ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

DATA MINING

2023/2024

Harokopio University of Athens

Kokkalis Konstantinos – it22047

Professor: Varlamis Iraklis

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Contents

A.DATA PRE-PROCCESSING

STEPS

1. Data frame Initialization.
2. Remove redundant columns.
3. Handle missing values.
4. Data formatting.
5. One Hot Encoding.
6. Delete duplicates.

```
def executePreprocess(self,type=None):
    df=initDataframe(self.fileToProcess)
    df=dropUseless(df,USELESS_COL) # DELETE USELESS COLUMNS
    df=columnDataFormating(df)
    df=numericMissingValues(df) # RETRIEVE MISSING VALUES
    df=stringMissingValues(df) # RETRIEVE MISSING VALUES
    df=columnDataFormating(df)
    df=oneHotEncoding(df) # ONE HOT ENCODING
    df=deleteDuplicate(df)  # CHECK FOR DUPLICATE ROWS
    df=dropUseless(df,['film','year']) # DELETE USELESS COLUMNS
```

Note:

At the beginning we save in some arrays the numeric columns, the redundant columns, the target strings, the no target strings as long as the script types. The reason we do that is to handle easier decision changes in the future, such as which columns we are not going to use in our classification model.

```
ALL_NUMERIC=['year', 'rotten tomatoes critics', 'metacritic
critics', 'average critics', 'rotten tomatoes audience',
'metacritic audience', 'rotten tomatoes vs metacritic deviance',
'average audience', 'audience vs critics deviance', 'opening
weekend', 'opening weekend ($million)', 'domestic gross', 'domestic
gross ($million)','foreign gross ($million)', 'foreign gross',
'worldwide gross', 'worldwide gross ($million)', 'budget ($million)
','of gross earned abroad', 'budget recovered','budget recovered
opening weekend','imdb rating','distributor','imdb vs rt disparity']
NO_TRAGET_STRINGS=['script type','primary genre','genre','release
date (us)'] #except 'film' 'oscar winners','oscar detail'
TARGET_STRINGS=['oscar winner','oscar detail']
TYPES=['adaptation','original','based on a true story','sequel',
'remake']
USELESS_COL=['rotten tomatoes critics','metacritic critics','rotten
tomatoes audience','metacritic audience','rotten tomatoes vs
metacritic deviance','audience vs critics deviance','primary genre',
'opening weekend ($million)','domestic gross ($million)','foreign
gross ($million)','worldwide gross ($million)','worldwide gross',
'budget recovered opening weekend','distributor','imdb vs rt
disparity']
```

ALL_NUMERICS = all the columns that they contain numeric values.

TARGET_STRINGS=all the string columns that have Oscar details.

NO_TARGET_STRINGS=all the other string columns except from target strings.

USELESS_COLUMNS=all the columns that we do not need for the classification model.

Data frame Initialization

1. We say the to the data frame to replace **'-'** and **'0'** with Not a Number values so we can handle them later.
2. We remove redundant spaces from column names and we transform them to lower case so we can handle them easier. We also do the same for the whole dataset.
3. We update the dataset.
4. We update the ALL_NUMERIC and the NO_TRAGET_STRINGS table according to the USELESS COLUMNS array.
5. Return the updated dataset.

```
df=initDataframe(self.fileToProcess)
```

```
def initDataframe(xFile):
    try:
        df=pd.read_excel(xFile, sheet_name = 'Sheet1',na_values=['-','0'])
        df.columns = df.columns.str.lower().str.replace(r'\s+', ' ', regex=True)
        df = df.map(lambda x: x.lower() if isinstance(x, str) else x)
        df.columns = df.columns.str.strip()
        df.to_excel(xFile, index=False)
        for item in USELESS_COL:
            if item in ALL_NUMERIC:
                ALL_NUMERIC.remove(item)
            if item in NO_TRAGET_STRINGS:
                NO_TRAGET_STRINGS.remove(item)
    except:
        raise RuntimeError(f'{colors.RED}A problem occured while initializing the excel file!
        {colors.END}')
    print(f"{colors.GREEN}INITIALIZATION HAS BEEN SUCCESFULLY CONVERTED!{colors.END}")
    return df
```

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Remove redundant columns

```
df=dropUseless(df,USELESS_COL) # DELETE USELESS COLUMNS
```

```
def dropUseless(file,uselessColumns):
  try:
    for item in uselessColumns:
      if item not in file.columns:
        uselessColumns.remove(item)
        continue
    if len(uselessColumns)!=0:
      file.drop(uselessColumns,axis=1,inplace=True) # deleting useless data from the excel
  except:
    raise RuntimeError(f'{colors.RED}A problem occured while dropping useless columns.{colors.
    END}')
  print(f"{colors.GREEN}USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!{colors.END}")
  return file
```

```
USELESS_COL=['rotten tomatoes critics','metacritic critics','rotten
tomatoes audience','metacritic audience','rotten tomatoes vs metacritic
deviance','audience vs critics deviance','primary genre','opening
weekend ($million)','domestic gross ($million)','foreign gross
($million)','worldwide gross ($million)','worldwide gross','budget
recovered opening weekend','distributor','imdb vs rt disparity']
```

Why each of them has been deleted?

distributor, imdb vs rt disparity, primary genre: We dropped these columns because we had no, or a few data comparing to the size of the tuples, which means they would not give any information and they would not help the training model.

opening weekend ($million), domestic gross ($million), foreign gross ($million), worldwide gross ($million): We dropped these columns because we have already the information of each, in other columns with a bigger precision (opening weekend, domestic gross, foreign gross).

rotten tomatoes vs metacritic deviance, audience vs critics deviance: We dropped these columns because they do not provide any useful information.

worldwide gross: We dropped this column because worldwide gross is a result of domestic and foreign gross columns that we already kept, so there is no point to keep those two features and at the same time the sum of those two features.

rotten tomatoes critics, metacrtic critics, rotten tomatoes audience, metacritic audience: Here we checked the corelation of these attributes and the averages columns to decide if we are going to keep the averages or each one individually. The result was to keep the averages and drop the individuals. The decision was made according to these results:

	rotten tomatoes critics	metacritic critics	average critics
rotten tomatoes critics	1.000000	0.941663	0.991123
metacritic critics	0.941663	1.000000	0.977929
average critics	0.991123	0.977929	1.000000

We can notice in the previous picture that all the corelations are very close to 1 and they also very similar between them. That means that we can keep only one of them. In that case we will keep average critics because it is also the mean of the other two features.

	rotten tomatoes audience	metacritic audience	average audience
rotten tomatoes audience	1.000000	0.719496	0.935264
metacritic audience	0.719496	1.000000	0.897723
average audience	0.935264	0.897723	1.000000

We can notice here that all the corelations are very close to 1 and they also very similar between them. That means that we can keep only one of them. In that case we will keep average audience because it is also the mean of the other two features.

budget recovered opening weekend: Here we checked again the corelation of this attribute, budget recovered and budget ($million). The decision was made according to these results:

	budget ($million)	budget recovered	budget recovered opening weekend
budget ($million)	1.000000	-0.001887	-0.002231
budget recovered	-0.001887	1.000000	0.998427
budget recovered opening weekend	-0.002231	0.998427	1.000000

We can notice here that all the corelations are very close to 0 (so there is no corelation) and they also very similar between them. We can only notice that budget recovered and budget recovered opening weekend they have corelation close to 1 so one of them could be deleted. So, we chose to keep budget recovered feature.

Code for getting corelation:

```
subset = df[['budget ($million)', 'budget recovered', 'budget recovered opening weekend']]
    # subset = df[['rotten tomatoes critics', 'metacritic critics','average critics']]
    # subset = df[['rotten tomatoes audience','metacritic audience','average audience']]
    getCorrelation(subset)
def getCorrelation(items):
  correlation_matrix = items.corr()
  print(correlation_matrix)
```

Note:

film, year: We delete film and year because the name of the film is a string that will not give helpful information to the classification model, on the other side it will confuse it. Year also does not provide any useful information, but we can use both to retrieve other data. So, we do not drop them yet.

script type, genre, Oscar detail: We will one-hot encode them and some of them so we will delete the main feature of one-hot encoding after this process.

