Do (wo)men talk too much in films?

Daniel Fransén

Andreas Hertzberg

Alhassan Jawad

Erik Magnusson

Abstract

The project aimed to answer if Machine Learning (ML) methods can be used to solve a classification problem of predicting the gender of a movie's lead character. Out of five used ML methods: *Logistic regression*, *Discriminant Analysis*, *k-Nearest Neighbor*, *Random Forest*, and *AdaBoost*, the project demonstrated that *Quadratic Discriminant Analysis* was the optimal method for this classification problem.

1 Introduction

In this study, a variety of machine learning techniques were employed to determine the gender of the lead actor in a movie. Using information from 1037 movies containing 13 inputs & 1 output, multiple supervised classification models were trained, individually, and their efficiency was compared to determine the most effective model for the particular purpose of determining if the lead actor is male/female. The evaluation process involved splitting the data into training and testing sets through k-fold cross-validation. The models tested were

1. *Logistic regression*
2. *Linear Discriminant Analysis* (LDA)
3. *Quadratic Discriminant Analysis* (QDA)
4. *k-Nearest Neighbor*
5. Tree-based methods: *Random Forest*
6. Boosting: *AdaBoost*

The optimal model will be proposed for implementation and will be further evaluated using a separate test set of 387 films.

2 Pre-data Analysis

The following questions were answered by pre-analysing the given dataset & mainly printing out different columns from it. The code can be seen in the appendix (*section: Pre-data analysis*).

2.1 Do men or women dominate speaking roles in Hollywood movies?

Do men or women dominate speaking roles in Hollywood movies? To answer the question we can pinpoint two points of discussion:

1. *Figure 1* which shows how many words were spoken by male leads vs female leads throughout the years.
2. Calculating the percentage of movies where men spoke more than females or vice-versa.

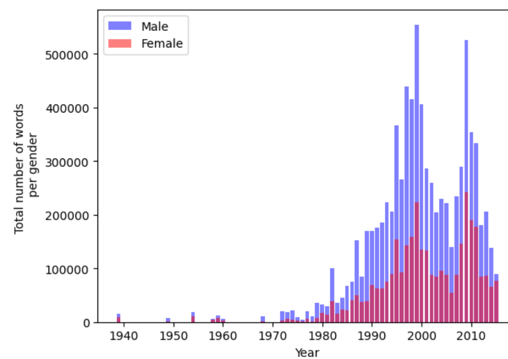


Figure 1: Number of words spoken by males/females throughout the years

Analysing *Fig 1* shows that in general, men speak more than women with approximately double the rate at some points. There exists some exceptions but the conclusion is still that men speak more than women. The reason *figure 1* is weirdly scaled is that what the figure is showing is the *total number of words* (lead+non-lead) per gender per year.

Printing the percentage of times where men spoke more than women in comparison with times where women spoke more shows the results that men spoke roughly 76.6% of the time while women the rest of the time (23.4%).

So an answer to the question: "Do men or women dominate speaking roles in Hollywood Movies?" is that men still dominate the speaking roles with more than half of the time.

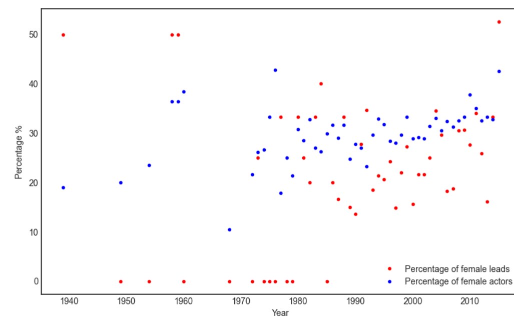
2.2 Has gender balance in speaking roles changed over time?

Concerning the question about gender balance, it can be understood in two different ways:

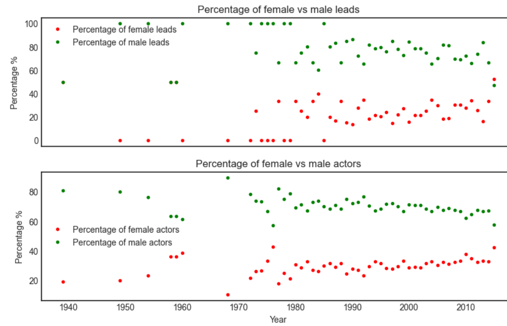
1. Has the gender balance for a specific role changed over time (i.e. years)?, *Figure 2a*
2. Has the gender balance (men/females) for speaking roles (mainly lead) changed over time (i.e. years)?, *Figure 2b*

Figure 2a answers the question regarding gender balance for a specific role by showing that the percentage of female actors saw a slow increase around the 1970s but then stagnated around 30%. The percentage of female leads, on the other hand, varied by a little bit but is still under 50% of the total leading roles.

Figure 2b shows two subplots in which the upper plot shows a comparison between the percentage of male & female leads while the lower plot shows a comparison between the percentage of male & female actors in the industry. Both suplots show two different plots that seem to be the



(a) Female leads vs Female actors (percentage)



(b) Percentage of male/female leads & male/female actors

reflections of themselves which brings us to the answer that the gender balance has changed over time.

2.3 Do films in which men do more speaking make a lot more money than films in which women speak more?

Which movies make more money, the movies where men speak more or those in which women speak more?

The solution can simply be obtained by computing the average gross for movies in which men spoke more frequently than women and vice-versa. The calculations revealed that movies with a predominant male dialogue had an average revenue of $118.6 million, while those with a female-led dialogue had an average gross income of $86.6 million. Additionally, movies with a male lead actor generated an average income of $115.2 million, compared to $98.7 million for movies with a female lead. This indicates that movies centered around males tend to generate higher gross than those centered around females.

