Statistical Machine Learning Mini Project 2022 - Do (wo)men talk too much in films?

Dap De Bruijckere
Engineering physics

Felix Dryselius
Social techincal systems engineering

Vidar Lundgren
Engineering physics

Irina Zarankina
Financial mathematics

Abstract

This project studies gender inequality in films and tries to create a classification model that can predict gender of the main character, given some data. In order to achieve this, multiple classification methods have been tuned and evaluated. In the end, QDA was chosen as the best model to predict gender of lead character.

1 Data analysis task

Questions to answer:

- Do men or women dominate speaking roles in Hollywood movies?
- Has gender balance in speaking roles changed over time (i.e. years)?
- Do films in which men do more speaking make a lot more money than films in which women speak more?

1.1 Answers

The data comes from the Film dialog data set created by Hanah Anderson and Matt Daniels in 2016 link. The full set contains 2000 entries but in this article a sub-set of 1039 randomly selected entries were used. This set is skewed with almost all features having a positive skewness value. In total skewness ranged from -1.27 to 3.79 and this was calculated using pandas.dataFrame.skew() function (6). The data is also imbalanced, with 785 points having class male and 254 having female. This means that a model that only predicts male will have a misclassification error of circa 24% ($\frac{254}{1039}$).

As visible from 1 gender imbalance is present in Hollywood. Women have both fewer roles and speak fewer words on average per movie for all years in the data set except one. The only exception is during 1958 when the proportion of words spoken by females exceeds those of men. This year might be an outlier though, because the data set is heavily skewed, with the mean of sampled films being in the year 1999.

Gender imbalance seems to improve over time though, and interestingly, movie gross (money earned) does not seem to correlate with male or female dominance.

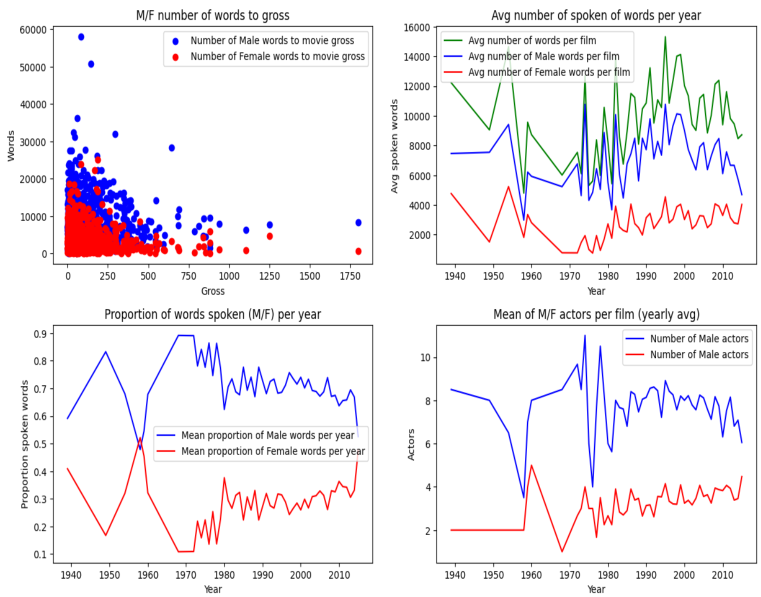

Figure 1: Inequality over time

2 Description of methods used

2.1 Logistic regression

Logistic regression can be seen as a modification of the linear regression model so it can be applied to a classification problem. This modification is obtained by using the logistic function to model a binary dependent variable based on some input variables. More specifically, let $g(x)$ be a function that approximates the conditional probability of the positive class, $g(x) = \frac{\epsilon^{\theta^T x}}{1+\epsilon^{\theta^T x}}$, where $\theta^T x$ is the linear regression. Thus, the linear regression can be "squeezed" into an interval [0,1] by using the logistic function (1). Then numerical optimisation is applied for learning the parameters θ of the model.

2.2 Discriminant analysis

The discriminant analysis methods: Linear Discriminant Analysis (LDA), and Quadratic Discriminant Analysis (QDA), are based on fitting a gaussian probability density to each class. For LDA, the same covariance matrix is assumed for each class, giving linear decision boundaries. QDA, as the name suggests, gives quadratic decision boundaries instead, due to the omission of the aforementioned assumption of covariance matrices being the same.

2.3 K-nearest neighbour

kNN is a distance-based method, that in classification, predicts class of a new data point by taking the majority vote from the k-nearest data points in the training set. It is a non-parametrized method and data must be normalized before use 8.

2.4 Tree-based methods

Tree-based methods work by splitting the data with as little error as possible. This was done by using gini index, which divides the data by creating the lowest amount of impurities on both sides. Because of that any redundant or useless data does not affect the accuracy of the method, but affects the runtime. To determine the best depth cross validation was used. Also when finding the minimum sample size, the minimum size that a node needs to be to allow it to be split, cross validation was used. To visualize the tree and see which data was used, graphviz was implemented.

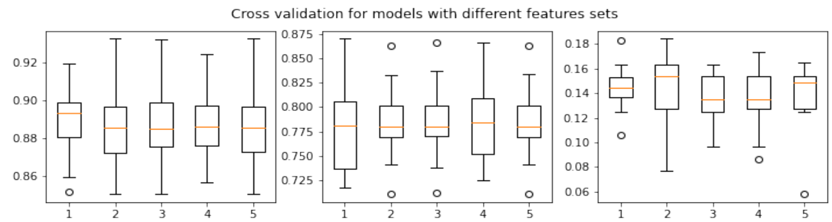

Figure 2: ROC AUC, PC AUC and error of 5 best log regression models

3 How the methods were applied to the data

3.1 Logistic regression

The base logistic regression model was implemented on the given data using scikit-learn package. The data was scaled using the 'StandardScaler' from the 'sklearn' library, since regularization works better when data is standardized. Evaluation metrics ROC AUC, PC AUC and misclassification error were used while tuning and evaluating performance.

At first, some assumptions were tested manually to tune the parameters. The data was divided into training and validation sets and a random seed was fixed to provide comparability and reproducibility of experiments. It was discovered that a 'liblinear' solver gave the best performance. Using parameter 'class weight' of the logistic regression function didn't improve the results.

After testing different subsets of features as input variables to the model (brute-force approach using library 'itertools'), it became clear that models with 8 and 9 input variables gave the best values of evaluation metrics. Thus, complexity of the models was reduced, which resulted in the improved performance of the models. (More details will follow in feature importance section) Then the 5 models with the best performance (ROC AUC) were compared using 10-fold cross-validation based on evaluation metrics: ROC AUC, PC AUC, misclassification error (the threshold rate was chosen comparing f1 coefficients).

As a result, the model number 3 was chosen (see the Figure 2 above). Then 10-fold cross-validation using randomized search was performed to determine if regularization l1 or l2 is needed. Cross-validation was also used to find regularization coefficient (the interval from 0.0001 to 100 of possible values of parameter C was checked, which is an inverse of the regularization parameter λ (1), ($C = 1/\lambda$) and threshold rate.

The chosen logistic regression model had the following features:

The list of features: ['Total words', 'Age Co-Lead', 'Number of female actors', 'Difference in words lead and co-lead', 'Number of words lead', 'Age Lead', 'Number words female', 'Number of male actors'] 'C': 18.31792545756584, 'penalty': 'l1', 'solver': 'liblinear', threshold rate 0.435

3.2 Discriminant analysis

Discriminant analysis does not have any hyperparameters that can be tuned, which can work as either an advantage or disadvantage. This means the implementation is fairly straight forward, and that the method will often perform well right away, as has proven to be the case in practice. (2)

Using the important features discussed in section 3.1 was the only way the methods were tuned to the problem. As some features were collinear, this greatly improved QDA performance in particular.

3.3 K-nearest neighbour

Application of kNN to the data was done in three steps: analysis and normalization of the data, decision of hyper-variables (k) and estimation of E[Accuracy_new], and estimation of other evaluation terms. For results see 4.2.

Normalization of the data *(all features except "Lead" was interpreted as quantitative)* was performed by using sklearn's StandardScaler method (5). This assumes that all features behave like a random

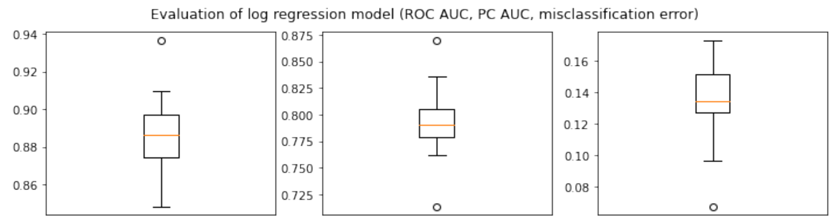

Figure 3: ROC AUC, PC AUC and error of 5 best logistic regression models

variable with normal distribution, and it normalizes each feature with mean 0 and standard deviation 1. Another re-scaler, "MinMaxScaler", was also evaluated but not used, because no significant performance difference was observed.

kNN was implemented using sklearn's KNeighborsClassifier with uniform weights, Euclidian distance, and the 'distance calculation algorithm' set to auto (in this generally low dimensional problem, the "KD Tree algorithm" was probably automatically chosen). This was done to reduce solution complexity and minimize calculation time (4). kNN was also evaluated using distance based weights but this was not implemented due to reduced performance. When plotting the missclassification error against the value of "k" with distance based weights it became clear that kNN only performed well for a specific k-value. This indicated that the model with distance weights would risk being over-fitted in a production environment, see 8.

For the evaluation of kNN, the whole data set was used with no splitting in a test and train data set. This was done after realising that performance improved considerably when using all 1039 entries. For calculations and estimations, cross validation was used instead (1). Cross validation was also used to estimate other performance data than missclassification error by taking the average or sum of the "evaluation term" in each fold.

