Movie recommendation using Machine Learning

Mini Project

Submitted in fulfillment of the Course

On

Machine Learning

In

Python

By

Shreyas Udupa, Kunal Kotkar, Sanket Jangale, Kedar Kharde

Under the guidance of

GURVANSH SINGH, MTech

Knowledge Solutions India

Skill development | Certification | Placement prep

ABSTRACT

Movie recommendation is an engine that filters movies and suggests them to the user based on their past viewing experiences. This engine has gained more popularity due to the impact of various OTT platforms like Netflix, Prime Video, Hotstar, etc. We have used 3 different algorithms to make our model and execute the engine. First is K-Means clustering, which makes clusters of the movie based on Rating and popularity. Second is the KNN algorithm, which takes the Genre as input, and it displays the nearest neighbors pertaining to the same Genre. Lastly, we have used the SVM algorithm to train our model and give the list of the movies according to the Genre given as input. Based on the execution of the models according to the algorithms as mentioned above, we can concur that the KNN model is more suited than the SVM model to execute the recommendation system.

TABLE OF CONTENTS

INTRODUCTION **5**
SOFTWARES **6**
ALGORITHMS **8**
LIBRARIES **10**
MATHEMATICS **11**
INFERENCE **13**
CONCLUSION **14**
CODE **15**
REFERENCES **32**

LIST OF FIGURES

Figure 1 - Elbow method 16
Figure 2 - Cluster plot for Movie Rating 17
Figure 3 - Box plot for Votes column 19
Figure 4 - Output of SVM 23
Figure 5 - Scatter plot of Original Movies vs. Predicted Movies (SVM) 24
Figure 6 - Scatter plot of Original Movies vs. Predicted Movies (KNN) 26
Figure 7 - Output for KNN 28
Figure 8 - Homepage GUI 29
Figure 9 - SVM menu 29
Figure 10 - SVM output 30
Figure 11 - KNN menu 30
Figure 12 - KNN output 31

INTRODUCTION

A recommendation engine is a type of information filtering system that predicts the preferences of the user. It makes a suggestion based on those preferences. Nowadays, the use of recommendation systems has increased manifold as most of the online platforms depend on this system to engage their users. Ecommerce sites like Amazon, Flipkart, Alibaba, and various other indigenous businesses use this system to suggest items based on their previous purchases and their search history. E.g., if a user wants to buy a new phone, the application recommends phones as well as the accessories associated with the search. OTT platforms like Netflix, Amazon Prime Video, Hotstar, etc. use this to suggest movies and tv shows based on previous movies viewed. If the user is using the platform for the first time, then the platform asks for the genres which the user prefers and then displays the associated movies and TV shows. Music apps like Apple Music, Spotify also use this to suggest songs based on the user's past song preferences. Each of the big companies have their own patented recommendation algorithm, which makes their application unique and attractive. People are so used to these apps, that the level of expectation has increased manifold. New users get disinterested by a new app if they don't find the recommendations according to their preferences. So all the new app developers are concentrating more on the development of their recommendation system.

SOFTWARES

We have mainly used Python and its vast array of libraries to assist us in the making of this project.

Python is a general-purpose interpreted, interactive, object-oriented, and high-level,[1] interpreted, interactive, and object-oriented scripting language. Python has a highly readable design. It uses English keywords frequently, whereas other languages use punctuation, and it has fewer syntactic constructions than other languages. Python is a must for students and working professionals to become a great Software Engineer,[1] especially when they are working in the Web Development Domain.

Following are essential characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.[1]

Here are a few essential reasons as to why Python is popular:

- Python has a vast collection of libraries.
- Python is known as the beginner's level programming language because of its simplicity and easiness.[2]
- From developing to deploying and maintaining, Python wants its developers to be more productive.[3]
- Portability is another reason for the vast popularity of Python.
- Python's programming syntax is simple to learn and is of high level compared to C, Java, and C++.[3]

Jupyter Notebook - The Jupyter Notebook is an open-source web application that allows us to create and share documents that contain live code, equations, visualizations, and narrative text.[4] Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.[4]

The code is written and implemented in both Jupyter Interactive Notebook and IDLE. We have implemented the GUI (Tkinter) part in IDLE.

Tkinter-The Tkinter module ("Tk interface") is the standard Python interface to the Tk GUI toolkit from Scriptics. Both Tk and Tkinter are available on most Unix platforms, as well as on Windows and Macintosh systems.[5] Starting with the 8.0 release, Tk offers native look and feel on all platforms.

It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with Tkinter is the fastest and easiest way to create the GUI applications.[6]

The several Python modules provide the public interface. The most crucial interface module is the Tkinter module itself.

The Tkinter module only exports widget classes and associated constants, so you can safely use the from-in form in most cases.

ALGORITHMS

1. **K-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. [7]
    a. It is popular for cluster analysis in data mining. *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem. [7]
    b. k-means clustering is rather easy to apply to even large data sets, mainly when using heuristics such as Lloyd's algorithm. [7]
    c. It has been successfully used in market segmentation, computer vision, and astronomy, among many other domains. It often is used as a preprocessing step for other algorithms[7]
2. **SVM Algorithm-**The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classify the data points.[8]
    a. Our objective is to find a plane that has the maximum margin, i.e., the maximum distance between data points of both classes. [8]
    b. Two main concepts are used in SVM:
        i. Hyperplanes- They are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. [8]
        ii. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. [8]
3. **KNN (K-Nearest Neighbors)-** The KNN algorithm is a robust and versatile classifier that is often used as a benchmark for more complex classifiers such as Artificial Neural [9] Networks (ANN) and Support Vector Machines (SVM).
    a. The KNN classifier is also a non-parametric and lazy learning algorithm.
        i. KNN is a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

    ii. Lazy learning means that the algorithm makes no generalizations. This means that there is little training involved when using this method. Because of this, all of the **training data is also used in testing when using KNN[10]**

b. KNN falls in the supervised learning family of algorithms. In KNN, there are a few hyper-parameters that we need to tune to get an optimal result. The distance metric is one of the important ones through which we calculate the distance between the data points.[11]
    i. KNN runs a formula to compute the distance between each data point and the test data. It then finds the probability of these points being similar to the test data and classifies it based on which points share the highest probabilities.[10]

c. A small value for K provides the most flexible fit, which will have low bias but high variance.[11] Larger values of K will have smoother decision boundaries, which[12] means lower variance but increased bias.[11]

