HAROKOPIO UNIVERSITY
DEPARTMENT OF INFORMATION & TELEMATICS

Final Report: Data Mining

Group 13:

Christos Kazakos, it22033

Konstantinos Katsaras, it22045

Manousos Linardakis, it22064

Utility Functions:

To implement the work, we made two helper functions, which help in the tokenization of the Genre, Distributor and Script Type columns.
The first function (get_dummies_genre) creates one hot encodings of a requested column (not necessarily the genre) taking into account the "repetitions" of the values of the column which cannot be separated with get_dummies as they are strings with commas. For example, a value of the genre column can be 'horror, horror' or 'drama, comedy', so the function creates one hot encoding of the horror column or (one hot encodings) of the two columns 'drama' and 'comedy' respectively . The function also takes spelling into account.

Because of the first function, and because the primary genre column has few and similar values to the genre, we decided to make a combine_primary_genre function, which takes the string values of the primary_genre column to the genre (separating them with commas). Then we can use get_dummies_genre to make the one hot encodings.

Data Preparation:

Starting the implementation of the work we see the type of features (numerical, categorical) so that we have a first picture.

```
# Summary statistics for numerical variables
print(df.describe().T)

print("========== ==========")

# Summary statistics for categorical variables
print(df.describe(include='object').T)
```

Next, we see the percentages of missing values per column. We notice that the IMDb Rating, IMDB vs RT disparity and Distributor columns are empty so we discard them (for now).

```
missing_data = df.isnull().sum()
missing_percentage = (missing_data[missing_data >0] / df.shape[0]) *100
missing_percentage.sort_values(ascending=True, inplace=True) print(missing_percentage)

df.drop(columns=['IMDb Rating'], inplace=True)
df.drop(columns=['IMDB vs RT disparity'], inplace=True) df.drop
(columns=['Distributor'], inplace=True)
```

Then we get data from csv of imdb (sourcehere) and we re-add the IMDb Rating column by filling it with all the data we have from the imdb csv. To fill this column, we join the movies.xlsx with the imdb csv on the movie name and year.

Then we check if there are duplicate tuples and if there are, we delete them. We also check if there are duplicates based on the name of the films, as after the union with the imdb csv it is possible that one film has the same name and date as another, with the result that they are considered "same" in the join. We keep the first of the two.

```
df= df.drop_duplicates()
df= df.drop_duplicates(subset='Film', keep='first')
```

Then we discard all columns that do not have at least one of the rest:

```
df.dropna(subset=[Genre',RottenTomatoes critics',Metacriticcritics', MetacriticAudience',Rotten
Tomatoes Audience','runtimeMinutes', 'directors'], inplace=True)
```

Because the csv doesn't have all the movies in movies.xlsx we additionally use the library PyMovieDb (it's a wrapper that represents the IMDB API. It gets the movie/TV series information from IMDB by scraping). Thus, we can get the ratings for almost all movies (only 1.5% of movies without IMDb Rating remain). To fill in any entries that don't have an IMDb Rating, we take the average difference between Average Audience and IMDb Rating, and subtract it from the row's Average Audience. The resulting value is the IMDb Rating estimate. The reason we chose the Average Audience column is because it had the smallest average difference with IMDb Rating.

Next, we webscrape using beautifulsoup to gather the movie distributors from rotten tomatoes. For movies that don't have distributors, we put the value -1, while for those that we can't webscrape, we put nan. As the movies we cannot webscrape are only 6.9% of the data, we consider these movies to have no distributor (we fill the nans of the distributor column with
- 1).

Then we delete some columns that we don't consider as important as some others, as their information already exists in the other columns. Specifically, we throw out Worldwide Gross ($million), Release Date (US), Opening Weekend ($million), Domestic Gross ($million), Foreign Gross ($million). In general, for "money" we deleted all values in millions. We even delete the Budget ($million) column and create the Budget column which results from Bugdet ($million) * 1000000.

Also, using the utility function “combine_primary_genre” we can discard the primary genre column as well (and we do). Another column we threw away is oscar details, as it does not exist in the anonymized dataset.

We then categorize the films based on whether they have won an Oscar (1) and those that have not (0). In the lines where there is no value (ie they have not won) we put the number 0 to make them easier to manage.

```
df['Oscar Winners'] = df['Oscar Winners'].map({'Oscar winner':1,'Oscar Winner':1})

df['Oscar Winners'].fillna(0, inplace=True)
```

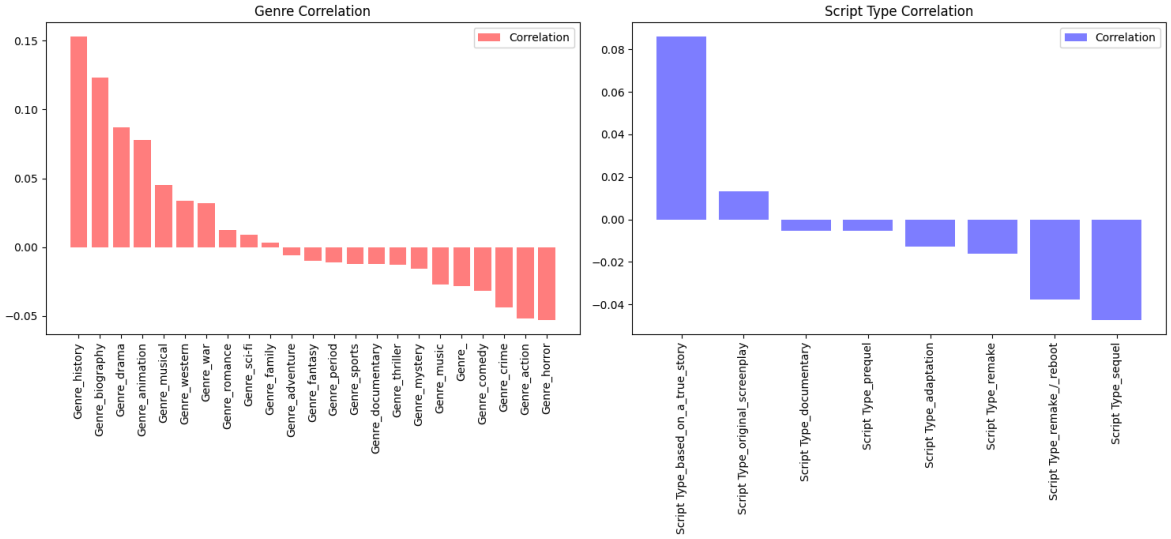
Finally, we convert the remaining columns to numeric:

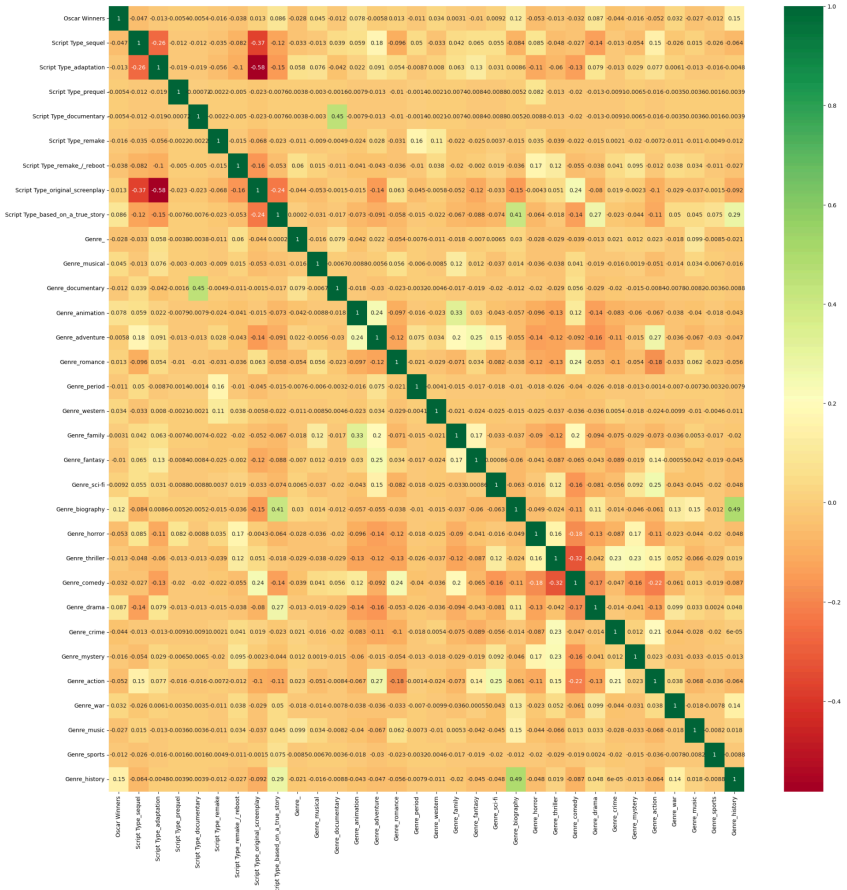
```
df['Rotten Tomatoes critics'] = pd.to_numeric(df['Rotten Tomatoes critics']) df['Metacritic critics']
= pd.to_numeric(df['Metacritic critics'])
df['Metacritic Audience'] = pd.to_numeric(df['Metacritic Audience'])

df['Opening Weekend'] = pd.to_numeric(df["Opening Weekend"].replace(',','', regex=True))

df['Domestic Gross'] = pd.to_numeric(df["Domestic Gross"].replace(',','', regex=True))

df['Foreign Gross'] = pd.to_numeric(df["Foreign Gross"].replace(',','', regex=True))


df['Rotten Tomatoes vs Metacritic deviance'] = pd.to_numeric(df['Rotten Tomatoes vs
Metacritic deviance'])
df['Audience vs Critics deviance'] = pd.to_numeric(df['Audience vs Critics deviance'])

df['Average critics'] = pd.to_numeric(df['Average critics'])
df['Average audience'] = pd.to_numeric(df['Average audience'])
df['Worldwide Gross'] = pd.to_numeric(df['Worldwide Gross'].replace(',','', regex=True))

df['Worldwide Gross'] = pd.to_numeric(df['Worldwide Gross'])

df['of Gross earned abroad'] = pd.to_numeric(df["of gross earned abroad"
].replace('%','', regex=True))
df['Budget recovered'] = pd.to_numeric(df["Budget recovered"].replace('%', '', regex=True))

df['Budget recovered opening weekend'] = pd.to_numeric(df["Budget recovered opening
weekend"].replace('%','', regex=True))
```

Having done the above we ended up with a dataset (1384x25) with no missing values. We save this dataset in an xlsx file for easy testing of experiments later.

Selection of Knowledge:

To select the features that we will finally use, we took into account various metrics such as feature importance and correlation with the Oscar Winners column. First we decided to see the correlation of the one hot encoding columns we kept, namely genre, script type and distributor. Here are the relevant charts:

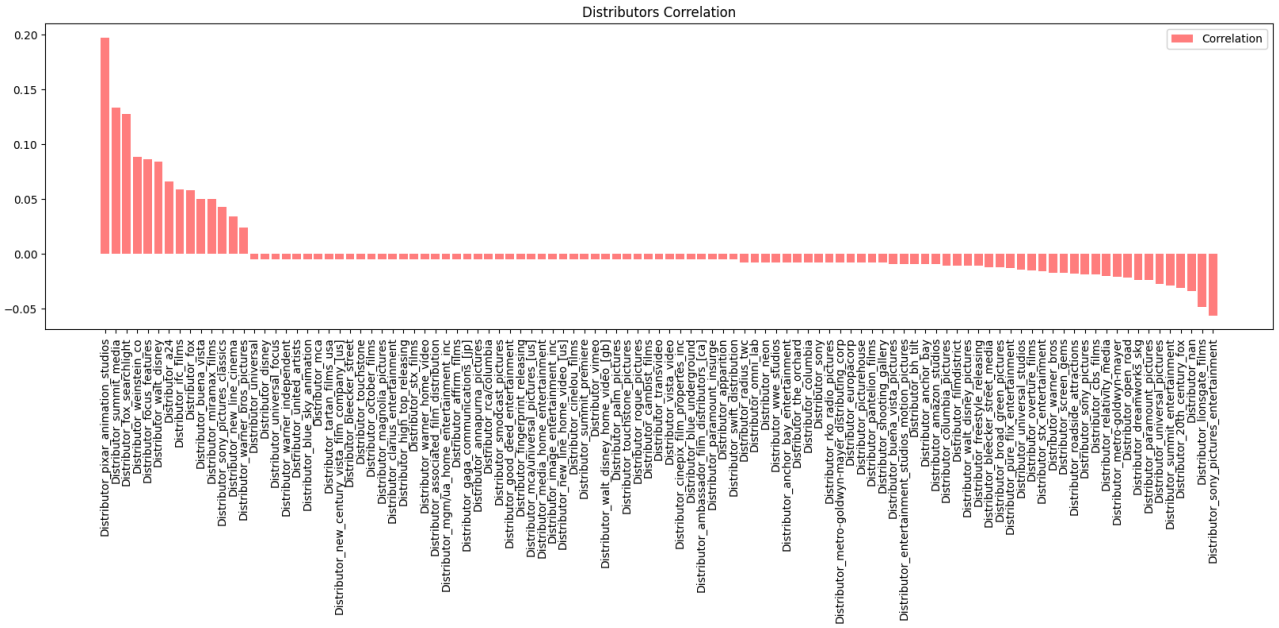




From these diagrams we can easily conclude that:

- For genre selection, we take the columns ['Genre_history', 'Genre_biography', 'Genre_drama', 'Genre_animation'], as these have the highest (by absolute value) correlation.
- To select the script type, we select the ['Script Type_based on a true story'] column, as this has the highest (by absolute value) correlation.

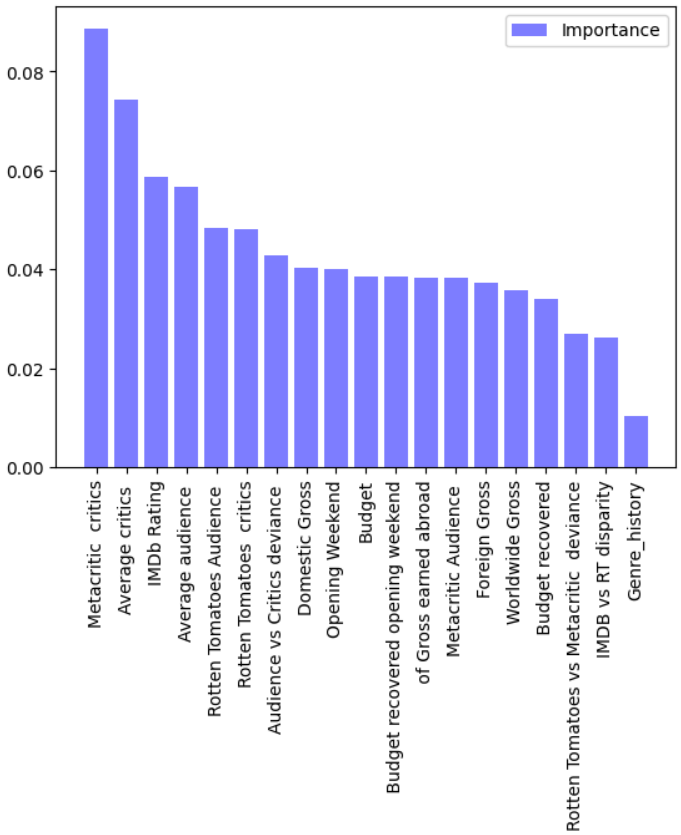
Here is the correlation chart for the distributors:



From the chart above we see that the columns ['Distributor_pixar_animation_studios', 'Distributor_summit_media', 'Distributor_fox_searchlight'] have a high (in absolute value) correlation with oscar winners, so we select them.

Then, for the selection of the numeric columns, we calculated the feature importance of the columns based on the RandomForest model, through the function calc_importance, which returns a dictionary with keys the names of the features and values the corresponding feature importance scores.