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Missing values

```
df=numericMissingValues(df) # RETRIEVE MISSING VALUES
df=stringMissingValues(df) # RETRIEVE MISSING VALUES
```

```
def numericMissingValues(file): # replacing ',' and missing values with
the mean of the year it belongs to
  try:
    for item in ALL_NUMERIC:
      if item not in file.columns:
        ALL_NUMERIC.remove(item)
        continue
    if len(ALL_NUMERIC)==0:
      print(f'{colors.GREEN}NO NUMERIC COLUMNS TO BE
PROCESSED.SUCCESFULLY COMPLETED{colors.END}')
      return file
  # file=externalIMDb(file) # IMDb
    for element in ALL_NUMERIC:
      for index, row in file.iterrows():
        year = row['year']
        mean = file[file['year'] == year][element].mean()
        file[element] = pd.to_numeric(file[element], errors='coerce')
        if pd.isna(row[element]):
          if pd.isna(mean):
            file[element]=file[element].ffill()
            continue
          file.loc[index, element] = mean
      file[element].bfill(inplace=True)
  except:
    print(element,index,row,mean)
    raise ValueError(f'{colors.RED}A problem occured while processing
numeric missing values{colors.END}')
  print(f"{colors.GREEN}NUMERIC MISSING VALUES HAS BEEN SUCCESFULLY
RESTORED!{colors.END}")
  return file
```

For numeric missing values we get **external knowledge** only for imdb rating feature. For all the others we take the mean value of each numeric feature according to year the film released.

(e.g Missing value at foreign gross. Year=2007, we complete the missing value with the foreign gross mean of the films released in 2007).

In case we cannot retrieve external knowledge or calculate the mean for a missing value we are doing a ffil() and if this is also not possible we do bfill().

```
def stringMissingValues(file):
  try:
    for item in NO_TRAGET_STRINGS:
      if item not in file.columns:
        NO_TRAGET_STRINGS.remove(item)
        print(f"{item} column removed from the array because it doesn't exist in the dataset")
        continue
      if len(ALL_NUMERIC)==0:
        print(f'{colors.GREEN}NO STRING COLUMNS TO BE PROCESSED.SUCCESFULLY COMPLETED{colors.END}')
        return file
    file=externalGenre(file)
    for index, row in file.iterrows():
        if 'oscar winners' in file.columns:
          if (pd.isna(row['oscar winners'])):
            file.loc[index, 'oscar winners'] = 0
          else:
            file.loc[index, 'oscar winners'] = 1
    for j in NO_TRAGET_STRINGS:
      if file[j].isnull().any():
        file[j]=file[j].ffill()
      else:
        file[j]=file[j].bfill()
  except:
    raise RuntimeError(f'{colors.RED}A problem occured while processing string missing values{colors.END}')
  print(f"{colors.GREEN}OTHER MISSING VALUES HAS BEEN SUCCESFULLY RESTORED!{colors.END}")
  return file
```

In string missing values we get **external knowledge** only for genre.

If any other value is missing we are doing a ffill() and if this is also not possible we do bfill().

External knowledge

In this part of external knowledge we just use a python library of IMDb and we get the genre.If we cannot get the genre then we complete the value as NaN.

```
def externalGenre(file):
  ia=IMDb()
  try:
    for index,row in file.iterrows():
      if pd.isna(row['genre']):
        movies = ia.search_movie(row['film'])
        if movies:
            movie = ia.get_movie(movies[0].movieID)
            genre=",".join(movie['genres']).lower()
        else:
            print("Movie not found.")
            genre=np.nan
        file.at[index, 'genre'] = genre
  except:
      raise RuntimeError(f'{colors.RED}A problem occured while 
receiving external knowledge in string missing values{colors.END}')
  return file
```

In this part of external knowledge we just use a python library of IMDb and we get the imdb rating. If we cannot get the rating then we complete the value as NaN and they will be replaced later with the mean, ffill() or bfill()

```
def externalIMDb(file):
  ia = IMDb()
  i=2
  for movieTitle in file['film']:
    movies = ia.search_movie(f"{movieTitle}")
    try:
      if movies:
          movie = ia.get_movie(movies[0].movieID)
          print(f"Title: {movie['title']}")
          year=movie['year']
          rating=movie['rating']
      else:
          print("Movie not found.")
          continue
      print(i," ",rating)
      if not rating:
        file.at[i-2, 'imdb rating'] = np.nan
        continue
      else:
        file.at[i-2, 'imdb rating'] = rating
        file.at[i-2,'year']=year
    except:
      file.at[i-2, 'imdb rating'] = np.nan
      continue
    i=i+1
  print(f'{colors.GREEN}EXTERNAL KNOWLEDGE \'IMDb\' HAS BEEN 
SUCCESFULLY ADDED{colors.END}')
  # file.to_excel("clone.xlsx")
  return file
```

Note:

1.This process has already been executed and we copied the data to another file (**moviesUpdated.xlsx**), because the process to fetch data for almost 1400 rows takes a lot of time.

2.Textblob library is used in the dataset for the genre feature, because a lot of genres are misspelled.

Data formatting

```
df=columnDataFormating(df)
```

After receiving all the new values and we have already dropped the most of the columns we do not need, we have to format all the data to be the same so the model can work correctly.

```
def columnDataFormating(file):
  try:
    #  Convert each value of the df that is ending with the '%' to
decimal (divide by 100)
    file = file.map(lambda val: float(val.rstrip('%')) / 100 if
isinstance(val, str) and val.endswith('%') else val)
  except:
    raise RuntimeError(f'{colors.RED}A problem occured while converting
precentages to decimal.{colors.END}')
  try:
    file[ALL_NUMERIC] = file[ALL_NUMERIC].replace(',', '',
regex=True).apply(pd.to_numeric, errors='coerce')
    file['budget ($million)'] = file['budget ($million)'] * 1000000
    # file[NO_TRAGET_STRINGS] = file[NO_TRAGET_STRINGS].replace(',',
'', regex=True)
    file['genre'] = file['genre'].str.replace(',', '
').str.replace('.', ' ').str.replace('\s+', ' ',
regex=True).str.strip()
    # file['oscar detail'] = file['oscar
detail'].str.extract(r'([^\(]+)')
    file['oscar detail'] = file['oscar detail'].str.split('(',
n=1).str[0].str.strip()
  except:
    raise RuntimeError(f'{colors.RED}A problem occured while replacing
charachters.{colors.END}')
  print(f"{colors.GREEN}ALL COLUMNS HAS BEEN SUCCESFULLY
FORMATED!{colors.END}")
  return file
```

One-Hot encoding

```
    df=oneHotEncoding(df) # ONE HOT ENCODING
```

We do have nominal features that we need to represent them with a way to help the model to work better. Those are script type, oscar detail, release date (us), genre

```
# SCRIPT TYPE
  if 'script type' in file.columns:
    try:
      one_hot_encoded = pd.get_dummies(file['script type'].apply(lambda x: next((t for t in TYPES if str(x).startswith(t)), None))).astype(int)
      file = pd.concat([file, one_hot_encoded], axis=1)
      file=dropUseless(file,['script type'])
    except:
      raise RuntimeError(f'{colors.RED}A problem occured while one-hot encoding script-type{colors.END}')
```

```
# OSCAR DETAILS
  if 'oscar detail' in file.columns:
    try:
      one_hot_encoded = file['oscar detail'].str.get_dummies(', ').astype(int)
      file = pd.concat([file, one_hot_encoded], axis=1)
      file=dropUseless(file,['oscar detail'])
    except:
      raise RuntimeError(f'{colors.RED}A problem occured while one-hot encoding oscar details{colors.END}')
```

In the previous ones we did directly normal one-hot encoding but genre and release date (us) need a special treatment.

