Book Recommendation System

Develop a machine learning model leveraging natural language processing to recommend books titles based on description, utilizing supervised learning on a dataset of book descriptions and titles for accurate predictions

Importing the necessary libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sn
```

Importing the gensim library for various natural language processing tasks

```
import gensim.downloader as api
wv = api.load('word2vec-google-news-300')
```

importing spacy library

```
import spacy
nlp = spacy.load("en_core_web_lg")
```

Reading the dataset

```
dataframe = pd.read_excel('priyapooji.xlsx')

dataframe.head(10)     #first 10 rows of the dataset
```

```
                                     title  \
0  Shadow Strike: A Special Forces Mission
1      Rogue Agent: The Pursuit of Justice
2           Code Red: Crisis in the Jungle
3    Dark Horizon: The Battle for Survival
4           Final Hour: Countdown to Chaos
5    Deadly Pursuit: Hunted Across Borders
6           Nightfall: Shadows of Betrayal
```

```
7             Storm Front: Clash of Titans
8            Black Ops: Behind Enemy Lines
9                   Inferno: Fire and Fury

                                         description   genre
0   An elite team embarks on a high-stakes covert...  action
1   A renegade CIA operative races against time t...  action
2   A group of mercenaries must navigate treacher...  action
3   Survivors of a plane crash must fight against...  action
4   A bomb expert races against the clock to disa...  action
5   A fugitive must evade capture by both law enf...  action
6   A retired assassin is forced back into action...  action
7   Two rival factions collide in a battle for su...  action
8   A covert operative infiltrates enemy territor...  action
9   Firefighters battle against impossible odds t...  action

dataframe['genre'].value_counts()      #printing the value counts

genre
action            100
adventure         100
gfiction          100
ghost             100
monster           100
sfiction          100
zombie            100
dark fantasy      100
fairy tales       100
heroic fantasy    100
fables            100
legends           100
romance           100
autobiography      98
Name: count, dtype: int64
```

Labeling the genre types

```
label = {'action' : 1, 'adventure' : 2, 'gfiction' : 3, 'ghost' : 4,
'monster' : 5, 'sfiction':6, 'zombie':7, 'dark fantasy':8, 'fairy
tales':9, 'heroic fantasy':10, 'fables':11, 'legends':12,
'romance':13,'autobiography' : 14 }

dataframe['output'] = dataframe['genre'].map(label)    #adding label
to the dataset

dataframe.head()     #printing the data

                                    title  \
0  Shadow Strike: A Special Forces Mission
1      Rogue Agent: The Pursuit of Justice
```

```
2           Code Red: Crisis in the Jungle
3    Dark Horizon: The Battle for Survival
4           Final Hour: Countdown to Chaos

                                         description   genre  output
0   An elite team embarks on a high-stakes covert...  action       1
1   A renegade CIA operative races against time t...  action       1
2   A group of mercenaries must navigate treacher...  action       1
3   Survivors of a plane crash must fight against...  action       1
4   A bomb expert races against the clock to disa...  action       1

ml_final = dataframe[~dataframe['genre'].isin(['dark fantasy',
'monster','gfiction','fables', 'zombie'])]
```

Preprocessing and Vectorizing

The funtion preprocess_and_vectorize takes the input as description and convert the description into tokens, eliminates the stopwords and punctuations using is_stop and is_punt, lemmatizing tokens using .lemma_(spacy functions) and Convert the description into 300 dimensional vector using get_mean_ vector.

```
def preprocess_and_vectorize(text):
    if not isinstance(text, str):
        text = str(text)

    doc = nlp(text)
    filtered_tokens = []
    for token in doc:
        if token.is_stop or token.is_punct:
            continue
        filtered_tokens.append(token.lemma_)

    return wv.get_mean_vector(filtered_tokens)

ml_final['vector'] = ml_final['description'].apply(lambda text:
preprocess_and_vectorize(text))     #calling the function

C:\Users\mpooj\AppData\Local\Temp\ipykernel_27196\2700383393.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  ml_final['vector'] = ml_final['description'].apply(lambda text:
preprocess_and_vectorize(text))     #calling the function
```