3 Implementation of Methods

Of the 13 known inputs, 12 of them (all except *Total words*) were used as a quantitative input subset. The input *Total words* was excluded because it is colinear with other inputs, specifically the inputs *Number words female* & *Number words male*. Furthermore, the models (all except *kNN*) were tuned using grid search or random search as a methodological approach and performance/evaluation is determined using k-fold cross-validation. We chose to use accuracy as the performance metric because it is intuitive to interpret. Mathematically, accuracy is the fraction of correctly classified samples and will be calculated using the *cross_val_score* function. Furthermore, the hyper parameter tuning is shortly described below, and for even more detail, see appendix *section: Choice of hyper parameters*. Additionally, the hyper-parameter optimization results are summarized in table 1.

3.1 Logistic Regression

3.1.1 Description & Implementation

Logistic regression is a method within statistical machine learning that uses both continuous and discrete measurements to classify new samples. One can with this interpret *logistic regression* as a linear regression model that is more fitted for binary classification, since the model will use values of 0 and 1 instead of continuous values. In short, the *logistic regression* uses numerical input values and gives a binary output value of 0 and 1, or *True* and *False*. In our case the output values are used to classify whether the class *Lead* is *Male* or *Female*.

With the input data and the binary output data we then fit a curve of a logistic function [1] $h(z) = \frac{e^z}{1+e^(z)}$ that is later used to calculate the probability of the class being either *Male*=0 or *Female*=1. This limit can be adjusted according to the needs of individual situations and calculations.

In our case, this threshold is set to 0.5 and means that if the probability of the class *Male* is 0.5 or higher, then the *logistic regression* method will classify that particular data point as *Male*. Finding a good limit or threshold to optimize this method is not always an easy task, and to find the optimum value usually means that one has to manually tune this and it could also require one to work with different projects for cross referencing. 0.5 is the standard threshold [2] and given the time and the complexity of this project this seems like a decent threshold to use.

3.1.2 Evaluation & Model tuning

With regards to model tuning the *logistic regression* model, we used the *RandomizedSearchCV* function which looped through different combinations of the hyper parameters passed into the model [3] and found the hyper parameters optimal for this specific classification problem.

A randomized search through 70 fits gave that the optimal model tuning for *Logistic Regression* was found and gave an accuracy of 87.5%. The reason why *GridsearchCV* was not used here is that *GridsearchCV* loops through 516096 fits which takes up to 3 hours per run and does not give any better accuracy.

3.2 Discriminant Analysis (LDA & QDA)

3.2.1 Description & Implementation

Discriminant analysis is based on the generative model *Gaussian Mixture Model (GMM)* which gives the insight that making predictions comes to computing the *conditional distribution* that results in the classifier [4]:

$$p\left(y=m \mid \mathbf{x}_{\star}\right)=\frac{\widehat{\pi}_{m} \mathcal{N}\left(\mathbf{x}_{\star} \mid \widehat{\boldsymbol{\mu}}_{m}, \widehat{\boldsymbol{\Sigma}}_{m}\right)}{\sum_{j=1}^{M} \widehat{\pi}_{j} \mathcal{N}\left(\mathbf{x}_{\star} \mid \widehat{\boldsymbol{\mu}}_{j}, \widehat{\boldsymbol{\Sigma}}_{j}\right)} \tag{1}$$

GMM can be used everywhere but in the *supervised learning* environment, the classifiers that we get are called *LDA* & *QDA* [4].

Linear Discriminant Analysis (LDA) is a Machine Learning model that is commonly used for *Regression Problems*, in particular when it is more than 2 classes that are being used. For our classification problem, the *categorical variable* **Lead** which can be either *Male* or *Female* is transformed into a binary number 1 & 0 so that we can have *numerical variables* which LDA can be applied on. The method works by structuring an axis that 1) maximizes the distance between the two classes mean values while 2) keeping the two classes internal variance minimized. The data points of each class are then extended on the shared axis which creates the *linear decision boundary* which helps in predicting, in our case, if the **Lead** gender is *Male*=1 or *Female*=0. Worth knowing is that in LDA, we assume that both classes have shared covariance matrices which is what results in a *linear decision boundary*.

Quadratic Discriminant Analysis (QDA) is a Machine Learning model similar to LDA but with an apparent difference of assuming that we have individual covariance matrices which results in a *quadratic decision boundary*. Another thing of note is that QDA is a model made for classification problems meaning that there is no need for transforming a *categorical variable* output into a *numerical variable* one. There are some other difference in how QDA & LDA behave for different data types (which model is more prone to overfit etc.) and it is up to the programmer to choose how and which model to implement.

Both LDA & QDA were implemented using the *sklearn.discriminant_analysis* import and then fitted using the package's *gridsearchCV*. As a pre-analysis step, we normalized the dataset for *LDA* mainly to improve and enhance performance metrics. In addition, for the LDA case we needed to transform the outputs into binary numbers using the *StandardScaler()* function.

We tried to normalize the data for *QDA* but it did not improve the performance metrics by any way.

3.2.2 Evaluation & Model tuning

Python's *GridSearchCV* function which used a grid search scheme to go through different combinations of the hyper parameters passed into each model [5] [6]. Note that the more hyper parameter

values are passed into the *param_grid* variable, the more computational demanding and complex the grid search will be and it doesn't necessarily mean that better parameter options will be found.

After grid searching through 2016 & 4480 compatible fits for both LDA & QDA, the optimal model tuning for the models was found and gave an accuracy of 85.8% & 89.7%, respectively.

3.3 k-Nearest Neighbor (kNN)

3.3.1 Description & Implementation

k-Nearest Neighbor evaluates a data point by looking at the closest neighboring data points. The method changes a bit if it is used on either a regression problem or classification problem. For a regression problem the average value of the neighboring data points will be the predicted value for the evaluated data point while for classification the predication will be the label that is the majority. The method only accepts numerical data and uses e.g. euclidean distance measurement between data point x_* and neighboring data points x_i to find the closest points [4].

$$\sqrt{\sum_{i=1}^{n} (x_i - x_*)^2}, \quad i = 1, 2, 3, ... \tag{2}$$

In this project, *kNN* works by setting a value to the number of data points you will use to make the classification, often labeled *k*. As a simplification, if *k* is set to 5 it means that the 5 nearest data points are compared to each other and the label that is a majority "wins" and becomes the value of x_*. One way of *kNN* optimization means finding the optimal *k* value which results in the minimum validation error.