LIBRARIES

1) **Pandas** - pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time-series data both easy and intuitive.[13] It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible[14] open-source data analysis/manipulation tool available in any language.
2) **Numpy** - NumPy is a python library used for working with arrays. It also has functions for working in the domain of linear algebra, Fourier transform, and matrices.
3) **Matplotlib** - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits.[15]
4) **SKLearn** - Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy.
5) **Import_ipynb** - Used to display output of imported ".ipynb" files in Jupyter.
6) **Columnar** - It is a library for creating columnar output strings using data as input.
    a) columnar() Arguments:
        i) Data:
            (1) An iterable of iterables, typically a list of lists of strings where each string occupies its cell in the table.
        ii) headers=None
            (1) A list of strings, one for each column in data, which will be displayed as the table headers. If left as None, this will produce a table that does not have a header row.
        iii) no_borders=False
            (1) Accepts a boolean value that specifies whether or not to display the borders between rows and columns. Passing True will hide all the borders and convert the headers to all caps for a more minimalistic look.[16]

MATHEMATICS

In the SVM algorithm, we are mainly passing the Kernel parameter to SVC. There are three types of kernel-

1) Linear:
   a) It is mostly used when data are linearly related.
   b) Equation is:

$$k(xi,xj) = a1xi + a2xj + c$$

Linear kernel equation

2) Rbf Kernel (Radial basis function):
   a) It is a general-purpose kernel; used when there is no prior knowledge about the data.
   b) Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Gaussian radial basis function (RBF)

3) Polynomial Kernel:
   a) It is popular in image processing.
   b) Equation is:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

      i) where d is the degree of the polynomial.\

Euclidean distance:
When implementing KNN, the first step is to <u>transform data points into feature vectors, or their numerical value.</u> The algorithm then works by finding the distance between the mathematical values of these points. The most common way to find this distance is the Euclidean distance, as shown below.[8]

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Minkowski distance:
Euclidean distance can be generalized using the Minkowski norm, also known as the p norm. The formula for Minkowski distance is:

D(x, y) = (Σd|xd – yd|^p)^(1/p)

Here we can see that the formula differs from the formula of Euclidean distance as we can see that instead of squaring the difference, we have raised the difference to the power of p and have also taken the p root of the difference. Now the most significant advantage of using such a distance metric is that we can change the value of p to get different types of distance metrics.

1) p = 2
    a) If we take the value of p as 2, then we get the Euclidean distance.
2) p = 1
    a) If we set p to 1, then we get a distance function known as the Manhattan distance.[11]

INFERENCE

In K-Means Cluster, data is fit into a variable, and k-means was performed on it using the factors like "Number of Votes" and "Ratings." The Result shows the movies divided into 4 categories as "Best," "Good," "Average," and "Worst." Naturally, very few movies deserved the "Best" category, and the majority of the movies belonged to either "Average" or "Worst" category.

As Genres contain multiple classes, it is not easily separable, i.e., the basic SVM uses linear hyperplanes to separate classes, and if we provide a different kernel (Non-linear kernel), then it will change the shape of the decision manifold that must be used.

Although KNN gives better results, but SVM is more trusted and is considered as a real-time classifier. For example, If we have a fixed data, then KNN may perform better, but if we have to build a prediction model and have to test it on real-time samples which were not previously available with the given dataset with whom we did the training, then SVM will perform better because it has right learning approach.

As in our dataset, there are no new movies to predict, and mostly in static data, KNN performs better.

CONCLUSION

The popularity and effectiveness of the Movie Recommendation System has been on the rise since the launch of various entertainment services. These services rely heavily on this system to retain its users. Based on the models that we have trained, we can concur that the KNN model is more accurate and faster in predicting the appropriate movies than the rest of the models.

Therefore based on the above inference, we can conclude that the KNN model is more accurate at predicting the movies than the SVM model.

CODE

Based on the problem statement, we have made two recommendation system variants. The first one gives the user a graphical and interactive interface that sorts the movies from bad to best. This model helps the user to decide on which movies are worth watching and which are worth skipping. We have achieved this by taking into consideration the Rating and the number of Votes each movie has received till the time it was in theatres. We have used the K-means clustering algorithm to achieve this Result.

#K-Means Clustering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

#Importing the dataset

```
dataset = pd.read_csv("IMDB-Movie-Data.csv")
x=dataset.iloc[:,[8,9]].values #taking values of ‘rating’ and ‘votes’
```