```
ml_final

                                                title  \
0     Shadow Strike: A Special Forces Mission
1         Rogue Agent: The Pursuit of Justice
2              Code Red: Crisis in the Jungle
3       Dark Horizon: The Battle for Survival
4              Final Hour: Countdown to Chaos
...                                       ...
1393                          Eleanor & Park
1394                    Call Me by Your Name
1395            A Court of Thorns and Roses
1396                       The Rosie Project
1397                        The Wedding Date

                                             description    genre
output  \
0      An elite team embarks on a high-stakes covert...   action
1
1      A renegade CIA operative races against time t...   action
1
2      A group of mercenaries must navigate treacher...   action
1
3      Survivors of a plane crash must fight against...   action
1
4      A bomb expert races against the clock to disa...   action
1
...                                                 ...      ...      .
..
1393   Rainbow Rowell's YA romance tells the story o...  romance
13
1394   André Aciman's novel of a passionate summer r...  romance
13
1395   Sarah J. Maas's fantasy romance follows Feyre...  romance
13
1396   Graeme Simsion's quirky romance follows Don T...  romance
13
1397   Jasmine Guillory's contemporary romance follo...  romance
13

                                                 vector
0     [-0.01199388, 0.027678106, 0.0064537725, 0.022...
1     [0.012646117, 0.026426949, 0.020831944, 0.0240...
2     [0.02572667, 0.029584605, -0.039901998, 0.0026...
3     [0.03676616, -0.0009867708, 0.007978628, 0.028...
4     [0.020591844, 0.004698135, 0.033796236, 0.0210...
...                                                 ...
1393  [0.03206649, 0.002541558, -0.034903083, 0.0201...
1394  [0.039164376, 0.029601324, -0.0057846084, 0.01...
1395  [0.045885768, -0.004113419, -0.012051461, 0.01...
```

```
1396  [0.019318914, -0.021177365, -0.028548038, 0.03...
1397  [0.031363852, 0.005924538, -0.035191517, 0.032...

[898 rows x 5 columns]
```

Convert the description of every class into string and store them in a dictionary

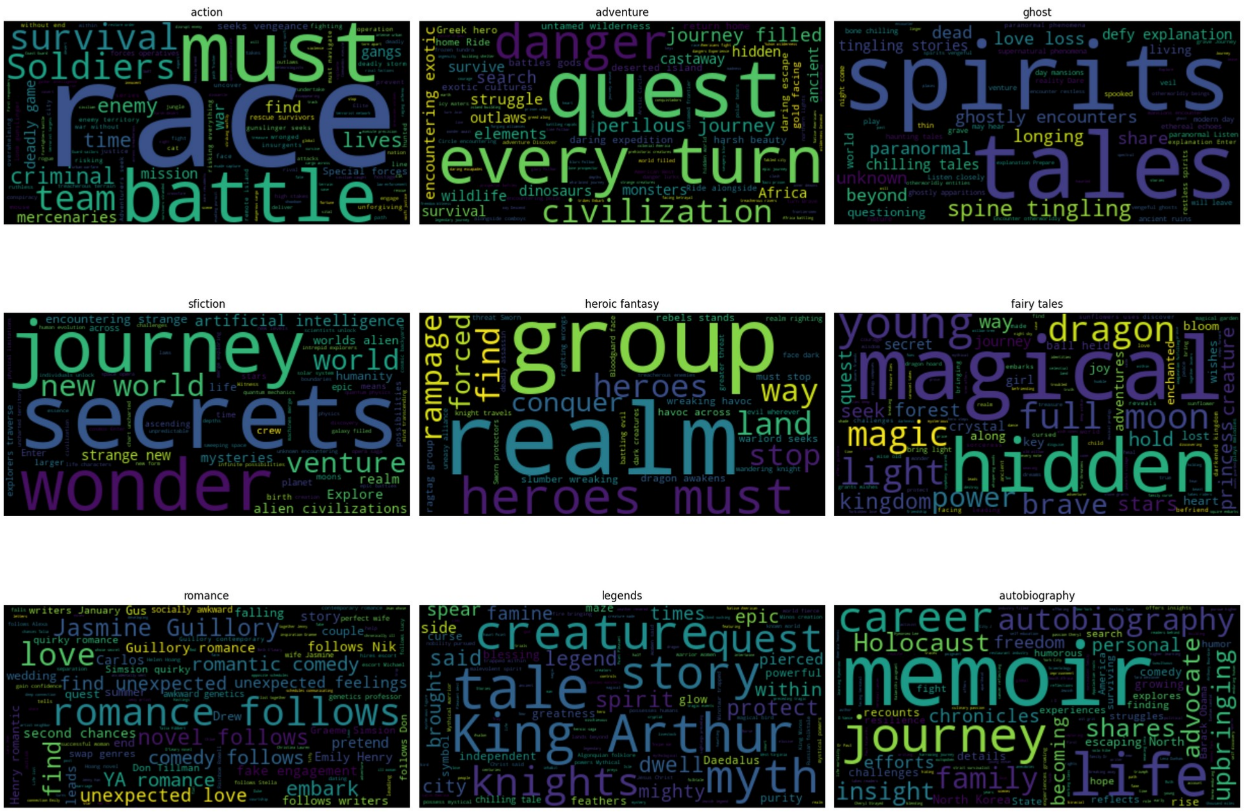
```
genre_texts = {}