As far as the genre is concerned there were a lot genres with wrong spelling mistakes or the same genre with another word , for example ‘thriller’ and ‘thrill’. All of those had to be handled before one-hot encoding.

```
# GENRE
  if 'genre' in file.columns:
    try:
      file['genre']=file['genre'].str.lower()
      genres=set()
      correctGenre=[]
      for item in file['genre']:
        for word in item.split():
          # correctedWords=[]
          correctedWord=str(Word(word).correct().lower())
          # print(correctedWord)
          correctGenre.append(correctedWord)
      genres.update(correctGenre)
      # print("c:",genres)
      words_to_remove = set()
      for word1 in genres:
            for word2 in genres:
                if word1 != word2 and len(word1) >3 and len(word2) > 3:
                    common_substrings = set([word1[i:i+5] for i in
range(len(word1)-4) if word1[i:i+5] in word2])
                    if common_substrings:
                        shorter_word = word1 if len(word1) < len(word2)
else word2
                        words_to_remove.add(shorter_word)
      genres.difference_update(words_to_remove)
      # print(genres)
      file['genre'] = file['genre'].apply(lambda cell: ' '.join(
        [next((word_set_word for word_set_word in genres if
word_set_word[:3] == word[:3]), word) for word in cell.split()]
    ))
      print('Set:::',genres)
      for genre in genres:
          file[genre] = file['genre'].apply(lambda x: 1 if genre in
x.split() else 0)
      file=dropUseless(file,['genre'])
    except:
```

```
      raise RuntimeError(f'{colors.RED}A problem occured while one-hot
encoding genres{colors.END}')
```

As far as the date is concerned before one hot encoding we played with the format and we kept only the released month instead of the whole day, month and year, because we believed all of these information will not give us any advantage.

```
# DATE
  if 'release date (us)' in file.columns:
    try:
      file['release date (us)'] = pd.to_datetime(file['release date
(us)'],format='mixed')
      # Extract month and day
      file['release date (us)'] = file['release date
(us)'].dt.strftime('%m').astype(int)
      monthMapping = {
        1: 'january',
        2: 'february',
        3: 'march',
        4: 'april',
        5: 'may',
        6: 'june',
        7: 'july',
        8: 'august',
        9: 'september',
        10: 'october',
        11: 'november',
        12: 'december'
    }

      # Nominalize the 'release date (us)' column
      file['release date (us)'] = file['release date
(us)'].map(monthMapping)
      one_hot_encoded = pd.get_dummies(file['release date (us)'],
prefix='').astype(int)
      file = pd.concat([file, one_hot_encoded], axis=1)
      file=dropUseless(file,['release date (us)'])
    except:
      raise RuntimeError(f'{colors.RED}A problem occured while one-hot
encoding dates{colors.END}')
```

Note:

Of course, after one-hot encoding we drop each feature that got one-hot encoded.

Delete duplicates

```
df=deleteDuplicate(df)   # CHECK FOR DUPLICATE ROWS
```

after all the process we have to check if we have duplicate data.

```
def deleteDuplicate(file):
  try:
    if (file.duplicated().sum() != 0) or (not
file[file.duplicated(subset=['film'])].empty):
      print(f'The dataset contains
{(file.duplicated(subset=["film"])).sum()} duplicate films that need to
be removed.')
      print(f'The dataset contains {file.duplicated().sum()} duplicate
rows that need to be removed.')
      file.drop_duplicates(inplace=True)
      file = file.drop_duplicates(subset=['film'],keep='first')
  except:
    raise RuntimeError(f'{colors.RED}A problem occured while deleting
duplicates{colors.END}')
  print(f"{colors.GREEN}DUPLICATE ROWS HAVE BEEN SUCCESFULLY
DELETED!{colors.END}")
  return file
```

we check the whole subset for duplicates, but also, we check the subset film for duplicate film names. If a film has a duplicate name, then we keep the data of the film we first found in our dataset.

Extra functions

We also created some functions for the preprocessing such as scaling and normalization because we do not know yet what classification model we are going to use and what are its prerequires for it to work.

```
def scaling(file):
  for item in ALL_NUMERIC:
    if item not in ALL_NUMERIC:
      ALL_NUMERIC.remove(item)
      continue
  if len(ALL_NUMERIC)!=0:
    try:
      scaler = StandardScaler()
      file[ALL_NUMERIC] = scaler.fit_transform(file[ALL_NUMERIC])
    except:
      raise ValueError(f'{colors.RED}A problem occured while scaling values{colors.END}')
  print(f"{colors.GREEN}SCALING HAS BEEN SUCCESFULLY COMPLETED!{col-ors.END}")
  return file

def normalization(file):
  for item in ALL_NUMERIC:
    if item not in ALL_NUMERIC:
      ALL_NUMERIC.remove(item)
      continue
  if len(ALL_NUMERIC)!=0:
    try:
      columns_to_normalize = file[ALL_NUMERIC]
      scaler = MinMaxScaler()
      normalized_columns = pd.DataFrame(scaler.fit_transform(col-umns_to_normalize), columns=ALL_NUMERIC)
      file[ALL_NUMERIC] = normalized_columns
    except:
      raise ValueError(f'{colors.RED}A problem occured while normalis-ing values{colors.END}')
  print(f"{colors.GREEN}NORMALISING HAS BEEN SUCCESFULLY COM-PLETED!{colors.END}")
  return file
```

Result

```
INITIALIZATION HAS BEEN SUCCESFULLY CONVERTED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
ALL COLUMNS HAS BEEN SUCCESFULLY FORMATED!
NUMERIC MISSING VALUES HAS BEEN SUCCESFULLY RESTORED!
OTHER MISSING VALUES HAS BEEN SUCCESFULLY RESTORED!
ALL COLUMNS HAS BEEN SUCCESFULLY FORMATED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
ONE HOT ENCODING HAS BEEN SUCESSFULLY COMPLETED!
The dataset contains 6 duplicate films that need to be removed.
The dataset contains 0 duplicate rows that need to be removed.
DUPLICATE ROWS HAVE BEEN SUCCESFULLY DELETED!
USELESS COLUMNS HAS BEEN SUCCESFULLY DELETED!
------------------------PRE-PROCESSING-FINISHED------------------------

DataFrame does not contain NaN values.
   average critics  average audience  opening weekend  domestic gross  ...  _may  _november  _october  _september
0             56.0              80.0         70885301       210614939  ...     0          0         0           0
1             82.0              80.0         14035033        53606916  ...     0          0         0           1
2             52.0              61.0         15951902        39568996  ...     0          0         1           0
3             55.0              78.0          3824988        24343673  ...     0          0         1           0
4             22.0              38.0         10059425        41797066  ...     0          0         0           0

[5 rows x 83 columns]
PS C:\Users\kokka\GitHub projects\Oscars---Data-Mining>
```

Important Note:

The features that have been selected to be used they are not the final as long as the classification model might not give us the best results. So modifications to USELESS_COL might happen.

B1.CLASSIFICATION

Before we start explaining about the classification process is important to mention that the features we use now are different. That is because with the previous features we could not receive very good results so the model was overfitting or it was not giving any results. The way we pre process the data is exactly the same. The only thing that changes is the columns we use.

The way we decided which columns to drop since we were not getting good results was running the model and checking at the same time the correlation and feature importance (after classification) of each feature with the target column 'oscar winners'. Features with negative or very low correlation were deleted.

By droping these columns as a first step we can make easier our choice on which features we should still drop.

```
USELESS_COL=['id','imdb vs rt disparity','oscar detail','distributor','primary genre','release date (us)']
```

Correlation Matrix: **Feature Importance:**

```
Correlation Matrix:
oscar winners                              1.000000
metacritic critics                         0.329870
average critics                            0.291117
average audience                           0.265832
rotten tomatoes critics                    0.259006
rotten tomatoes audience                   0.252677
metacritic audience                        0.239778
audience vs critics deviance               0.188254
history                                    0.151222
biography                                  0.121439
domestic gross                             0.106747
domestic gross ($million)                  0.106690
worldwide gross                            0.105525
based on a true story                      0.102890
of gross earned abroad                     0.102483
foreign gross                              0.096799
worldwide gross ($million)                 0.095323
imdb rating                                0.092167
drama                                      0.091771
rotten tomatoes vs metacritic deviance     0.090909
animation                                  0.075737
western                                    0.033406
war                                        0.031641
foreign gross ($million)                   0.028307
original                                   0.010070
musical                                    0.008389
sci-fi                                     0.007970
romantic                                   0.007725
opening weekend ($million)                 0.004982
family                                     0.002003
opening weekend                           -0.000809
budget ($million)                         -0.003665
budget recovered                          -0.003945
adventure                                 -0.007016
budget recovered opening weekend          -0.007500
fantasy                                   -0.010517
period                                    -0.010856
adaptation                                -0.011803
documentary                               -0.012142
thrilled                                  -0.014573
mystery                                   -0.016673
sports                                    -0.027348
comedy                                    -0.033214
remake                                    -0.040624
crime                                     -0.044129
sequel                                    -0.048100
action                                    -0.053181
horror                                    -0.054118
Name: oscar winners, dtype: float64
```

```
42                               thrilled    0.105416
26                     metacritic critics    0.103682
6                         average critics    0.092655
0                                  action    0.090161
15                         domestic gross    0.048649
5                        average audience    0.047008
8                               biography    0.045434
25                    metacritic audience    0.039476
11       budget recovered opening weekend    0.038825
29                 of gross earned abroad    0.036651
46             worldwide gross ($million)    0.029583
3                               animation    0.028927
32                               original    0.027658
20                          foreign gross    0.015045
9                       budget ($million)    0.013528
40                                 sequel    0.012782
38 rotten tomatoes vs metacritic deviance    0.011469
30                        opening weekend    0.010324
4            audience vs critics deviance    0.010204
45                        worldwide gross    0.009149
37                rotten tomatoes critics    0.007785
16              domestic gross ($million)    0.006248
31             opening weekend ($million)    0.004746
14                            documentary    0.000000
13                                  crime    0.000000
33                                 period    0.000000
34                                 remake    0.000000
35                               romantic    0.000000
28                                mystery    0.000000
27                                musical    0.000000
7                   based on a true story    0.000000
39                                 sci-fi    0.000000
22                                history    0.000000
41                                 sports    0.000000
2                               adventure    0.000000
43                                    war    0.000000
44                                western    0.000000
18                                 family    0.000000
23                                 horror    0.000000>
```

So according to the results above we ended up dropping the following features

New Dropped Columns:

```
USELESS_COL=['id','imdb vs rt disparity','oscar detail','distribu-
tor','primary genre','genre','script type','release date (us)','opening
weekend','budget recovered','budget ($million)','budget recovered open-
ing weekend','domestic gross ($million)']
```

Now we can continue to classification.