By running this function, we ended up with the following graph of feature importances:

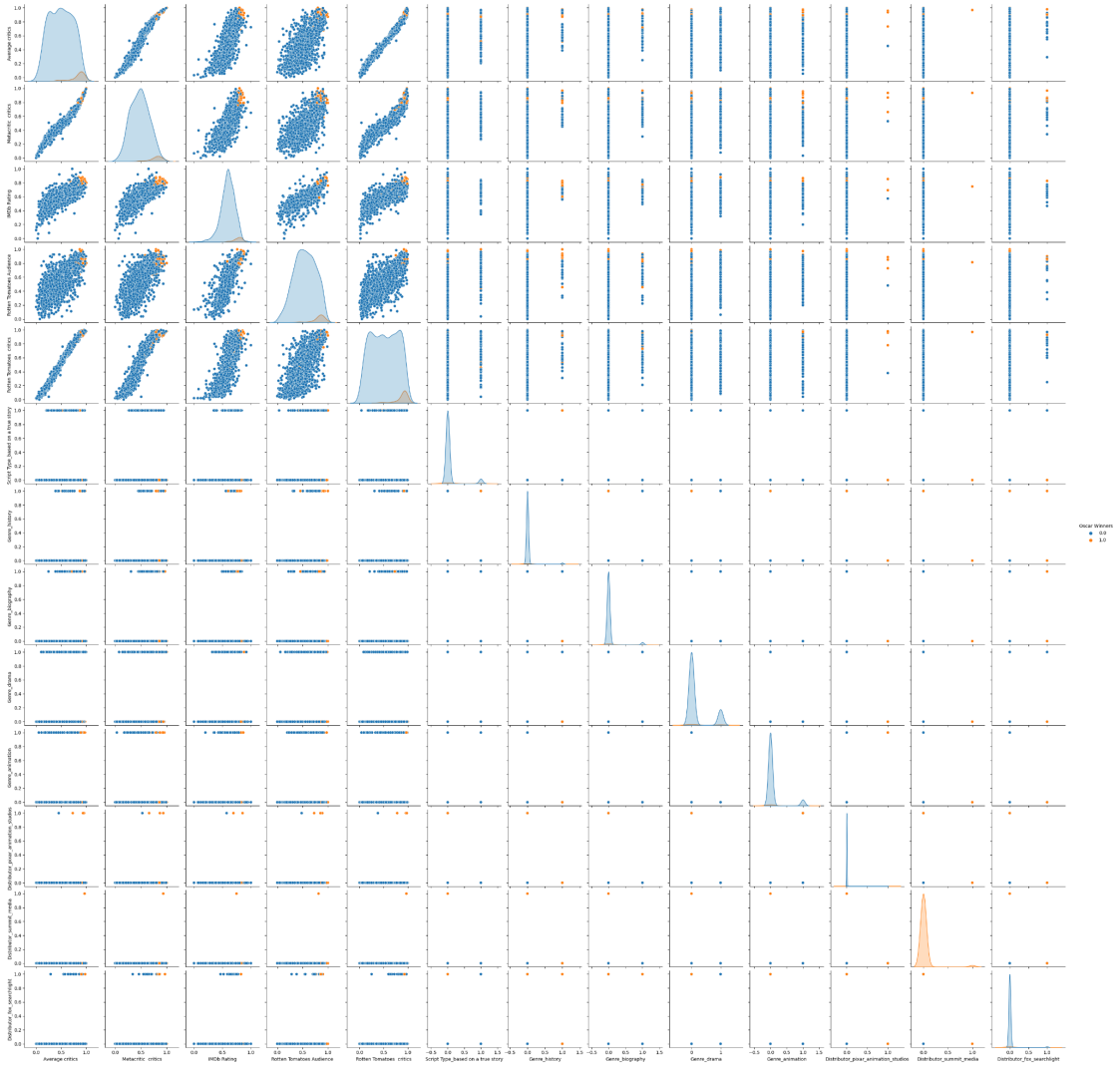


We notice that the columns ['Average critics ', 'Metacritic critics', 'IMDb Rating', 'Rotten Tomatoes Audience ', 'Rotten Tomatoes critics', 'Average audience '] are of high importance. But after experimenting, we chose all of them except 'Average audience', as (the rest) make better predictions in the train / test set.

Therefore the columns we choose are: ['Average critics ','Metacritic critics','IMDb Rating','Rotten Tomatoes Audience ', 'Oscar Winners', 'Rotten Tomatoes critics', 'Script Type_based on a true story', ' Genre_history', 'Genre_biography', 'Genre_drama', 'Genre_animation', 'Distributor_pixar_animation_studios', 'Distributor_summit_media', 'Distributor_fox_searchlight']. So we select them and then normalize the values with MinMaxScaler.

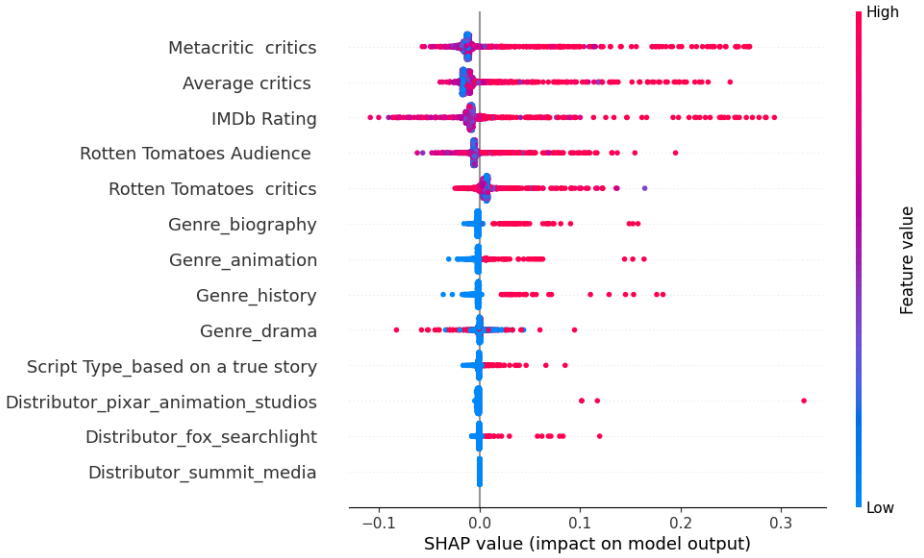
To visualize the selected features, we created the following graphs:

Pairplot:



We notice that the selected features give a relatively good separation between the films that have received an oscar and those that have not received an oscar (we can see that in most cases, the films with oscars are on the top right). However, it should be mentioned that this separation is not so "clean" as the films are "entangled" with each other.

SHAP summary plot of trait effects for the entire dataset:



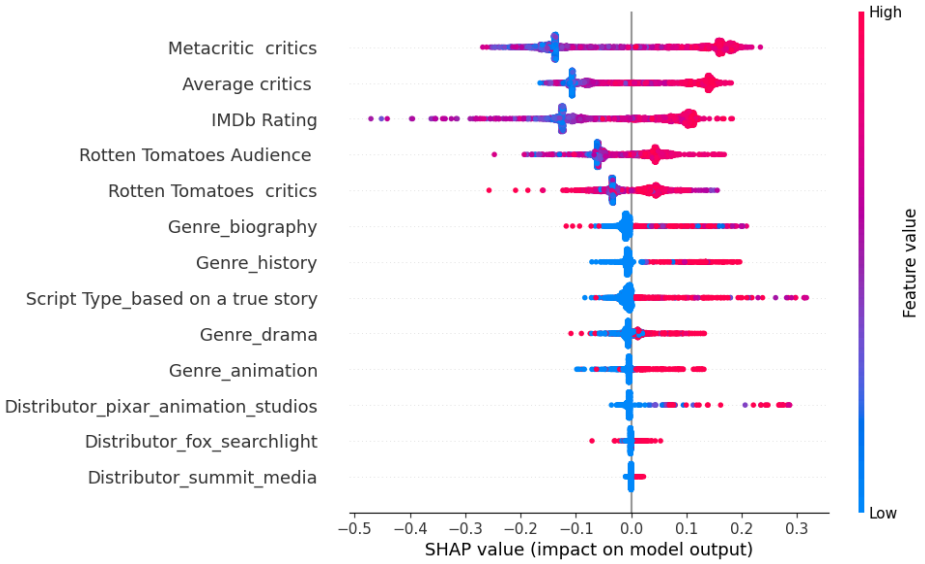
The most important features (according to this SHAP diagram) are those that are not one-hot encodings (eg Metacritic critics, Average critics, etc.). We see that as long as these values are "large" they increase (mainly) the SHAP result, i.e. the probability that the film won an oscar. But when they have medium prices or are low, the probability of them getting an oscar (in the forecast) also decreases (mainly).

