(Frame):

def \_\_init\_\_(self, master=None):

Frame.\_\_init\_\_(self, master)

self.master = master

self.pack(fill=BOTH, expand=1)

actor = Label(self, text="Actor name : "+entry2.get())

director = Label(self, text="Director name : "+entry1.get())

Accuracy = Label(self, text = metrics.accuracy\_score(y\_test, nb\_predict\_train))

text = Label(self, text="MOVIE WILL BE \*\*\*\*\*Hit\*\*\*\*\*")

text.place(x=70,y=90)

actor.place(x=50,y=60)

director.place(x=30,y=40)

Accuracy.place(x=90,y=100)

root = Tk()

app = Window(root)

root.wm\_title("PREDICTION RESULT")

root.geometry("500x500")

root.mainloop()

else:

print("FLOP")

class Window(Frame):

def \_\_init\_\_(self, master=None):

Frame.\_\_init\_\_(self, master)

self.master = master

self.pack(fill=BOTH, expand=1)

actor = Label(self, text="Actor name : "+entry2.get())

director = Label(self, text="Director name : " + entry1.get())

Accuracy = Label(self, text = metrics.accuracy\_score(y\_test, nb\_predict\_train))

text = Label(self, text="MOVIE WILL BE \*\*\*\*Flop\*\*\*\*")

text.place(x=70,y=90)

actor.place(x=70,y=60)

director.place(x=70,y=40)

Accuracy.place(x=100,y=120)

root = Tk()

app = Window(root)

root.wm\_title("PREDICTION RESULT")

root.geometry("500x500")

root.mainloop()

# Accuracy

print(" ACCURACY: {0:.4f}".format(metrics.accuracy\_score(y\_test, nb\_predict\_train)))

print()

#Import the required Libraries

import tkinter

from tkinter import \*

from tkinter import ttk

top = tkinter.Tk()

top.title('movie success prediction')

top.geometry("500x700")

top.configure(bg="black")

var = StringVar()

label = Label( top, textvariable=var, relief=RAISED )

var.set(" let's predict ")

label.pack()

var = StringVar()

label = Label( top, textvariable=var, relief=RAISED )

var.set("ANALYSING SUCCESS RATE OF MOVIES Based on director and actor")

label.pack()

nb = ttk.Notebook(top)

page1 = Frame(nb)

page2 = ttk.Frame(nb)

page3 = Frame(nb)

#frame = Frame(top, width=200, height=200,bg="blue")

#frame.pack()

lbl10 = Label(page1,text="movie name \*")

lbl10.pack()

entry10 = Entry(page1, bd =5)

entry10.pack()

B = tkinter.Button(page1, text ="PREDICT", command = recommend)

B.pack()

# PAGE TWO STARTS.

#entry3 = Entry(page2, bd =5)

#entry3.pack()

lbl1 = Label(page2,text="Director Name\*")

lbl1.pack()

entry1 = Entry(page2, bd =5)

entry1.pack()

lbl2 = Label(page2,text="Actor Name\*")

lbl2.pack()

entry2 = Entry(page2, bd =5)

entry2.pack()

lbl5 = Label(page2,text="Genre\*")

lbl5.pack()

OPTIONS = [

'Action','Adventure','Animation','Comedy','Drama','Family', 'Fantasy', 'Horror','Thriller', 'Romance','Sci-Fi'

]

variable = StringVar(page2)

variable.set(OPTIONS[0]) # default value

w = OptionMenu(page2, variable, \*OPTIONS)

w.pack()

lbl3 = Label(page2,text="IMDB Rating")

lbl3.pack()

entry3 = Entry(page2, bd =5)

entry3.pack()

lbl4 = Label(page2,text="Budget")

lbl4.pack()

entry4 = Entry(page2, bd =5)

entry4.pack()

lbl6 = Label(page2,text="Gross")

lbl6.pack()

entry6 = Entry(page2, bd =5)

entry6.pack()

lbl7 = Label(page2,text="profit")

lbl7.pack()

entry7 = Entry(page2, bd =5)

entry7.pack()

B = tkinter.Button(page2, text ="PREDICT", command = movie\_Predict)

B.pack()

#page three

lbl = Label(page1,text="movie name \*")

lbl.pack()

B = tkinter.Button(page3, text ="PREDICT", command = review\_res)