STEPS

1.Load Data

2.Prepare Data

3.Train Model

4.Predict

```
if __name__=='__main__':
    argOne,argTwo,argThree=handleArgvs()

    trainDataset,predictDataset=preprocess(argThree)
    trainTarget,trainData=seperateData(trainDataset) #seperate train from target data of the
    predictData=fixTestFile(trainData,predictDataset)
    scaledTrainData,scaledPredictData=scaleData(trainData,predictData)
    model,X_train,X_valid,y_train, y_valid=doTraining(scaledTrainData,trainTarget,argOne)
    y_pred = model.predict(X_valid)
    if sys.argv[2]=='stats':
        printStats(model,y_valid,y_pred,scaledTrainData,trainTarget)
    if sys.argv[1]!='knn' and sys.argv[1]!='lr':
        featureImportances(model,trainData)
    doPredictions(model,scaledPredictData,predictData) # Predict in the model
```

Load and Prepare Data

Before the training of the model and before making the predictions we have to be sure that there are no missing values in both training and test datasets as long as the two datasets must have exactly the same features. We also have to separate features from target in the training dataset and scale the data using MinMax scaler.

So first we preprocess both datasets

```
def preprocess(command='prepro'):
    if command=='prepro':
        dp=DataPreprocessor(TRAIN_PATH,TRAIN_PATH_PROCESSED)
        df=DataPreprocessor(PREDICT_PATH,PREDICT_PATH_PROCESSED)
        trainDataset=dp.executePreprocess()
        predictDataset=df.executePreprocess(predict=True)#options: pre-
dict=True/False
    else:
        # SAVING TIME-------------------
        trainDataset=pd.read_excel(TRAIN_PATH_PROCESSED, sheet_name =
'Sheet1')
        predictDataset=pd.read_excel(PREDICT_PATH_PROCESSED, sheet_name
= 'Sheet1')
    print('all files has been succesfully preprocessed')
    return trainDataset,predictDataset
```

and right after we make sure that both datasets have the same columns

```
def fixTestFile(train_file,test_file):
  diff_train_columns = set(train_file.columns) - set(test_file.columns)
  diff_test_columns = set(test_file.columns) - set(train_file.columns)

  for test_column in diff_test_columns:
      for train_column in diff_train_columns:
          # Check if the columns share at least four letters
          if len(set(test_column).intersection(train_column)) >= 4:
              # Replace the column name in test_file
              test_file = test_file.rename(columns={test_column:
train_column})
              break  # Stop searching for similar columns once a match
is found

  # Add missing columns to test_file with values set to 0
  for train_column in diff_train_columns:
      test_file[train_column] = 0
  test_file = test_file.loc[:, ~test_file.columns.dupli-
cated(keep='first')]
  test_file = test_file.drop(columns= set(test_file.columns) -
set(train_file.columns))

  test_file.to_excel(PREDICT_PATH_PROCESSED)
  # print(train_file.shape,test_file.shape)
  return test_file.sort_index(axis=1)
```

separating features from target - and scaling

```
def seperateData(ds):
  if 'oscar winners' not in ds.columns:
    raise ValueError('Oscar winners not in the dataset')
  target=ds['oscar winners']
  data=ds.drop(columns='oscar winners')
  data=data.sort_index(axis=1)
  return (target,data)
```

```
def scaleData(trainData,predictData):
    scaler=MinMaxScaler()
    scaledTrainData=scaler.fit_transform(trainData)
    scaledPredictData=scaler.transform(predictData)
    return scaledTrainData,scaledPredictData
```

Model Training

Now that the data are ready we can proceed to train the model by splitting our train dataset in train and test data.

```
def doTraining(scaledTrainData,trainTarget,modelName='knn'):
    X_train, X_valid, y_train, y_valid = 
train_test_split(scaledTrainData,trainTarget, test_size=0.25,ran-
dom_state=42)
    if modelName=='rf':
        model = RandomForestClassifier(random_state=42)
    elif modelName=='lr':
        model = LogisticRegression(max_iter=1500, random_state=42)
    elif modelName=='dtc':
        model = DecisionTreeClassifier(random_state=42)
    elif modelName=='knn':
        model=KNeighborsClassifier(n_neighbors=3)
    model.fit(X_train, y_train)
    return model,X_train,X_valid, y_train, y_valid
```

after the splitting we try different models to see which one works better for our dataset

RANDOM FOREST

```
Confusion Matrix:
 [[329   1]
 [ 11   8]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       330
           1       0.89      0.42      0.57        19

    accuracy                           0.97       349
   macro avg       0.93      0.71      0.78       349
weighted avg       0.96      0.97      0.96       349

Accuracy: 0.9656160458452722
Cross validation: 0.9648924731182795
```

```
#Oscar winners:  6
Winner IDs: [45, 111, 133, 147, 149, 466]
PREDICTIONS COMPLETED. YOU CAN CHECK the results of the predictions at .\Data\predictions.csv/xlsx
```

DECISION TREE CLASSIFIER

```
Confusion Matrix:
 [[314  16]
 [ 10   9]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       330
           1       0.36      0.47      0.41        19

    accuracy                           0.93       349
   macro avg       0.66      0.71      0.68       349
weighted avg       0.94      0.93      0.93       349

Accuracy: 0.9255014326647565
Cross validation: 0.94842037890425
```

```
#Oscar winners:  211
```

K-NEAREST NEIGHBOURS

```
Confusion Matrix:
 [[326   4]
 [ 11   8]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       330
           1       0.67      0.42      0.52        19

    accuracy                           0.96       349
   macro avg       0.82      0.70      0.75       349
weighted avg       0.95      0.96      0.95       349

Accuracy: 0.9570200573065902
Cross validation: 0.9641756272401434
#Oscar winners:  5
Winner IDs: [27, 45, 111, 147, 370]
PREDICTIONS COMPLETED. YOU CAN CHECK the results of the predictions at .\Data\predictions.csv/xlsx
```

GRADIENT BOOSTER

```
Confusion Matrix:
 [[323   7]
 [  9  10]]
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98       330
           1       0.59      0.53      0.56        19

    accuracy                           0.95       349
   macro avg       0.78      0.75      0.77       349
weighted avg       0.95      0.95      0.95       349

Accuracy: 0.9541547277936963
Cross validation: 0.968469022017409
```

```
#Oscar winners:  12
Winner IDs: [59, 80, 94, 111, 133, 147, 149, 308, 406, 413, 466, 561]
PREDICTIONS COMPLETED. YOU CAN CHECK the results of the predictions at .\Data\predictions.csv/xlsx
```

LOGISTIC REGRESSION

Linear Regression and other algorithms like SVC are not suitable for our dataset as long as they are ill defined and they don't return any useful results.

```
Confusion Matrix:
 [[330   0]
 [ 19   0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       330
           1       0.00      0.00      0.00        19

    accuracy                           0.95       349
   macro avg       0.47      0.50      0.49       349
weighted avg       0.89      0.95      0.92       349

Accuracy: 0.9455587392550143
Cross validation: 0.9606016385048642
#Oscar winners:  258
```

As we can see the True Negative is 0 which means 0 oscar winners and that is why it is ill-defined, so not suitable.

Note:

As long as the training models' parameters is concerned, the models were tested in a variance of parameters. Different *random_states* were giving vey similar results but huge importance had the splitting the dataset to train and test data. In our case training the model with *test_size=0.25* had better results than training the model with other *test_sizes*.