#Elbow Plot to determine the number of clusters

```
from sklearn.cluster import KMeans
# making a wcss list which will contain wcss values for the respective cluster number
wcss = []
# starting a for loop to provide the different cluster values
for i in range(1,11):
    k_means = KMeans(n_clusters = i, init = "k-means++", random_state = 0)
    k_means.fit(x)
# to get the wcss value KMeans class provides us an inbuilt function 'inertia_'
    wcss.append(k_means.inertia_)
print(wcss)
x_range = range(1,11)
plt.plot(x_range,wcss)
plt.title("The Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```

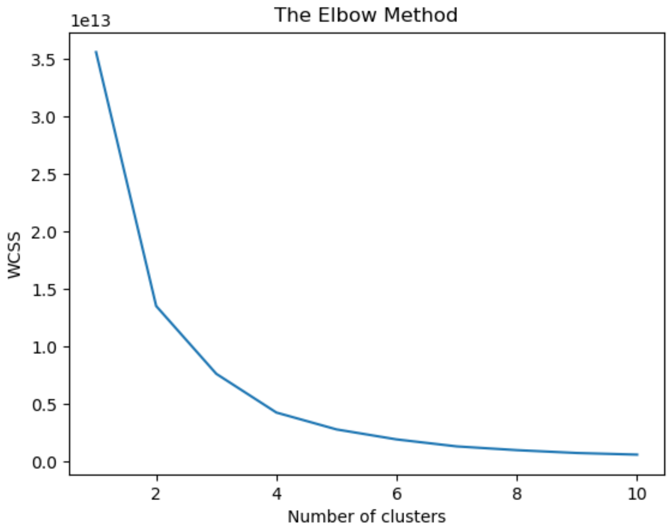

Figure 1- Elbow method

Cluster plot with n_cluster=4

kmeans = KMeans(n_clusters = 4, init = "k-means++", random_state = 0)
kmeans.fit(x)
y_kmeans = kmeans.predict(x)
plt.scatter(x[y_kmeans == 0,0] , x[y_kmeans == 0,1], s = 100, c = "red", label = "Average")
plt.scatter(x[y_kmeans == 1,0] , x[y_kmeans == 1,1], s = 100, c = "blue", label = "Good")
plt.scatter(x[y_kmeans == 2,0] , x[y_kmeans == 2,1], s = 100, c = "green", label = "Worst")
plt.scatter(x[y_kmeans == 3,0] , x[y_kmeans == 3,1], s = 100, c = "yellow", label = "Best")
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1], s = 200, c = "black", label = "Centroid")
plt.title("Best to Worst Movies")
plt.xlabel("Rating")
plt.ylabel("Votes")
plt.legend()
plt.show()

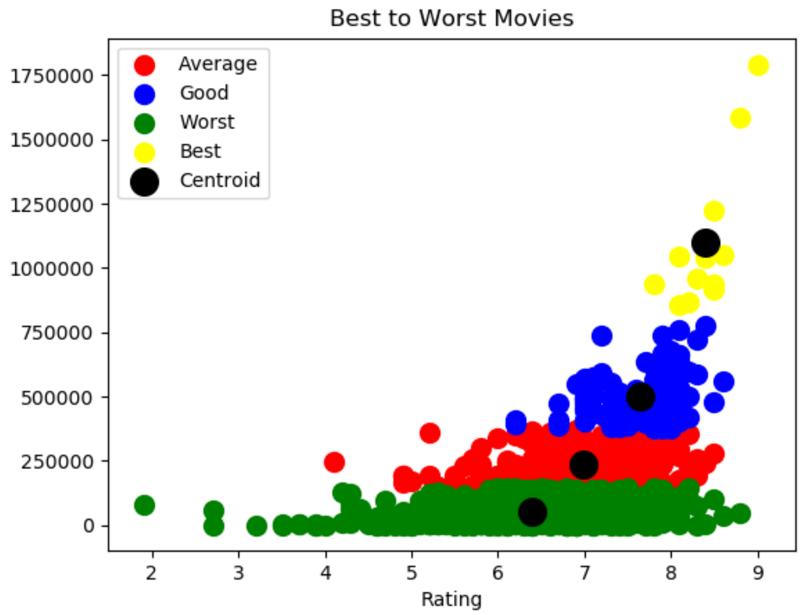

Figure 2- Cluster plot for Movie Rating

The second one gives the user an option to enter his/her desired Genre according to which the model recommends the movies. This model gives the user a comprehensive list of movies pertaining to the desired Genre, also sorted according to the Rating to make a simplified list that suggests the user the top 10 movies in the selected Genre. This model is beneficial for users who like to watch movies of a particular genre only, and hence they have a personalized list related to the same.

We have used the KNN algorithm and the SVM model to train our data and display the desired Result.

```
#Taking the input of the genre from the user
def get_key(val):
    for key, value in list1.items():
         if val == value:
             return key
    return
list1={2:"Action",3:"Adventure",4:"Animation",5:"Biography",6:"Comedy",7:"Crime",8:"Drama",9:"Family",10:"Fantasy",11:"History",12:"Horror",13:"Music",14:"Musical",15:"Mystery",16:"Romance",17:"Sci-Fi",18:"Sport",19:"Thriller",20:"War",21:"Western"}

list2=['Action',"Adventure","Animation","Biography","Comedy","Crime","Drama","Family","Fantasy","History","Horror","Music","Musical","Mystery","Romance","Sci-Fi","Sport","Thriller","War","Western"]
```

```
x_input=input('Enter the genre of your choice')

while x_input not in list2:
    print('Enter first letter capital or enter a correct genre')
    x_input=input('Enter the genre of your choice')

col=get_key(x_input)
print(col)
```

```
Enter the genre of your choiceAction
2
```

#Importing the dataset

```
dataset = pd.read_csv('IMDB-Movie-Data.csv')
print(dataset)
```

	Rank	Title	...	Revenue (Millions)	Metascore
0	1	Guardians of the Galaxy	...	333.13	76.0
1	2	Prometheus	...	126.46	65.0
2	3	Split	...	138.12	62.0
3	4	Sing	...	270.32	59.0
4	5	Suicide Squad	...	325.02	40.0
..	...	...	...	...	...
995	996	Secret in Their Eyes	...	NaN	45.0
996	997	Hostel: Part II	...	17.54	46.0
997	998	Step Up 2: The Streets	...	58.01	50.0
998	999	Search Party	...	NaN	22.0
999	1000	Nine Lives	...	19.64	11.0

#Sorting values according to rating in descending order

```
dataset = dataset.sort_values('Rating',ascending=False)
x= dataset.iloc[:,[2,8]]
print(x)
```

	Genre	Rating
54	Action,Crime,Drama	9.0
80	Action,Adventure,Sci-Fi	8.8
117	Action,Biography,Drama	8.8
36	Adventure,Drama,Sci-Fi	8.6
96	Animation,Drama,Fantasy	8.6

.. ..
968 Action, Horror, Thriller 3.5
647 Fantasy, Horror, Thriller 3.2
871 Action, Adventure, Fantasy 2.7
42 Horror 2.7
829 Comedy 1.9

dataset.boxplot(column=['Votes']) #plotting the column “Votes”

<matplotlib.axes._subplots.AxesSubplot at 0x151d7d48>

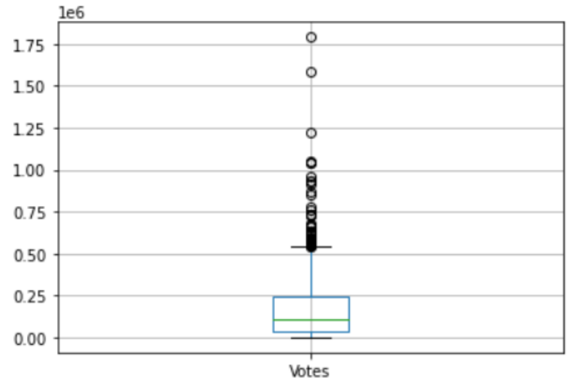

Figure 3- Box plot for Votes column

Encoding Genre using pd.str_get_dummies()

x = pd.concat([x.drop('Genre', 1), x['Genre'].str.get_dummies(sep=",")], 1)
print(x)

#Since the Genre column has multiple genres for a single movie separated by commas; we cannot directly use OneHotEncoder. So we used Pandas.str.get_dummies() instead. Pandas str.get_dummies() is used to separate each string in the caller series at the passed separator. A data frame is returned with all the possible values after splitting every string. If the text value in the original data frame at the same index contains the string (Column name/Split values), then the value at that position is 1 otherwise, 0.

Output-

	Rank	Rating	Action	Adventure	...	Sport	Thriller	War	Western
54	55	9.0	1	0	...	0	0	0	0
80	81	8.8	1	1	...	0	0	0	0
117	118	8.8	1	0	...	0	0	0	0
36	37	8.6	0	1	...	0	0	0	0
96	97	8.6	0	0	...	0	0	0	0
..	...	...	... ...	...	...	...		...	...
968	969	3.5	1	0	...	0	1	0	0
647	648	3.2	0	0	...	0	1	0	0
871	872	2.7	1	1	...	0	0	0	0

42	43	2.7	0	0	...	0	0	0	0
829	830	1.9	0	0	...	0	0	0	0