The value of "k" was set in order to minimize missclassification error and was chosen from a range of 1-80. This range was analysed using cross-validation with 200 folds and the results can be seen in table 4.2. These ranges were capped at these values when the results started converging. Because the data set was imbalanced, evaluation was also performed with different thresholds. One tested the threshold was r=24% for positive class ('Male') and these results are also visible in 4.2. This threshold was chosen because it mimics the statistical chance of randomly selected data point having class ('Female').

3.4 Tree-based methods

The tree-based model was implemented using the 'DecisionTreeClassifier' from the 'sklearn' library. This uses the Gini index to partition the data several times until the depth is reached or the minimum amount of samples in a node is too low. The Gini index is calculated as $G = \Sigma_{i=1}^{2} p(i) * (1 - p(i))$, where G is the Gini index and p is the probability that it is a male or a female lead. This is done with all the possible places to split and with all the data. The combination with the lowest Gini index is used. Instead of the Gini index entropy can be used, but it wont make a difference in the accuracy of the tree. To determine both the best depth and the minimum sample a 40 fold cross validation was used 10 times. the range of depth to be evaluated was between 2 and 15 and the minimum sample was between 1 and 40.

4 Performance evaluation

4.1 Logistic regression

Results of the model's performance evaluation using 10-fold cross validation are shown in table 1.

4.2 K-nearest neighbour evaluation

The results from the model are shown in table 4.2. As is visible, the model tends to over-predict on the positive class (Male) with the FPR being around 73%, however do to the imbalance in the data

Table 1: Results from logistic regression evaluation

Term	Value
ROC AUC	0.887
PC AUC	0.792
TPR	0.675
FPR	0.068
E[Accuracy_new]	0.867
F1	0.712

set this does not translate to a high missclassification rate. Note also that changing the threshold does not translate into a similar change of the results, this is somewhat expected given that kNN is a non-parametrized model. For a full table and graphs please see A and B.

Table 2: Results from kNN evaluation with two thresholds

Term (r=50%)	Values
Optimal_k	16
TPR	0.971975
FPR	0.728346
E[Accuracy_new]	0.80077
F1	0.880554

Term (r=24%)	Values
Optimal_k	5
TPR	0.950318
FPR	0.700787
E[Accuracy_new]	0.791145
F1	0.873025

Table 3: Discriminant Analysis Confusion Matrices

LDA	Term
Accuracy	0.860
TPR	0.975
FPR	0.497

QDA	Term
Accuracy	0.888
TPR	0.868
FPR	0.227

4.3 Discriminant analysis

Because there is no tuning in LDA or QDA, the methods could not be evaluated using cross-validation. Instead, 200 runs were performed with random splits of the test and training data for each run. The accuracy was estimated by the mean average accuracy of the 200 runs. Please see 3 for results.

4.4 Tree-based method

Cross validation showed that the best depth was either 8 or 9 with the average being 8.6. It also showed that the minimum sample size varied from 7 to 15,

Table 4: Results from tree-based method evaluation

Term	Value
TPR	0.862
FPR	0.399
E[Accuracy_new]	0.805
missclassification	0.204
Best depth	8.6
Best min sample size	12.1

4.5 Gini importance

The Gini importance looks at how many times each feature has been used to split a node, divided by all the split in the tree. figure(4). shows that "Number words female", "Number of female actors" and

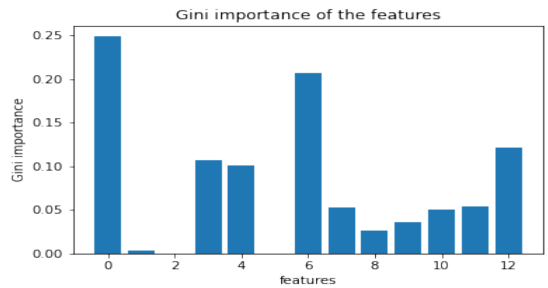

Figure 4: Inequality over time

"Age Co-Lead" are the most used and "Total words", "Number of words lead" and "Year" are not used.

5 Choice of method

Usually when picking which method to use, one would look at a comparison of different performance metrics for the available methods and weigh in how important each metric is for the problem at hand. For example, if you were working on a machine learning algorithm for identifying disease of some kind, one of the top concerns would be to minimise the false negative rate so not to risk the condition go untreated. For this project, the outcome of false negatives and false positives does not matter in the same way; therefore, our primary metric of choice is accuracy, $\frac{TN+TP}{n}$

In place of unseen data, cross validation (when applicable) has been used to estimate the accuracies in table 5. From the table it is evident that QDA gives the best results and should be used "in production".

Table 5: Accuracies

Method	Accuracy
Logistic Regression	0.867
LDA	0.860
QDA	0.888
Tree-based	0.804
k-NN	0.801

6 Conclusions

It is clear that model performance depends a lot on the features used. With the feature optimization performed, the best performing model is QDA: Quadratic Discriminant Analysis when studying accuracy, TPR and FPR. If a new test set was given we estimate that QDA would have an accuracy of 0.888.

7 Feature importance and selection

Based on the data analysis and correlation matrix between features (figure 5), the assumption was made that some variables would not be significant for the model.

Running logistic regression with different number of features showed the following – models with 8 and 9 features had the the highest ROC-AUC, PC-AUC and misclassification error values.

ROC-AUC was chosen as the main metric based on following factors:

- During manual experimentation it was discovered that if one metric improves or degrades, so does the others in the majority of cases.
- It is easier to compare logistic regression models by ROC-AUC and PC-AUC as there is no need to take in account a threshold rate

- The data is imbalanced, but not severely imbalanced

As there could have been several models with close values of ROC-AUC, the decision was made to examine which features subsets gave the best average results (ROC-AUC) on 20 runs. As a result, 5 best subsets of features were selected: 3 with 8 features, 2 with 9 features.

The best subsets of features varied just by 1-2 features. The most important features are those, which were in all these subsets. More specifically, 'Number words female', 'Age Co-Lead''Number of female actors', 'Difference in words lead and co-lead', 'Number of words lead', 'Age Lead', 'Number of male actors'. 'Age Lead' was also a popular feature.

The feature 'Number words female' was included in all subsets of features, which gave the best ROC-AUC metric. Moreover, if we look at the models' coefficients, this factor had the highest coefficient value (approx. 2.0). The feature 'Number words male' was in two of the 5 feature subsets with best performance, and its coefficients were not that heavy, just approximately 0.6.

Such features as Year of release ('Year') and Money made by film ('Gross') were not included in the feature subsets at all. So, these factors give worse prediction than chosen features. They are not important as when we try the logistic regression model with all features, their weight coefficients are very low (<0.1), while 'Number words female' and 'Number words male' have higher coefficients.

The models with one variable didn't perform well, they didn't have true positives or even false positive at all, so they performed like the worst-case classifer, which always predicts the same output class. It has a misclassification error approx. 25 percent according to the proportion of female lead roles to the number of all movies.

8 Discussion

The model choice was based on evaluation using an accuracy metric. However, the models differ in various metrics, for example, TPR, FPR, F1. These metrics demonstrate the characteristics of the models that differ from accuracy.

In real life it would be right to choose an evaluation metric or combination of metrics depending on how the results of the prediction would be used.

For example, assume, we have a client, i.e. a person who will use the model. If the main reason of the prediction is to obtain the list of films with women leads, then TPR, precision (the proportion of TP to all predicted positives), and F1 metrics should be taken in account as well as accuracy. This is because the data is imbalanced, and we can have a case when true positives are 100% correct, but we have 13% misclassification rate. This is, in fact, would be the ratio of false positives to data. Thus, if we had 100 films, we would get 13 FP and 25 TP (given the data is imbalanced in proportion 1:3). I.e., every third film from the list with women leads wouldn't belong there.

Maybe it is not important how many films with men leads got in the list with women leads and the client cares just about TPR and accuracy.

Another case would occur if both lists of films with men and women leads should be used by the client and he/she doesn't care how many films with men leads got on the list with women leads. We would be interested in the accuracy and some balance between TPR and FPR in this case.

Probably, if the client wants just one list with films classified by the gender of the lead actor, the most important is to obtain the minimum misclassification rate.

Since the project description does not specify how the model's predictions will be used, accuracy was chosen as a "safe" bet. However, if TPR and FPR are considered, the best model could be log regression as it has low FPR (0.068), good values of TPR (0.675), and the accuracy (0.867).

References

[1] A. Lindholm, N. Wahlström, F. Lindsten, T. B. Schön, MACHINE LEARNING - *A First Course for Engineers and Scientists*, Draft version: April 30, 2021

[2] 1.2. Linear and Quadratic Discriminant Analysis, Scikit-learn.org. Retrieved February 19, 2022.

[3] Scikit-learn documentation, (KNeighborsClassifier) Scikit-learn.org. Retrieved February 21, 2022.

[4] Scikit-learn documentation, (Nearest Neighbors) Scikit-learn.org. Retrieved March 3, 2022.

[5] Scikit-learn documentation, (StandardScaler) Scikit-learn.org. Retrieved February 21, 2022.

[6] Pandas documentation, (DataFrame.skew) pandas.pydata.org. Retrieved February 21, 2022.