B2.PREDICTIONS

In the models above we can see pretty similar results between some algorithms but at the end the model that is more useful for us is **gradient booster** and that is because it returns a high f-score a high recall, as long as a reasonable amount of Oscar winners. So **Gradient Booster** model is our selection to predict the oscar winners to unknown movies dataset.

f-score=0.56 - recall=0,42 – Accuracy=95 – Cross Validation=96

```
Confusion Matrix:
 [[323   7]
 [  9  10]]
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.98      0.98       330
           1       0.59      0.53      0.56        19

    accuracy                           0.95       349
   macro avg       0.78      0.75      0.77       349
weighted avg       0.95      0.95      0.95       349

Accuracy: 0.9541547277936963
Cross validation: 0.968469022017409
```

The predictions are located in **'Data/predictions.csv'** or **'Data/predictions.xlsx'**

C.CLUSTERING

STEPS

1. Load Dataset
2. Scale
3. Cluster
4. Plot Results

```
y,X,dataSet=loadDataset()
Xscaled=scale(X)
cl=Clustering(y=y,X=X,Xscaled=Xscaled,dataFrame=dataSet)
cl.executeClustering(alg=sys.argv[1],numClusters=int(sys.argv[2]))
print('FINISHED')
```

Load Dataset and Scale

```
def scale(X):
    if sys.argv[3] == 'ss':
        scaler = StandardScaler()
    elif sys.argv[3] == 'rs':
        scaler = RobustScaler()
    else:
        scaler = MinMaxScaler()
    Xscaled = scaler.fit_transform(X)
    return Xscaled

def loadDataset():
    dataset = pd.read_excel(TRAIN_PATH_PROCESSED, sheet_name='Sheet1')
    y, X = separateData(dataset)
    print(type(X))
    return y,X,dataset
```

Before clustering is very important our data to be scaled. The type of scaling is very important for the results of our clustering.

Clustering

```
def executeClustering(self,alg='KM',numClusters=2):
        # Perform PCA on the scaled data
        pca = PCA(n_components=2)
        Xnew = pca.fit_transform(self.Xscaled)
        print("PCA Components:")
        print(pca.components_)
        y=pd.DataFrame(self.y)
        print('EXECUTING: ')
        if alg=='KM':
            clusteringAlgorithm = KMeans(n_clusters=numClusters,
n_init='auto', random_state=42)
            print('KMeans')
        elif alg=='HAC':
            clusteringAlgorithm = AgglomerativeClustering(n_clus-
ters=numClusters)
            print('HAC')
        elif alg=='DBSCAN':
            eps_value = 0.5
            clusteringAlgorithm = DBSCAN(eps=eps_value, min_sam-
ples=numClusters)
            print('DBSCAN')
        else:
            clusteringAlgorithm = Birch(n_clusters=numClusters)
            print('Birch')
        clusterLabels = clusteringAlgorithm.fit_predict(Xnew)
        # Add cluster labels in the dataframe
        dfCluster = pd.DataFrame(clusterLabels, columns=['cluster'])
        dfPCA = pd.DataFrame(Xnew, columns=["PC1", "PC2"])
        dfClass = pd.DataFrame(self.y, columns=['oscar winners'])
        dfAllClusters = pd.concat([dfPCA, dfClass, dfCluster], axis=1)
        pc1Osc,pc2Osc=self.printStats(y,dfAllClusters,Xnew,self.X,clus-
teringAlgorithm,numClusters,pca,alg)
        self.plotDiagrams(dfAllClusters,Xnew,pc1Osc,pc2Osc,numClusters)
```

Before doing clustering another important step is to find the two most important principal components. After that we do clustering according to them and our target feature (oscar winners).

Plot Results

```
def plotDiagrams(self,dfAllClusters,Xnew,pc1Osc,pc2Osc,numClusters):
        fig, axs = plt.subplots(1, 2, figsize=(15, 5))

        # Plot the silhouette score comparing to Fowlkes-Mallows score
        xs = range(2, 50)
        sils = []
        fms = []

        for i in xs:
            clusteringAlgorithm = AgglomerativeClustering(n_clusters=i)
            clusterLabels = clusteringAlgorithm.fit_predict(Xnew)
            sils.append(metrics.silhouette_score(Xnew, clusterLabels, 
metric='euclidean'))
            fms.append(metrics.fowlkes_mallows_score(dfAllClusters['os-
car winners'], clusterLabels))

        axs[0].plot(xs, sils)
        axs[0].plot(xs, fms)
        axs[0].set_xlabel('Number of clusters (k)')
        axs[0].set_ylabel('Score')
        axs[0].set_title('Silhouette and Fowlkes-Mallows Scores')
        axs[0].legend(['Silhouette', 'Fowlkes-Mallows'])

        # Plot the Clusters PC1=x PC2=y
        for i in range(numClusters):
            clusterDF = dfAllClusters[dfAllClusters['cluster'] == i]
            axs[1].scatter(clusterDF['PC1'], clusterDF['PC2'], la-
bel=f'Cluster {i + 1}')
        axs[1].scatter(pc1Osc, pc2Osc, label=f'Oscars', color='red')
        axs[1].set_xlabel("PC1")
        axs[1].set_ylabel("PC2")
        axs[1].set_title("Clusters in PC1-PC2 Space")
        axs[1].legend()

        plt.tight_layout()
        plt.show()



    def printStats(self,y,dfAllClusters,Xnew,X,clusteringAlgo-
rithm,numClusters,pca,alg):
```

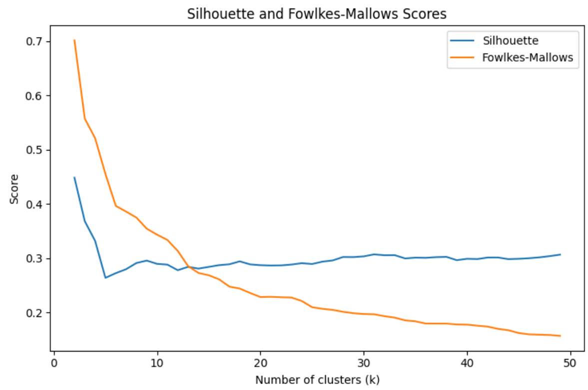
```
        # Print clustering stats
        print("Confusion Matrix:")
        print(confusion_matrix(dfAllClusters['oscar winners'], dfAll-
Clusters['cluster']))
        print("Calinski-Harabasz Score:", metrics.ca-
linski_harabasz_score(Xnew, dfAllClusters['cluster']))
        print("Silhouette Score:", metrics.silhouette_score(Xnew,
dfAllClusters['cluster'], metric='euclidean'))

        # Explore cluster characteristics (centroid values)
        if alg!='KM':
            centroids = dfAllClusters.groupby('cluster').mean().values
        else:
            centroids = clusteringAlgorithm.cluster_centers_

        for i in range(numClusters):
            print(f"Cluster {i + 1} Centroid Values:")
            print("PC1:", centroids[i, 0])
            print("PC2:", centroids[i, 1])
            print()

        featureImportancePC1 = pca.components_[0]
        featureImportancePC2 = pca.components_[1]

        print("Top features contributing to PC1:")
        topFeaturesPC1 = X.columns[np.argsort(featureImportancePC1)[::-
1]][:5].tolist()
        print(topFeaturesPC1)

        print("\nTop features contributing to PC2:")
        topFeaturesPC2 = X.columns[np.argsort(featureImportancePC2)[::-
1]][:5].tolist()
        print(topFeaturesPC2)

        # Loop through the clusters and print information for points
with prediction equal to 1
        predictionOneIds = getRowsWithPredictionOne(y)
        pc1Osc = []
        pc2Osc = []
        for index, row in dfAllClusters.iterrows():
            pointId = index
            pc1Value = row['PC1']
            pc2Value = row['PC2']
            clusterLabel = row['cluster'] + 1
```

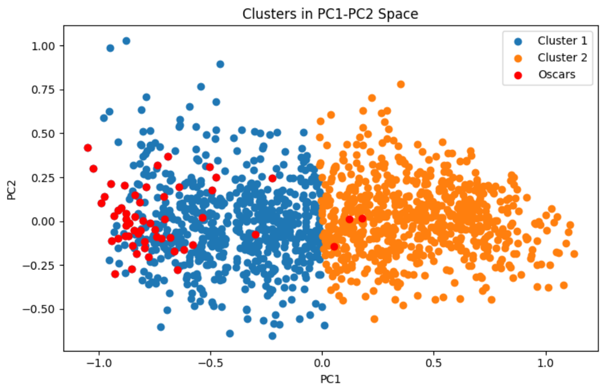
```
        if pointId in predictionOneIds:
            pc1Osc.append(pc1Value)
            pc2Osc.append(pc2Value)

    # centroids = clusteringAlgorithm.cluster_centers_
    # Display top features contributing to each cluster
    print("Top features contributing to each cluster:")
    for i in range(numClusters):
        print(f"Cluster {i + 1}:")
        topFeaturesIndices = np.argsort(centroids[i])[::-1][:7]
        topFeatures = X.columns[topFeaturesIndices].tolist()
        print(topFeatures)

    return pc1Osc,pc2Osc
```

Different algorithms with different scaling will provide different results.