# Iterate over each genre
for genre in ml_final['genre'].unique():
    # Concatenate text data for the current genre
    genre_text = ml_final[ml_final['genre'] == genre]
['description'].str.cat(sep=' ')
    # Store the concatenated text data in the dictionary
    genre_texts[genre] = genre_text
```

Word Cloud

Word cloud for different genres. This visualizes the most frequent text in the description in each genre

```
genres_of_interest = ['action', 'adventure', 'ghost', 'sfiction',
'heroic fantasy', 'fairy tales', 'romance', 'legends',
'autobiography']

# Define the number of rows and columns for subplots
num_rows = 3  # Adjust as needed based on the number of genres
num_cols = 3

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(20,15))

# Flatten axes if needed
if num_rows > 1:
    axes = axes.flatten()

for i, genre in enumerate(genres_of_interest):
    genre_text = genre_texts[genre]


    wordcloud =
WordCloud(background_color='black').generate(genre_text)


    axes[i].imshow(wordcloud, interpolation='bilinear')
    axes[i].set_title(genre)
```

```
    axes[i].axis('off')

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




```
sample = ml_final['vector'].iloc[0]

sample

array([-1.19938804e-02,  2.76781060e-02,  6.45377254e-03,
2.26829685e-02,
       -1.67668760e-02,  1.60643458e-03, -2.81919427e-02, -
4.23460938e-02,
        5.55780530e-02,  3.49348225e-03,  3.27580869e-02, -
1.82840936e-02,
        1.72826555e-02,  1.33724706e-02, -4.08946089e-02,
2.92749405e-02,
       -2.12888215e-02,  2.10861862e-02,  8.87039863e-03,
4.29179054e-03,
       -1.75357983e-02,  2.16857418e-02,  3.82898119e-03,
2.53107287e-02,
        5.48995435e-02, -2.07007974e-02, -5.78572750e-02,
4.62411717e-03,
        2.60032108e-03, -4.65378202e-02,  6.06090315e-02, -
3.31293531e-02,
```

```
         2.93650199e-03,  1.46736149e-02, -1.41730336e-02, -
6.56976539e-04,
         1.08747575e-02, -3.29440795e-02,  4.51020077e-02,
3.15123275e-02,
         2.85499804e-02, -1.26030073e-02,  2.62327194e-02,
1.49766253e-02,
        -1.55604947e-02, -3.97745706e-02, -1.33417873e-02, -
5.83102275e-03,
        -7.71748181e-03,  4.28199768e-02,  1.34740947e-02,
2.07937844e-02,
         1.98091157e-02, -1.24915512e-02,  2.91331264e-04, -
4.28250954e-02,
        -1.70061532e-02, -3.19818743e-02, -1.85566507e-02, -
2.18495093e-02,
        -1.58350151e-02,  1.69698335e-02, -1.61700789e-02, -
1.33346859e-02,
        -2.07941849e-02, -9.28584859e-03, -1.40174376e-02,
1.97822731e-02,
         1.61829367e-02,  7.19527900e-03, -1.02250632e-02,
3.67437233e-03,
         3.75072435e-02,  2.44661737e-02, -1.53042970e-03, -
5.29513881e-02,
         3.79155092e-02,  2.70136807e-04,  1.94374453e-02, -
1.91570781e-02,
        -3.35868564e-03,  5.33219054e-03,  1.70418415e-02,
1.02238907e-02,
         3.88034107e-03, -5.35847433e-02, -2.93791257e-02,
3.25313620e-02,
         1.83456540e-02,  4.40232232e-02, -5.47202289e-05, -
3.90328690e-02,
        -1.77220311e-02,  6.82614604e-03,  2.49056984e-02, -
3.00847888e-02,
        -5.70725929e-03, -2.96019427e-02, -1.88957751e-02,
8.95486306e-03,
         3.23585980e-02, -2.29329991e-04,  2.52287579e-03,
3.01727350e-03,
        -3.81308096e-03, -3.65429732e-04,  3.97190433e-06, -
1.64701082e-02,
         8.33290163e-03,  4.90026921e-03,  5.31905051e-03, -
2.80070100e-02,
        -5.51736243e-02,  2.48465082e-03,  1.52130034e-02,
1.80789363e-02,
         2.04972439e-02,  3.03514488e-02,  3.60771343e-02, -