A Table Appendix

Table 6: Results from kNN evaluation with two thresholds

Term (r=50%)	Values
P	785
N	254
P_star	948
N_star	91
TN	69
FP	185
FN	22
TP	763
Optimal_k	16
TPR	0.971975
FPR	0.728346
E[Accuracy_new]	0.80077
E[Error_new]	0.2005
precision	0.804852
recall	0.971975
F1	0.880554

Term (r=24%)	Values
P	785
N	254
P_star	924
N_star	115
TN	76
FP	178
FN	39
TP	746
Optimal_k	5
TPR	0.950318
FPR	0.700787
E[Accuracy_new]	0.791145
E[Error_new]	0.2165
precision	0.807359
recall	0.950318
F1	0.873025

B Graph Appendix

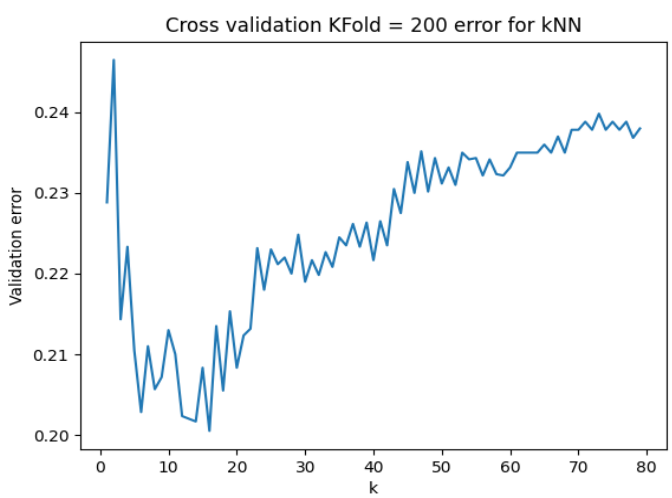

Figure 6: kNN, CV missclassification to number of 'k', r=50%

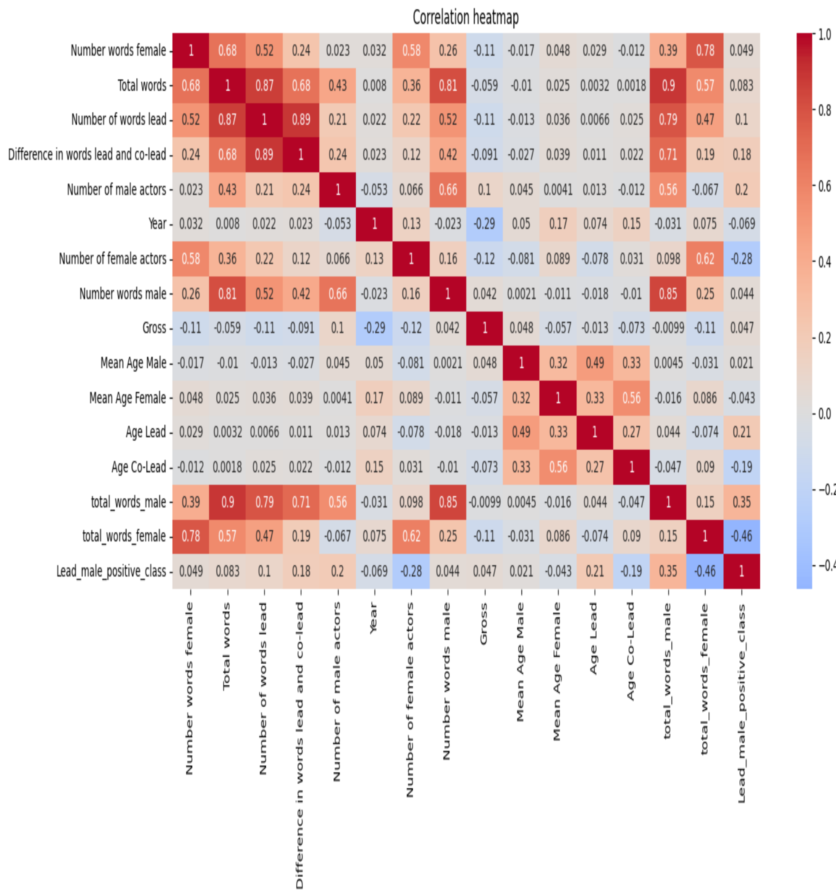

Figure 5: Feature correlation heatmap

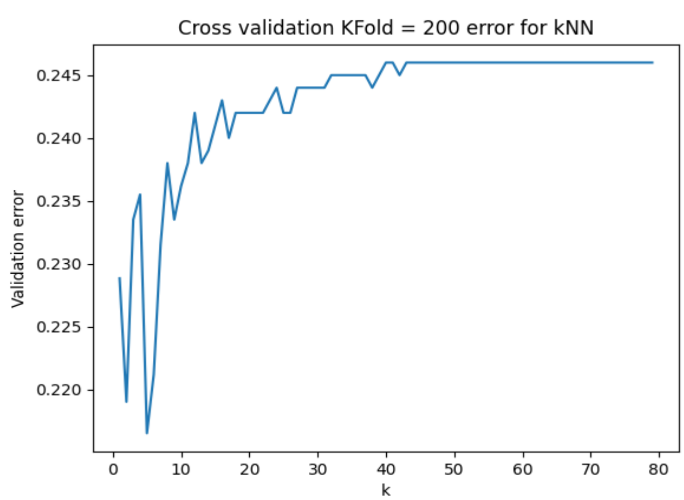

Figure 7: kNN, CV missclassification to number of 'k', r=24%

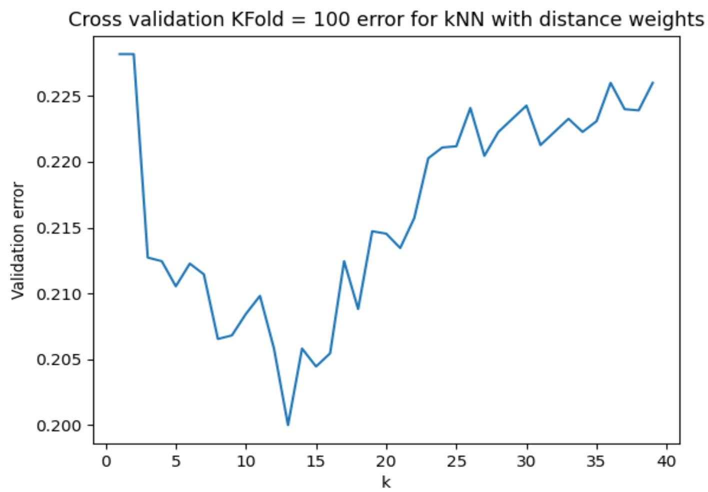

Figure 8: kNN, CV missclassification to 'k', distance weights, r=50%

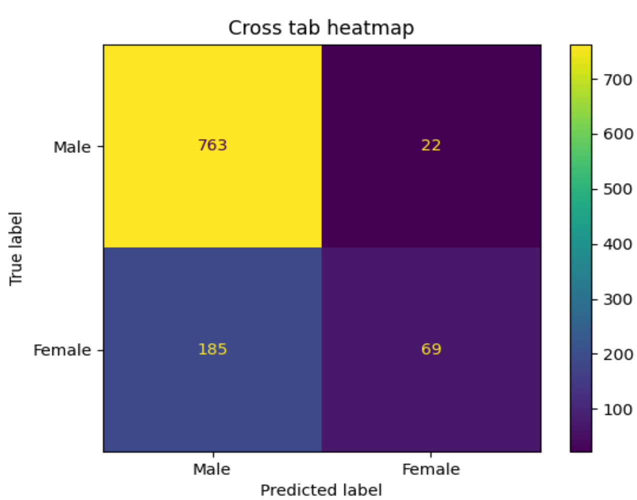

Figure 9: kNN, Crosstab heatmap, r=50%

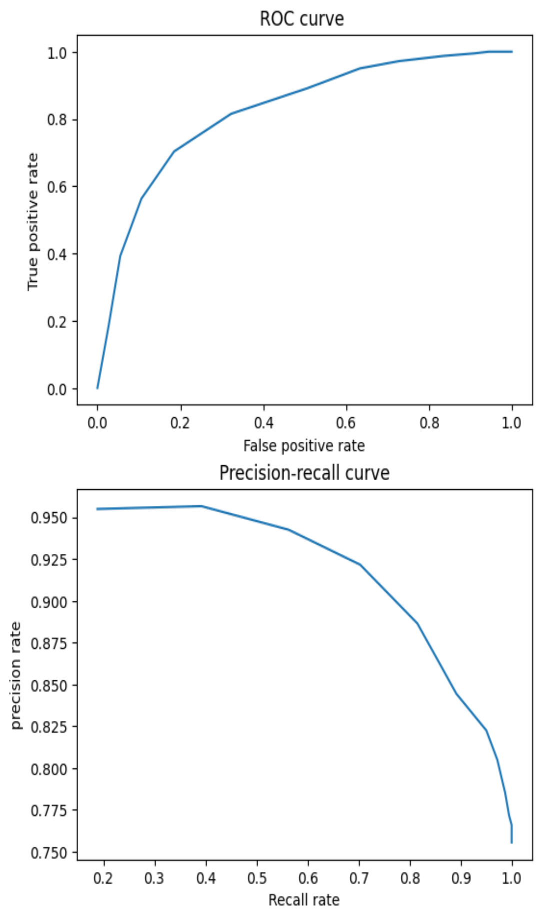

Figure 10: kNN, ROC and Precision-recall

C Code Appendix

C.1 Data equality plotting

```
# ------------------ Data analysis ------------------

# This program creates the plots to answers the task:
# "Data analysis task"


import pandas as pd
import matplotlib.pyplot as plt
import os

cwd = os.getcwd()
URI = (cwd+"\\train.csv")
film_data = pd.read_csv(URI, dtype={"Lead":str}).dropna().reset_index(
    drop=True)

# Creating dummy values of the category "Lead"
film_data_dummies = pd.get_dummies(film_data).copy()

# Adding two columns that summerizes the number
# of words M/F with the lead's words if  it is
# the same gender
film_data_dummies = film_data_dummies.assign(total_words_male=lambda
    row: (row['Number words male'] + row['Number of words lead'] * row
    ['Lead_Male']))
film_data_dummies = film_data_dummies.assign(total_words_female=lambda
     row: (row['Number words female'] + row['Number of words lead'] *
    row['Lead_Female']))


# Only saving "Lead_Male" as category and
# renaming it to "Lead"
film_data_dummies.drop("Lead_Female",1, inplace=True)
y = film_data_dummies['Lead_Male'].rename("Lead")

# Creating the features set without the category
x = film_data_dummies.drop(columns=['Lead_Male'])

# ------------------ Scatter plot ------------------
# Creates a scatter plot of movie earnings vs
# total number words M/F The aim is to determine
# if movie earnings correlates to M/F dominace

df = film_data_dummies.copy()

plt.subplot(2,2,1)
ax_male_word_gross = plt.scatter(df["Gross"], df["total_words_male"],
    color="b", label="Number of Male words to movie gross")
ax_female_word_gross = plt.scatter(df["Gross"], df["total_words_female
    "], color="r", label="Number of Female words to movie gross")
plt.xlabel("Gross")
plt.ylabel("Words")
plt.title("M/F number of words to gross")
plt.legend()


# ------------------ Line plot 1/3 ------------------
# Creates a line plot where the yearly mean of
# "Total words", total_words_female", and
# "total_words_male" are plotted against "Year".
# The aim is to discern if eqality has
# improved over time.
```

```

df = film_data_dummies.copy()

#average nr spoken words (M/F) per film per year
df_avg_male_words = df.groupby("Year")["total_words_male"].mean()
df_avg_female_words = df.groupby("Year")["total_words_female"].mean()
df_avg_total_words= df.groupby("Year")["Total words"].mean()
plt.legend()

plt.subplot(2,2,2)
df_avg_total_words_plot = df_avg_total_words.plot(kind="line", y="Total words", x="Year", color="g", label="Avg number of words per film")
df_avg_male_words_plot = df_avg_male_words.plot(kind="line", y="total_words_male", x="Year", color="b", label="Avg number of Male words per film", ax=df_avg_total_words_plot)
df_avg_female_words_plot = df_avg_female_words.plot(kind="line", y="total_words_female", x="Year", color ="r",label="Avg number of Female words per film", xlabel="Year", ylabel="Avg spoken words", title="Avg number of spoken of words per year", ax=df_avg_male_words_plot)
plt.legend()

# ------------------ Line plot 2/3 ------------------
# Creates a line plot where yearly mean of
# proportion of "total_words_female" and
# "total_words_male" to "Total words" is
# plotted against "Year".
# The aim is to discern if eqality has
# improved over time.

# Proportion of spoken words (M/F) per film per year
def proportion(x, y):