KMEANS

MinMaxScaler



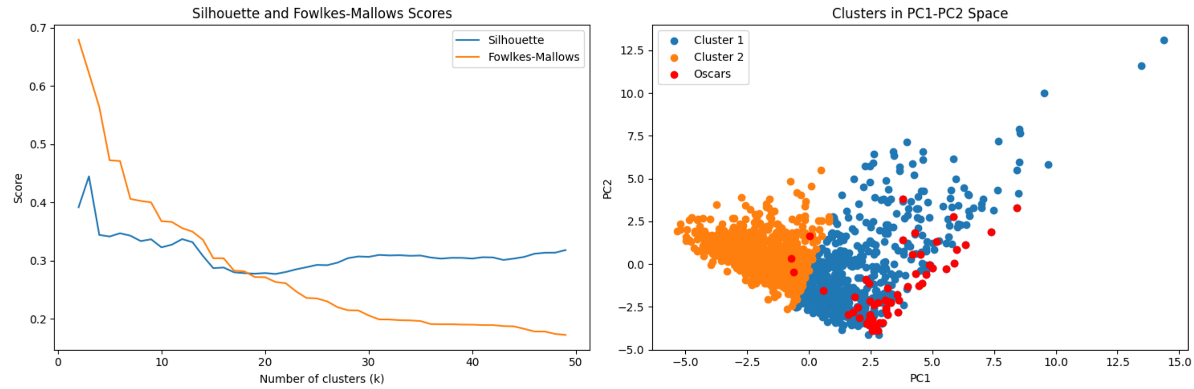



```
KMeans
Confusion Matrix:
[[642 699]
 [ 52   3]]
Calinski-Harabasz Score: 2084.3617525235445
Silhouette Score: 0.4886135763689926
Top features contributing to PC1:
['foreign gross ($million)', 'of gross earned abroad', 'imdb rating', 'foreign gross', 'worldwide gross ($million)']

Top features contributing to PC2:
['rotten tomatoes audience', 'opening weekend ($million)', 'domestic gross', 'average audience', 'worldwide gross']
Top features contributing to each cluster:
Cluster 1:
['metacritic critics', 'rotten tomatoes critics']
Cluster 2:
['rotten tomatoes critics', 'metacritic critics']
FINISHED
```

Comment:

Clustering here seems to work well as long as most of the oscar winners except from 3 are on the same cluster. Shilhouette score is good.

StandardScaler

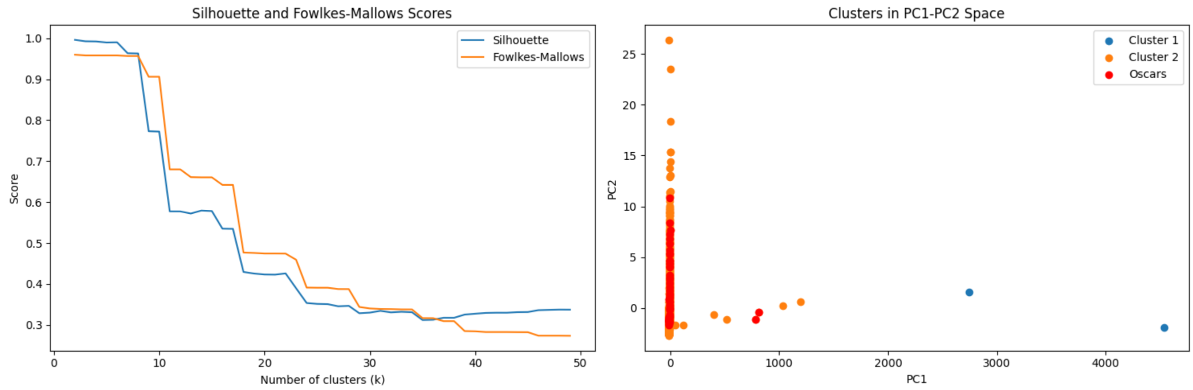



```
KMeans
Confusion Matrix:
[[608 733]
 [ 52   3]]
Calinski-Harabasz Score: 957.8909833622955
Silhouette Score: 0.4102691299296032
Top features contributing to PC1:
['average audience', 'average critics', 'rotten tomatoes critics', 'rotten tomatoes audience', 'metacritic critics']

Top features contributing to PC2:
['worldwide gross', 'opening weekend ($million)', 'worldwide gross ($million)', 'foreign gross', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'metacritic critics']
Cluster 2:
['metacritic critics', 'rotten tomatoes critics']
FINISHED
```

Comment:

Clustering here seems to work also well as long as most of the oscar winners except from 3 are on the same cluster. Shilhouette score is good.

RobustScaler



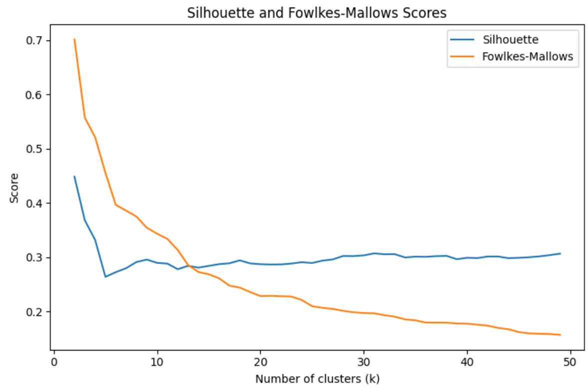

```
KMeans
Confusion Matrix:
[[   2 1339]
 [   0   55]]
Calinski-Harabasz Score: 6266.974569477474
Silhouette Score: 0.9961892714764893
Top features contributing to PC1:
['foreign gross ($million)', 'average audience', 'metacritic critics', 'rotten tomatoes audience', 'metacritic audience']

Top features contributing to PC2:
['worldwide gross', 'worldwide gross ($million)', 'foreign gross', 'opening weekend ($million)', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'metacritic critics']
Cluster 2:
['metacritic critics', 'rotten tomatoes critics']
FINISHED
```

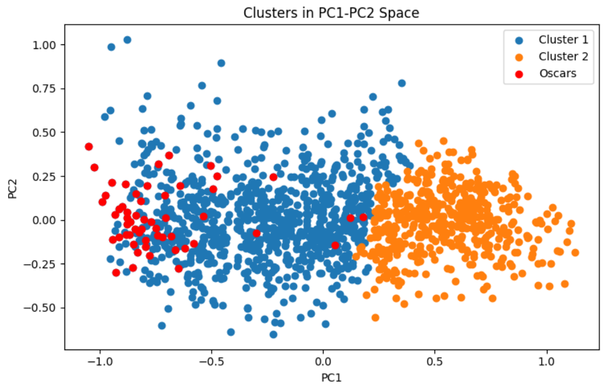
Comment:

Clustering here is not working well despite the fact that Silhouette score is high.