7.63252145e-03,
        -3.72054204e-02, -3.44676897e-02, -3.35300192e-02,
3.18491347e-02,
        -3.66611965e-02, -2.88788918e-02,  7.87569396e-03, -
1.84571762e-02,
        -1.85596757e-04,  3.46756577e-02,  1.03231240e-03,
```

```
2.01416481e-02,
         2.51274109e-02, -3.36654484e-04,  6.89149508e-03,
4.68806271e-03,
         4.10584845e-02,  2.80173030e-02, -6.01161309e-02, -
7.18011567e-03,
        -2.61561871e-02, -5.82967401e-02,  7.28051970e-03,
5.22717945e-02,
         2.82804426e-02, -1.32498201e-02, -6.94039324e-03, -
1.83963589e-02,
        -2.64206640e-02, -3.25470194e-02,  5.30327335e-02,
1.61668025e-02,
        -2.44244933e-02,  2.15323996e-02,  1.07375477e-02, -
3.43296602e-02,
        -4.40250523e-03, -4.88221124e-02,  1.61033198e-02, -
1.26208086e-02,
         1.01553379e-02,  1.66074783e-02,  2.92182323e-02, -
2.14752369e-02,
        -1.63627584e-02, -4.86181974e-02,  1.48483468e-02, -
5.88358156e-02,
         1.25893916e-03,  1.08380690e-02,  5.90921054e-03, -
3.18167619e-02,
         1.46602618e-03, -7.29132444e-02,  4.91598621e-02, -
1.82594843e-02,
         1.28647462e-02, -5.44023979e-03, -4.64665564e-03, -
2.22128481e-02,
        -2.59038918e-02, -4.60980013e-02, -2.18864647e-03, -
3.52245616e-03,
         1.24672949e-02,  4.74946853e-03,  1.03389127e-02,
1.27377091e-02,
         3.77969518e-02,  1.64150484e-02,  7.70763448e-03,
4.90118098e-03,
         1.34692285e-02,  3.07424515e-02, -4.54997718e-02, -
1.47437351e-02,
         1.58545939e-04,  2.09338181e-02,  1.93707552e-02, -
3.26902159e-02,
         2.73140669e-02,  3.61428447e-02,  2.51150392e-02,
2.88097169e-02,
         9.44474712e-04, -2.27629803e-02, -1.90060996e-02, -
2.36385651e-02,
         1.57843847e-02, -2.53727436e-02, -8.67177546e-03, -
1.60874184e-02,
        -2.06768159e-02, -1.52127352e-02, -1.11150229e-02,
2.01648399e-02,
         8.61712638e-03, -9.34025925e-03, -6.37198910e-02, -
1.47662815e-02,
         7.07081938e-03,  3.50473216e-03, -1.97989084e-02,
4.17788811e-02,
         3.85540212e-03, -1.65159125e-02,  1.30861355e-02,
6.05173316e-03,
```

```
         2.17586122e-02, -2.78245118e-02,  6.19081501e-03, -
3.02882530e-02,
         4.49911226e-03,  4.48894035e-03,  1.74667705e-02, -
2.06959359e-02,
         1.87602267e-03, -1.19841658e-03,  3.68306562e-02, -
3.33955921e-02,
        -1.80764813e-02, -3.84330895e-04, -4.12911586e-02, -
1.63569190e-02,
         1.19666215e-02,  6.73098629e-03, -6.04139082e-03, -
2.52514295e-02,
        -1.64093785e-02, -1.39928265e-02, -6.74690399e-03, -
2.22017616e-02,
         3.16361198e-03,  1.07495124e-02,  3.23630497e-02,
2.34250519e-02,
         3.63200828e-02,  1.10410794e-03, -5.89396581e-02, -
5.38484380e-03,
         1.84691828e-02, -8.60794622e-04,  4.80171433e-03,
4.29252982e-02,
         9.71564651e-03,  3.78920361e-02, -6.70176297e-02, -
5.14952317e-02,
        -7.26294070e-02,  6.01052423e-04, -9.83959157e-03,
4.26544212e-02,
         1.04332166e-02,  2.30962355e-02,  2.06075720e-02, -
3.40553150e-02,
         2.17553079e-02, -1.67980511e-02, -3.16908844e-02,
2.18269434e-02,
        -4.40596417e-02,  6.18229446e-04,  7.70715903e-03,
6.81341905e-03,
        -6.94406172e-03, -3.12092248e-02, -3.18492129e-02,
2.85526477e-02,
         3.45781818e-02, -4.15961444e-02, -1.23750074e-02,
2.50208192e-02,
        -4.80974913e-02,  2.20844243e-03, -5.37249558e-02, -
2.88978573e-02,
         4.69745602e-04, -6.60136342e-03,  1.93090399e-03, -
1.00802798e-02],
      dtype=float32)
```