We also notice that for features with one-hot encodings (eg Genre_biography, Genre_history, etc.) when they are 1 (high value) they increase the probability of predicting an oscar (vice versa when we reduce the value of one-hot encodings). It is important to note that the "same" behavior also emerges in the correlation plots, as they have a positive correlation with the oscar winners column and therefore a "balanced" relationship.

After selecting the features, we aimed to balance the dataset, so that the model could "see" more examples of oscar movies (since in the beginning, oscar movies were about 4% of all movies in the dataset). So the model could better predict oscar movies.

For this we used the SMOTE technique (Synthetic Minority Oversampling Technique) which creates synthetic samples from the minority class (movies with oscars) to balance the data. The SMOTE technique creates these synthetic data by taking the Euclidean distances with the nearest neighbors, therefore the normalization of the data before its application is necessary. After SMOTE, the dataset increased in size (2660, 14) and half of these rows are oscar movies.

SHAP summary plot of effects (after SMOTE):



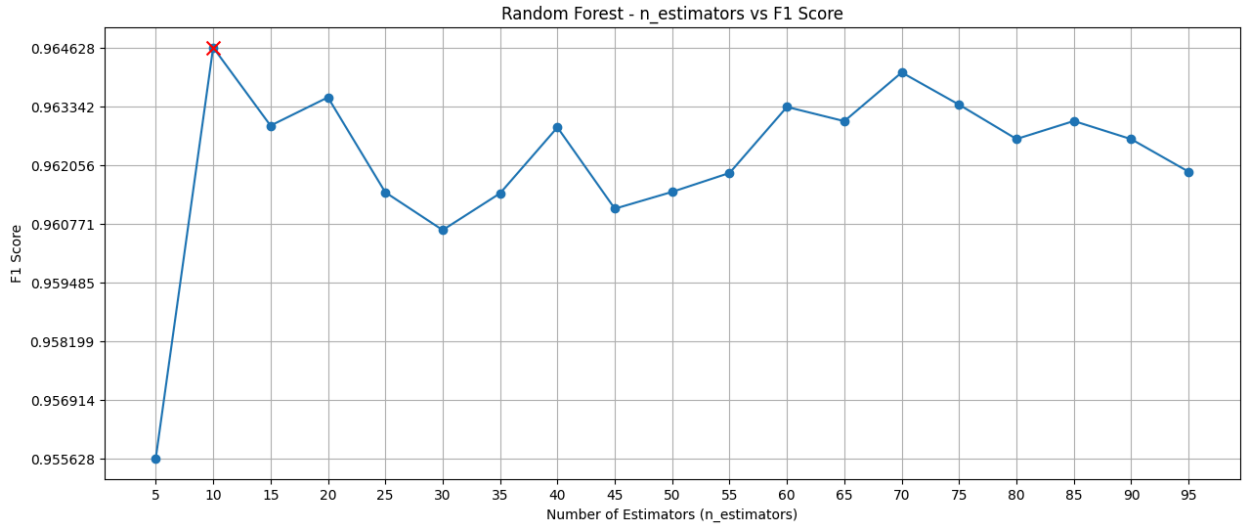
Doing the SHAP graph one more time (after SMOTE) we notice that the results are close to the original SHAP (but having more "samples" from movies with oscars). Finally we save the dataset (with SMOTE) in moviesNew3.xlsx to use it later in train/test.

Predictions & Model Evaluation:

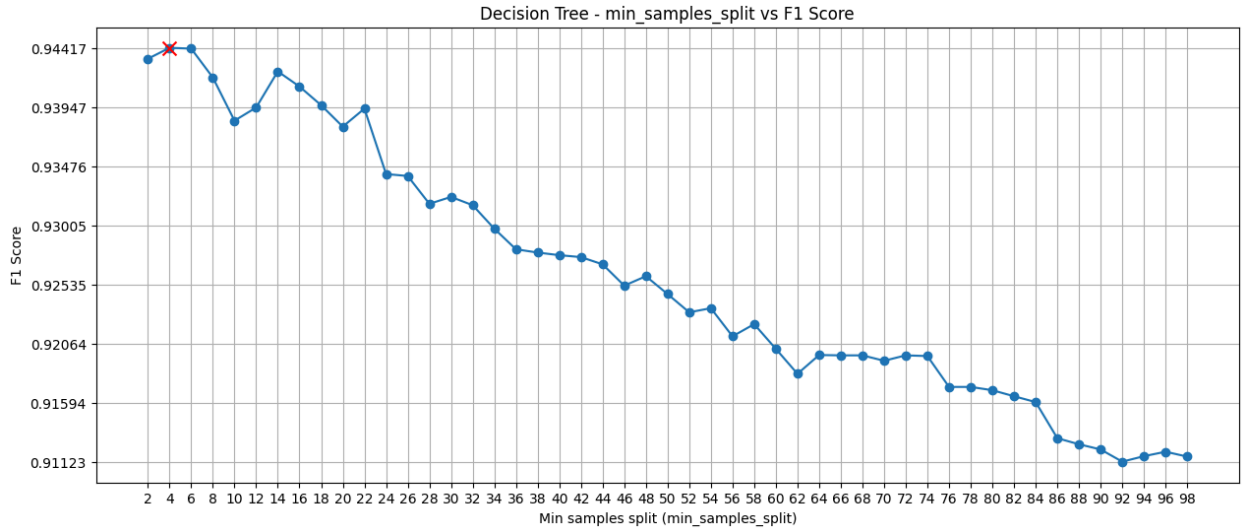
For predictions we compared 3 models: the random forest, logistic regression and decision tree.

First, we read it from the moviesNew3.xlsx of the preparation. To avoid overfit and underfit, we do 10-cross validation in our methods. By also using GridSearchCV we can find the best parameters for our model by essentially doing multiple 10-cross validations. To evaluate the parameters we use fmeasure. Here are the performance charts of the random forest and decision tree models for various values of the parameters n_estimators and min_samples_split respectively.

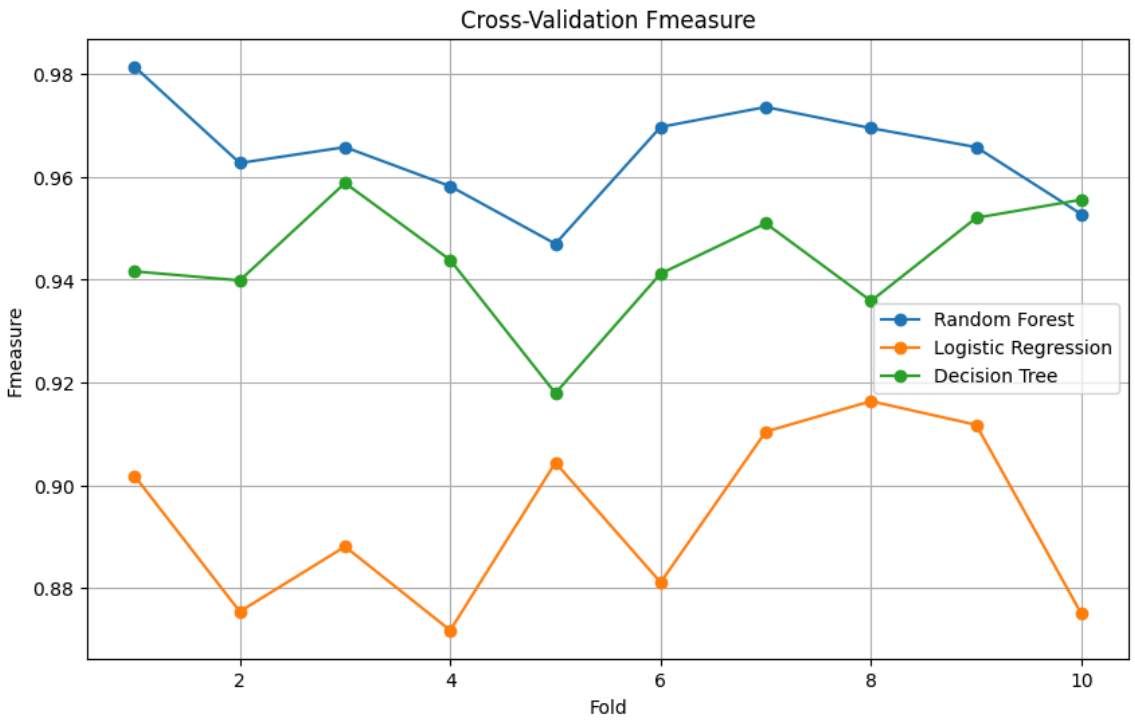
Random Forest:



Decision Tree:



For seed = 42, the best parameters of the random forest are n_estimators = 10 while for the decision tree the best min_samples_split is 4. For the logistic regression we do not put any parameter. Then we do another 10-cross validation with the best parameters and put all the Fmeasures of the models in the following plot:



We also print the Fmeasure means of the diagram above:

```
Average Random Forest Fmeasure: 0.9646277540782844
Average Logistic Regression Fmeasure: 0.8936251285334764
Average Decision Tree Fmeasure: 0.9437541633353028
```

Looking at the plots and taking the Fmeasure means, the highest is the random forest (with n_estimators=10), so this is what we choose for the next predictions.