AGGLOMERATIVE CLUSTERING (HAC)

MinMaxScaler



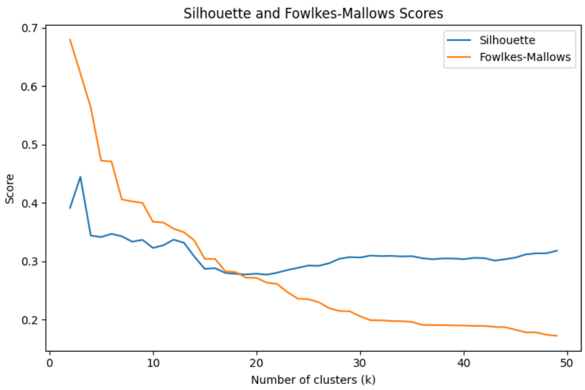


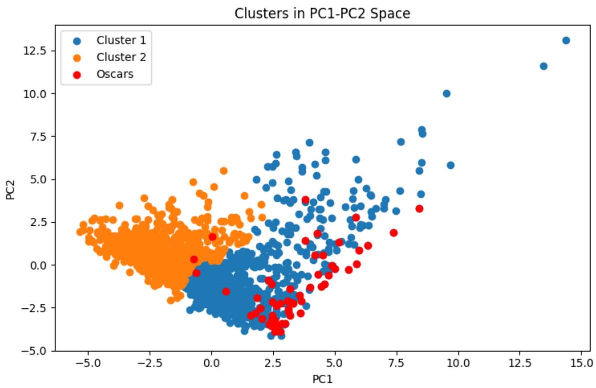

```
HAC
Confusion Matrix:
[[859 482]
 [ 55   0]]
Calinski-Harabasz Score: 1637.2076841641754
Silhouette Score: 0.44840625858757355
Top features contributing to PC1:
['foreign gross ($million)', 'of gross earned abroad', 'imdb rating', 'foreign gross', 'worldwide gross ($million)']

Top features contributing to PC2:
['rotten tomatoes audience', 'opening weekend ($million)', 'domestic gross', 'average audience', 'worldwide gross']
Top features contributing to each cluster:
Cluster 1:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
Cluster 2:
['rotten tomatoes critics', 'average critics', 'metacritic critics']
FINISHED
```

Comment:

Clustering here seems to work even better as long as allof the oscar winners in the same cluster. Shilhouette score is good.

StandardScaler



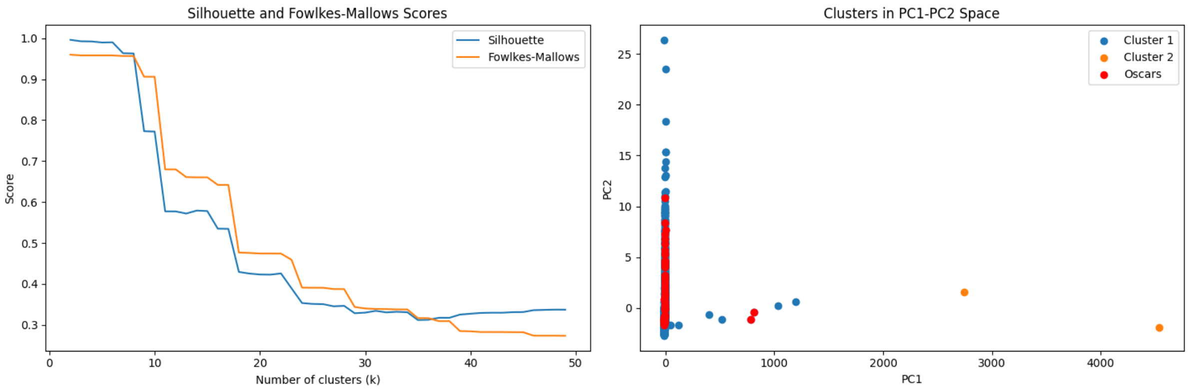



```
HAC
Confusion Matrix:
[[693 648]
 [ 53   2]]
Calinski-Harabasz Score: 885.1865404758329
Silhouette Score: 0.3914789852677304
Top features contributing to PC1:
['average audience', 'average critics', 'rotten tomatoes critics', 'rotten tomatoes audience', 'metacritic critics']

Top features contributing to PC2:
['worldwide gross', 'opening weekend ($million)', 'worldwide gross ($million)', 'foreign gross', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'average critics', 'metacritic critics']
Cluster 2:
['metacritic critics', 'average critics', 'rotten tomatoes critics']
FINISHED
```

Comment:

Clustering here seems to work well as long as most of the oscar winners, except from 2, are on the same cluster. Shilhouette score is good.

RobustScaler




```
HAC
Confusion Matrix:
[[1339    2]
 [  55    0]]
Calinski-Harabasz Score: 6266.974569477474
Silhouette Score: 0.9961892714764893
Top features contributing to PC1:
['foreign gross ($million)', 'average audience', 'metacritic critics', 'rotten tomatoes audience', 'metacritic audience']

Top features contributing to PC2:
['worldwide gross', 'worldwide gross ($million)', 'foreign gross', 'opening weekend ($million)', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
Cluster 2:
['rotten tomatoes critics', 'average critics', 'metacritic critics']
FINISHED
```

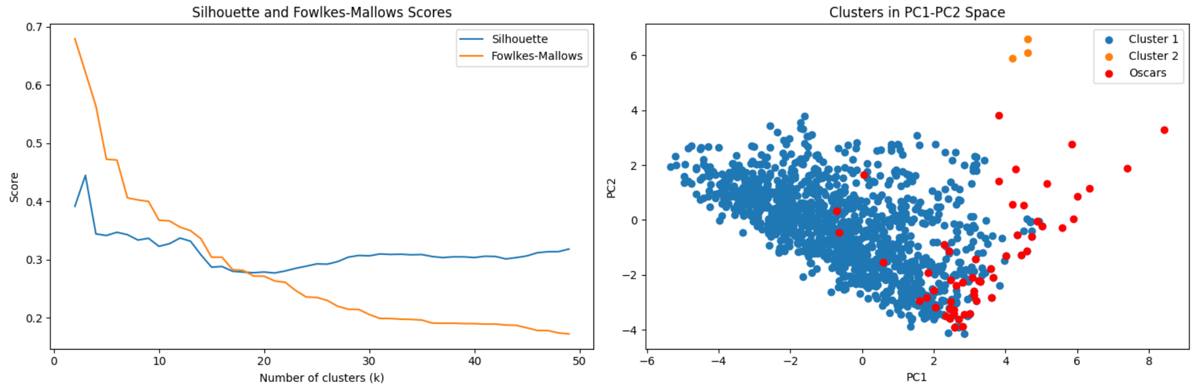
Comment:

Clustering here is not working well despite the fact that Silhouette score is high.

DBSCAN

MinMaxScaler(not applicable in DBSCAN)

StandardScaler

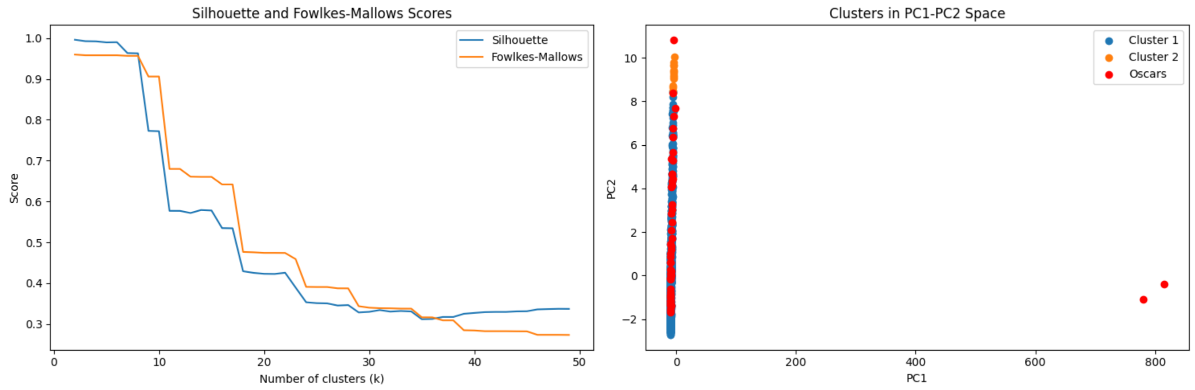



```
Calinski-Harabasz Score: 43.08638071913865
Silhouette Score: 0.10282909241003808
Top features contributing to PC1:
['average audience', 'average critics', 'rotten tomatoes critics', 'rotten tomatoes audience', 'metacritic critics']

Top features contributing to PC2:
['worldwide gross', 'opening weekend ($million)', 'worldwide gross ($million)', 'foreign gross', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['metacritic critics', 'rotten tomatoes critics', 'average critics']
Cluster 2:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
FINISHED
```

Comment:

Clustering here is not working well because Silhouette score is low and Oscars seem to be not in the same cluster.

