SVM Classification

In [2]:

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

In [3]:

```
df = pd.read_csv("Movie_classification.csv", header=0)
```

In [5]:

```
df.head()
```

Out[5]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
Marketing expense      506 non-null float64
Production expense     506 non-null float64
Multiplex coverage     506 non-null float64
Budget                 506 non-null float64
Movie_length           506 non-null float64
Lead_ Actor_Rating     506 non-null float64
Lead_Actress_rating    506 non-null float64
Director_rating        506 non-null float64
Producer_rating        506 non-null float64
Critic_rating          506 non-null float64
Trailer_views          506 non-null int64
3D_available           506 non-null object
Time_taken             494 non-null float64
Twitter_hastags        506 non-null float64
Genre                  506 non-null object
Avg_age_actors         506 non-null int64
Num_multiplex          506 non-null int64
Collection             506 non-null int64
Start_Tech_Oscar       506 non-null int64
dtypes: float64(12), int64(5), object(2)
memory usage: 75.2+ KB
```

In [7]:

```
df.describe()
```

Out[7]:

	Marketing	Production	Multiplex	Budget	Movie_length	Lead	Lead_Actress_rating	Director_rating	Producer

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.(
mean	92.270471	77.273557	0.445305	34911.144022	142.074901	8.014002	8.185613	8.019664	8.1
std	172.030902	13.720706	0.115878	3903.038232	28.148861	1.054266	1.054290	1.059899	1.(
min	20.126400	55.920000	0.129000	19781.355000	76.400000	3.840000	4.035000	3.840000	4.(
25%	21.640900	65.380000	0.376000	32693.952500	118.525000	7.316250	7.503750	7.296250	7.5
50%	25.130200	74.380000	0.462000	34488.217500	151.000000	8.307500	8.495000	8.312500	8.4
75%	93.541650	91.200000	0.551000	36793.542500	167.575000	8.865000	9.030000	8.883750	9.(
max	1799.524000	110.480000	0.615000	48772.900000	173.500000	9.435000	9.540000	9.425000	9.6

Missing Value Imputation

In [8]:

```
df['Time_taken'].mean()
```

Out[8]:

```
157.39149797570855
```

In [9]:

```
df['Time_taken'].fillna(value = df['Time_taken'].mean(), inplace = True)
```

In [10]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 19 columns):
Marketing expense      506 non-null float64
Production expense     506 non-null float64
Multiplex coverage     506 non-null float64
Budget                 506 non-null float64
Movie_length           506 non-null float64
Lead_ Actor_Rating     506 non-null float64
Lead_Actress_rating    506 non-null float64
Director_rating        506 non-null float64
Producer_rating        506 non-null float64
Critic_rating          506 non-null float64
Trailer_views          506 non-null int64
3D_available           506 non-null object
Time_taken             506 non-null float64
Twitter_hastags        506 non-null float64
Genre                  506 non-null object
Avg_age_actors         506 non-null int64
Num_multiplex          506 non-null int64
Collection             506 non-null int64
Start_Tech_Oscar       506 non-null int64
dtypes: float64(12), int64(5), object(2)
memory usage: 75.2+ KB
```

Dummy Variable Creation

In [11]:

```
df.head()
```

Out[11]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [12]:

```
df = pd.get_dummies(df,columns = ["3D_available","Genre"],drop_first = True)
```

In [13]:

```
df.head()
```

Out[13]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

5 rows × 21 columns

X-y split

In [14]:

```
X = df.loc[:,df.columns!="Start_Tech_Oscar"]
type(X)
```

Out[14]:

```
pandas.core.frame.DataFrame
```

In [15]:

```
X.head()
```

Out[15]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	Crit
0	20.1264	59.62	0.462	36524.125	138.7	7.825	8.095	7.910	7.995	
1	20.5462	69.14	0.531	35668.655	152.4	7.505	7.650	7.440	7.470	
2	20.5458	69.14	0.531	39912.675	134.6	7.485	7.570	7.495	7.515	
3	20.6474	59.36	0.542	38873.890	119.3	6.895	7.035	6.920	7.020	
4	21.3810	59.36	0.542	39701.585	127.7	6.920	7.070	6.815	7.070	

In [16]:

```
X.shape
```

Out[16]:

```
(506, 20)
```

In [17]:

```
y = df["Start_Tech_Oscar"]
type(y)
```

Out[17]:

```
pandas.core.series.Series
```

In [18]:

```
y.head()
```

Out[18]:

```
0    1
1    0
2    1
3    1
4    1
Name: Start_Tech_Oscar, dtype: int64
```

In [19]:

```
y.shape
```

Out[19]:

```
(506,)
```

Test-Train Split

In [20]:

```
from sklearn.model_selection import train_test_split
```

In [21]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=0)
```

In [22]:

```
X_train.head()
```

Out[22]:

	Marketing expense	Production expense	Multiplex coverage	Budget	Movie_length	Lead_ Actor_Rating	Lead_Actress_rating	Director_rating	Producer_rating	C
220	27.1618	67.40	0.493	38612.805	162.0	8.485	8.640	8.485	8.670	
71	23.1752	76.62	0.587	33113.355	91.0	7.280	7.400	7.290	7.455	
240	22.2658	64.86	0.572	38312.835	127.8	6.755	6.935	6.800	6.840	
6	21.7658	70.74	0.476	33396.660	140.1	7.065	7.265	7.150	7.400	
417	538.8120	91.20	0.321	29463.720	162.6	9.135	9.305	9.095	9.165	

In [23]:

```
X_train.shape
```

Out[23]:

```
(404, 20)
```

In [24]:

```
X_test.shape
```

Out[24]:

```
(102, 20)
```

Standardizing Data

In [26]:

```
from sklearn.preprocessing import StandardScaler
```

In [27]:

```
sc = StandardScaler().fit(X_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:625:
DataConversionWarning: Data with input dtype uint8, int64, float64 were all converted to float64 b
y StandardScaler.
  return self.partial_fit(X, y)
```

In [28]:

```
X_train_std = sc.transform(X_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data wi
th input dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

In [29]:

```
X_test_std = sc.transform(X_test)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: Data wi
th input dtype uint8, int64, float64 were all converted to float64 by StandardScaler.
  """Entry point for launching an IPython kernel.
```

In [30]:

```
X_test_std
```

Out[30]:

```
array([[-0.40835869, -1.12872913,  0.83336883, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [ 0.71925111,  0.9988844 , -0.65283979, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [-0.40257488,  0.39610829,  0.05115377, ...,  1.50268577,
        -0.48525664, -0.75225758],
       ...,
       [-0.3982601 , -0.85812418,  0.89420778, ..., -0.66547513,
        -0.48525664,  1.3293319 ],
       [-0.39934279, -0.07637654,  0.58132175, ...,  1.50268577,
        -0.48525664, -0.75225758],
       [-0.40088071, -0.36702631,  0.31189212, ..., -0.66547513,
        -0.48525664, -0.75225758]])
```

In [ ]:

```

```

Training SVM

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

In [31]:

```
from sklearn import svm
```

In [32]:

```
clf_svm_l = svm.SVC(kernel='linear', C=100)
clf_svm_l.fit(X_train_std, y_train)
```

Out[32]:

```
SVC(C=0.01, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [ ]:

Predict values using trained model

In [33]:

```
y_train_pred = clf_svm_l.predict(X_train_std)
y_test_pred = clf_svm_l.predict(X_test_std)
```

In [34]:

```
y_test_pred
```

Out[34]:

```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0], dtype=int64)
```

Model Performance

In [35]:

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [36]:

```
confusion_matrix(y_test, y_test_pred)
```

Out[36]:

```
array([[11, 33],
       [ 5, 53]], dtype=int64)
```

In [37]:

```
accuracy_score(y_test, y_test_pred)
```

Out[37]:

```
0.6274509803921569
```

In [38]:

```
clf_svm_l.n_support_
```

Out[38]:

```
array([186, 189])
```

Grid Search

In [41]:

```
from sklearn.model_selection import GridSearchCV
```

In [42]:

```
params = {'C':(0.001,0.005,0.01,0.05, 0.1, 0.5, 1, 5, 10, 50,100,500,1000)}
```

In [43]:

```
clf_svm_l = svm.SVC(kernel='linear')
```

In [44]:

```
svm_grid_lin = GridSearchCV(clf_svm_l, params, n_jobs=-1,
                            cv=10, verbose=1, scoring='accuracy')
```

In [45]:

```
svm_grid_lin.fit(X_train_std, y_train)
```

```
Fitting 10 folds for each of 13 candidates, totalling 130 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 130 out of 130 | elapsed:  2.7min finished
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841:
DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0
.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[45]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': (0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=1)
```

In [46]:

```
svm_grid_lin.best_params_
```

Out[46]:

```
{'C': 0.5}
```

In [47]:

```
linsvm_clf = svm_grid_lin.best_estimator_
```

In [48]:

```
accuracy_score(y_test, linsvm_clf.predict(X_test_std))
```

Out[48]:

```
0.5980392156862745
```

In [49]:

```
clf_svm_p3 = svm.SVC(kernel='poly', degree=2, C=0.1)
clf_svm_p3.fit(X_train_std, y_train)
```

Out[49]:

```
SVC(C=0.1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=2, gamma='auto_deprecated',
  kernel='poly', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [50]:

```
y_train_pred = clf_svm_p3.predict(X_train_std)
y_test_pred = clf_svm_p3.predict(X_test_std)
```

In [51]:

```
accuracy_score(y_test, y_test_pred)
```

Out[51]:

```
0.5588235294117647
```

In [52]:

```
clf_svm_p3.n_support_
```

Out[52]:

```
array([185, 194])
```

Radial

In [53]:

```
clf_svm_r = svm.SVC(kernel='rbf', gamma=0.5, C=10)
clf_svm_r.fit(X_train_std, y_train)
```

Out[53]:

```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.5, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [54]:

```
y_train_pred = clf_svm_r.predict(X_train_std)
y_test_pred = clf_svm_r.predict(X_test_std)
```

In [55]:

```
accuracy_score(y_test, y_test_pred)
```

Out[55]:

```
0.6176470588235294
```

In [ ]:

```
clf_svm_r.n_support_
```

Radial Grid

In [56]:

```
params = {'C':(0.01,0.05, 0.1, 0.5, 1, 5, 10, 50),
          'gamma':(0.001, 0.01, 0.1, 0.5, 1)}
```

In [57]:

```
clf_svm_r = svm.SVC(kernel='rbf')
```

In [58]:

```
svm_grid_rad = GridSearchCV(clf_svm_r, params, n_jobs=-1,
                            cv=3, verbose=1, scoring='accuracy')
```

In [59]:

```
svm_grid_rad.fit(X_train_std, y_train)
```

```
Fitting 3 folds for each of 40 candidates, totalling 120 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   54.7s
[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:   55.6s finished
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:841:
DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0
.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[59]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'C': (0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50), 'gamma': (0.001, 0.01, 0.1, 0.5,
1)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=1)
```

In [60]:

```
svm_grid_rad.best_params_
```

Out[60]:

```
{'C': 50, 'gamma': 0.001}
```

In [61]:

```
radsvm_clf = svm_grid_rad.best_estimator_
```

In [62]:

```
accuracy_score(y_test, radsvm_clf.predict(X_test_std))
```

Out[62]:

```
0.6176470588235294
```