```
X = x.iloc[:,0:21].values

x1=dataset.iloc[:,[0,2,8]]
x1 = pd.concat([x1.drop('Genre', 1), x1['Genre'].str.get_dummies(sep=",")], 1)

from sklearn.decomposition import PCA
pca = PCA(n_components=None) #Since we do not know how many eigenvectors are needed, we keep the value of ‘n components=None' so that we can get the eigenvalues of all the eigenvectors to figure out the best one.
X = pca.fit_transform(X)
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.33238783 0.14317095 0.10089998 0.0705231  0.05540225 0.0459295
 0.03789689 0.03530911 0.03500896 0.02613341 0.02515884 0.02438035
 0.01652608 0.01301041 0.01021592 0.00877265 0.00573457 0.00520926
 0.00417902 0.00240437 0.00174652]
```

```
np.set_printoptions(precision=2)
print(X)
```

```
[[-2.29e+00  2.63e-01 -5.17e-01 ... -4.49e-02 -2.25e-03 -1.07e-02]
[-1.87e+00  1.59e+00 -1.47e-01 ... -2.09e-02 -1.38e-02 -1.34e-03]
[-2.15e+00  2.38e-01 -3.57e-01 ... -3.96e-02 -1.30e-03 -6.62e-03]
...
[ 4.06e+00  5.71e-01 -1.62e-01 ...  2.73e-02 -2.60e-04  8.54e-03]
[ 4.06e+00 -6.30e-01 -1.74e-01 ... -2.81e-02 -2.25e-02 -9.17e-03]
[ 4.77e+00 -7.58e-01  6.74e-01 ... -2.09e-02 -2.54e-02 -3.07e-03]]
```

```
y= dataset.iloc[:,0].values
print(y)
```

```
[  55   81  118   37   97  250  134   65  100  992  125  477  635  431
  862  145    7  456  500   78  689  479   27  766   83  242  646  714
  743   17  195  144  231  155  193  428   68   75   77  146   84   91
  199   93  239  137  335  300  112  115  198  139    1  773  642   19
  696  609   51  165  670  333  103  686  262  274  366  641  130   20
  236  141  163   34  174  469  185  490  448  171  247   22  898  253
  104  695  172  753  599  201  505  272  204   36  657  519  160   82
   13  904  550  815  233  471  486  224  449  256  261  951  206  217
  271  143  510  794  591  960  511  848  840  786  408  692  363  360
   88   89  189  411  860  339  564  337  760  574  418  312  426  385
   12  659  663  685  484  220  890  735  278  294  311  325  350  358
  364  404  419  508  443  475  871  347   66  106  114  110   14  958
  579  148  712  728  612  241  246  258  585  744  282   38  980  855
  461  378  381  384  438  816  818  149  829  687  177  673  154  464
  126  637   42   73  837  442  466  255  159  990  882  682  761  291
  302  614  175   60  568  886  644  268   32   69  919   11  850  315
```

```
  736  207  208  858  916  474  476  852  798  124   90  179  248  120
   95  975   58  499  444   67  833  328  651  147  709  314  983  407
  334  434  357  542   96  234  158  342  940  606  382  783  320  105
  351  762  763  797  281  539  348  878  504   47  332  775  756  169
  413  203    3  713  968  638  138  621  620  136  180  846  168  131
  927  939  252   56  600  952  589  323  460   79  698  420   70  414
  512  481   85   54  393  861  802    4  496   57  396  502   44  292
  521  301  680  280  273  590  751  954  831  678  947  604  676  119
  596  913  914  541  780  699  587  535  191  152  607   76  379  593
  930  808  738  447  386  135  707    9  703  213   72  436  140  403
  437  690  222  722  227  843  844  249  665  897  283  514  864  544
   40  552  555  299  557  295  498  516   94  572  573   33  895  277
  893  896  260  353   49  989  854  981  656  493  537  824  661  525
   10  533  636  433  451  800    2  979  633  813  610   59  452  732
  702  162   92  183  369  226  229  316  225  874  329  215  876  303
  961  102  101   86  284  264  966  178  279  157  276  877  375  908
  531  868  468  867  377  618  785  492  946  368  518  733  361  331
  608  881  559  417  884  223   39  662  972  196  870  821  781  965
  729  344  421  787  497  237  903  547  549   26  298  563  985  887
  575  577  578  313  275  453  809  845  853  372  397   62   74  209
  924  182  851  805  653  932  170  938  822  306  601  963  388  546
  457  389   23  767  811   61  847  254  122  677  405  121  181  406
  681  995  918  660  654  269   18  627  487  721   46  488  480   21
  324  631  354  789  967  730  210  470  221  640  613  530  374  907
  915   16  717  742  647  188  128  597  727  910  704  666  991  664
  623  615  917  734  632  764  524  771  409  869  527  439  793  446
  341  795  352  489  857  359  390  383  463  875  807  936  560  758
  296  970  548  899  507  561  503  340  562  752  491  716  866  427
  570  167  799  454  482  888  708  669  705  782  244  683  176  679
  536   24  901  672  909  693  784  321  701  202  984  412   63  832
  459  945  151    8  835  921  435  842  401  923  911  655  259  603
  305  133  534  529  652  228  622  322  594  129  790  720  343  859
   87   15  472  955  467  602  883  754  934  993  769  731  964  836
  814  792  841  865  922  849  988  501  485  554   99  365  543  197
  200  391  626  619  111  616  558  567  580   45   41   31  187  108
  373  668  355  190  441  495  465  483  812  977  150  804  803  974
  959  345  319  424  307  117  796  776  863  956  745  520  768  142
  718  263  892  725  726   29  611  906    5  996  598  998  710  630
  935  592  905  885  920  697  540  944  950  688  336  164  400  235
  455  671  113  629  929  286  746  941  925  290  380  569  410  565
  211  219  595    6  310  749  458  462  806  628  445  161  186  588
  430  132  399  267  285  265  309  723  625  517  326  791  571  327
  711  976  371  123  931  425  856  184  973  450  987  684  240  675
  245  658  994   30  928  750   71  962  338  214  346  289  880  879
  257  194  287  243  392  473  706  715  719  576   52  566   98  109
  116  551  423  153  943  825  823  834  422  218  739  770  584  748
  667  370  293  297  765  376  205  545  127  538  478  362  515  506
  624  999  674  650  415  639  902  948  737  387  192  986   35   48
  774  440   53  997  216  694  900  367  838  308  304  356  889  398
  645  894  700  395  330  691  819  416  817  873  528  532  107  741
  933  494  522   80  982  826   25  828  649 1000  212  288  318  166
  757  957  317  759  429  810  586  740  173  232  605  724  583  777
  349  755  513  978  394  634  251  581  949  942  266  926  971  230
  643  779  912  156  778   28  523  432  556  820  788  747  827  772
  937  953  801  509  553  839  582   64  617  238  402  526  270   50
  891  969  648  872   43  830]
```

SVM Classifier