    x_new = x/(y)
    return x_new

df['Proportion female words'] = df.apply(lambda row : proportion(row['total_words_female'],row['Total words']), axis = 1)
df['Proportion male words'] = df.apply(lambda row : proportion(row['total_words_male'], row['Total words']), axis = 1)

df_male_words_proportion = df.groupby("Year")["Proportion male words"].mean()
df_female_words_proportion = df.groupby("Year")["Proportion female words" ].mean()

plt.subplot(2,2,3)
df_male_words_proportion_plot = df_male_words_proportion.plot(kind="line", y="Proportion male words", x="Year", color="b", label="Mean proportion of Male words per year")
df_female_words_proportion_plot = df_female_words_proportion.plot(kind="line", y="Proportion female words", x="Year", color ="r", \
    label="Mean proportion of Female words per year", xlabel="Year", ylabel="Proportion spoken words", title="Proportion of words spoken (M/F) per year", \
        ax=df_male_words_proportion_plot)
plt.legend()


# ------------------ Line plot 3/3 ------------------
# Creates a line plot where yearly mean of
# "Number of male actors" and
# "Number of female actors" is plotted against "Year".
# The aim is to discern if eqality has
# improved over time.
```

```

df_male_actors = df.groupby("Year", as_index=True)["Number of male actors"].mean()
df_female_actors = df.groupby("Year", as_index=True)["Number of female actors"].mean()

df_male_actors.columns = ["Year", "Number of male actors"]
df_female_actors.columns = ["Year", "Number of female actors"]

plt.subplot(2,2,4)
df2 = df_male_actors.plot(kind="line", y="Number of male actors",x="Year", color="b", label="Number of Male actors")
df3 = df_female_actors.plot(kind="line", y="Number of female actors",x="Year", color="r", label="Number of Male actors", ax=df2, xlabel="Year", ylabel="Actors", title="Mean of M/F actors per film (yearly avg)",figsize=(15,10))


plt.legend()
plt.show()
```
Listing 1: Code for equality analysis

C.2 Discriminant Analysis

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

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb


url = 'train.csv'
indata = pd.read_csv(url, na_values='?', dtype={'ID': str}).dropna().
    reset_index()

allparams = ['Number words female', 'Total words', 'Number of words
    lead', 'Difference in words lead and co-lead', 'Number of male
    actors', 'Year', 'Number of female actors', 'Number words male', '
    Gross', 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age Co-
    Lead']
optparams = ['Number words female', 'Total words', 'Number of words
    lead', 'Difference in words lead and co-lead', 'Number of male
    actors', 'Number of female actors', 'Age Lead', 'Age Co-Lead']

N = 200          # Number of random data sets tested
seeds = []       # For np.random

for i in range(0, N):
    seeds.append(int(np.random.random()*1000))    # Random seed 0-999

LDA_Accuracy = []
QDA_Accuracy = []

for s in seeds:

    np.random.seed(s)
    trainI = np.random.choice(indata.shape[0], size=300, replace=False
    )
    trainIndex = indata.index.isin(trainI)
    train = indata.iloc[trainIndex]
    test = indata.iloc[~trainIndex]

    X_train = train[optparams]
    Y_train = train['Lead']
    X_test = test[optparams]
    Y_test = test['Lead']


    # ---------  LDA  ---------

    model = skl_da.LinearDiscriminantAnalysis()
    model.fit(X_train, Y_train)

    predict_prob_L = model.predict_proba(X_test)

    prediction_L = np.empty(len(X_test), dtype=object)
    prediction_L = np.where(predict_prob_L[:, 0]>=0.5, 'Female', 'Male
    ')

    # Accuracy
    LDA_Accuracy.append(np.mean(prediction_L == Y_test))


    # ---------  QDA  ---------
```

```
    model = skl_da.QuadraticDiscriminantAnalysis()
    model.fit(X_train, Y_train)

    predict_prob_Q = model.predict_proba(X_test)

    prediction_Q = np.empty(len(X_test), dtype=object)
    prediction_Q = np.where(predict_prob_Q[:, 0]>=0.5, 'Female', 'Male')

    # Accuracy
    QDA_Accuracy.append(np.mean(prediction_Q == Y_test))

# Accuracy results and sample confusion matrix:

print(f"LDA Accuracy: {np.mean(LDA_Accuracy):.3f}")
print(f"QDA Accuracy: {np.mean(QDA_Accuracy):.3f} \n")