According to the documentation [7], there are 8 hyper parameters in the sklearn.neighbors.KNeighborsClassifier function. However, because *n_neighbors* was the only parameter covered in this course, we deemed it better to exclude them during model tuning.

3.3.2 Evaluation & Model tuning

The first approach for determining the optimal *k* value was to perform a training and validation split then calculating the average misclassification error for each *k* value across 10 different datasets for which the *k* value with the lowest average misclassification error was selected. For the second approach, we used n-fold cross-validation to determine the optimal *k* value. The n-fold cross-validation method was used to train and test the model on random subsets of the data. Thereafter the performance of the model was evaluated by calculating the average misclassification error for all cross-validation runs. The *k* value with the lowest misclassification error was the optimal one. Both approaches found that setting *k* to 5 resulted in an optimal kNN model with an accuracy of 81.0% which is the worst one out of the other methods.

3.4 Tree-based methods: Random forest

3.4.1 Description & Implementation

Decision trees can be used for regression or classification problems, and function by partitioning the input space into disjoint regions and fitting a simple model in each region. Since finding the globally optimal partitioning with respect to error minimization is computationally infeasible, decision trees use a greedy optimization algorithm based on recursive binary splitting.

A *Random Forest* is an ensemble method that improves the performance of decision trees by reduction of variance through bootstrap aggregation (Bagging) of both samples and features. For regression, the aggregation function is averaging, and in more precise mathematical terms the ensemble variance of N trees with correlation ρ and variance σ^2 can be shown to be $\text{Var}\left[\frac{1}{N}\sum_{n=1}^{N} z_n\right] = \frac{1-\rho}{N}\sigma^2 + \rho\sigma^2$ or approximately $\rho\sigma^2$ for large ensembles. This is smaller than the variance of the individual decision trees (σ^2), and there is a similar result for classification.

The most obvious way of reducing the ensemble variance is by having a large ensemble (many trees, large N), but this also implies a large computational cost. Apart from the ensemble size, the remaining variables ρ and σ are both affected by the bagging of samples and features. For both

sample and feature bagging, there is an optimum in the number of samples/features drawn during the bootstrapping, since training with a large number of samples/features makes the variance σ^2 of the individual trees small, but increases their correlation (ρ).

3.4.2 Evaluation & Model tuning

Random Forest was implemented using python's package *scikit-learn*. The model was tuned using the *GridSearchCV* function which explores different combinations of hyper parameters [8]. Using *Grid Search* on 40824 parameter combinations gave an accuracy of 85.0% for the optimally tuned model.

3.5 Boosting: AdaBoost

3.5.1 Description & Implementation

AdaBoost is an ensemble method that attempts to form a strong model from a linear combination of weak models. The weak models are learned sequentially from reweighed data such that each new model tries to correct the errors of the one before. More specifically, samples that are misclassified by one learner will be assigned a larger coefficient in the error function of the subsequent learner. Mathematically, the prediction of the boosted classifier is taken to be a weighted majority vote $\widehat{y}^B_{\text{boost}} = \text{sign}\left(\sum_{b=1}^B \alpha^b \widehat{y}^b\right)$. Here, $\widehat{y}^b$, $b = 1, ..., B$ are the predictions of the weak models. In each step the error of the next weak model is taken to be

$$E^b_{\text{train}} = \sum_{i=1}^{n} w^b_i \mathbb{I}\{\widehat{y}^{(b)}(\mathbf{x}_i) \neq y_i\} \quad \text{with} \quad w^b_i = w^{b-1}_i \exp\left(-\alpha^{(b-1)} y_i \widehat{y}^{(b-1)}(\mathbf{x}_i)\right), \tag{3}$$

for data $(\mathbf{x}_i, y_i)$. After normalization of $\mathbf{w}^b$ the remaining step in each iteration is to calculate the optimal coefficients α. It can be shown that the error of the boosted classifier formed so far by the weak models, can be minimized by setting $\alpha^b = \frac{1}{2}\ln\left(\frac{1-E^b_{\text{train}}}{E^b_{\text{train}}}\right)$.

3.5.2 Evaluation & Model tuning

We chose to train a model using *AdaBoost* with classification tree as the base model, implemented in *scikit-learn*. For some reason it seems that *scikit-learn* does not implement the theoretical optimal weighing, but instead all weak models are given the same weight, by default $\alpha = 1$. Regarding model tuning, a *grid Search* using the passed hyper parameters [9] to loop through 20160 fits gave that the optimally tuned model for *AdaBoost* has an accuracy of 87.8% which is the second best after QDA.

RandomizedSearchCV was not implemented for *Random Forest* or *AdaBoost* because it would have decreased the amount of searched fits/combinations to 196 and that decreased the optimal models accuracy. Even if the *RandomizedSearchCV* would have decreased the computational time by a more than 97.5%.

Table 1: Some of the most important hyper parameters for each model, along with their respective optimized values. The hyper parameters were optimized through grid or random search. The names are from *scikit-learn*, and are largely self-explanatory, but see refs. [3], [5]–[9] for more details.

Method	**Parameter 1**	**Parameter 2**	**Parameter 3**	**Parameter 4**	**Parameter 5**
LDA	shrinkage = None				
QDA	reg_param = 0.2	priors = [0.25, 0.75]			
Logistic regression	penalty = 'l2'	C = 1.0	fit_intercept = False		
Random forest	n_estimators = 1000	bootstrap = False	min_samples _split= 5	max_depth = None	min_samples _leaf = 5
AdaBoost tree	n_estimators = 1000	learning_rate = 1.0	min_samples _split = 200	max_depth = 5	
kNN	n_neighbors = 5				

3.6 Model Selection

For a good comparison with the implemented methods *(section 3.1-3.5)*, a naive classifier that always predicts the gender of *Lead* as *Male* was created. The classifier's accuracy reached 75.6% and methods with accuracy higher than the naive classifiers will be deemed useful for the given *classification problem*.