```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf',random_state=0)
classifier.fit(X,y)
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
coef0=0.0,
    decision_function_shape='ovr', degree=3,
gamma='scale',kernel='rbf',
    max_iter=-1, probability=False, random_state=0, shrinking=True,
tol=0.001,     verbose=False)
```

```
y_pred = classifier.predict(X) #Predicting the rank of movies
```

```
data_svm=[]
i=0
for j in range(0,1000):
   trry:
                  if(x1[x1.Rank==y_pred[j]][x_input].values[0]==1):
               list_svm=[]
               list_svm.append(dataset[dataset.Rank==y_pred[j]]['Title'].values[0])
               list_svm.append(dataset[dataset.Rank==y_pred[j]]['Genre'].values[0])
               list_svm.append(dataset[dataset.Rank==y_pred[j]]['Rating'].values[0])
               data_svm.append(list_svm)
               list_svm=None
               i+=1
         if i==10:
               break
         else:
               pass
   except IndexError:
         Break
```

Recommending Movies using SVM

```
!{sys.executable} -m pip install columnar
from columnar import columnar
print('Your Input genre is ',x_input)
print('Recommending best movies of ',x_input,' Genre : \n\n')
headers = ['Title', 'Genres', 'Rating']
data_svm = sorted(data_svm,key=lambda lr:lr[2], reverse=True)
table = columnar(data_svm, headers, no_borders=False)
print(table)
```

```
Requirement already satisfied: columnar in c:\users\sanket\appdata\local\programs\python\python37\lib\site-packages (1.3.1)
Requirement already satisfied: toolz in c:\users\sanket\appdata\local\programs\python\python37\lib\site-packages (from columna
r) (0.10.0)
Requirement already satisfied: wcwidth in c:\users\sanket\appdata\local\programs\python\python37\lib\site-packages (from column
ar) (0.1.9)
Your Input genre is  Action
Recommending best movies of  Action  Genre :
```

Title	Genres	Rating
The Dark Knight	Action,Crime,Drama	9.0
Inception	Action,Adventure,Sci-Fi	8.8
Dangal	Action,Biography,Drama	8.8
The Dark Knight Rises	Action,Thriller	8.5
Bahubali: The Beginning	Action,Adventure,Drama	8.3
Warrior	Action,Drama,Sport	8.2
The Bourne Ultimatum	Action,Mystery,Thriller	8.1
Mad Max: Fury Road	Action,Adventure,Sci-Fi	8.1
The Avengers	Action,Sci-Fi	8.1
Rush	Action,Biography,Drama	8.1

Figure 4- Output of SVM

print(data_svm)

```
[['The Dark Knight', 'Action,Crime,Drama', 9.0], ['Inception',
'Action,Adventure,Sci-Fi', 8.8], ['Dangal', 'Action,Biography,Drama',
8.8], ['The Dark Knight Rises', 'Action,Thriller', 8.5], ['Bahubali:
The Beginning', 'Action,Adventure,Drama', 8.3], ['Warrior',
'Action,Drama,Sport', 8.2], ['The Bourne Ultimatum',
'Action,Mystery,Thriller', 8.1], ['Mad Max: Fury Road',
'Action,Adventure,Sci-Fi', 8.1], ['The Avengers', 'Action,Sci-Fi',
8.1], ['Rush', 'Action,Biography,Drama', 8.1]]
```

#Getting the accuracy , mean_square and r-Square of our model
import sklearn.metrics as met
from sklearn.metrics import accuracy_score
print(met.r2_score(y,y_pred))
print(met.mean_squared_error(y,y_pred))
print(accuracy_score(y,y_pred))

```
0.5659642339642339
36169.611
0.79
```

Plot of original movies in dataset vs. predicted movies(ALL Genres)

x2=x.iloc[:,[0]].values

```
plt.scatter(x2,y,color='red') #original movies(entire set)
plt.scatter(x2,y_pred,color='green') #predicted movies(entire set)
plt.xlabel('Rating')
plt.ylabel('Rank')
plt.show()
```

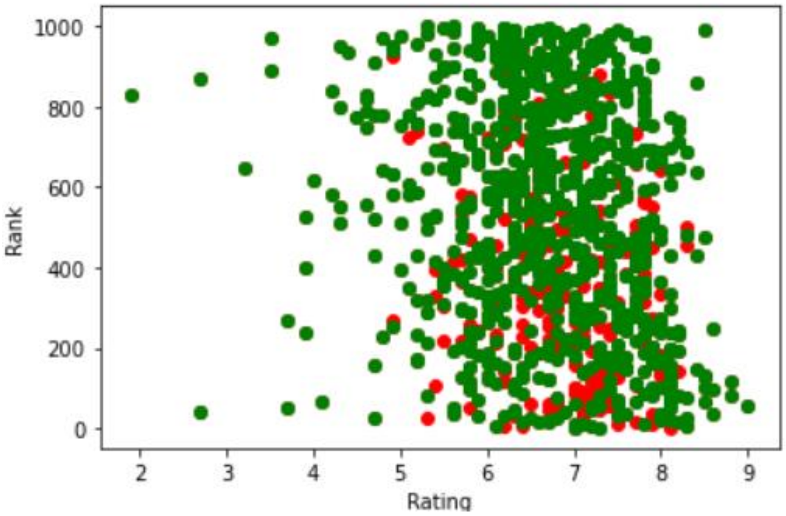

Figure 5- Scatter plot for predicting data (SVM)

KNN Algorithm

```
X_KNN = x1.values
print(X_KNN)
[[ 55.    9.    1.   ...   0.    0.    0. ]
 [ 81.    8.8   1.   ...   0.    0.    0. ]
 [118.    8.8   1.   ...   0.    0.    0. ]
 ...
 [872.    2.7   1.   ...   0.    0.    0. ]
 [ 43.    2.7   0.   ...   0.    0.    0. ]
 [830.    1.9   0.   ...   0.    0.    0. ]]
```

#preparing training data with input Genre