Train Test Split

Splits the 80% dataset into train subset and 20% dataset into test subset

```
X_train, X_test, y_train, y_test = train_test_split(
    ml_final.vector.values,
    ml_final.output,
    test_size=0.2,
    random_state=2022,
```

```
    stratify=ml_final.output
)
```

Reshaping the training set and testing set and printing their shape before and after reshaping

```
print("Shape of X_train before reshaping: ", X_train.shape)
print("Shape of X_test before reshaping: ", X_test.shape)


X_train_2d = np.stack(X_train)
X_test_2d =  np.stack(X_test)

print("Shape of X_train after reshaping: ", X_train_2d.shape)
print("Shape of X_test after reshaping: ", X_test_2d.shape)
```

```
Shape of X_train before reshaping:  (718,)
Shape of X_test before reshaping:  (180,)
Shape of X_train after reshaping:  (718, 300)
Shape of X_test after reshaping:  (180, 300)
```

MinMaxScaler

We employ MinMax scaling on the vector containing negative values since the machine learning model cannot handle negative data directly.

```
scaler = MinMaxScaler()
scaled_train = scaler.fit_transform(X_train_2d)
scaled_test = scaler.transform(X_test_2d)
```

SVC

```
svc = cross_val_score(SVC(C=3),scaled_train, y_train,cv = 3)
svc.mean()
```

```
0.9568224081822408
```

The SVC Model gives 0.967 accuracy which meansthe model performed exceptionally well

Multinomial Naive Bayes

```
mnb = cross_val_score(MultinomialNB(), scaled_train, y_train, cv = 3 )
mnb.mean()
```

```
0.8941480706648072
```

The Multinomial Naive Bayes has accuarcy of 0.909 which shows it performed well but not as good as SVC

KNN

```
knn = cross_val_score(KNeighborsClassifier(n_neighbors=
7),scaled_train,y_train,cv = 3)
knn.mean()
```

```
0.8370467224546724
```

The KNN Classifier has the accuracy of 0.798 which shows it does not perform well for high dimensional data

Random Forest Classifier

```
rf = cross_val_score(RandomForestClassifier(), scaled_train,y_train,
cv = 3 )
rf.mean()
```

```
0.9331299395629938
```

The Random Forest Classifier has the accuracy of 0.936 which shows it performed well and is the next best model after SVC

Fitting the SVC Model

```
clf = SVC(C=3 , kernel = 'rbf')
clf.fit(scaled_train,y_train )
```

```
SVC(C=3)
```