We compared the implemented methods using k-fold cross-validation with *k=14* and the results are shown in the box-plot displayed below.

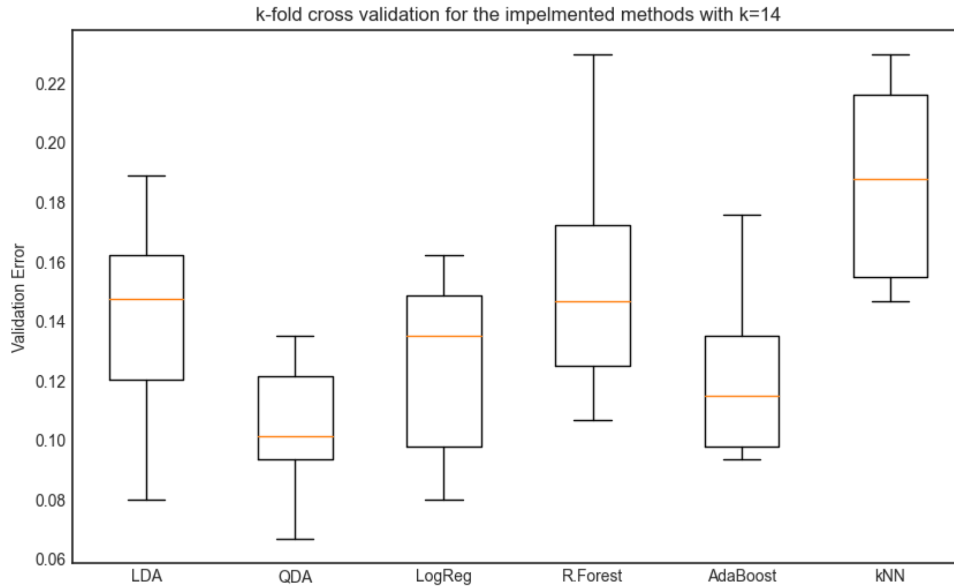


Figure 3: Box-plot showing the validation error for different methods

Looking at *figure 3* we can presume that *kNN* can be eliminated as the chosen method because it has the worst validation error of the 5 examined families. *Random Forest*, *Logistic Regression* & *LDA* show a smaller validation error than *kNN* but they too will be eliminated because both *AdaBoost* & *QDA* are performing better. This leaves us with one of two choices and we choose to use *QDA "in production"* because it is performing better than *AdaBoost* by a good margin.

4 Conclusion

The chosen method/model to be put into production for this classification problem is *Quadratic Discriminant Analysis* (QDA) with the hyper parameters shown in appendix (section: *Choice of hyper parameters*). The script "*Comparing the methods*" gives that the implemented methods accuracy are all around 80% with the QDA model performing better than the others with an impressive accuracy of 89.7%.

Overall, this study provides valuable insights into the usage of machine learning methods and shows that even tasks such as gender classification based on movies can be solved using Machine learning.

References

[1] I. B. Machines. “What is logistic regression?” IBM. (2023), [Online]. Available: https://www.ibm.com/topics/logistic-regression (visited on 02/25/2023).

[2] G. Harrison. “Calculating and setting thresholds to optimise logistic regression performance,” Towards Data Science. (2023), [Online]. Available: https://towardsdatascience.com/calculating-and-setting-thresholds-to-optimise-logistic-regression-performance-c77e6d112d7e (visited on 02/22/2023).

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.linear_model.logisticregression,” Scikit-learn. (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (visited on 02/22/2023).

[4] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *MACHINE LEARNING: A First Course for Engineers and Scientists*, New Edition. Cambridge: Cambridge University Press, 2022, ISBN: 9781108843607.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.discriminant$_a nalysis.lineardiscriminantanalysis$,” Scikit-learn. (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html (visited on 02/22/2023).

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.discriminant$_a nalysis.quadraticdiscriminantanalysis.$” (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html (visited on 02/22/2023).

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.neighbors.kneighborsclassifier.” (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (visited on 02/22/2023).

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.ensemble.randomforestclassifier.” (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html (visited on 02/22/2023).

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* “Sklearn.ensemble.adaboostclassifier.” (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html (visited on 02/22/2023).

Appendix

Choice of hyper parameters

Logistic Regression have 14 different optional & required hyper parameters:
The parameter *penalty* is something the that have been talked about when introducing *logistic regression* (in the course SML). In addition, this parameter have to do with the regularization parameter which means it may have an influence on how the optimal model is structured.

The parameters *tol*, *C*, *fit_intercept*, *random_state*, *solver*, *max_iter* & *multi_class* will be set to different values and looped through because of their importance to find the optimal *logistic regression* model.

1. *tol* is all about the tolerance for the stopping criteria
2. *C* specifies the inverse of regularization strength, must be non-negative float
3. *fit_intercept* specifies if a constant θ_0 should be added to the function or not
4. *random_state* to decide if data should be shuffled or not, for reproducible reasons too
5. *solver* specifies which solver method would be used in the described problem
6. *max_iter* is the *maximum number of iterations taken for the solver to converge* [3]
7. *multi_class* decides how the solver will go through solving the given problem

The parameters *Dual*, *intercept_scaling*, *verbose*, *warm_start*, *l1_ratio* & *class_weight* is not something that have been talked about in the course SML. Thus, they will not be used when tuning *logistic regression*.

n_jobs controls how many CPU cores are used when working with Logistic Regression. According to the documentation [3], *n_jobs* is "*used when parallelizing over classes if multi_class='ovr'*", which means that *n_jobs* will be set ONLY if the *logistic regression* code takes a long time to run and/or loops through many parameter fits.

Linear Discriminant Analysis have 7 hyper parameters: The parameter *solver* specifies which solver method would be used in the described problem as is thus something worth looping over. In addition, this parameter is something that we talked about when *LDA* was introduced in the course.

Concerning the parameter *shrinkage*, it controls the strength of this models regularization term (regularization applied to the covariance matrix of the input data) in which smaller/larger values may dictate if the model overfits/underfits. Thus this parameter is of much importance to have when looping through different fits.