Using random forest with seed = 42 and n_estimators = 235, we train a model and see how the predictions went. The results obtained from the training are:

```
Mean Squared Error: 0.03634085213032581
[[377 13]
 [ 16 392]]

              precision    recall  f1-score   support

           0       0.96      0.97      0.96       390
           1       0.97      0.96      0.96       408

    accuracy                           0.96       798
   macro avg       0.96      0.96      0.96       798
```

```
weighted avg          0.96      0.96      0.96       798

Roc auc score0.9637254901960786 F1
Score: 0.964329643296433
```

In the above table (confusion matrix) we see that our model has correctly predicted as negative 377. It has incorrectly predicted as positive 13. It has incorrectly predicted as negative 16 and finally it has correctly predicted as positive 377.
Also, 97% of the lines that our model predicts as class 1 actually are class 1 and correspondingly 96% of the lines that our model classifies as class 0 are actually class 0.

Regarding precision:

- For class 0: 0.96. This means that 96% of the examples that the model predicts as class 0 actually are class 0.
- For class 1: 0.97. This means that 97% of the examples that the model predicts as class 1 actually are class 1.

Regarding the recall:

- For class 0: 0.97. This means that 97% of the real examples of class 0 are correctly detected by the model.
- For class 1: 0.96. This means that 96% of real class 1 examples are correctly identified by the model.

For the F1-score (It is a combination of precision and recall.)

- For class 0: 0.96
- For class 1: 0.96

Accuracy: This means that 96% of the total examples are correctly classified by the classifier.

Also from the Roc Auc Score and F1 Score, we see that the model does a very good job of classifying the data (about 96%).

Therefore the model classifies Oscar-winning or non-Oscar-winning films very well.

Test on Anonymous Sample Data:

After choosing the best model (random forest) we proceed to predict the oscar winners for the sample of anonymous data. We read the anonymous dataset and transform its columns to look like the movies.xlsx data used for training. As many one hot encoding columns that existed in the training but do not exist in the anonymous dataset, we create them (in the anonymous) and fill them with 0 (eg 'Distributor_pixar_animation_studios' which does not exist in the anonymous dataset, the

add and fill it with 0). Also specifically for the distributors, we saw that in the anonymous data there were the companies (such as trainnng) but written differently, so we made the necessary names so that the values (and then the one hot encodings) are the same as those in trainning (for example the train dataset had the value summit_media while the anonymous dataset had the value summit_entertainment, so we simply renamed summit_entertainment to summit_media to then produce the corresponding one hot encoding).

Then we made sure that all the data of the anonymous dataset had the same types as the train dataset (eg numeric and not categorical) and we filled the nans of the columns either with the M.O from previous values of the column or with more specific types that we also used in train dataset (eg to populate the nans of the 'IMDb Rating' column in the anonymous dataset, we took the average difference between Average Audience and IMDb Rating, and subtracted it from the row's Average Audience – as we did in train). Finally, we select the columns we got in train, convert the Distributor, Genre and Script Type columns into one hot using the function get_dummies_genre (as in train) and normalize the data using MinMaxScaler (as we did in train). The final dataset is a dataset that has been created with the same procedures and has the same columns as the train dataset.

Having done all this preparation, we continue to train the random forest model with n_estimaros = 10 (as we had chosen before). For training we use the entire train dataset from before.

We finally predicted that 57 movies won oscars in the anon dataset. Here are the results of our prediction (as it is in 1st round results in eclass):

```
Omada Xriston 13 (it22064@hua_gr).csv Oscars
Predicted: 57
Precision: 0.14035087719298245 Recall:
0.7272727272727273 F-measure:
                    0.23529411764705882
Coverage:     1.0
Accuracy:     0.9079646017699115
```

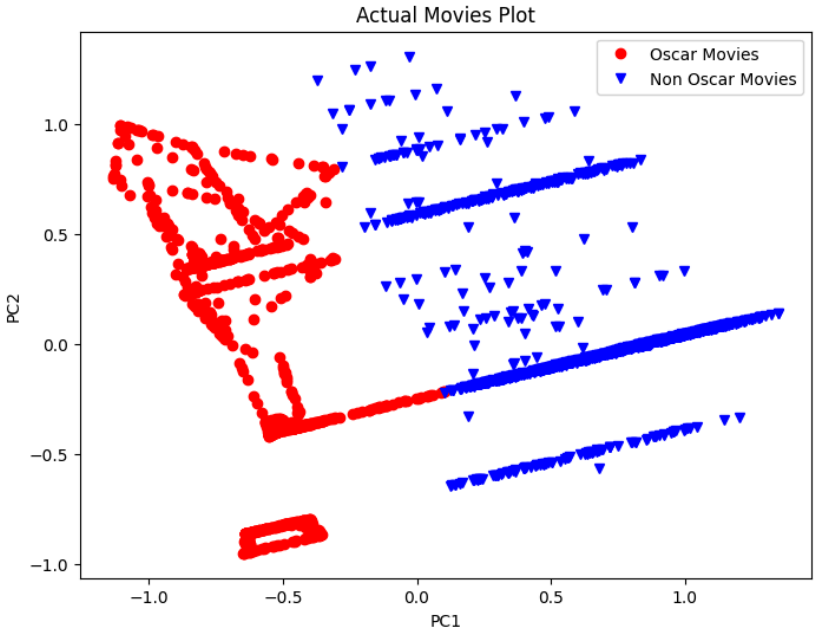
Clustering:

For clustering we read the SMOTE dataset used in training and evaluation. Then we reduce the dimensions to 2 using PCA to better find the clusters.

```
import pandas as pd
df = pd.read_excel('/content/drive/My Drive/DataMining/moviesNew3.xlsx') X = df

y = df['Oscar Winners'] seed
= 42
```

```
# PCA:
from sklearn.decomposition import PCA pca =
PCA(n_components = 2)
X = pca.fit(X).transform(X)
dfpca =pd.DataFrame(X,columns=['PC1','PC2']) df =
pd.concat([dfpca, y],axis=1)
```

First, we plot the data as it is in the dataset. In red we put the films with Oscars and in blue the films without Oscars:



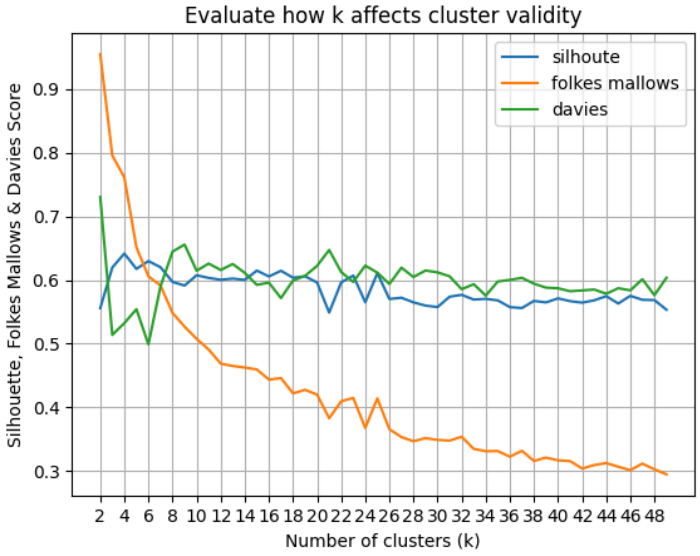
We already notice that the films are somewhat divided into 2 groups. Right and below are the ones with oscar, while left and above are the ones without (oscar).