CLASSIFICATION REPORT

```
from sklearn.metrics import classification_report

y_pred = clf.predict(scaled_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.90      0.90      0.90        20
           2       0.90      0.95      0.93        20
           4       1.00      1.00      1.00        20
           6       0.95      1.00      0.98        20
           9       0.83      0.95      0.88        20
          10       0.94      0.75      0.83        20
          12       1.00      0.95      0.97        20
          13       1.00      1.00      1.00        20
          14       1.00      1.00      1.00        20

    accuracy                           0.94       180
   macro avg       0.95      0.94      0.94       180
weighted avg       0.95      0.94      0.94       180
```

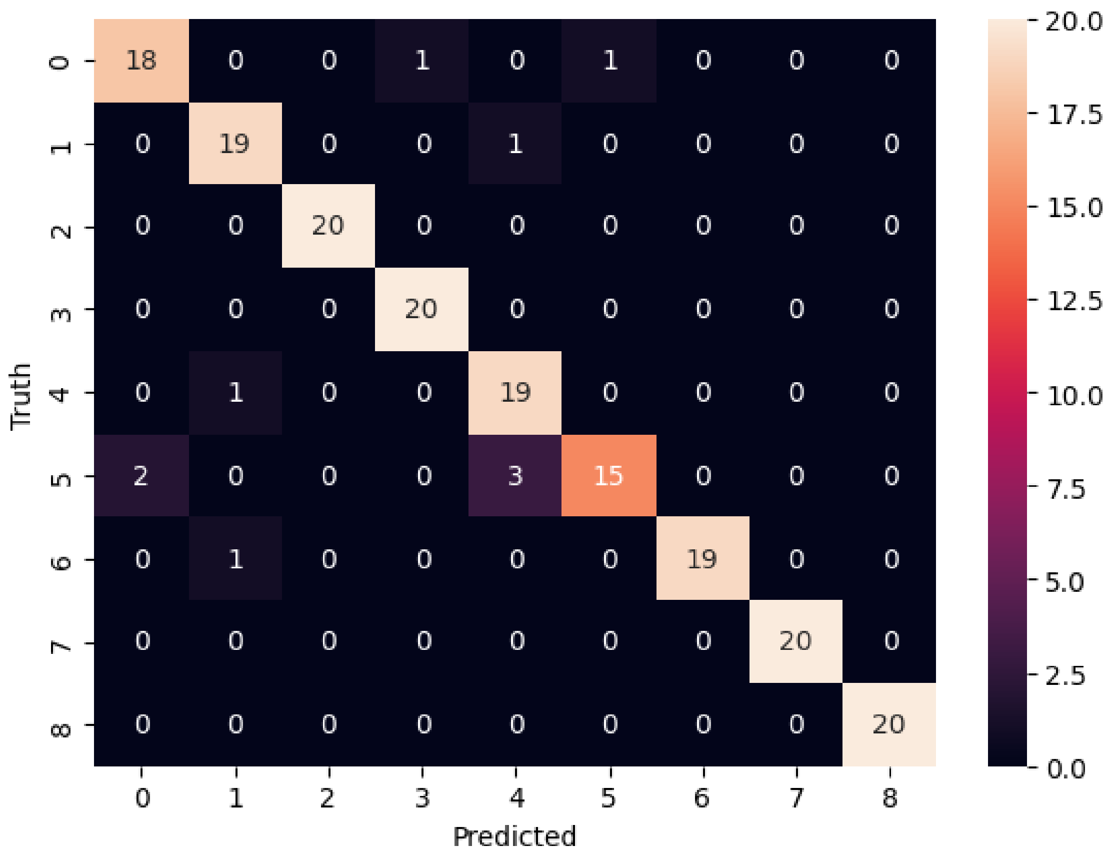
CONFUSION MATRIX

```
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[18,  0,  0,  1,  0,  1,  0,  0,  0],
       [ 0, 19,  0,  0,  1,  0,  0,  0,  0],
       [ 0,  0, 20,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 20,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0, 19,  0,  0,  0,  0],
       [ 2,  0,  0,  0,  3, 15,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0, 19,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 20,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0, 20]], dtype=int64)
```