The parameters *priors*, *n_components*, *store_covariance* and *covariance_estimator* is not something that have been talked about in the course and will not be taken into account.

tol will be set to different values and looped through because of its importance to find the optimal *LDA* model. Observe that *tol* is a parameter that is all about the tolerance for the stopping criteria.

Quadratic Discriminant Analysis have 4 required hyper parameters and thus all of them will be included in the grid search even though two of the parameters are not something we have talked about in the course. The four hyper parameters are *priors*, *reg_param*, *store_covariance* & *tol*.

kNN have 8 hyper parameters and of them only the most known one (namely *n_neighbors*) will be used because this is the only parameter talked about in the course. The other 7 parameters are:

1. *weights*
2. *algorithm*
3. *leaf_size*
4. *metric*
5. *metric_params*
6. *n_jobs*

Observe that *n_jobs* decides how many CPU cores are used for calculations but it will not be needed as no more than one hyper parameter is used for *kNN*.

Random Forest have 18 different optional and required parameters and of them only 7 will be used due to the code taking a long time to run. The used parameters are:

1. *n_estimators*
2. *criterion*
3. *max_depth*
4. *min_samples_split*
5. *min_samples_leaf*
6. *bootstrap*
7. *random_state*

Note that *n_jobs* for the *gridsearchCV* function will be manually set to -1 so that all CPU processors/cores are used.

AdaBoost have 5 parameters and beside the *base_estimator* that will be passed in as pre-made parameter(s), the parameters:

1. *n_estimators*
2. *learning_rate*
3. *random_state*

will be included as they are parameters that we went through in the course and/or feels important for model tuning.

Due to time constraints and the fact that including 7 & 3 hyper parameters for *Random Forest* & *AdaBoost* gives us between 56000-700000 fits/combinations to search through (takes up to 320 minutes for 1 run), it was decided that each parameter will be limited to contain a maximum of 3 & 10 values for *Random Forest* & *AdaBoost*, respectively. This is done to conserve computational power and computational time per run.

Main Code

The code will be attached as a zip file and should be opened as a *Jupyter Notebook*. Some basic comments that will not be shown in appendix will be found on the real code. Observe that after model tuning the individual models, their accuracy was calculated in the *Compare the methods* script as to have the same datasets from all methods and not give any method some advantage or disadvantage.

Imports

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

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
import sklearn.model_selection as skl_ms
import sklearn.ensemble as skl_en
```

Pre-data steps

```
# Read the training data and save it into variable "train"
train = pd.read_csv('train.csv')

# Information about "train" variable
#   -> mean('Number of male actors')=~7.8
#   -> mean('Number of female actors')=~3.5
train.describe()
```

Pre-data Analysis

```
# First Question: Do men or women dominate speaking roles in
    Hollywood movies?

# Create sorted list of years with at least one movie
Years = train['Year'].unique() # .unique() returns the specific
    values (unduplicated) in an usorted array
Years.sort()

# Determines & calculates the percentage of when females/males
    spoke more and then prints the results
more_female_words = train.loc[(train['Number␣words␣male']\
                                <train['Number␣words␣female'])]
more_male_words = train.loc[(train['Number␣words␣female']\
                              <train['Number␣words␣male'])]
print(f"The␣percentage␣of␣movies␣where␣females␣speak␣more␣than␣
    males:\␣{(len(more_female_words)/len(train))*100:.1f}%")
print(f"The␣percentage␣of␣movies␣where␣males␣speak␣more␣than␣
    females:\␣{(len(more_male_words)/len(train))*100:.1f}%")

total_words_female = []
total_words_male = []
for year in Years:
     current_movies = train[train['Year'] == year]

     words_female_actors = np.sum(current_movies['Number␣words␣
         female'])
     female_lead = (current_movies['Lead']=='Female')
     words_female_leads = np.sum(current_movies['Number␣of␣words␣
         lead'] * female_lead)
     total_words_female.append(words_female_actors +
         words_female_leads)

     words_male_actors = np.sum(current_movies['Number␣words␣male'
         ])
     male_lead = (current_movies['Lead']=='Male')
     words_male_leads = np.sum(current_movies['Number␣of␣words␣lead
         '] * male_lead)
     total_words_male.append(words_male_actors + words_male_leads)
# Plot that shows which gender spoke more throughout the years
plt.figure(1)
plt.bar(Years, total_words_male, color='blue', alpha = 0.5, label=
    'Male')
plt.bar(Years, total_words_female, color='red', alpha = 0.5, label
    ='Female')

plt.xlabel('Year')
plt.ylabel('Total␣number␣of␣words␣\n␣per␣gender')
plt.legend()
plt.show()
```

```
# Second Question: Has gender balance in speaking roles changed
    over time (i.e. years)?

Years = train['Year'].unique() # .unique() returns the specific
    values (unduplicated) in an usorted array
Years.sort()
female_lead_percentage = []
female_actors_percentage = []
male_lead_percentage = []
male_actors_percentage = []

for i in Years:
    data = train.loc[(train['Year'] == i)]

    # Calculates how many female lead there have been each year (
        given in percentage)
    lead_precentage_female = (data[data['Lead'] == 'Female'].shape
        [0])/len(data)*100
    female_lead_percentage.append(lead_precentage_female)

    # Calculates how many female actors there have been each year
        (given in percentage)
    actors_percentage_female = data['Number of female actors'].sum
        ()\
         /(data['Number of female actors'].sum() + data['Number of
             male actors'].sum())*100
    female_actors_percentage.append(actors_percentage_female)

    # Calculates how many male led there have been each year (
        given in percentage)
    lead_precentage_male = (data[data['Lead'] == 'Male'].shape[0])
        /len(data)*100
    male_lead_percentage.append(lead_precentage_male)

    # Calculates how many male actors there have been each year (
        given in percentage)
    actors_percentage_male = data['Number of male actors'].sum()\
         /(data['Number of female actors'].sum() + data['Number of
             male actors'].sum())*100
    male_actors_percentage.append(actors_percentage_male)