# Confusion Matrix LDA
print('LDA Confusion Matrix:')
print(pd.crosstab(prediction_L, Y_test), '\n')

# Confusion Matrix QDA
print('QDA Confusion Matrix:')
print(pd.crosstab(prediction_Q, Y_test))
```
Listing 2: Code for LDA & QDA

C.3 Decision-tree based method

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


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

from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
import graphviz


# In[19]:


Oscar = pd.read_csv('train.csv')
np.random.seed(1)
Oscar_index=np.random.choice(Oscar.shape[0],size=800,replace=False)
train_Oscar=Oscar.iloc[Oscar_index]
test_Oscar=Oscar.drop(Oscar_index)
x_train_Oscar=train_Oscar.drop(columns=['Lead'])
y_train_Oscar=train_Oscar['Lead']
y_test_Oscar=test_Oscar['Lead']


# In[3]:


def create_right_train(x_train_Oscar,test_Oscar,name_of_new_x):
    #removes the parameters that are not used in the new training and test set
    new_x_train_Oscar=x_train_Oscar[name_of_new_x]
    new_x_test_Oscar=test_Oscar[name_of_new_x]
    return new_x_train_Oscar,new_x_test_Oscar


# In[4]:


new_one_train,new_one_test=create_right_train(x_train_Oscar,test_Oscar,["Number words female","Total words"])
new_one_test


# In[5]:


def create_model_samples(x_train_Oscar,y_train_Oscar,depth,min_samples):
    #Create the decision-tree with specified sample rate
    model=tree.DecisionTreeClassifier(max_depth=depth,min_samples_split=min_samples)
    model.fit(X=x_train_Oscar,y=y_train_Oscar)
    return model


# In[6]:
```

```python


def create_model(x_train_Oscar,y_train_Oscar,depth):
    #Create the decision-tree
    model=tree.DecisionTreeClassifier(max_depth=depth)
    model.fit(X=x_train_Oscar,y=y_train_Oscar)
    return model


# In[7]:


def make_graph(model,x_train_Oscar):
    #Make a graph of the decision-tree
    dot_data=tree.export_graphviz(model,out_file=None,feature_names=x_train_Oscar.columns,
                                  class_names=model.classes_,filled=True,rounded=True,
                                 leaves_parallel=True,proportion=True)
    graph=graphviz.Source(dot_data)
    return graph


# In[8]:


def test_graph(test_Oscar,y_test_Oscar,model):
    #Test the decision-tree on the test set
    #x_test_Oscar=test_Oscar.drop(columns=['Lead'])
    #y_test_Oscar=test_Oscar['Lead']

    y_predict=model.predict(test_Oscar)
    true_male=0
    false_male=0
    true_female=0
    false_female=0
    y_tests=y_test_Oscar.tolist()
    for nr in range(0,len(y_predict)):
        if y_predict[nr]==y_tests[nr]:
            if y_predict[nr]=="Male":
                true_male+=1
            else:
                true_female+=1
        else:
            if y_predict[nr]=="Female":
                false_male+=1
            else:
                false_female+=1

    err=np.mean(y_predict != y_test_Oscar)
    #print('Error rate for tree: '+ str(err))
    #print('Accuracy rate is %.2f' % np.mean(y_predict==y_test_Oscar))
    #print("TM "+str(true_male)+" FM: "+str(false_male)+" TF: "+ str(true_female)+" FF: "+str(false_female))
    return err,true_male,false_male,true_female,false_female


# In[9]:


def cross_val(new_x_train,depths,min_sample):

    cv=40
    scoring='accuracy'
    cross_scores=[]
```

```
    cross_std=[]
    cross_mean=[]
    accuracy=[]
    for depth in depths:
        model=create_model_samples(new_x_train,y_train_Oscar,depth,
    min_sample)
        #err=test_graph(new_x_test,y_test_Oscar,model)
        cross_score=cross_val_score(model,new_x_train,y_train_Oscar,cv
    =cv,scoring=scoring)
        cross_scores.append(cross_score)
        cross_mean.append(cross_score.mean())
        cross_std.append(cross_score.std())
        accuracy.append(model.fit(new_x_train,y_train_Oscar).score(
    new_x_train,y_train_Oscar))
    cross_mean=np.array(cross_mean)
    cross_std= np.array(cross_std)
    accuracy=np.array(accuracy)
    return cross_mean,cross_std,accuracy


# In[10]:


def cross_val_samples(new_x_train,depths,min_samples):

    cv=40
    scoring='accuracy'
    cross_scores=[]
    cross_std=[]
    cross_mean=[]
    accuracy=[]
    for sample in min_samples:
        model=create_model_samples(new_x_train,y_train_Oscar,depth,
    sample)
        #err=test_graph(new_x_test,y_test_Oscar,model)
        cross_score=cross_val_score(model,new_x_train,y_train_Oscar,cv
    =cv,scoring=scoring)
        cross_scores.append(cross_score)
        cross_mean.append(cross_score.mean())
        cross_std.append(cross_score.std())
        accuracy.append(model.fit(new_x_train,y_train_Oscar).score(
    new_x_train,y_train_Oscar))
    cross_mean=np.array(cross_mean)
    cross_std= np.array(cross_std)
    accuracy=np.array(accuracy)
    return cross_mean,cross_std,accuracy


# In[37]:


#Show the gini importance of the
names_of_char=["Number words female","Total words","Number of words
    lead","Difference in words lead and co-lead",
              "Number of male actors","Year","Number of female actors"
    ,
               "Number words male","Gross","Mean Age Male","Mean Age
    Female","Age Lead","Age Co-Lead"]
new_x_train,new_x_test=create_right_train(x_train_Oscar,test_Oscar,
    names_of_char)
depth=9
model=create_model_samples(new_x_train,y_train_Oscar,depth,15)
for l in range(1,101):
    gini+=model.feature_importances_
gini=gini/100
```

```python
plt.bar([i for i in range(len(gini))],gini)
plt.xlabel("features")
plt.ylabel("Gini importance")
plt.title("Gini importance of the features")
plt.savefig("Gini_importance.png")
#plt.bar(names_of_char,gini)
print(str(names_of_char[0])+", "+str(names_of_char[6])+ " and "+str(names_of_char[12]))
print(str(names_of_char[1])+", "+str(names_of_char[2])+ " and "+str(names_of_char[5]))
print(sum(gini))


# In[17]:


#Find the accuracy, misclassification and true/false male/female rate
names_of_char=["Number words female","Total words","Number of words lead","Difference in words lead and co-lead",
              "Number of male actors","Year","Number of female actors",
               "Number words male","Gross","Mean Age Male","Mean Age Female","Age Lead","Age Co-Lead"]
new_x_train,new_x_test=create_right_train(x_train_Oscar,test_Oscar,names_of_char)
depth=9
err_all=[]
TMA=[]
FMA=[]
TFA=[]
FFA=[]
for n in range(1,101):
    model=create_model_samples(new_x_train,y_train_Oscar,depth,15)
    err,TM,FM,TF,FF=test_graph(new_x_test,y_test_Oscar,model)
    err_all.append(err)
    TMA.append(TM)
    FMA.append(FM)
    TFA.append(TF)
    FFA.append(FF)
TMS=sum(TMA)/len(TMA)
FMS=sum(FMA)/len(FMA)
TFS=sum(TFA)/len(TFA)
FFS=sum(FFA)/len(FFA)
print("Accuracy :"+str(1-sum(err_all)/len(err_all)))
print("False male: "+str(FMS/(FMS+TFS)))
print("True male: "+str(TMS/(TMS+FFS)))
print("True female: "+ str(TFS/(TFS+FMS)))
print("False Female: "+ str(FFS/(FFS+TMS)))
print("Misclassification : "+ str((FFS+FMS)/(FFS+FMS+TFS+TMS)))


# In[12]:


#Find the accuracy of the decision-tree
names_list=[0,1,2,3,4,5,6,7,8,9,10,11,12]
#top 0
#second row 6
#third row 3 4
#fourth row 12 6 0 3
#fifth row 11  10
#names_list=[0,3,4,6,12]
#names_list=[0,6,7,10]
names_of_char=["Number words female","Total words","Number of words lead","Difference in words lead and co-lead",
```

```
              "Number of male actors","Year","Number of female actors",
               "Number words male","Gross","Mean Age Male","Mean Age Female","Age Lead","Age Co-Lead"]
name_of_new_x=[]
depth=9
min_sample=12
accuracies=[]
for nr in names_list:
    name_of_new_x.append(names_of_char[nr])
new_x_train,new_x_test=create_right_train(x_train_Oscar,test_Oscar,name_of_new_x)
#cross_mean,cross_std,accuracy=cross_val(new_x_train,depth,min_sample)
for n in range(1,51):
    model=create_model_samples(new_x_train,y_train_Oscar,depth,min_sample)
    err=test_graph(new_x_test,y_test_Oscar,model)
#print(err)
    accuracy=model.fit(new_x_train,y_train_Oscar).score(new_x_test,y_test_Oscar)
    accuracies.append(accuracy)
print(max(accuracies))
print(sum(accuracies)/50)
print(min(accuracies))


# In[13]:


#Plot the model
dot_data=tree.export_graphviz(model,out_file=None,feature_names=new_x_train.columns,
                              class_names=model.classes_,filled=True,rounded=True,
                             leaves_parallel=True,proportion=True)
graph=graphviz.Source(dot_data)
graph


# In[14]:


#plot the
names_list=[0,1,2,3,4,5,6,7,8,9,10,11,12] #0.8225 #0.81125
names_list_2=[0,3,4,6,12] #0.81875
#names_list_2=[1,2,5,7,8,9,10,11] #0.775
#names_list_2=[0,3,4,6,10,12] #0.81500
#names_list_2=[0,3,4,6,11,12] #0.815
#names_list_2=[0,3,4,6,7,10,11,12] #0.82875
#names_list_2=[0,3,4,6,7,10] #0.8024
#names_list_2=[0,3,4,6,12] #0.821
names_of_char=["Number words female","Total words","Number of words lead","Difference in words lead and co-lead",
              "Number of male actors","Year","Number of female actors",
               "Number words male","Gross","Mean Age Male","Mean Age Female","Age Lead","Age Co-Lead"]
name_of_new_x_1=[]
name_of_new_x_2=[]
depths=[2,3,4,5,6,7,8,9,10,11,12,13,14,15]
min_samples=12
for nr in names_list:
    name_of_new_x_1.append(names_of_char[nr])
    if nr in names_list_2:
        name_of_new_x_2.append(names_of_char[nr])
```

```
new_x_train_1,new_x_test_1=create_right_train(x_train_Oscar,test_Oscar
    ,name_of_new_x_1)
new_x_train_2,new_x_test_2=create_right_train(x_train_Oscar,test_Oscar
    ,name_of_new_x_2)
cross_mean_1,cross_std_1,accuracy_1=cross_val(new_x_train_1,depths,
    min_samples)
cross_mean_2,cross_std_2,accuracy_2=cross_val(new_x_train_2,depths,
    min_samples)
fig,ax=plt.subplots(1,1,figsize=(15,5))
ax.plot(depths,cross_mean_1,color='red')
ax.plot(depths,cross_mean_2)
ax.fill_between(depths,cross_mean_1-2*cross_std_1,cross_mean_1+2*
    cross_std_1,alpha=0.2,color='red')
id_max_1=cross_mean_1.argmax()
id_max_2=cross_mean_2.argmax()
best_depth_1=depths[id_max_1]
best_depth_2=depths[id_max_2]
best_score_1=cross_mean_1[id_max_1]
best_score_2=cross_mean_2[id_max_2]
print("NR 1 the depth: "+str(best_depth_1)+" with score: "+ str(
    best_score_1))
print("NR 2 the depth: "+str(best_depth_2)+" with score: "+ str(
    best_score_2))


# In[15]:


#Determine depth
names_of_char=["Number words female","Total words","Number of words
    lead","Difference in words lead and co-lead",
               "Number of male actors","Year","Number of female actors"
    ,
                "Number words male","Gross","Mean Age Male","Mean Age
    Female","Age Lead","Age Co-Lead"]
depths=[2,3,4,5,6,7,8,9,10,11,12,13,14,15]
min_samples=9
times=10
best_depths=[]
new_x_train,new_x_test=create_right_train(x_train_Oscar,test_Oscar,
    names_of_char)
for n in range(1,times+1):
     cross_mean_1,cross_std_1,accuracy_1=cross_val(new_x_train,depths,
    min_samples)
     id_max_1=cross_mean_1.argmax()
     best_depth_1=depths[id_max_1]
     best_depths.append(best_depth_1)
print("Max depth "+ str(max(best_depths))+" Min depth "+ str(min(
    best_depths)))
print("Mean "+ str(sum(best_depths)/10))


# In[16]:


#Determine min samples
name_of_new_x=["Number words female","Total words","Number of words
    lead","Difference in words lead and co-lead",
               "Number of male actors","Year","Number of female actors"
    ,
                "Number words male","Gross","Mean Age Male","Mean Age
    Female","Age Lead","Age Co-Lead"]
depths=9
min_samples=list(range(2,40))
times=10
```

```
best_samples=[]
new_x_train,new_x_test=create_right_train(x_train_Oscar,test_Oscar,name_of_new_x)
for n in range(1,times+1):
    print(n)
    cross_mean,cross_std,accuracy=cross_val_samples(new_x_train,depths,min_samples)
    id_max=cross_mean.argmax()
    best_sample=min_samples[id_max]
    best_score_1=cross_mean[id_max]
    best_samples.append(best_sample)

print("Max sample size: "+str(max(best_samples))+" Min sample size: "+str(min(best_samples)))
print(sum(best_samples)/10)


# In[ ]:
```