B.pack()

nb.add(page2, text='Movie\_success\_prediction')

nb.pack(expand=1, fill="both")

nb.add(page1, text='MOVIE RECOMENDATION')

nb.pack(expand=1, fill="both")

nb.add(page3, text='review analysis')

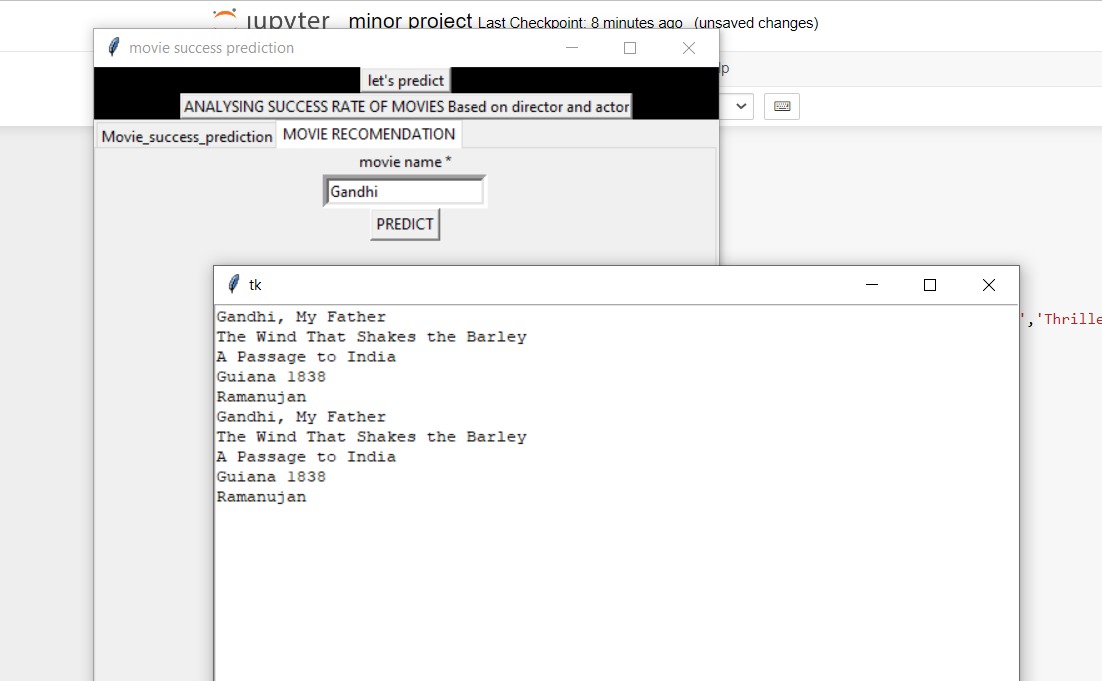
nb.pack(expand=1, fill="both")

top.mainloop()