# Plot the results
#        Figure 2: Female leads vs Female actors (percentage)
plt.figure(2)
plt.plot(Years, female_lead_percentage, 'r.', label='Percentage of
    female leads')
plt.plot(Years, female_actors_percentage, 'b.', label='Percentage
    of female actors')
plt.xlabel('Year')
plt.ylabel('Percentage $\%$')
plt.legend()
plt.show()

#  Figure 3: Percentage of male/female leads \& male/female actors
plt.figure(3)
fig, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

# Female leads vs Male leads (percentage)
```

```
ax1.plot(Years, female_lead_percentage, 'r.', label='Percentage␣of
    ␣female␣leads')
ax1.plot(Years, male_lead_percentage, 'g.', label='Percentage␣of␣
    male␣leads')
ax1.set_ylabel('Percentage␣$\%$')
ax1.set_title('Percentage␣of␣female␣vs␣male␣leads')
ax1.legend()

# Female actors vs Male actors
ax2.plot(Years, female_actors_percentage, 'r.', label='Percentage␣
    of␣female␣actors')
ax2.plot(Years, male_actors_percentage, 'g.', label='Percentage␣of
    ␣male␣actors')
ax2.set_xlabel('Year')
ax2.set_ylabel('Percentage␣$\%$')
ax2.set_title('Percentage␣of␣female␣vs␣male␣actors')
ax2.legend()

plt.show()
```

```
# Third Question: Do films in which men do more speaking make a
    lot more money than films in which women speak more?

# Calulates & prints average gross made on movies where males
    speak more than females
male_gross_mean = more_male_words['Gross'].mean()
female_gross_mean = more_female_words['Gross'].mean()
print(f'Average␣Gross␣for␣movies␣where␣males␣speak␣more␣than␣
    females:\
␣␣␣␣␣␣${male_gross_mean:.1f}m')
print(f'Average␣Gross␣for␣movies␣where␣females␣speak␣more␣than␣
    males:\
␣␣␣␣␣␣${female_gross_mean:.1f}m')

# Calulates & prints average gross made on movies with male/female
     leads
lead_male = train.loc[(train['Lead'] == 'Male')]
lead_female = train.loc[(train['Lead'] == 'Female')]
lead_gross_male = lead_male['Gross'].mean()
lead_gross_female = lead_female['Gross'].mean()
print(f"Average␣Gross␣for␣movies␣where␣the␣lead␣is␣male:␣${
    lead_gross_male:.1f}m")
print(f"Average␣Gross␣for␣movies␣where␣the␣lead␣is␣female:␣${
    lead_gross_female:.1f}m")
```

Implementation of methods

```
# Used all 13 inputs except 'Total words' which is colinear with '
    Number words female' & 'Number words male' and was excluded
    beacuse of warnings

# Linear Discriminant Analysis (LDA)
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

# Implementation:

np.random.seed(1) # For reproducibility
train = pd.read_csv('train.csv')
normalizer = StandardScaler()

# Convert the target variable to binary representation where "Male
    "=1 & "Female"=0
encoder = LabelEncoder()
train['Lead'] = encoder.fit_transform(train['Lead'])

# Create X & y
X = train[['Number␣words␣female', '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']]
X = normalizer.fit_transform(X) # To normalize the data
y = train['Lead']

# Model structuring
LDA_model = skl_da.LinearDiscriminantAnalysis()

# Model tuning & fitting
cv = skl_ms.KFold(n_splits=14, shuffle=True, random_state=None)
     param_grid = {'solver': ['svd', 'lsqr', 'eigen'], 'tol':
         [0.001, 0.0001, 0.00001, 0.000001],\
     'shrinkage': [None, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
         0.8, 0.9, 1.0]}
grid_search = GridSearchCV(estimator=LDA_model, param_grid=
    param_grid, scoring='accuracy', verbose=1, cv=cv)
grid_search.fit(X, y)
best_params = grid_search.best_params_

# Note:
#        The warning that we get is because:
#             1) Shrinkage is not supproted with the solver='svd'
#        We tested differnt combinations and this warning doesn't
    affect finding the optimal model!
```

```python
# Quadtratic Discriminant Analysis (QDA)
from sklearn.model_selection import GridSearchCV

# Implementation:

np.random.seed(1) # For reproducibility
train = pd.read_csv('train.csv')

# Create X & y
X = train[['Number words female', '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']]
y = train['Lead']

# Model structuring
QDA_model = skl_da.QuadraticDiscriminantAnalysis()

# Model tuning & fitting
param_grid = {'reg_param': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
    0.8,       0.9, 1],\
     'tol': [0.001, 0.0001, 0.00001, 0.000001],\
     'priors': [None, [0.25, 0.75], [0.5, 0.5], [0.75, 0.25]],\
     'store_covariance': [True, False]}
cv = skl_ms.KFold(n_splits=14, shuffle=True, random_state=None)
grid_search = GridSearchCV(estimator=QDA_model, param_grid=
    param_grid, scoring='accuracy', verbose=1, cv=cv)
grid_search.fit(X, y)
best_params = grid_search.best_params_
```

```
# Logistic Regression
from sklearn.model_selection import RandomizedSearchCV

# Read in the data training(Observe that the file is in this case
    loaded locally on the computer)
url = 'train.csv'
training = pd.read_csv(url, na_values='?', dtype={'ID': str}).
    dropna().reset_index()

#sampling indices for the data set in a training set and test set
np.random.seed(1)
trainI = np.random.choice(training.shape[0],size = 500, replace=
    False)
trainIndex = training.index.isin(trainI)
train = training.iloc[trainIndex] #Training set
test = training.iloc[~trainIndex] #test set

#Set the model using sklearn to solve with logistic regression
LogReg_model = skl_lm.LogisticRegression()

X_train = train[['Number␣words␣female','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']]
Y_train = train['Lead']
X_test = test[['Number␣words␣female','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']]
Y_test = test['Lead']