```
x_test = []
y_test = []
#Getting ranks
Ranks  = []
for i in range(0,len(X_KNN)):
    if X_KNN[i,col]==1:
        Ranks.append(X_KNN[i,0])
        x_test.append(X_KNN[i,1:])
        y_test.append(X_KNN[i,0])
x_test=np.array(x_test)
y_test=np.array(y_test)
y_test=y_test.ravel()
```

```
X_train_predict = x_test.copy()
Y_train_predict = y_test.copy()
```

 # Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_predict = sc.fit_transform(X_train_predict)
```

#KNN Model

```
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(n_neighbors=1,metric='minkowski',p=2)
classifier_knn.fit(X_train_predict,Y_train_predict)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1,
p=2,
                     weights='uniform')
```

#Predicting

```
Y_pred_predict = classifier_knn.predict(X_train_predict)
print(Y_pred_predict)
```

```
[ 55.  81. 118. 125.  27. 195. 428.  68.  77. 335.  68. 773.  51. 165.
141. 141.  34. 469. 201. 201. 201. 160. 201. 904. 233. 449. 206. 206.
271. 206.  88. 411. 760. 449. 220.  66. 114. 579. 148. 241. 258. 282.
  38. 855. 829. 177. 154. 466. 159. 291. 302. 124.  90.  95. 434.  96.
434.  90. 320. 105. 332. 169. 105. 939. 600. 320.  79.  70. 512.  85.
  85.  85.  70. 280. 273. 604. 699.  76. 738. 447. 386. 135.   9. 703.
213. 738. 403. 703.  76. 213. 895. 703. 893. 896. 213. 493. 537. 661.
533. 433. 451. 800. 610. 661. 610. 369. 229. 316. 610. 610. 451. 178.
451. 276. 618. 492. 518. 492. 559.  39. 196. 870. 781. 729. 421. 729.
577. 578. 453. 845. 372. 578. 845. 170. 601. 388. 221. 767. 811. 221.
221. 677. 221. 221.  18. 627. 627. 324. 470. 221. 530. 374. 742. 597.
991. 664. 917. 764. 917. 409. 527. 664. 341. 341. 857. 409. 991. 936.
970. 561. 427. 167. 454. 705. 782. 244. 176. 705. 945. 921. 435. 401.
945. 529. 529. 622. 435. 343.  15. 945. 467. 435. 754. 922. 501. 485.
543. 200. 391. 626. 754. 111. 616. 558. 567. 108. 616. 616. 150. 804.
803. 804. 345. 424. 768. 725. 768. 598. 905. 688. 164. 565. 688. 286.
688. 380. 565. 565.   6. 588. 430. 285. 265. 309. 625. 517. 326. 711.
371. 425. 684. 240. 675. 658. 994.  30. 214. 346. 289. 880. 879. 879.
287. 392. 879. 719. 880. 880. 423. 879. 834. 834. 370. 376. 205. 127.
834. 674. 650. 415. 415. 387.  35. 216. 694. 838. 216. 691. 873. 494.
  80.  25. 828.  25. 318. 957. 317. 759. 810. 586. 759. 581. 949. 156.
778. 523. 788. 772. 553. 582. 526. 969. 872.]
```

#Checking accuracy

```
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
print('Accuracy :',accuracy_score(Y_train_predict,Y_pred_predict))
```

```
r2 = met.r2_score(Y_train_predict,Y_pred_predict)
print('R-square_score : ',r2)
mse = met.mean_squared_error(Y_train_predict,Y_pred_predict)
print('MSE : ',mse)
rmse = np.sqrt(mse)
print('RMSE : ',rmse)
```

```
Accuracy : 0.7755775577557755
R-square_score :  0.674491469927109
MSE :  24725.102310231025
RMSE:  157.24217726243498
```

#Plotting predicted Rank with respect to original Rating of User input Genre

```
plt.scatter(x_test[:,0],Y_train_predict,color='magenta') #original rank
plt.scatter(x_test[:,0],Y_pred_predict,color='blue') #predicted rank
plt.xlabel('Rating')
plt.ylabel('Movie Title Ranks')
plt.show()
```

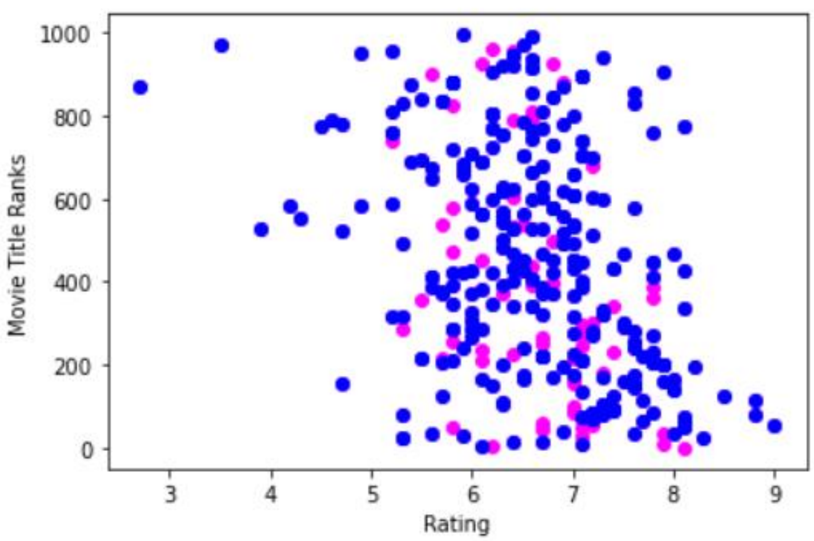

Figure 6- Scatter plot for predicting data (KNN)

Recommendation using Nearest Neighbor

```
from sklearn.neighbors import NearestNeighbors
nbrs =
NearestNeighbors(n_neighbors=6,metric='cosine',algorithm='brute').fit(X)

distances , indices = nbrs.kneighbors(X,n_neighbors=100)
data = list()
```

