Movie Popularity Prediction

CS_2

Team Members

Name	ID	Department
أحمد حسن حسن خطاب	*20201700033*	Computer Science(CS).
انس سعيد صبري	*20201700159*	Computer Science(CS).
عمر رضا عبدالدايم عبدالعزيز	*20201701267*	Computer Science(CS).
عبدالرحمن هاني خيري الدين	*20201700488*	Computer Science(CS).
أحمد السيد محمود الباز	*20191700863*	Computer Science(CS).
أحمد هاني حامد محمد	*20191700875*	Computer Science(CS).

1. Data Splitting

Using **train_test_split()** to split the data with the following parameters:

- shuffle = True
- test_size = 0.2
- random_state = 10

X_train [2431 rows x 19 columns]

Index	udge	genres	omepag	id	keywords	inal langu	riginal tit	overview	iewercoun
0	250…	[{"id"…	http:/…	33870	[{"id": 4…	en	Mao's …	At the…	1.87681
1	380…	[{"id"…	nan	193	[{"id": 1…	en	Star T…	Captai…	14.779
2	200…	[{"id"…	http:/…	10139	[{"id": 2…	en	Milk	The st…	30.9097
3	230…	[{"id"…	nan	11632	[{"id": 2…	en	Vanity Fair	Beauti…	6.61815
4	520…	[{"id"…	http:/…	26389	[{"id": 9…	en	From P…	James …	27.9163
5	280…	[{"id"…	http:/…	277216	[{"id": 3…	en	Straig…	In 198…	61.7623
6	260…	[{"id"…	nan	14181	[{"id": 6…	en	Boiler Room	A coll…	11.2331
7	0	[{"id"…	nan	10413	[{"id": 1…	en	Nowhere to Run	Escape…	11.6893
9	120…	[{"id"…	http:/…	101267	[{"id": 1…	en	Katy P…	Giving…	8.41069

y_train [2431 rows]

Index	te avera
0	6.8
1	6.4
2	7.1
3	5.5
4	6.1
5	7.7
6	6.5
7	5.5
9	6.5

X_test [608 rows x 19 columns]

Index	budget	genres	omepage	id	eyword	inal langu	riginal titl	overview	iewercoun
8	4000000	[{"id…	nan	2370	[{"id…	en	Topaz	A Fre…	5.9756
16	50000000	[{"id…	http://…	3298…	[{"id…	en	Zooland…	Derek…	37.2538
19	38000000	[{"id…	http://…	44264	[{"id…	en	True Grit	Follo…	49.2924
20	15000000	[{"id…	nan	586	[{"id…	en	Wag the Dog	Durin…	13.582
25	0	[{"id…	nan	70670	[{"id…	en	Hodejeg…	An ac…	20.8218
27	78000000	[{"id…	nan	9533	[{"id…	en	Red Dragon	Forme…	10.0839
28	90000000	[{"id…	nan	18	[{"id…	en	The Fif…	In 22…	109.529
29	0	[{"id…	nan	16441	[{"id…	en	The Bea…	Dar, …	6.37752
37	70000000	[{"id…	http://…	59981	[]	en	Legends…	Dorot…	6.68201

y_test [608 rows]

Index	te avera
8	6.1
16	4.7
19	7.2
20	6.7
25	7.1
27	6.7
28	7.3
29	6
37	5.9

2. Preprocessing

2.1. Handling Missing Values

Using print(Movie_Data.isna().sum())

Columns contains missing values:

Training data	
column	Null count
homepage	1530
overview	1
runtime	1
Tagline	299

Testing data	
column	Null count
homepage	380
Tagline	84

These nulls were handled as the following:

Training data:

- homepage →

  by replacing all null values by

  "http://www.+[original title]+.com/"

- overview → by dropping the row containing this null value.
- runtime → by replacing the missing value with the mean of the column.
- tagline → by dropping the rows containing this null value.

Testing data:

Same as training data.

Note : if 'runtime' columns has null values in testing data they will be replaced with the mean of 'runtime' column in the training data not testing data.

2.2. Handling release-date column

By converting the column to datetime format [using **pd.to_datetime**], then splitting this columns to three columns that are release_day, release_year, release_month, then drop the release_date column in both training & testing .