RobustScaler



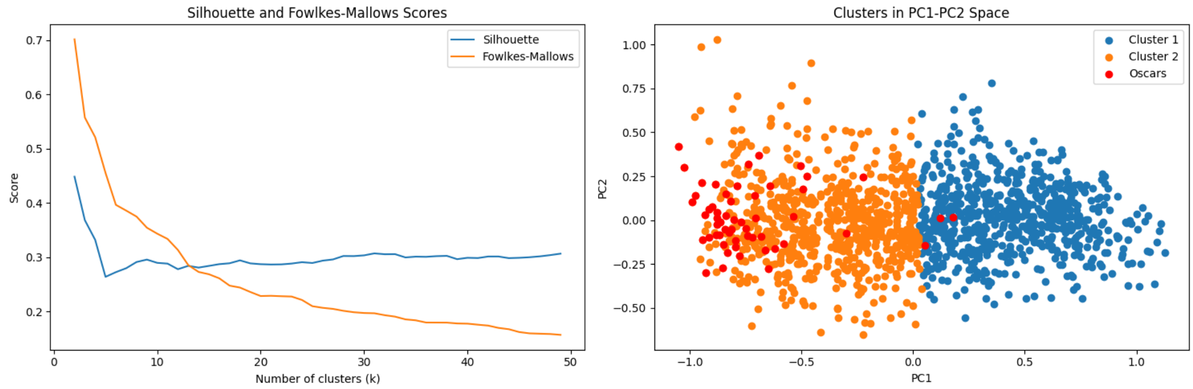

```
DBSCAN
Confusion Matrix:
[[   0    0    0    0    0    0    0]
 [  35 1284   13    2    2    3    2]
 [   5   47    1    1    0    1    0]
 [   0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0]]
Calinski-Harabasz Score: 30.379422989709216
Silhouette Score: 0.5214747626658898
Top features contributing to PC1:
['foreign gross ($million)', 'average audience', 'metacritic critics', 'rotten tomatoes audience', 'metacritic audience']

Top features contributing to PC2:
['worldwide gross', 'worldwide gross ($million)', 'foreign gross', 'opening weekend ($million)', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'metacritic critics', 'average critics']
Cluster 2:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
FINISHED
```

Comment:

Clustering here is not working well despite the fact that Silhouette score is good.

BIRCH

MinMaxScaler

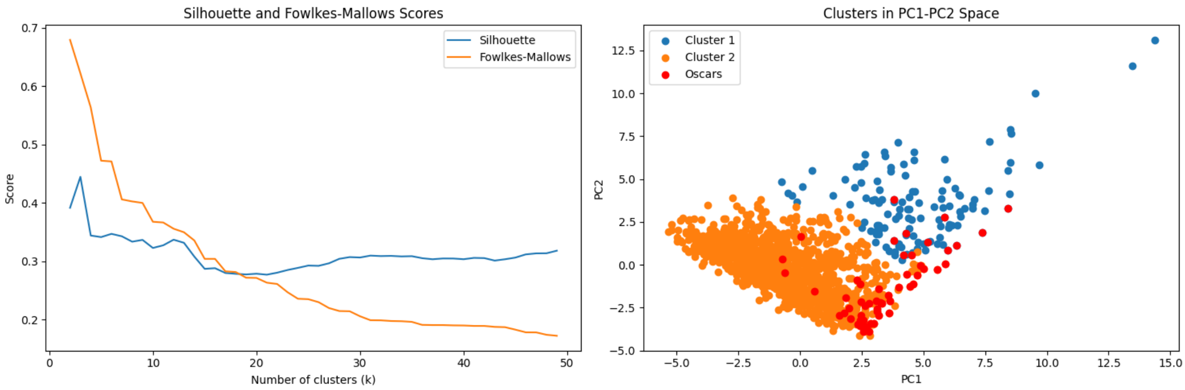



```
Birch
Confusion Matrix:
[[670 671]
 [  3  52]]
Calinski-Harabasz Score: 2080.6073320331116
Silhouette Score: 0.4883898547669409
Top features contributing to PC1:
['foreign gross ($million)', 'of gross earned abroad', 'imdb rating', 'foreign gross', 'worldwide gross ($million)']

Top features contributing to PC2:
['rotten tomatoes audience', 'opening weekend ($million)', 'domestic gross', 'average audience', 'worldwide gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'metacritic critics', 'average critics']
Cluster 2:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
FINISHED
```

Comment:

Clustering here seems to work well as long as most of the oscar winners, except from 3, are on the same cluster. Shilhouette score is good.

StandardScaler

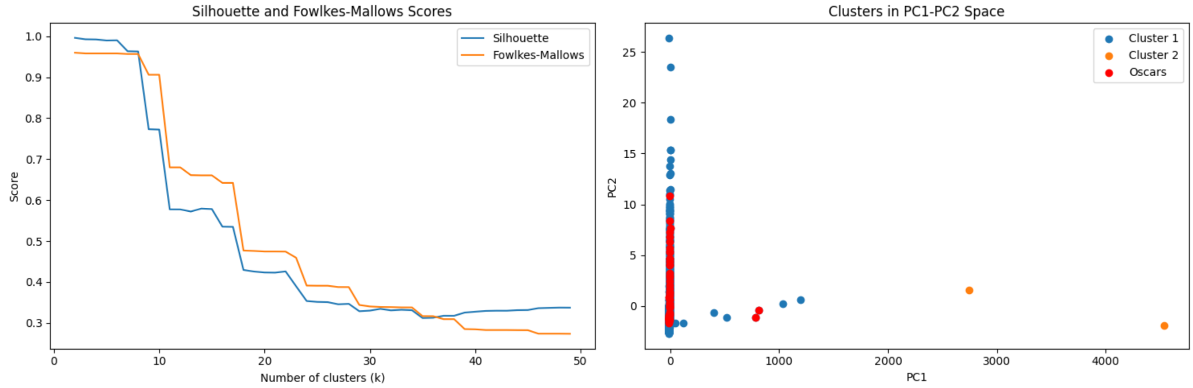



```
Birch
Confusion Matrix:
[[ 114 1227]
 [  10   45]]
Calinski-Harabasz Score: 666.2026707946574
Silhouette Score: 0.5283997583434148
Top features contributing to PC1:
['average audience', 'average critics', 'rotten tomatoes critics', 'rotten tomatoes audience', 'metacritic critics']

Top features contributing to PC2:
['worldwide gross', 'opening weekend ($million)', 'worldwide gross ($million)', 'foreign gross', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['rotten tomatoes critics', 'metacritic critics', 'average critics']
Cluster 2:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
```

Comment:

Clustering here is not working so well as long as most of the oscar winners, except from 2, are on the same cluster. Silhouette score is good.

RobustScaler




```
Birch
Confusion Matrix:
[[1339    2]
 [  55    0]]
Calinski-Harabasz Score: 6266.974569477474
Silhouette Score: 0.9961892714764893
Top features contributing to PC1:
['foreign gross ($million)', 'average audience', 'metacritic critics', 'rotten tomatoes audience', 'metacritic audience']

Top features contributing to PC2:
['worldwide gross', 'worldwide gross ($million)', 'foreign gross', 'opening weekend ($million)', 'domestic gross']
Top features contributing to each cluster:
Cluster 1:
['average critics', 'metacritic critics', 'rotten tomatoes critics']
Cluster 2:
['rotten tomatoes critics', 'average critics', 'metacritic critics']
FINISHED
```

Clustering here is not working well despite the fact that Silhouette score is good.

Conclusion for Clustering:

As you notice it is very important the way of scaling the dataset. The reason we use 2 clusters in all the algorithms is because silhouette score has the highest value with 2 clusters (see blue line in figures). The clustering seems to be good in HAC and Kmeans as long as most of the Oscars are in the same cluster, but we did not make it to characterize each cluster according to their features, because cluster1 and cluster2 in all cases have the same features as most important.

CLOSING

.readme

Install necessary packages

```
pip install -r requirements.txt
```

To run preprocessing and classification

```
FORMAT: python3 main.py <lr/knn/rf/dtc/gb> <stats/nostats> <prepro/no-prepro> <corel/nocorel>
```

```
python3 main.py gb stats prepro corel
```

To run classification (without doing preprocessing again)

```
python3 main.py gb stats noprepro
```

To run clustering

```
python3 clusteringMain.py <KM/DBSCAN/HAC/Birch > <numClusters> <rs/ss/mm >
```

```
python3 clusteringMain.py HAC 2 mm
```

To run the final and correct model use this command for the predictions I uploaded:

```
python3 main.py gb stats prepro corel
python3 clusteringMain.py HAC 2 mm
```

NOTE:

For preprocessing is recommended to process the file **moviesUpdated.xlsx** because it is already filled with the imdb rating missing values.

In the file **movies.xlsx** imdb rating missing values are missing in all the rows so it will take a lot of time to fill all the missing values .

moviesUpdated.xlsx is a copy of **movies.xlsx** with the only difference that **moviesUpdated.xlsx** has also the IMDb ratings column ready.

Links

GitHub: https://github.com/Kokkales/Oscars---Data-Mining