#Function to get index of input Movie Title

```
def get_index(title):
    return dataset[dataset["Title"]==title].index.values.astype(int)[0]
#Function to print Similar movies based on distances
def print_similar_movies(query=None):
   global data
   if query:
      found_id = get_index(query)
      for id in indices[found_id][0:]:
            if x.loc[id][x_input] == 1:
                  if dataset.loc[id]["Rating"] > 6.5:
                        list3=[]
                        list3.append(dataset.loc[id]["Title"])
                        list3.append(dataset.loc[id]["Genre"])
                        list3.append(dataset.loc[id]["Rating"])
                        list3.append(x.loc[id][x_input])
                        data.append(list3)
                        list3 = None

rank = Ranks[0]
movie_input = np.array(dataset[dataset['Rank']==rank].Title)
movie = movie_input[0]


title = np.array(dataset[dataset['Rank']==rank].index)
print(title)

print_similar_movies(movie)
```

#Recommending movies using KNN

```
from columnar import columnar
print('Your Input genre is ',x_input)
print('Recommending best movies of ',x_input,' Genre : \n\n')
headers = ['Title', 'Genres', 'Rating', 'Is {}'.format(x_input)]
data = sorted(data,key=lambda lr:lr[2], reverse=True)
table = columnar(data, headers, no_borders=False)
print(table)
```

```
Your Input genre is  Action
Recommending best movies of  Action  Genre :
```

Title	Genres	Rating	Is Action
The Dark Knight	Action,Crime,Drama	9.0	1.0
Inception	Action,Adventure,Sci-Fi	8.8	1.0
Bahubali: The Beginning	Action,Adventure,Drama	8.3	1.0
The Avengers	Action,Sci-Fi	8.1	1.0
Guardians of the Galaxy	Action,Adventure,Sci-Fi	8.1	1.0
Avatar	Action,Adventure,Fantasy	7.8	1.0
Kingsman: The Secret Service	Action,Adventure,Comedy	7.7	1.0
300	Action,Fantasy,War	7.7	1.0
Kick-Ass	Action,Comedy	7.7	1.0
Watchmen	Action,Drama,Mystery	7.6	1.0
Sherlock Holmes	Action,Adventure,Crime	7.6	1.0
Avengers: Age of Ultron	Action,Adventure,Sci-Fi	7.4	1.0
Pirates of the Caribbean: Dead Man's Chest	Action,Adventure,Fantasy	7.3	1.0
The Man from U.N.C.L.E.	Action,Adventure,Comedy	7.3	1.0
The Lost City of Z	Action,Adventure,Biography	7.1	1.0
Thor: The Dark World	Action,Adventure,Fantasy	7.0	1.0
Oblivion	Action,Adventure,Mystery	7.0	1.0
Pirates of the Caribbean: On Stranger Tides	Action,Adventure,Fantasy	6.7	1.0
Batman v Superman: Dawn of Justice	Action,Adventure,Sci-Fi	6.7	1.0
Jason Bourne	Action,Thriller	6.7	1.0
The Wolverine	Action,Adventure,Sci-Fi	6.7	1.0
X-Men Origins: Wolverine	Action,Adventure,Sci-Fi	6.7	1.0

Figure 7 Output for KNN

Representation of the code in Tkinter GUI

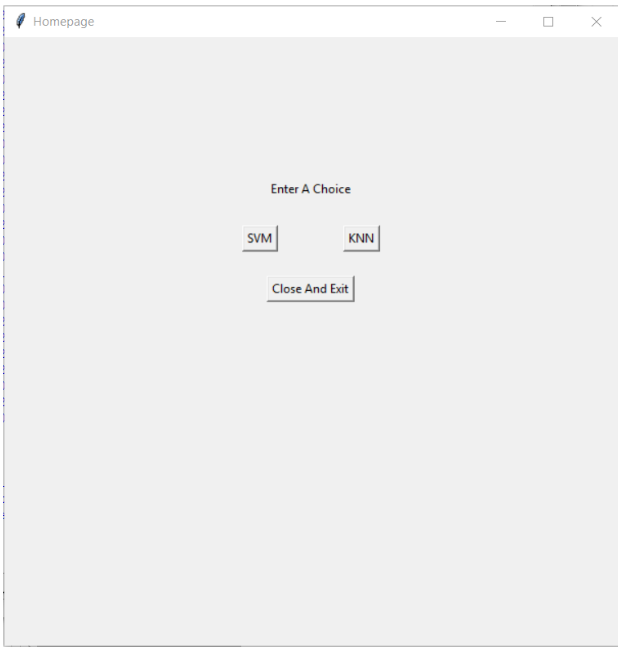

Figure 8 Homepage GUI

Here we have given the user a choice to choose which algorithm he/she prefers

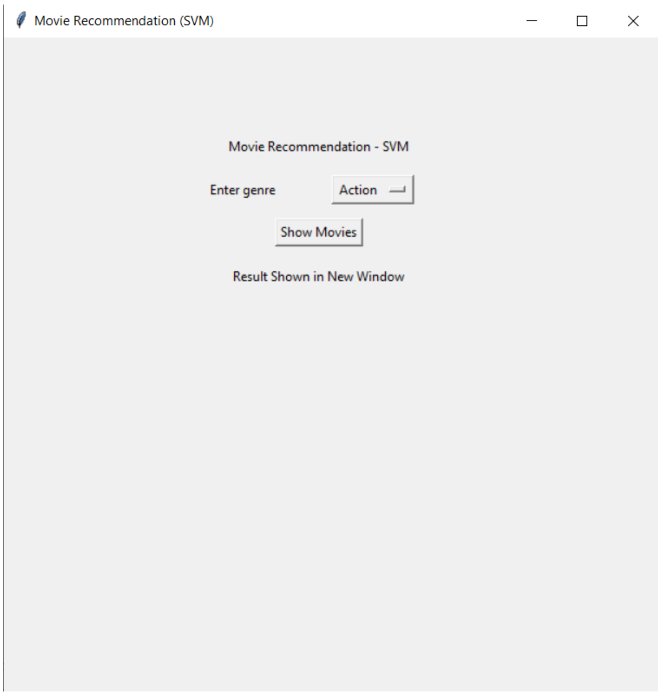

Figure 9 SVM menu

the user can select the list of genres from the dropdown list. On clicking the show movies button, the Result is shown in a new window.