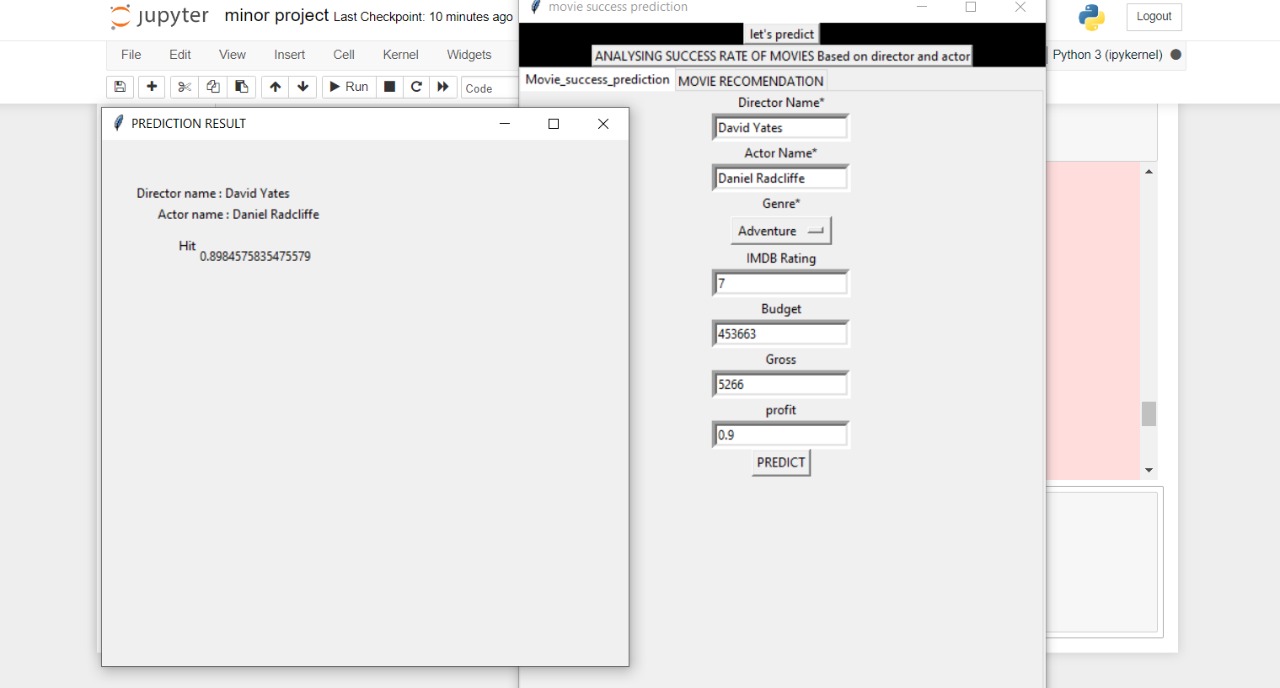
# 

# INTERFACES

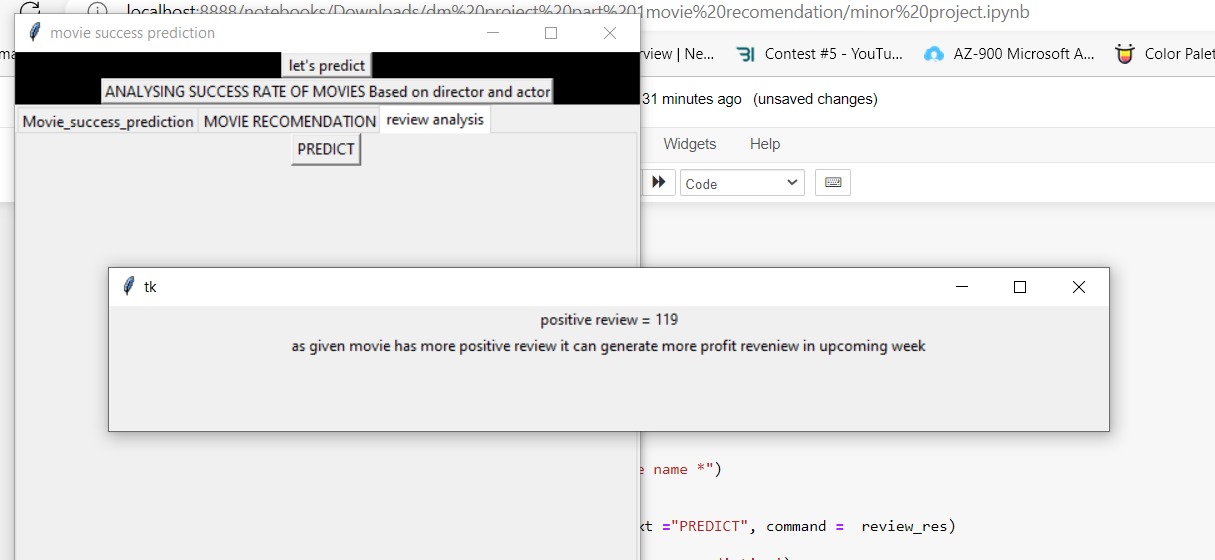
**MOVIEW RECOMEDATION**

****

**MOVIE SUCCESS PREDICTION**

****

**REVIEW ANALYSIS**

****

# CONCLUSION

# When a user enters a movie name into our system, it creates a list of 05 movies that are comparable to that movie (recent watched movie name). The computer system can estimate whether a movie will be a hit or a flop after receiving some information. The system will evaluate user-made movie reviews and produce a count of both favorable and negative reviews. The addition of features like "Mark Movies as Watched or To-Watch" and "Collection of Better Dataset with Current Movies and TV Shows" will further improve this project.