Index	release_date
0	10/1/2009
1	11/17/1994
2	11/26/2008
3	9/1/2004
4	2/5/2010
5	8/13/2015
6	2/18/2000
7	1/15/1993
8	12/18/1969

Index	release_day	release_month	release_year
0	1	10	2009
1	17	11	1994
2	26	11	2008
3	1	9	2004
4	5	2	2010
5	13	8	2015
6	18	2	2000
7	15	1	1993
8	18	12	1969

2.3. Handling 'overview' column [Text Preprocessing]

By using **TextBlob** sentiment polarity analysis in both training & testing data to get either overview is 'Positive' , 'Negative' or 'Natural' ,as the following:

Polarity	Rank	Rank encoded	Rank scaled
> 0.5	Positive	**2**	**1**
< 0.5	Negative	**1**	**0.5**
= 0.5	Natural	**0**	**0**

Index	overview
0	At the age of 11, Li was plucked from a poo…
1	Captain Jean-Luc Picard and the crew of the…
2	The story of California's first openly gay …
3	Beautiful, funny, passionate, and calculati…
4	James Reese has a good job as an ambassador…
5	In 1987, five young men, using brutally hon…
6	A college dropout gets a job as a broker fo…
7	Escaped convict Sam Gillen single handedly …

Index	overview
0	1
1	0.5
2	0
3	1
4	1
5	0.5
6	0.5
7	0.5

2.4. Label Encoding

By using ***LabelEncoder()*** on categorial columns which are:

('status', 'original_language', 'original_title', 'tagline', 'homepage', 'title') + 'Rate' in Classification Dataset.

[fit-transform on training data ,transform only on testing data]

Note: any unseen labels in testing data while transforming will be assigned to label <others>.

Encoding example: 'original_language' column

Index	original_language
18	en
19	en
20	en
21	en
22	ja
23	en
24	en
25	en
26	en

Index	original_language
18	4
19	4
20	4
21	4
22	11
23	4
24	4
25	4
26	4

another example: 'status' column

Index	status
1430	Released
1431	Released
1432	Released
1433	Released
1434	Released
1435	Released
1436	Released
1437	Post Production
1438	Released

Index	status
1430	1
1431	1
1432	1
1433	1
1434	1
1435	1
1436	1
1437	0
1438	1

2.5. handling with List of dictionaries columns

By using **MultiLabelBinarizer()** on 'list of dictionaries' columns which are:

(' genres', ' spoken_languages', ' production_countries', ' production_companies', ' keywords')

Then removing the columns that contains one counts less than ¼ the training size in both train and test data.

For example: ' genres ' column

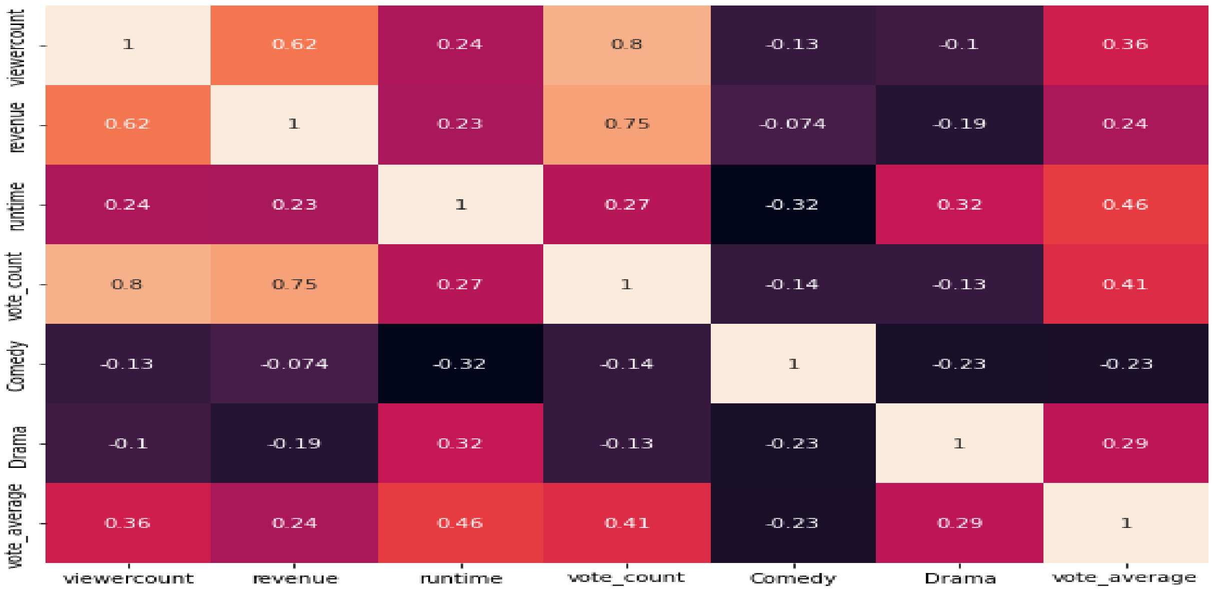
Index	genres
0	[{"id": 35, "name": "Comedy"}, {"id": 10749, "name": "Romance"}, {"id": 18, "name": "Drama"}]
1	[{"id": 18, "name": "Drama"}, {"id": 10749, "name": "Romance"}]
2	[{"id": 35, "name": "Comedy"}, {"id": 10749, "name": "Romance"}]
3	[{"id": 35, "name": "Comedy"}]
4	[{"id": 35, "name": "Comedy"}, {"id": 14, "name": "Fantasy"}]
5	[{"id": 28, "name": "Action"}, {"id": 80, "name": "Crime"}, {"id": 18, "name": "Drama"}, {"id": 53, "name": "Thriller"}]
6	[{"id": 35, "name": "Comedy"}, {"id": 10751, "name": "Family"}, {…
7	[{"id": 27, "name": "Horror"}, {"id": 18, "name": "Drama"}, {"id": 53, "name": "Thriller"}, {"id": 878, "name": "Science Fiction"}]
8	[{"id": 53, "name": "Thriller"}, {"id": 28, "name": "Action"}, {"id": 80, "name": "Crime"}]