Listing 3: Code for tree-based methods

C.4 Logistic regression and feature selection

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

import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
import sklearn.preprocessing as skl_pre

from google.colab import files
uploaded = files.upload()

import io
data = pd.read_csv(io.BytesIO(uploaded['train.csv']), na_values = '?')
    .dropna().reset_index(drop=True)
# Dataset is now stored in a Pandas Dataframe

data['Lead'] = np.where(data['Lead'] == 'Female', 1, 0)

# Split the data randomly into a training set and a test set of
    approximately similar size.

np.random.seed(2)
trainI = np.random.choice(data.shape[0], size= 700, replace=False)
trainIndex = data.index.isin(trainI)

train = data.iloc[trainIndex]
test = data.iloc[~trainIndex]

# Try different feature sets (with data scaling)

import itertools
def findsubsets(s, n):
    return list(itertools.combinations(s, n))

from sklearn.metrics import roc_auc_score
from sklearn.metrics import auc
from sklearn.metrics import precision_recall_curve

min_error = data.shape[0]
error_a = 0
features_max = []
coef =0
conf_matrix =0
chosen_model =0
prob = 0

pc_auc = 0
roc_auc = 0
f1 = 0
roc_auc_a = 0
pc_auc_a = 0
pc_auc_max = 0
roc_auc_max = 0
f1_a = 0

s = {'Number words female', 'Total words', 'Number of words lead', '
    Difference in words lead and co-lead', 'Number of male actors', '
    Year', 'Number of female actors', 'Number words male', 'Gross', '
    Mean Age Male', 'Mean Age Female','Age Lead', 'Age Co-Lead'}
n = 13

# define model
```

```
model = skl_lm.LogisticRegression(solver ='liblinear')

subsets = findsubsets(s, n)
for s in subsets:
  input_variables = list(s)

  x_train = train[input_variables]
  y_train = train['Lead']
  x_test = test[input_variables]
  y_test = test['Lead']

  scaler = skl_pre.StandardScaler().fit(x_train)
  model.fit(scaler.transform(x_train), y_train )
  predict_prob = model.predict_proba(scaler.transform(x_test))
  prediction = np.where(predict_prob[:,1] > 0.5, 1, 0)

  # misclassification error
  error = np.mean(prediction != y_test)

  # calculate roc auc
  roc_auc = roc_auc_score(y_test, predict_prob[:,1])

  # calculate the precision-recall auc
  precision, recall, _ = precision_recall_curve(y_test, predict_prob
    [:,1])
  pc_auc = auc(recall, precision)
  f1 = f1 = 2*precision*recall/(precision+recall)

  # chose the model with the max ROC AUC
  if roc_auc > roc_auc_max:
    roc_auc_max = roc_auc
    error_a = error
    features_max = input_variables
    chosen_model = model
    conf_matrix = pd.crosstab(prediction, y_test)
    prob = predict_prob
    pc_auc_a = pc_auc
    f1_a = f1

print(features_max)
print('Misclassification error',error_a)
print(f'ROC AUC = {roc_auc_max}')
print('PC AUC =', pc_auc_a)
print('F1 =', np.max(f1_a))
print(chosen_model.coef_)
print(chosen_model.classes_)
print(conf_matrix)

# Find best features subsets (with 8,9,10 features)
X = data
y = data['Lead']

import sklearn.model_selection as skl_ms
import sklearn.preprocessing as skl_pre
n_runs = 20
features_lsts_with_big_roc_auc =[]
s = {'Number words female', 'Total words', 'Number of words lead', '
    Difference in words lead and co-lead', 'Number of male actors', '
    Year', 'Number of female actors', 'Number words male', 'Gross', '
    Mean Age Male', 'Mean Age Female','Age Lead', 'Age Co-Lead'}

subsets_1 = findsubsets(s, 8)
subsets_2 = findsubsets(s, 9)
subsets_3 = findsubsets(s, 10)

```

```python
subsets = subsets_1 + subsets_2 + subsets_3
roc_auc_n = np.zeros((n_runs, len(subsets)))
pc_auc = 0
error = 0
pc_auc = 0
f1 = 0

for i in range(n_runs):
  X_train, X_val, y_train, y_val = skl_ms.train_test_split(X, y,
    test_size = 0.3)

  for j, s in enumerate(subsets):
    input_variables = list(s)

    x_train = X_train[input_variables]
    x_test = X_val[input_variables]
    y_test = y_val

    model = skl_lm.LogisticRegression(solver ='liblinear')
    scaler = skl_pre.StandardScaler().fit(x_train)

    model.fit(scaler.transform(x_train), y_train)
    predict_prob = model.predict_proba(scaler.transform(x_test))
    prediction = np.where(predict_prob[:,1] > 0.5, 1, 0)

    roc_auc = roc_auc_score(y_test, predict_prob[:,1])
    roc_auc_n[i,j] = roc_auc

roc_auc_avg = np.mean(roc_auc_n, axis = 0)

for idx ,el in enumerate(roc_auc_avg):
  if el > np.max(roc_auc_avg)*(1-0.001):
    features_lsts_with_big_roc_auc.append(list(subsets[idx]))

for el in features_lsts_with_big_roc_auc:
  print(el)
  print('Number of features',len(el))

# find the decision threshold rate
def find_r(prob, y_test):
  recall = []
  precision = []
  f1_max = 0
  r_f1 = 0

  positive_class = 1
  negative_class = 0

  P = np.sum(y_test == positive_class)
  prediction = np.empty(len(x_test), dtype ='object')
  tr_val = np.linspace(0.00 , 1, num =101)

  for r in tr_val:
    prediction = np.where(prob[:,1] > r, positive_class,
    negative_class)
    P_star = np.sum(prediction == positive_class)
    tr_pos = np.sum((prediction == y_test)&(prediction ==
    positive_class))

    rec = tr_pos/P
    prec = tr_pos/P_star

    f1 = 2*prec*rec/(prec+rec)
    if f1_max < f1:
      f1_max =f1
```

```
      r_f1 = r

  return r_f1

# cross-validation of the threshold rate
def calculate_average_r(features, X, y):
# features -list of features
# X - data with all features
# y - column 'Lead'
  n_runs = 10

  tr_rates_n = np.zeros((n_runs))
  for i in range(n_runs):

    X_train, X_val, y_train, y_val = skl_ms.train_test_split(X, y,
    test_size = 0.3)
    model = skl_lm.LogisticRegression(solver ='liblinear')
    scaler = skl_pre.StandardScaler().fit(X_train[features])
    model.fit(scaler.transform(X_train[features]), y_train)
    predict_prob = model.predict_proba(scaler.transform(X_val[features
    ]))

    tr_rates_n[i] = find_r(predict_prob, y_val)

  return np.average(tr_rates_n)

# cross-validation of models with different feature sets with k-fold
features_list = features_lsts_with_big_roc_auc

n_runs = 10

pc_auc_n =np.zeros((n_runs, len(features_list)))
roc_auc_n = np.zeros((n_runs, len(features_list)))
misclassification_n = np.zeros((n_runs, len(features_list)))
tr_rate_models = []

X = data.drop(columns = 'Lead')
y = data['Lead']

n_fold = 10
cv = skl_ms.KFold(n_splits = n_fold, random_state = 1, shuffle = True)

for i, (train_index, val_index) in enumerate(cv.split(X)):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  for j,features in enumerate(features_list):
    model = skl_lm.LogisticRegression(solver ='liblinear')
    scaler = skl_pre.StandardScaler().fit(X_train[features])
    model.fit(scaler.transform(X_train[features]), y_train)
    predict_prob = model.predict_proba(scaler.transform(X_val[features
    ]))

    precision, recall, _ = precision_recall_curve(y_val, predict_prob
    [:,1])
    pc_auc_n[i,j] = auc(recall, precision)
    roc_auc_n[i,j] = roc_auc_score(y_val, predict_prob[:,1])

    tr_rate = calculate_average_r(features, X, y)
    prediction = np.where(predict_prob[:,1] > tr_rate, 1, 0)
    misclassification_n[i,j] = (np.mean(prediction != y_val))
    tr_rate_models.append(tr_rate)

fig, axs = plt.subplots(1, 3, figsize=(12, 3), sharey=False)
axs[0].boxplot(roc_auc_n)
```

```python
axs[1].boxplot(pc_auc_n)
axs[2].boxplot(misclassification_n)

fig.suptitle('Cross validation for models with different features sets')
plt.show()

# Model (logistic regression) validation: find regularization type and rate
from scipy.stats import loguniform
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV

def validate_model(X,y):
  model = LogisticRegression()
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  space = dict()
  space['solver'] = ['liblinear']
  space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
  space['C'] = loguniform(1e-5, 100)

  search = RandomizedSearchCV(model, space, n_iter=500, scoring='roc_auc', n_jobs=-1, cv=cv, random_state=1)
  res = search.fit(X, y)

  print('Score: %s' % res.best_score_)
  print('Hyperparameters: %s' % res.best_params_)
  return res.best_params_