For choosing the number of clusters we take the silhouette, fowlkes mallows and davies bouldin metrics.

We also created the function print_cluster_scores which prints these scores along with the clustering's Adjusted Rand Index Score (we will use it later).

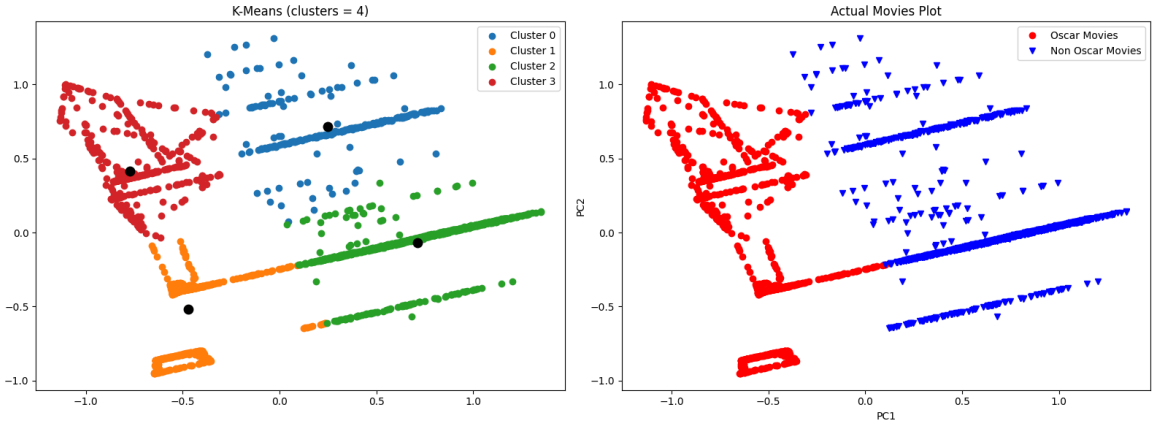
Means:

To decide the best number of clusters for Kmeans, we create the following chart with the scores reported:



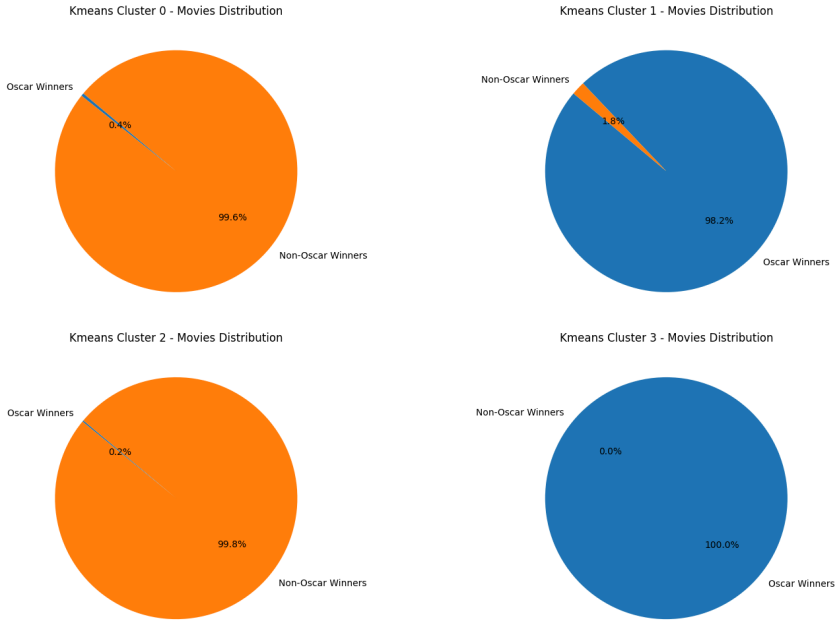
At first we notice that folkes mallows are very high. But davies bouldering is also the tallest there is, so we don't choose k = 2 or 3. Looking at the whole figure, we notice that for k = 4 the silhouette is the tallest there is, folkes mallows is tall and davies has pretty low price. For k = 6 we observe a similar behavior, but the folkes mallows is more reduced than k = 4. In the continuation of k we see that the scores folkes mallows and silhouette keep decreasing, while davies increases which is not desirable. For this we concluded that the best choice is k = 4.

To see how well the clustering did, we compare the clustered data with the real data through the following plot:



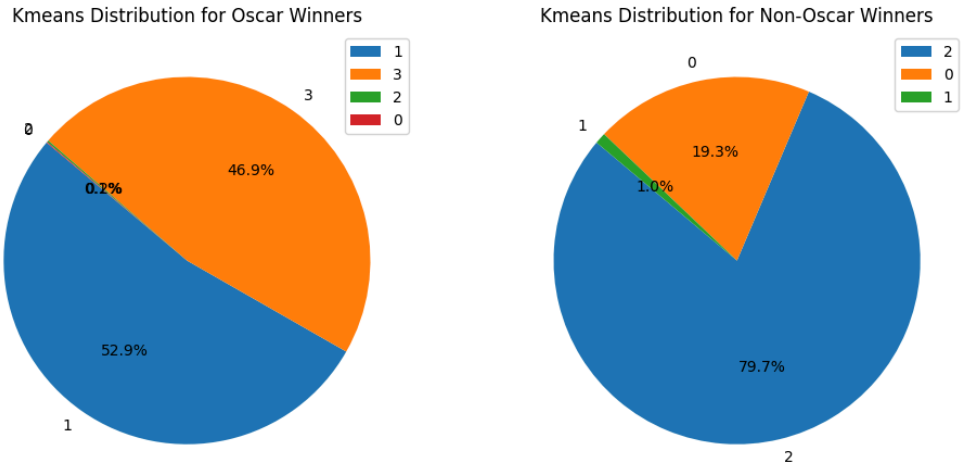
We notice that the data has been clustered very well. In particular, almost all the movies in cluster 1 and cluster 3 can be considered as the movies with oscar, while clusters 2, 3 are the movies without oscar.

This observation is confirmed if we look at the piechart:



As we noticed before, we see that cluster 3 only has movies with oscars and in cluster 1 98% are movies with oscars. In clusters 0, 2 are mainly films without oscar.

Here is the piechart of the films (and which cluster they belong to):



Based on these results we see that almost all films with oscars (about 99.8%) are in clusters 1, 3 while almost 99% of films without oscars are in clusters 0, 2. As before it seems that in cluster 3 there are only movies with oscars.

Printing the scores, we see the following:

```
Silhouette Score: 0.6416936239627882
Adjusted Rand Index(ARI): 0.5793397831624262
Fowlkes-Mallows Score: 0.7611292608123397
```

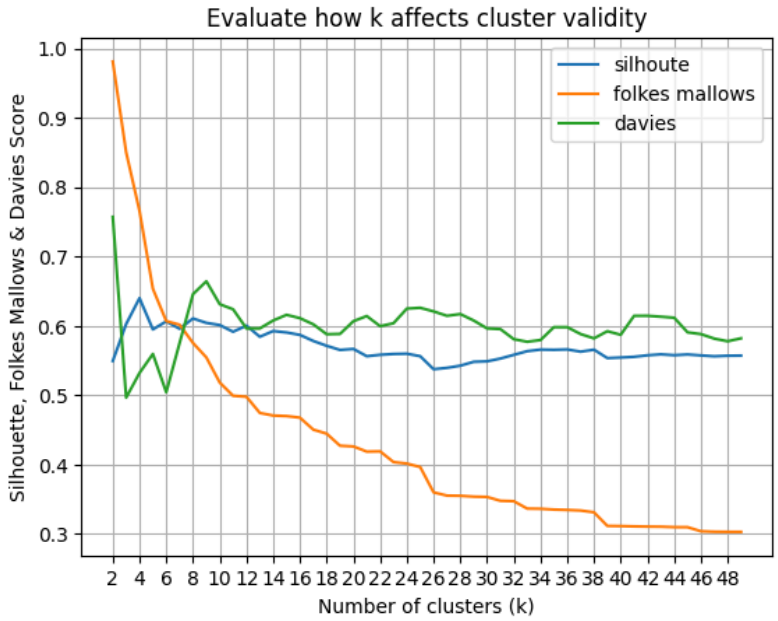
According to the ARI we conclude that the clustering level is good since if ARI = 1 then it means perfect clustering (i.e. perfect "agreement" between the predicted clusters and the actual values) while if ARI < 0 it means random (or worse) clustering.