Index	Action	Comedy	Drama	Thriller
0	0	1	1	0
1	0	0	1	0
2	0	1	0	0
3	0	1	0	0
4	0	1	0	0
5	1	0	1	1
6	0	1	0	0
7	0	0	1	1
8	1	0	0	1

2.6.a Features Selection (Regression Phase)

For training data:

By using corr() for all features according to the label column 'vote_average' , the selected features will be the features that has **corr > 0.15** ,which are:



Index	viewercount	revenue	runtime	vote_count	Comedy	Drama	vote_average
0	0.0218388	0.00405086	0.313609	0.0370128	1	1	7.1
1	0.0362011	0.00373368	0.286982	0.0594095	0	1	7.1
2	0.0256818	0.0467837	0.319527	0.0295957	1	0	5.4
3	0.0323408	0.0238412	0.263314	0.0313409	1	0	5.5
4	0.00359684	0.00251079	0.245562	0.00145433	1	0	4.1
5	0.0257737	0.0235189	0.322485	0.0263962	0	1	6.3
6	0.00149321	0.00620251	0.254438	0.00239965	1	0	5.4
7	0.00733843	0	0.369822	0.00734439	0	1	6.6
8	0.110351	0.0689861	0.390533	0.214805	0	0	7.1

Number of Selected features : 6

For testing data:

Testing data will be assigned to only the testing data that contains top features selected by corr of training data so it will have the same top features.

2.6.b Features Selection (Classification Phase)

For training data:

By using corr() for all features according to the label column 'vote_average' , the selected features will be the features that has **corr > 0.04** ,which are:

	budget	id	original_language	viewercount	revenue	runtime	status	vote_count	release_month	release_year	Action	Comedy	Drama	Thriller	English	United States of America	Rate
budget	1	-0.017	-0.061	0.47	0.69	0.23	0.02	0.54	0.056	0.21	0.3	-0.094	-0.21	0.0072	0.063	0.19	-0.05
id	-0.017	1	0.016	0.15	0.023	-0.042	-0.093	0.081	-0.019	0.47	0.013	-0.051	0.00057	0.022	0.012	-0.07	0.045
original_language	-0.061	0.016	1	-0.037	-0.042	0.048	0.062	-0.046	0.0056	0.041	0.05	-0.055	0.058	0.014	-0.32	-0.3	-0.04
viewercount	0.47	0.15	-0.037	1	0.62	0.24	0.018	0.8	0.051	0.16	0.15	-0.13	-0.1	0.05	0.035	0.11	-0.29
revenue	0.69	0.023	-0.042	0.62	1	0.23	0.013	0.75	0.06	0.12	0.19	-0.074	-0.19	-0.042	0.029	0.14	-0.2
runtime	0.23	-0.042	0.048	0.24	0.23	1	0.022	0.27	0.17	-0.2	0.054	-0.32	0.32	0.019	-0.0027	0.0078	-0.4
status	0.02	-0.093	0.062	0.018	0.013	0.022	1	0.014	-0.019	-0.026	0.013	0.017	-0.024	0.014	-0.0033	-0.0076	-0.046
vote_count	0.54	0.081	-0.046	0.8	0.75	0.27	0.014	1	0.037	0.15	0.2	-0.14	-0.13	0.025	0.041	0.12	-0.34
release_month	0.056	-0.019	0.0056	0.051	0.06	0.17	-0.019	0.037	1	-0.073	-0.041	-0.066	0.088	-0.038	0.04	0.027	-0.13
release_year	0.21	0.47	0.041	0.16	0.12	-0.2	-0.026	0.15	-0.073	1	0.0014	0.049	-0.046	0.026	-0.008	-0.056	0.12
Action	0.3	0.013	0.05	0.15	0.19	0.054	0.013	0.2	-0.041	0.0014	1	-0.19	-0.25	0.27	0.0061	0.022	0.097
Comedy	-0.094	-0.051	-0.055	-0.13	-0.074	-0.32	0.017	-0.14	-0.066	0.049	-0.19	1	-0.23	-0.39	-0.015	0.035	0.21
Drama	-0.21	0.00057	0.058	-0.1	-0.19	0.32	-0.024	-0.13	0.088	-0.046	-0.25	-0.23	1	-0.018	-0.038	-0.075	-0.27
Thriller	0.0072	0.022	0.014	0.05	-0.042	0.019	0.014	0.025	-0.038	0.026	0.27	-0.39	-0.018	1	0.047	0.0079	0.052
English	0.063	0.012	-0.32	0.035	0.029	-0.0027	-0.0033	0.041	0.04	-0.008	0.0061	-0.015	-0.038	0.047	1	0.31	0.072
United States of America	0.19	-0.07	-0.3	0.11	0.14	0.0078	0.0076	0.12	0.027	-0.056	0.022	0.035	-0.075	0.0079	0.31	1	0.053
Rate	-0.05	0.045	-0.04	-0.29	-0.2	-0.4	-0.046	-0.34	-0.13	0.12	0.097	0.21	-0.27	0.052	0.072	0.053	1

Index	budget	id	original_language	viewercount	revenue	runtime	status	vote_count	release_month	release_year	Action	Comedy	Drama	Thriller	English	United States of America	Rate
0	0.04…	0.0…	0.25	0.0218388	0.004…	0.313…	1	0.0370128	0.818182	0.898876	0	1	1	0	1	1	0
1	0.02…	0.0…	0.25	0.0362011	0.003…	0.286…	1	0.0594095	0.272727	0.808989	0	0	1	0	1	1	0
2	0.19…	0.0…	0.25	0.0256818	0.046…	0.319…	1	0.0295957	0.545455	0.88764	0	1	0	0	1	1	1
3	0.05…	0.0…	0.25	0.0323408	0.023…	0.263…	1	0.0313409	1	0.831461	0	1	0	0	1	1	1
4	0.07…	0.0…	0.25	0.00359684	0.002…	0.245…	1	0.001454…	0.818182	0.910112	0	1	0	0	1	1	2
5	0.07…	0.0…	0.25	0.0257737	0.023…	0.322…	1	0.0263962	0.272727	0.910112	1	0	1	1	1	1	1
6	0.04…	0.0…	0.25	0.00149321	0.006…	0.254…	1	0.002399…	0.818182	0.831461	0	1	0	0	1	1	1
7	0	0.0…	0.25	0.00733843	0	0.369…	1	0.007344…	0.818182	0.573034	0	0	1	1	1	1	1
8	0.19…	0.3…	0.25	0.110351	0.068…	0.390…	1	0.214805	0.727273	0.977528	1	0	0	1	1	1	0

Number of Selected features : 16

For testing data:

Testing data will be assigned to only the testing data that contains top features selected by corr of training data so it will have the same top features.

2.7. Features Normalization

By using **MinMaxScaler()** on all columns in the data that already exist and the features that had been extracted.