# insert input variables (list of features) to validate the model
features = ['Total words', 'Age Co-Lead', 'Number of female actors', 'Difference in words lead and co-lead', 'Number of words lead', 'Age Lead', 'Number words female', 'Number of male actors']
validate_model(X[features], y)

# Find threshold rate for the best model (result = 0.435)
features = ['Total words', 'Age Co-Lead', 'Number of female actors', 'Difference in words lead and co-lead', 'Number of words lead', 'Age Lead', 'Number words female', 'Number of male actors']
# parameters {'C': 18.31792545756584, 'penalty': 'l1', 'solver': 'liblinear'}

n_runs = 10
tr_rates_n = np.zeros((n_runs))
X = data[features]
y = data['Lead']

for i in range(n_runs):

  X_train, X_val, y_train, y_val = skl_ms.train_test_split(X, y, test_size = 0.3)

  model = skl_lm.LogisticRegression(solver ='liblinear', C = 18.31792545756584, penalty ='l1')
  scaler = skl_pre.StandardScaler().fit(X_train)
  model.fit(scaler.transform(X_train), y_train)
  predict_prob = model.predict_proba(scaler.transform(X_val))
  tr_rates_n[i] = find_r(predict_prob, y_val)

r_res = np.average(tr_rates_n)
print(r_res)

# Evaluate performance of logistic regression model
```

```
from sklearn.metrics import f1_score
features = ['Total words', 'Age Co-Lead', 'Number of female actors', 'Difference in words lead and co-lead', 'Number of words lead', 'Age Lead', 'Number words female', 'Number of male actors']
X = data[features]
y = data['Lead']
pc_auc_n2 =np.zeros((n_runs))
roc_auc_n2 = np.zeros((n_runs))
misclassification_n2 = np.zeros((n_runs))
true_positive_rate = []
false_positive_rate = []
f1_rate = []
tr_pos = 0
fal_pos = 0

positive_class = 1
negative_class = 0

n_fold = 10
cv = skl_ms.KFold(n_splits = n_fold, random_state = 1, shuffle = True)

for i, (train_index, val_index) in enumerate(cv.split(X)):
  X_train, X_val = X.iloc[train_index], X.iloc[val_index]
  y_train, y_val = y.iloc[train_index], y.iloc[val_index]

  P = np.sum(y_val == positive_class)
  N = np.sum(y_val == negative_class)

  model = skl_lm.LogisticRegression(solver ='liblinear', C = 18.31792545756584, penalty ='l1')

  scaler = skl_pre.StandardScaler().fit(X_train)
  model.fit(scaler.transform(X_train[features]), y_train)
  predict_prob = model.predict_proba(scaler.transform(X_val))
  # calculate the metrics
  precision, recall, _ = precision_recall_curve(y_val, predict_prob[:,1])

  pc_auc_n2[i] = auc(recall, precision)
  roc_auc_n2[i] = roc_auc_score(y_val, predict_prob[:,1])

  tr_rate = 0.435
  prediction = np.where(predict_prob[:,1] > tr_rate, 1, 0)

  misclassification_n2[i] = (np.mean(prediction != y_val))
  tr_pos = np.sum((prediction == y_val)&(prediction == positive_class))
  fal_pos = np.sum((prediction != y_val)&(prediction == positive_class))
  true_positive_rate.append(tr_pos/P)
  false_positive_rate.append(fal_pos/N)

  f1 = f1_score(y_val, prediction, average='binary')
  f1_rate.append(f1)

print('ROC AUC', np.average(roc_auc_n2))
print('PC AUC', np.average(pc_auc_n2))
print('Error', np.average(misclassification_n2))
print('TPR', np.average(true_positive_rate))
print('FPR', np.average(false_positive_rate))
print('f1 coef', np.average(f1_rate))

fig, axs = plt.subplots(1, 3, figsize=(12, 3), sharey=False)
axs[0].boxplot(roc_auc_n2)
axs[1].boxplot(pc_auc_n2)
```

```
axs[2].boxplot(misclassification_n2)

axs[0].set_xticks(range(1))
axs[1].set_xticks(range(1))
axs[2].set_xticks(range(1))

fig.suptitle('Evaluation of log regression model (ROC AUC, PC AUC,
    misclassification error)')
plt.show()
```

Listing 4: Code for logistic regression and feature selection

C.5 kNN evaluation code

```python
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.neighbors as skl_nb
import sklearn.model_selection as skl_ms
import sklearn.metrics as met
import os
from tabulate import tabulate


def main():
    # Load data which has already been normalized
    # using StandardScaler()
    cwd = os.getcwd()
    #URI = (cwd+"\\train.csv")
    URI = (cwd+"\\train_standardscaler.csv")
    #URI = (cwd+"\\train_minmaxscaler.csv")
    film_data = pd.read_csv(URI, dtype={"Lead":str}).dropna().reset_index(drop=True)

    x = film_data.drop(columns=['Lead']) #, 'Year', 'Mean Age Male', 'Mean Age Female', 'Number of male actors', 'Number of female actors', 'Age Co-Lead'])
    y = film_data['Lead']
    positive_class = "Male"
    negative_class = "Female"



    # Setting values to k-range, number of folds, and threshold
    k_max = 80
    n_folds = 200
    threshold = 0.24

    # Call of function "evaluate_with_kfold",
    # please see function def for more info
    missclassification_k_error,  evaluation_terms, plotting_terms = evaluate_with_kfold(x, y,positive_class, negative_class, threshold, n_folds, k_max)

    # Finds what k-value minimizes missclassification
    # and find missclassification at that point
    min_error_k = find_best_k(missclassification_k_error)
    new_error_estimate = np.min(missclassification_k_error)
    evaluation_terms["Optimal_k"] = min_error_k
    evaluation_terms["E[Error_new]"] = new_error_estimate

    # Creates list for latex
    data = list(evaluation_terms.items())
    data.insert(0, ["Term","Value"])
    data_np = np.array(data)
    print(tabulate(data_np, headers=("firstrow"), tablefmt="latex"))


    # Create the missclassification to 'k' value
    # graph
    plt.plot(np.arange(1,k_max), missclassification_k_error)
    plt.title(f"Cross validation KFold = {n_folds} error for kNN")
    plt.xlabel("k")
    plt.ylabel("Validation error")
    plt.show()

```

```python
    # Create crosstab heatmap
    cm=np.array([ [evaluation_terms["TP"],evaluation_terms["FN"]],\
        [evaluation_terms["FP"],evaluation_terms["TN"]] ])
    disp = met.ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Male", "Female"])
    disp.plot()
    disp.ax_.set_title("Cross tab heatmap")
    plt.show()

    # Create ROC and Recall-Precision graph
    figure, axis = plt.subplots(2)
    figure.set_constrained_layout(True)

    axis[0].plot(plotting_terms["FPR_curve"], plotting_terms["TPR_curve"]);
    axis[0].set_title("ROC curve ")
    axis[0].set_xlabel("False positive rate")
    axis[0].set_ylabel("True positive rate")

    axis[1].plot(plotting_terms["recall_curve"], plotting_terms["precision_curve"]);
    axis[1].set_title("Precision-recall curve")
    axis[1].set_xlabel("Recall rate")
    axis[1].set_ylabel("precision rate")
    plt.show()

def evaluate_with_kfold(x, y, positive_class, negative_class, threshold=0.5, n_folds=None, k_max=None):
    # ----------------- function description -----------------
    # This function has three returns:
    # 'missclassification_k_error', 'evaluation_terms', and
    # 'plotting_terms'.
    # These are all dictionaries with 'values' estimated
    # by cross validation and 'key' as the name of the
    # variable or list of plotting points.
    #
    # 'missclassification_k_error' is a np.array where index+1
    # represents k and value at index represents
    # missclassification error for that k
    #
    # 'evaluation_terms' is a dictionary containing values
    # of 'evaluation terms', (ex. TP,FP...) where the name
    # of the value is the key
    #
    # 'plotting_terms' is a dictionary containing np.arrays.
    # Each np.array has elements representing the value of
    # an 'evalueation term' at a certain 'r' threshold. The
    # index represents the 'r' value which ranges from 0 to 1
    # with increments of 0.01.


    if n_folds==None:
        n_folds=x.shape[0]//4
    if k_max == None:
        k_max= x.shape[1]*4

    # Defining the splits and setting random_state to have reproductable results
    Kfold_cv = skl_ms.KFold(shuffle=True,n_splits=n_folds, random_state=1)
    k_range = np.arange(1,k_max)

    # Initializing variables for later usage
    missclassification_k_error = np.zeros(len(k_range))
    evaluation_terms = {}
```

```
    threshold_terms = {}
    plotting_terms = {}

    # ----------------- 1st cross validation -----------------
    # Finds the optimal value for 'k' and an estimation of
    # E[Error_new] using cross validation with uniform weights.
    # Number of folds and range of k-values tested are
    # given by n_folds and k_max

    for train_index, val_index in Kfold_cv.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[val_index]
        y_train, y_test = y.iloc[train_index], y.iloc[val_index]

        temp_missclassification_k_error = evaluate_k_kNN(k_range, x_train, y_train, x_test, y_test,positive_class, negative_class,"uniform", "auto",threshold)
        missclassification_k_error = np.add(missclassification_k_error,temp_missclassification_k_error)

    missclassification_k_error /= n_folds
    min_error_k = find_best_k(missclassification_k_error)

    # ----------------- 2nd cross validation -----------------
    # Finds values of 'evaluation terms' and 'plotting
    # ranges(terms)'

    for train_index, val_index in Kfold_cv.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[val_index]
        y_train, y_test = y.iloc[train_index], y.iloc[val_index]
        model = skl_nb.KNeighborsClassifier(n_neighbors=min_error_k, weights="uniform",algorithm="auto")
        model.fit(x_train, y_train)

        temp_evaluation_terms, temp_threshold_terms = get_evaluation_terms(model, x_test, y_test, positive_class, negative_class, threshold)