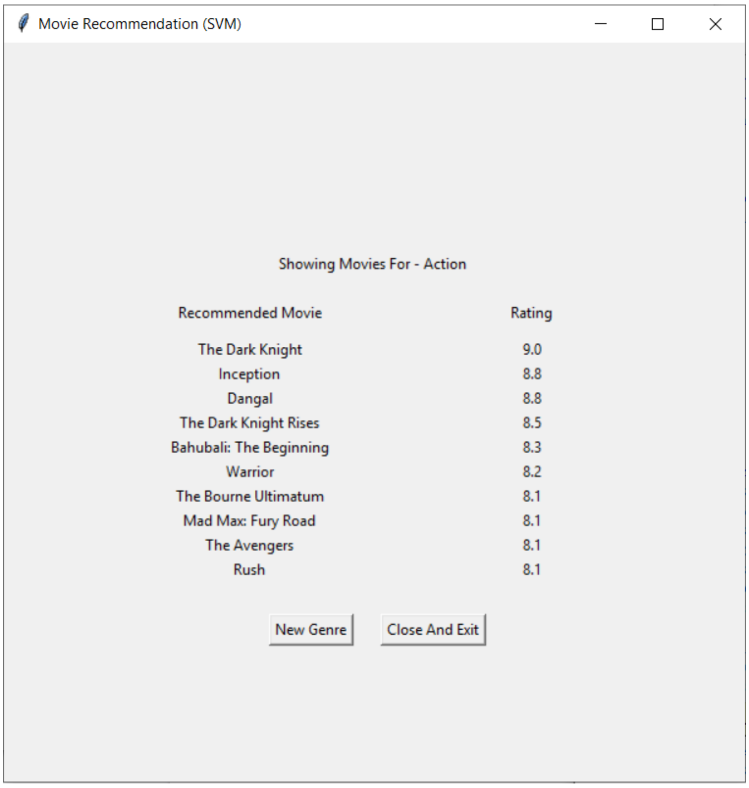

Figure 10 SVM output

On this window, we have given the user an option to select a new genre or to go back to the main menu.

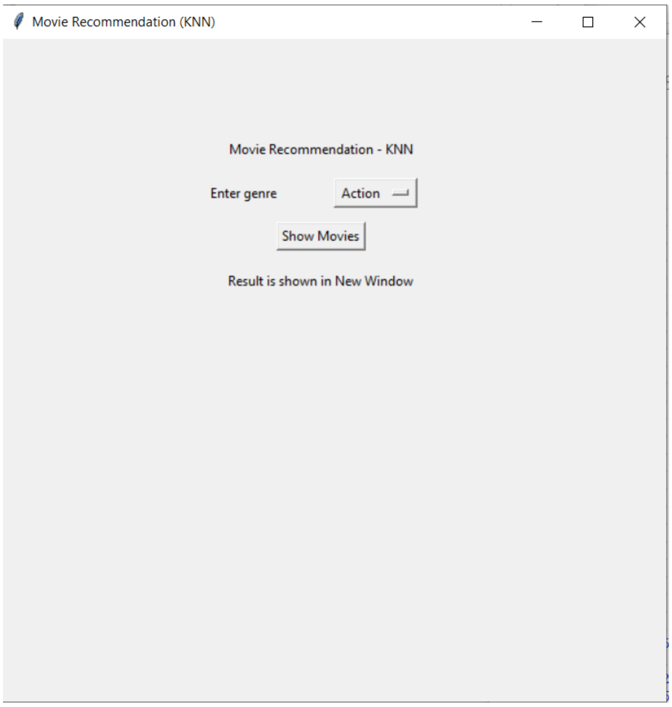

Figure 11-KNN menu

Here the Result is shown by executing the KNN algorithm.

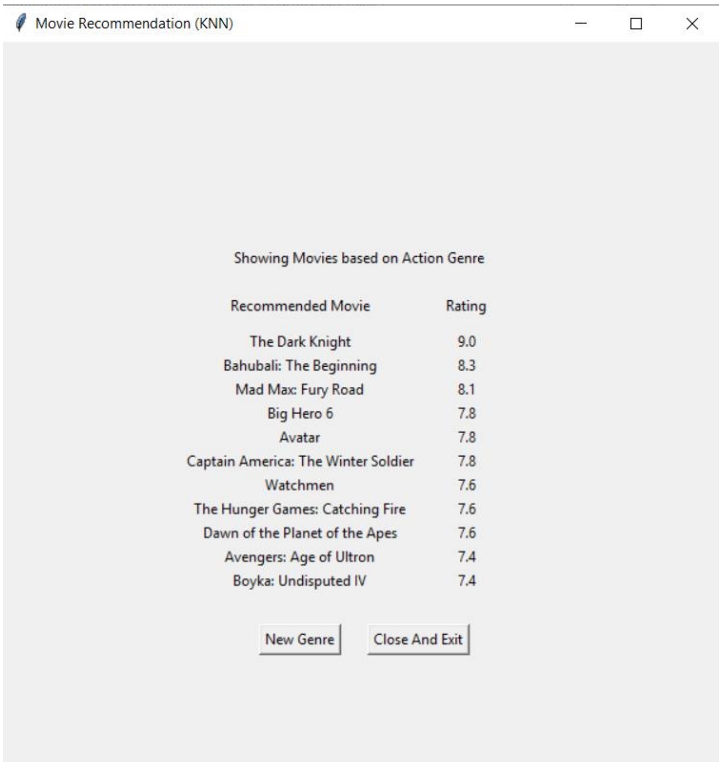

Figure 12- KNN output

Result window shows the Movies and the respective Rating of the movie.

REFERENCES

1. https://www.tutorialspoint.com/python/index.htm
2. https://www.edureka.co/blog/python-libraries/
3. https://hackernoon.com/top-10-libraries-in-python-to-implement-machine-learning-12602cf5dc61
4. https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-jupyter.html
5. https://docs.python.org/2.4/lib/module-Tkinter.html
6. https://www.geeksforgeeks.org/python-tkinter-spinbox/
7. k-means clustering - Wikipedia. https://en.wikipedia.org/wiki/K-means_algorithm
8. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47
9. Pros and Cons of K-Nearest Neighbors - From The GENESIS. https://www.fromthegenesis.com/pros-and-cons-of-k-nearest-neighbors/
10. K-Nearest Neighbors (KNN) Algorithm for Machine Learning. https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26
11. https://www.datavedas.com/knn-distance-metrics/
12. https://towardsdatascience.com/knn-visualization-in-just-13-lines-of-code-32820d72c6b6
13. https://towardsdatascience.com/12-amazing-pandas-numpy-functions-22e5671a45b8
14. https://medium.com/research-studies-by-alvaro-bartolome/investpy-a-python-library-for-historical-data-extraction-from-the-spanish-stock-market-ad4d564dbfc5
15. https://medium.com/@pranav3nov/data-visualization-in-python-c49fda103218
16. Columnar · PyPI. https://pypi.org/project/Columnar/
17. https://medium.com/capital-one-tech/k-nearest-neighbors-knn-algorithm-for-machine-learning-e883219c8f26