[fit-transform on training data ,transform only on testing data]

Index	budget	id	inal langu	riginal tit	overview	iewercoun	revenue
0	12000000	6615	4	868	2	15.8171	11293663
1	6000000	1443	4	1890	1	26.2189	10409377
2	54000000	1819	4	2098	0	18.6004	130431368
3	16000000	11397	4	1050	2	23.4231	66468332
4	20000000	13948	4	122	2	2.60537	7000000
5	20000000	1266	4	1430	1	18.6669	65569869
6	12000000	34549	4	956	1	1.08182	17292381
7	0	16241	4	1528	1	5.31521	0
8	55000000	156022	4	1587	1	79.9221	192330738

Index	budget	id	inal langu	original title	overview	viewercount	revenue
0	0.0428571	0.0158048	0.25	0.407321	1	0.0218388	0.00405086
1	0.0214286	0.00342708	0.25	0.886908	0.5	0.0362011	0.00373368
2	0.192857	0.00432693	0.25	0.984514	0	0.0256818	0.0467837
3	0.0571429	0.0272491	0.25	0.492726	1	0.0323408	0.0238412
4	0.0714286	0.0333542	0.25	0.0572501	1	0.00359684	0.00251079
5	0.0714286	0.00300348	0.25	0.671046	0.5	0.0257737	0.0235189
6	0.0428571	0.0826569	0.25	0.448616	0.5	0.00149321	0.00620251
7	0	0.0388419	0.25	0.717034	0.5	0.00733843	0
8	0.196429	0.373368	0.25	0.744721	0.5	0.110351	0.0689861

3. Regression Models

3.1. Linear Regression

From scratch linear regression function with GD algorithm to minimize the error:

- #epochs = 100000
- α = 0.663 “Learning Rate”

resulted coefficients:-

θ	value
θ0	4.469857
θ1	1.216461
θ2	-1.784756
θ3	4.195088
θ4	3.638630
θ5	-0.055620
θ6	0.404850

Evaluation:

MSE	R2 Score
0.47160445037702553	0.28879882778053434

3.2. Polynomial Regression

Using built-in modules LinearRegression() , PolynomialFeatures() with:

- Degree = 2

Evaluation:

MSE	R2 Score
0.5338504005318015	0.1949290759565221

3.3. Lasso Regression

Using built-in module Lasso() with:

- Alpha = 0.001

Evaluation:

MSE	R2 Score
0.47357818006233854	0.285822352760596

3.4. Ridge Regression

Using built-in module Ridge() with:

- Alpha = 1.0

Evaluation:

MSE	R2 Score
0.4745641855460691	0.2843354112033579

4. Classification Models

4.1. Logistic Regression

Using built-in model LogisticRegression() with:

- C = 9
- penalty = ‘l1’
- solver = ‘saga’

Evaluation:

Accuracy Score
0.7461832061068703

	Confusion Matrix		
True Label	85	76	0
	22	304	1
	0	34	2
	Predicted Label		

4.2. Support Vector Machine

Using built-in model SVC() with:

- C = 100
- kernel = ‘rbf’
- gamma = ‘auto’
- max_iter = 2400

Evaluation:

Accuracy Score
0.7595419847328244

	Confusion Matrix		
True Label	90	71	0
	19	308	0
	0	36	0
	Predicted Label		

4.3. Decision Tree

Using built-in model DecisionTreeClassifier() with:

- max_depth = 10
- max_leaf_nodes = 25

Evaluation:

Accuracy Score
0.7213740458015268

	Confusion Matrix		
True Label	93	68	0
	42	279	6
	0	30	6
	Predicted Label		

4.4. K Nearest Neighbors

Using built-in model KNeighborsClassifier() with:

- n_neighbors = 9

Evaluation:

Accuracy Score
0.6965648854961832

	Confusion Matrix		
True Label	65	96	0
	28	299	0
	4	31	1
	Predicted Label		

5. Tuning the hyperparameters for Classifiers

5.1. Logistic Regression

Notice the change of **C** parameter in Logistic Regression:

1- lr = LogisticRegression(C=1,penalty='l1',solver='saga')

```
Accuracy : 0.7385496183206107
```

2- lr = LogisticRegression(C=10,penalty='l1',solver='saga')

```
Accuracy : 0.7461832061068703
```

3- lr = LogisticRegression(C=20,penalty='l1',solver='saga')

```
Accuracy : 0.7442748091603053
```

With these accuracies we find that C=10 is the best value to get higher accuracy.