        # The sum of each 'evaluation terms' generated
        # per fold
        for key in temp_evaluation_terms.keys():
            if key in evaluation_terms:
                evaluation_terms[key] += temp_evaluation_terms[key]
            else:
                # If evaluation_terms is empty
                evaluation_terms[key] = temp_evaluation_terms[key]

        # The sum of each 'plotting term' per index, per fold
        # This works because all plotting_terms have the same
        # size
        for key in temp_threshold_terms.keys():
            if key in threshold_terms:
                for i, k in enumerate(temp_threshold_terms[key]):
                    val = threshold_terms[key][i] + temp_threshold_terms[key][i]
                    threshold_terms[key][i]= val
            else:
                # If plotting_terms is empty
                threshold_terms[key] = temp_threshold_terms[key]



    # Creation of additional evaluation terms and addition to
    # dictionary "evaluation_terms"
    evaluation_terms["TPR"] = evaluation_terms["TP"]/evaluation_terms["P"]
```

```
    evaluation_terms["FPR"] = evaluation_terms["FP"]/evaluation_terms["N"]
    evaluation_terms["accuracy"] = (evaluation_terms["TP"]+evaluation_terms["TN"])/(evaluation_terms["N"]+evaluation_terms["P"])
    evaluation_terms["precision"] = evaluation_terms["TP"]/evaluation_terms["P_star"]
    evaluation_terms["recall"] = evaluation_terms["TP"]/(evaluation_terms["TP"]+evaluation_terms["FN"])
    evaluation_terms["F1"] = 2*(evaluation_terms["precision"]*evaluation_terms["TPR"])/(evaluation_terms["precision"]+evaluation_terms["TPR"])


    # Creation of specific plotting curves
    plotting_terms["FPR_curve"] = threshold_terms["FP_threshold"]/evaluation_terms["N"]
    plotting_terms["TPR_curve"] = threshold_terms["TP_threshold"]/evaluation_terms["P"]
    plotting_terms["recall_curve"] = threshold_terms["TP_threshold"]/(threshold_terms["FN_threshold"]+threshold_terms["TP_threshold"])
    plotting_terms["precision_curve"] = threshold_terms["TP_threshold"]/(threshold_terms["FP_threshold"]+threshold_terms["TP_threshold"])


    return missclassification_k_error, evaluation_terms, plotting_terms



def evaluate_k_kNN(k_range,x_train, y_train, x_test, y_test, positive_class, negative_class, weight_type=None, algorithm_type=None, threshold=0.5):
    # ----------------- function description -----------------
    # This function returns the missclassification error
    # of 'k' in range (1-'k_range').
    # It returns an np.array where index is the value
    # of ('k'-1) and the value the missclassification error.

    if weight_type == None:
        weight_type = "uniform"

    if algorithm_type==None:
        algorithm_type = "auto"


    missclassification_k_error = np.zeros(len(k_range))
    for index, k in enumerate(k_range):
        model = skl_nb.KNeighborsClassifier(n_neighbors=k,weights=weight_type,algorithm=algorithm_type)
        model.fit(x_train, y_train)
        missclassification_k_error[index] += get_mean_missclassification(model,x_test,y_test, threshold, positive_class, negative_class)

    return missclassification_k_error


def get_mean_missclassification(model,x_test,y_test, threshold, positive_class, negative_class):
    # ----------------- function description -----------------
    # This function returns the mean missclassification error
    # from a 'model' evaluated on 'y_test' and with
    # 'threshold'
```

```python

    # set threshold
    positive_class_index = np.argwhere(model.classes_== positive_class).squeeze()
    prediction = np.where(model.predict_proba(x_test)[:, positive_class_index] > threshold, positive_class, negative_class)

    # calc missclasification error
    mean_missclassification =  np.mean(prediction != y_test)
    return mean_missclassification


def get_evaluation_terms(model, x_test, y_test, positive_class, negative_class,threshold):
    # ----------------- function description -----------------
    # This function returns the evaluation and plotting terms
    # given a model, test set and class labels.

    positive_class_index = np.argwhere(model.classes_== positive_class).squeeze()

    # Setting based on 'threshold'
    prediction = np.where(model.predict_proba(x_test)[:, positive_class_index] > threshold, positive_class, negative_class)
    prediction = model.predict(x_test)
    predict_prob = model.predict_proba(x_test)
    P = np.sum(y_test == positive_class) #the same as TP+FN
    N = np.sum(y_test == negative_class) #the same as TN+FP

    # All variables with *_threshold are lists that contain
    # plotting values.
    # Index represents the value of the threshold 'r' in
    # range 0-1 with 0.01 as increments.
    FP_threshold = np.zeros(101)
    TP_threshold = np.zeros(101)
    FN_threshold = np.zeros(101)
    TN_threshold = np.zeros(101)
    threshold_range = np.linspace(0,1,101)

    i=0
    for r in threshold_range:
        prediction_curve = np.where(predict_prob[:, positive_class_index]> r, positive_class, negative_class)
        FP_threshold[i] = np.sum((prediction_curve == positive_class) & (y_test == negative_class))
        TP_threshold[i] = np.sum((prediction_curve == positive_class) & (y_test == positive_class))
        FN_threshold[i] = np.sum((prediction_curve == negative_class) & (y_test == positive_class))
        TN_threshold[i] = np.sum((prediction_curve == negative_class) & (y_test == negative_class))
        i +=1

    FP = np.sum((prediction == positive_class) & (y_test == negative_class))
    TP = np.sum((prediction == positive_class) & (y_test == positive_class))
    FN = np.sum((prediction == negative_class) & (y_test == positive_class))
    TN = np.sum((prediction == negative_class) & (y_test == negative_class))


    P_star = np.sum(prediction == positive_class) #the same as TP+FP
    N_star = np.sum(prediction == negative_class) #the same as TN+F
```

```
    evaluation_terms = {"P":P, "N":N,"P_star":P_star, "N_star":N_star,
     "TN":TN, "FP":FP, "FN":FN, "TP":TP}
    plotting_terms = {"FP_threshold":FP_threshold,"TP_threshold":
    TP_threshold,"FN_threshold":FN_threshold,"TN_threshold":
    TN_threshold}

    return evaluation_terms, plotting_terms


def find_best_k( missclassification_k_error):
    # ----------------- function description -----------------
    # This function returns the index+1 of the minimum valued
    # element in the missclassification_k_error list.
    # This equates to the 'k' value for that point
    #
    min_error = np.min(missclassification_k_error)
    min_error_k = [i for i, x in enumerate(missclassification_k_error)
     if x == min_error] [0]+1

    return min_error_k



# Functions for creating scaled data sets

def generate_standard_scaled_datafile(film_data):
    import sklearn.preprocessing as skl_pre
    #CREATE NEW DATA WITH STANDARDSCALING
    x = film_data.drop(columns=['Lead'])
    y = film_data['Lead']

    standard_scaler = skl_pre.StandardScaler(with_mean=True,with_std=
    True)

    # StandardScaler: mean=0, variance=1
    scaled_film_data_array = standard_scaler.fit_transform(x)

    x_scaled = pd.DataFrame(scaled_film_data_array, columns = list(x.
    columns))

    film_data_scaled = x_scaled.join(y)

    film_data_scaled.to_csv('train_standardscaler.csv',index=False)

    print("New standardScaled data saved to file: '
    train_standardscaler.csv'")

def generate_MinMax_scaled_datafile(film_data):
    import sklearn.preprocessing as skl_pre
    #CREATE NEW DATA WITH MINMAXSCALER
    x = film_data.drop(columns=['Lead'])
    y = film_data['Lead']

    minmax_scaler = skl_pre.MinMaxScaler(feature_range=(0,1))

    # MinMax scaler: min=0, max=1
    scaled_film_data_array = minmax_scaler.fit_transform(x)

    x_scaled = pd.DataFrame(scaled_film_data_array, columns = list(x.
    columns))

    film_data_scaled = x_scaled.join(y)

    film_data_scaled.to_csv('train_minmaxscaler.csv', index=False)
```

```
    print("New MinMax data saved to file: 'train_minmaxscaler.csv'")



if __name__=="__main__":
    main()
```

Listing 5: Code for evaluating the performance of kNN

C.6 Quadratic Discriminant Analysis Predictions

```
import pandas as pd
import numpy as np

import sklearn.discriminant_analysis as skl_da
import csv

url_train = 'train.csv'
train = pd.read_csv(url_train, na_values='?', dtype={'ID': str}).
    dropna().reset_index()
url_test = 'test.csv'
test = pd.read_csv(url_test, na_values='?', dtype={'ID': str}).dropna
    ().reset_index()

allparams = ['Number words female', 'Total words', 'Number of words
    lead', 'Difference in words lead and co-lead', 'Number of male
    actors', 'Year', 'Number of female actors', 'Number words male', '
    Gross', 'Mean Age Male', 'Mean Age Female', 'Age Lead', 'Age Co-
    Lead']
optparams = ['Number words female', 'Total words', 'Number of words
    lead', 'Difference in words lead and co-lead', 'Number of male
    actors', 'Number of female actors', 'Age Lead', 'Age Co-Lead']


X_train = train[optparams]
Y_train = train['Lead']
X_test = test[optparams]
#Y_test = test['Lead']


# ---------  QDA  ---------

model = skl_da.QuadraticDiscriminantAnalysis()
model.fit(X_train, Y_train)

predict_prob_Q = model.predict_proba(X_test)

prediction_Q = np.empty(len(X_test), dtype=object)
prediction_Q = np.where(predict_prob_Q[:, 0]>=0.5, '1', '0')

np.savetxt("predictions.csv", prediction_Q, newline=',', fmt='%s')
```

Listing 6: Code for the predictions, QDA