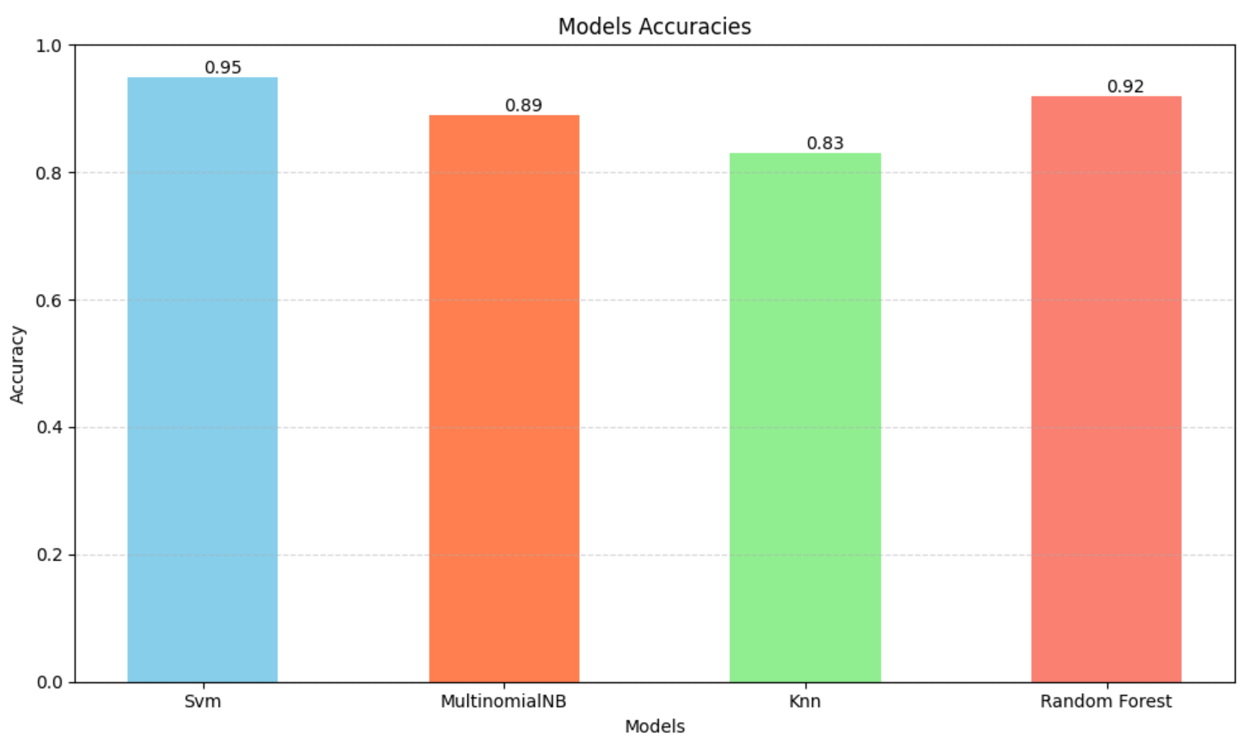
```
%matplotlib inline
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(58.222222222222214, 0.5, 'Truth')
```



Accuracy of different models

```
models = ['Svm', 'MultinomialNB', 'Knn', 'Random Forest']

# Accuracy scores
acc = [0.95, 0.89, 0.83, 0.92]

# Create bar plot
plt.figure(figsize=(10, 6))
b = plt.bar(models, acc, color=['skyblue', 'coral', 'lightgreen',
'salmon'], width=0.5)
for bar in b:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2),
va='bottom')

plt.title('Models Accuracies')
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.xticks()

# Show plot
plt.tight_layout()
plt.show()
```



INPUT AND RECOMMENDATION

```
t1 = '''Join a team of elite operatives as they embark on a high-stakes mission
to thwart a terrorist plot and save the world from imminent danger.'''

sam = preprocess_and_vectorize(t1)  # Assuming preprocess_and_vectorize function returns a 1D array

sam_reshaped = sam.reshape(1, -1)

scaled_sam = scaler.transform(sam_reshaped)

ip = clf.predict(scaled_sam)

type(ip)
```

```
numpy.ndarray
```

```
ml_final[ml_final['output'].values == ip]['title'].sample(n=5)
```

```
10      Bulletproof: The Shield of Justice
60             Final Assault: War on Terror
54    Deadly Cargo: Race Against the Clock
50       Final Stand: Last Line of Defense
9                     Inferno: Fire and Fury
Name: title, dtype: object
```