Notice the change of **penalty** parameter in Logistic Regression:

1- lr = LogisticRegression(C=1,penalty='l1',solver='saga')

```
Accuracy : 0.7461832061068703
```

2- lr = LogisticRegression(C=10,penalty='l2',solver='saga')

```
Accuracy : 0.7442748091603053
```

3- lr = LogisticRegression(C=20,penalty='none',solver='saga')

```
Accuracy : 0.7442748091603053
```

With these accuracies we find that `penalty='l1'` is the best value to get higher accuracy.

5.2. Support Vector Machine

Notice the change of **C** parameter in SVM:

1- svm = SVC(C=1,kernel='rbf',gamma='auto',max_iter=2400)

```
Accuracy : 0.6927480916030534
```

2- svm = SVC(C=10,kernel='rbf',gamma='auto',max_iter=2400)

```
Accuracy : 0.7290076335877863
```

3- svm = SVC(C=100,kernel='rbf',gamma='auto',max_iter=2400)

```
Accuracy : 0.7595419847328244
```

With these accuracies we find that C=100 is the best value to get higher accuracy.

Notice the change of **kernel** parameter in SVM:

1- svm = SVC(C=100,kernel='linear',gamma='auto',max_iter=2400)

```
Accuracy : 0.6698473282442748
```

2- svm = SVC(C=100,kernel='rbf',gamma='auto',max_iter=2400)

```
Accuracy : 0.7595419847328244
```

3- svm = SVC(C=100,kernel='poly',gamma='auto',max_iter=2400)

```
Accuracy : 0.7423664122137404
```

With these accuracies we find that kernel='rbf' is the best value to get higher accuracy.

5.3. Decision Tree

Notice the change of **max_Depth** parameter in Decision Tree:

1- clf = DecisionTreeClassifier(max_depth = 4,max_leaf_nodes= 25)

```
Accuracy : 0.6851145038167938
```

2- clf = DecisionTreeClassifier(max_depth = 5,max_leaf_nodes= 25)

```
Accuracy : 0.6774809160305344
```

3- clf = DecisionTreeClassifier(max_depth = 6,max_leaf_nodes= 25)

```
Accuracy : 0.7041984732824428
```

4- clf = DecisionTreeClassifier(max_depth = 7,max_leaf_nodes= 25)

```
Accuracy : 0.7213740458015268
```

With these accuracies we find that max_depth=7 is the best value to get higher accuracy.

Notice the change of **max_Leaf_Nodes** parameter in Decision Tree:

1- clf = DecisionTreeClassifier(max_depth = 6,max_leaf_nodes= 15)

```
Accuracy : 0.6755725190839694
```

2- clf = DecisionTreeClassifier(max_depth = 6,max_leaf_nodes= 20)

```
Accuracy : 0.6927480916030534
```

3- clf = DecisionTreeClassifier(max_depth = 6,max_leaf_nodes= 25)

```
Accuracy : 0.7213740458015268
```

4- clf = DecisionTreeClassifier(max_depth = 6,max_leaf_nodes= 30)

```
Accuracy : 0.7118320610687023
```

With these accuracies we find that max_leaf_nodes=25 is the best value to get higher accuracy.

5.4. K Nearest Neighbors

Notice the change of **weights** parameter in KNN:

1- knn=KNeighborsClassifier(n_neighbors=9, weights='distance')

```
Accuracy : 0.683206106870229
```

2- knn=KNeighborsClassifier(n_neighbors=9, weights='uniform')

```
Accuracy : 0.6965648854961832
```

3- knn=KNeighborsClassifier(n_neighbors=9, weights= lambda x : x)

```
Accuracy : 0.6889312977099237
```

With these accuracies we find that weights='uniform' is the best value to get higher accuracy.

Notice the change of **n_neighbors** parameter in KNN:

1- knn=KNeighborsClassifier(n_neighbors=8, weights='uniform')

```
Accuracy : 0.6851145038167938
```

2- knn=KNeighborsClassifier(n_neighbors=9, weights='uniform')

```
Accuracy : 0.6965648854961832
```

3- knn=KNeighborsClassifier(n_neighbors=10, weights='uniform')

```
Accuracy : 0.6908396946564885
```

4- knn=KNeighborsClassifier(n_neighbors=11, weights='uniform')

```
Accuracy : 0.6851145038167938
```

With these accuracies we find that n_neighbors=9 is the best value to get higher accuracy.