# Model tuning & fitting
param_grid = {'penalty': ['l1', 'l2', 'elasticnet', None],\
               'C': [0.0001, 0.001, 0.01, 0.1, 1.0],\
               'fit_intercept': [True, False],\
               'solver': ['lbfgs', 'liblinear', 'newton-cg', '
                   newton-cholesky', 'sag', 'saga'],\
               'multi_class': ['auto', 'ovr', 'multinomial'],\
               'tol': [0.001, 0.0001, 0.00001, 0.000001],\
               'random_state': [None, 0, 1234, 4321], 'max_iter':
                   [100, 1000, 10000, 10000]}
random_search = RandomizedSearchCV(estimator=LogReg_model,
    param_distributions=param_grid,\
                n_iter=14, scoring='accuracy', n_jobs=-1,
                    random_state=1, verbose=1)
random_search.fit(X_train, Y_train)
best_params = random_search.best_params_

#Fitting the model and also make sure that we are using logistic
    regression
model = skl_lm.LogisticRegression(solver='newton-cholesky', tol
    =0.0001, penalty='l2',\
        multi_class='ovr', fit_intercept=False, C=1.0, max_iter
            =10000,\
        random_state=0)
```

```
model.fit(X_train, Y_train)
#print('model summary:')
#print(model)

#Calculate predicted values, with the first 10 results.
predict_prob = model.predict_proba(X_test)
#print('Classes')
#print(model.classes_)
#print(predict_prob[0:10])

#Predict the the results based on the parameters before, the first
     20 is chosen here
#simply because to check if the algorithm works properly since the
     first 10 results in 'Male'.
prediction = np.empty(len(X_test), dtype = object)
prediciton = np.where(predict_prob[:, 0]>=0.5,'Female','Male')
#print(prediciton[0:20])

#n-fold Cross validation with logistic regression, using n = 14,
    random_state = 42 and
#1400 max iterations to exclude problems running the code
log_cv =skl_lm.LogisticRegressionCV(cv=14,random_state=42,max_iter
    =1400)

#Fitting the crossvalidation
log_cv.fit(X_train,Y_train)
```

```
# Random Forest & AdaBoost
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold, GridSearchCV
from sklearn import tree
from sklearn.ensemble import AdaBoostClassifier,
    RandomForestClassifier

# Load the data set
df = pd.read_csv('train.csv')

# If refit=True then models are run even if there already exist
    results for them.
# However, if refit=False then models are not refitted once stored
     results exist.
refit = True
results = {}

# Make one-hot encoding for the 'Lead'-variable
df_prep = pd.get_dummies(df, columns=['Lead'])

# Use all data samples for cross validation since the final test
    data is stored in a separate file.
train = df_prep

# Train with all features except the lead ('Lead_Male' and '
    Lead_Female')
X_train = train.drop(columns=['Total␣words', 'Lead_Male', '
    Lead_Female'])

# Normalize the data
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns =
    X_train.columns)

# Training label is 'Lead_Female' (1 or 0)
y_train = train['Lead_Female']

# Define jobs to (re-)run
job_list = [
    'RF',
    'AB-T',
]

# Define models and parameters for the grid search parameter
    optimiaztion
full_names = {
    'RF': 'Random␣Forest',
    'AB-T': 'AdaBoosted␣tree',
}

base_models = {
    'RF': RandomForestClassifier(),
    'AB-T': AdaBoostClassifier(base_estimator=tree.
        DecisionTreeClassifier()),
}

model_parameters = {
    'RF': {
        'max_depth': [None, 5, 15],
```

```
        'min_samples_split': [5, 10, 20],
        'n_estimators': [500, 1000, 2000],
        'max_samples': [None, 0.5, 0.8],
        'criterion': ['gini', 'entropy'],
        'min_samples_leaf': [5, 15, 25],
        'bootstrap': [True, False],
        'random_state': [None, 0, 1234]},
    'AB-T': {
        'base_estimator__max_depth': [None, 5, 10, 15],
        'base_estimator__min_samples_split': [1, 5, 20, 200],
        'n_estimators': [500, 1000, 2000],
        'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
            0.9, 1.0],
        'random_state': [None, 0, 1234]}
}

# Cross validation sheme
n_splits = 14
cv = KFold(n_splits=n_splits, random_state=None, shuffle=True)

# Fit models and print respective best parameter values and
    accuracy
n_jobs = len(job_list)
for i, job_name in enumerate(job_list):
    if refit==False and model_name in results.keys(): continue #
        Don't refit models if refit==False

    # Prepare model
    base_model = base_models[job_name]
    param_grid = model_parameters[job_name]
    model_name = full_names[job_name]
    clf = GridSearchCV(base_model, param_grid, scoring='accuracy',
         cv=cv, n_jobs=-1, verbose=1)

    # Fit model and search for best parameters
    #print(f'\n\nFitting {job_name} (Job {i+1}/{n_jobs}) \nModel:
        {model_name} \n----------')
    clf.fit(X_train, y_train)
```

```
# k-Nearest-Neigbor
def perform_kNN_cross_validation_to_get_k(K, columns, n_folds,
    data, test):
    missclassification = np.zeros(len(K))
    cv = skl_ms.KFold(n_splits=n_folds, random_state=2, shuffle=
        True)
    data_X = data.iloc[:, columns]
    data_Y = data.iloc[:, -1]
    test_X = test.iloc[:, columns]

    for j, k in enumerate(K):
        model = skl_nb.KNeighborsClassifier(n_neighbors=k)
        scores = skl_ms.cross_val_score(model, data_X, data_Y,
            scoring='accuracy', cv=cv)
        missclassification[j] = 1 - np.mean(scores)

    plt.plot(K, missclassification)
    plt.title('Cross␣validation␣error␣for␣kNN␣(n␣fold)')
    plt.xlabel('k')
    plt.ylabel('Validation␣error')
    plt.show()

    min_index = np.argmin(missclassification)
    min_value = missclassification[min_index]


    return min_value, min_index+1

def perform_kNN_10_val_to_get_k(data, columns, n_runs, K):
    missclassification = np.zeros((n_runs, len(K)))