The Silhouette score describes how well an object of a class fits the space around it (its neighborhood). It takes values from -1 to 1. The closer to 1 its value is, the better for our model. Therefore, the value 0.641 gives us the security that the objects of each cluster fit satisfactorily with each other.

The Fowlkes-Mallows Score is a metric (like the ARI) that measures how close the predicted clusters are to the actual data (using a combination of precision and recall). The bigger it is, the better for our model. Therefore 0.76 is a number that assures us that the clustering is good.

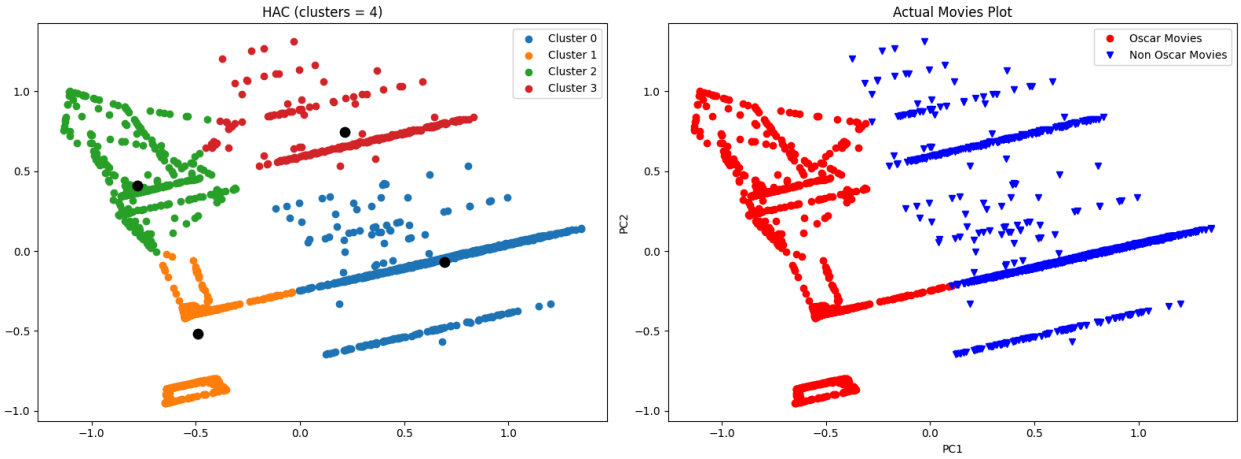
HAC:

Accordingly for HAC, we make the following diagram:



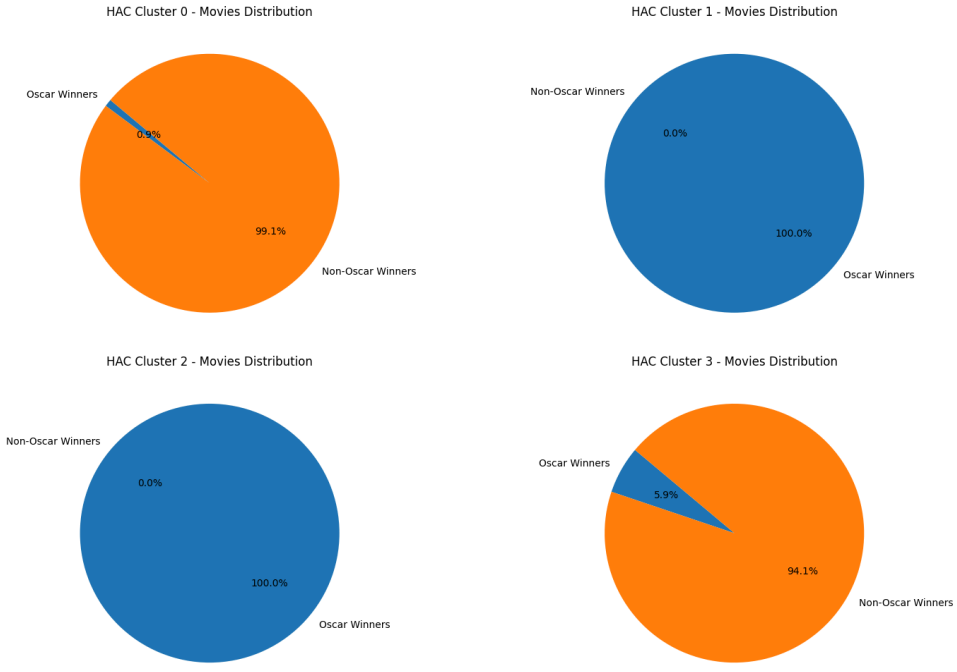
Again we notice that the highest value of the silouette score is at k = 4. Also, for k = 4, davies is relatively low and folkes mallows is high, so we choose k = 4.

To see how well the clustering did, we compare the clustered data with the real data through the following plot:



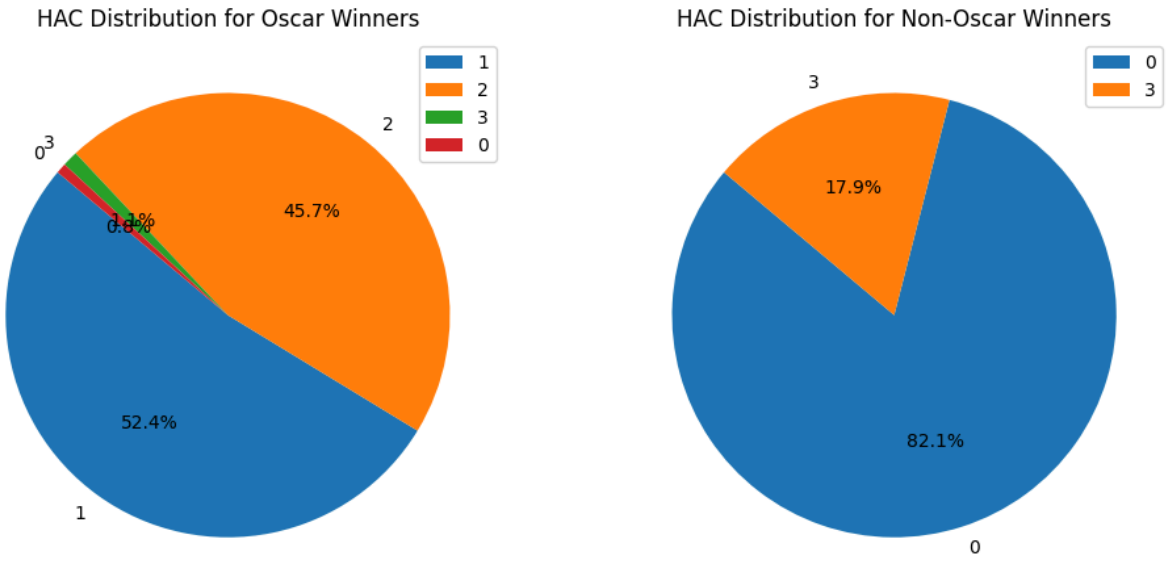
We notice that the data is again very well grouped. We could again say that almost all oscar movies are in 2 clusters (in cluster 1, 2) while the rest are in clusters 0, 3. Of course, we can also see that cluster 0, 3 also has some movies with oscar at their "limits" on the left.

These observations are confirmed if we look at the piechart:



We notice that 100% of clusters 1, 2 consist of films with oscars. Clusters 0, 3 on the other hand have mostly non-oscar movies, but there are also some oscar movies in them.

Here is the piechart of the films (and which cluster they belong to):



Based on these results we see that 100% of films without oscars are in clusters 0, 3 while almost all films with oscars (about 98.1%) are in clusters 1, 2 (as pointed out before).

Printing the scores we see:

```
Silhouette Score: 0.6400645055001494
Adjusted Rand Index(ARI): 0.5865672567916242
Fowlkes-Mallows Score: 0.7658883670748172
```

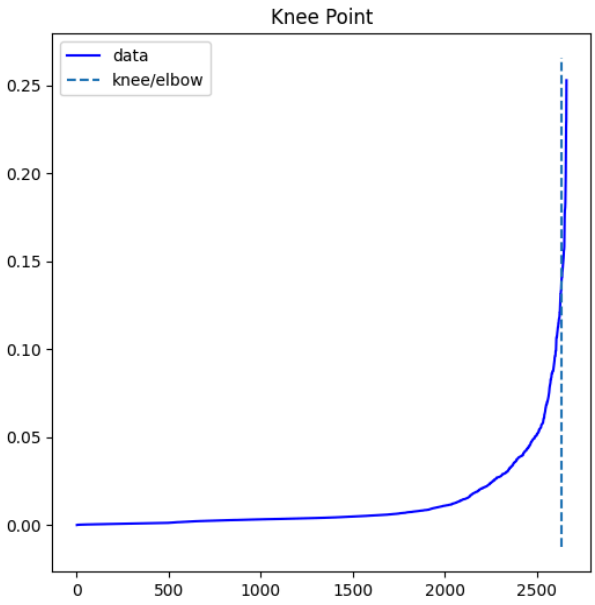
ARI score slightly higher than Kmeans, showing that HAC has predicted the clusters better and closer to the real data (as well as fowlkes-mallows). The silhouette score, on the other hand, is slightly lower than Kmeans, showing that the data within the HAC clusters are not so "similar" to each other (unlike Kmeans).

DBSCAN:

For DBSCAN we do not choose the number of clusters, but we must choose the epsilon (the radius around a data point within which the algorithm searches for neighboring data points). Specifically, if the distance between two points is less than or equal to eps, they are considered neighbors. We also need to set min_samples (the minimum number of data points needed to form a kernel). min_samples is usually equal to twice the number of features (according to this source). And because of the PCA from before, the features are 2, we set min_samples = 4.

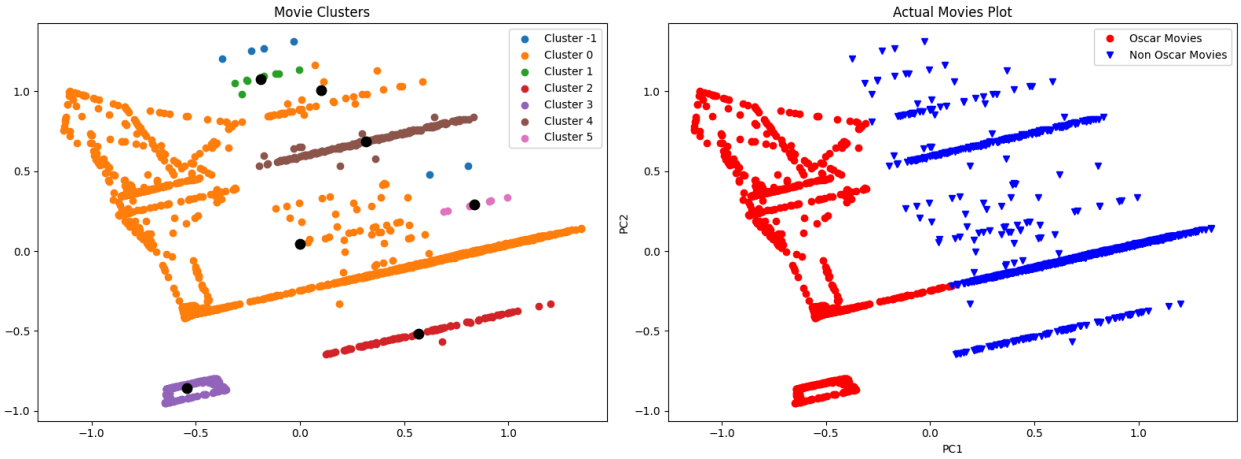
To select the eps we use from the ready library[kneeled](), the KneeLocator class, which draws and finds the knee-point (ie the eps we will use).

Specifically, the knee point (eps) finds it equal to 0.14, and produces the following figure to justify the choice:



And indeed at this point we notice that there is a sharp change in the curve, so we set eps = 0.14.

Having defined eps and min samples, we proceed to implement DBSCAN. By setting these parameters, DBSCAN made 7 clusters which look like this:



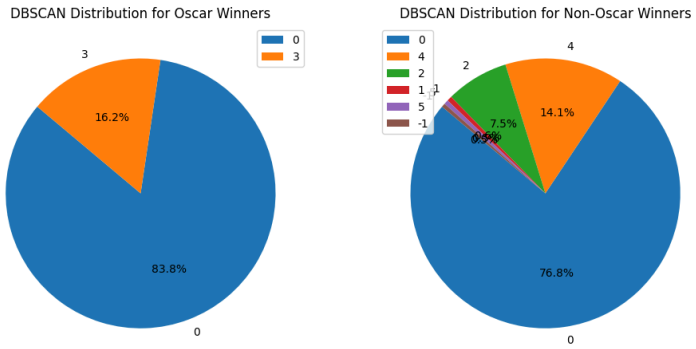
Already from the figure we can see that DBSCAN did not do a good categorization of the movies. We can see that it put most movies (oscar and not) in cluster 0. Cluster 3, seems to have correctly predicted the oscar movies. Accordingly, the rest of the clusters (except 0) we see that they describe films without an oscar. An important

observation (according todocumentation of sklearn) is that cluster -1 has the movies considered as outliers.

These observations are also confirmed by the following piecharts:

From these piecharts we see that all clusters (except cluster = 0) of DBSCAN have one movie category. As observed before, cluster 3 only has movies with oscars, while clusters -1, 1, 2, 4, 5, 6, 7 have movies without oscars. Cluster 0 has almost the same amount of oscar movies (52.2%) and no oscar movies (47.8%), with slightly more oscar movies, which is quite wrong considering (as we mentioned before) that the movies

The same result can be seen from the piechart of the films (and which cluster they belong to):



Similarly we notice that only cluster 3 has oscar movies, while -1, 1, 2, 4, 5, 6, 7 have no oscar movies. Cluster 0 has the largest number of movies and (unfortunately) has the largest percentage of both Oscar and non-Oscar movies.

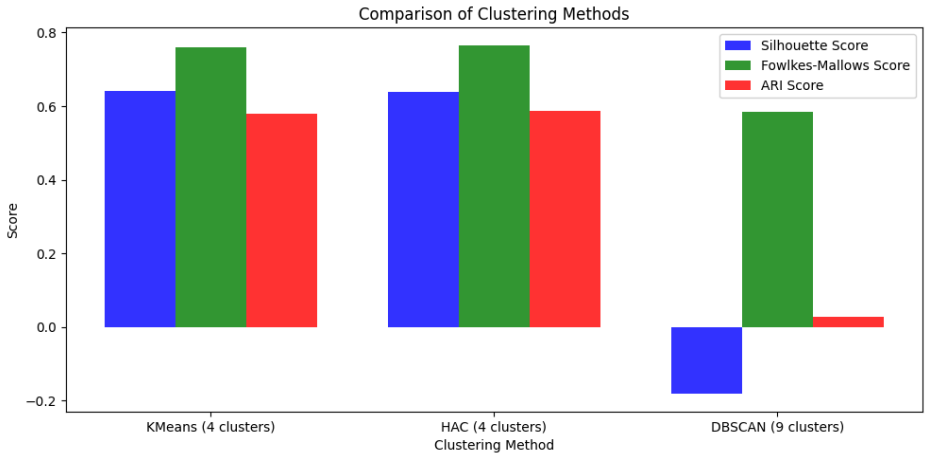
Here are the metric results:

```
Silhouette Score:-0.1820774640078261
Adjusted Rand Index(ARI): 0.028241188392073 75
Fowlkes-Mallows Score: 0.5854114173823263
```

We notice that all scores are the lowest so far. From the low ARI (and fowlkes-mallows) we see that the predicted clusters are not close to the real data (and in fact because they are close to 0, it means random clustering). The silhouette score is much lower than before, showing that the data within the clusters are not similar to each other.

<u>Final clustering Conclusions:</u>

We create a barchart with all the scores (annotated) of the tested models:



We notice that KMeans and HAC have very similar (and relatively good) scores compared to DBSCAN which has all its scores lower than both KMeans and HAC.

Therefore, for the selection of the best model in relation to the separation of Oscar films from the rest, we can choose either KMeans or HAC, as we have already seen that they do a good job in this categorization.

You can see all the code we usedhere .