    for i in range(n_runs):
        X_train, X_val, Y_train, Y_val = skl_ms.train_test_split(
            data.iloc[:, columns],\
             data.iloc[:, -1], test_size=0.3)
        for j, k in enumerate(K):
            model = skl_nb.KNeighborsClassifier(n_neighbors=k)
            model.fit(X_train, Y_train)
            prediction = model.predict(X_val)
            missclassification[i, j] = np.mean(prediction != Y_val
                )
    average_mis = np.mean(missclassification, axis=0)
    min_error = np.min(average_mis)
    index = np.argmin(average_mis)
    k_value = K[index]

    plt.plot(K, average_mis)
    plt.title('Cross␣validation␣error␣for␣kNN␣(n␣run␣average)')
    plt.xlabel('k')
    plt.ylabel('Validation␣error')
    plt.show()

    return min_error, k_value

def prediction(k, columns, data, test):
    data_X = data.iloc[:, columns]
    data_Y = data.iloc[:, -1]
    test_X = test.iloc[:, columns]
    model = skl_nb.KNeighborsClassifier(n_neighbors=k)
```

```
    model.fit(data_X, data_Y)
    prediction = model.predict(test_X)

    return prediction

# Set random seed
np.random.seed(1)

# Get data from train.csv
data = pd.read_csv('train.csv')

# Get test data from test.csv
test = pd.read_csv('test.csv')

# General parameters
K = np.arange(1, 199)  #k = 1,2,...,198
columns = [0,2,3,4,5,6,7,8,9,10,11,12] # better accuracy without 1

# Get best k for K-NN using fold cross validation
n_folds = 14
min_value, k_fold = perform_kNN_cross_validation_to_get_k(K,
    columns, n_folds, data, test)
#print(k)
#print(f'The best k value according to n fold cross validation: {
    k_fold}')
print(f"The␣minimum␣missclassification␣value␣is␣{min_value}␣at␣k␣=
    ␣{k_fold}\
␣according␣to␣n␣fold␣cross␣validation")

# Get best k for K-NN by doing 10 runs with different validation
    sets
n_runs = 10
min_error, k_10_val = perform_kNN_10_val_to_get_k(data, columns,
    n_runs, K)
print(f"The␣minimum␣missclassification␣value␣is␣{min_error}␣at␣k␣=
    ␣{k_10_val}\
␣␣␣␣␣␣#␣according␣to␣average␣from␣n␣runs")
```

Which inputs to consider

```
# Implementation:

np.random.seed(1) # For reproducibility
train = pd.read_csv('train.csv')
X = train.copy().drop(columns='Lead')
plt.figure(5)
pd.plotting.scatter_matrix(X, figsize=(40,40))
plt.show()
```

Naive Classifier

```
# Implementation:
train = pd.read_csv('train.csv')

# Naive Classifier that always predicts lead gender as male
def naive_classifier(train):
    male_prediction = []
    for i in train['Lead']:
        male_prediction.append('Male')
    return male_prediction
# Function to calculate accuracy
def accuracy(y_prediction, y_validation):
    accuracy = np.mean(y_prediction == y_validation)
    return accuracy
```

Comparing the methods

```
# Performing k-fold cross-validation (with k=14) on all the chosen
     models to choose best optimal method for predicting lead.

# Implementation:
from sklearn import tree

np.random.seed(1) # For reproducibility
train = pd.read_csv('train.csv')
X = train[['Number␣words␣female', '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']]
y = train['Lead']
cross_validation = skl_ms.KFold(n_splits=14, shuffle=True,
    random_state=1)

#                  LinearDiscriminantAnalysis (LDA)
# Evaluate optimal LDA using k-fold cross-validation where k=14
LDA_model = skl_da.LinearDiscriminantAnalysis(solver='svd', tol
    =0.001, shrinkage=None)
Accuracy = skl_ms.cross_val_score(LDA_model, X, y, scoring='
    accuracy', cv=cross_validation)
print(f"Accuracy␣with␣LDA␣using␣k-fold␣cross-validation␣with␣k=14:
    ␣{(np.mean(Accuracy))*100:.1f}%")

#                  QuadtraticDiscriminantAnalysis (QDA)
# Evaluate optimal QDA using k-fold cross-validation where k=14
QDA_model = skl_da.QuadraticDiscriminantAnalysis(reg_param=0.2,
                         tol=0.001,\
              store_covariance=True, priors=[0.25, 0.75])
Accuracy = skl_ms.cross_val_score(QDA_model, X, y, scoring='
    accuracy', cv=cross_validation)
print(f"Accuracy␣with␣QDA␣using␣k-fold␣cross-validation␣with␣k=14:
    ␣{(np.mean(Accuracy))*100:.1f}%")

#                          Logistic Regression
# Evaluate optimal Logistic Regression using k-fold cross-
    validation where k=14
LogReg_model = skl_lm.LogisticRegression(solver='newton-cholesky',
      tol=0.0001, penalty='l2',\ multi_class='ovr', fit_intercept=
    False, C=1.0, max_iter=10000, random_state=0)
Accuracy = skl_ms.cross_val_score(LogReg_model, X, y, scoring='
    accuracy', cv=cross_validation)
print(f"Accuracy␣with␣Logistic␣Regression␣using␣k-fold␣cross-
    validation␣with␣k=14:\
␣␣␣␣␣␣{(np.mean(Accuracy))*100:.1f}%")

#                  Tree-based methods: Random Forest
# Evaluate optimal Random Forest using k-fold cross-validation
    where k=14
forest_model = skl_en.RandomForestClassifier(n_estimators=1000,
    min_samples_split=5,\
          max_depth=None, max_samples=None, bootstrap=False,\
          criterion='entropy', min_samples_leaf=5, random_state=0)
```

```
Accuracy = skl_ms.cross_val_score(forest_model, X, y, scoring='accuracy', cv=cross_validation)
print(f"Accuracy using Random Forest & k-fold cross-validation with k=14:\
{(np.mean(Accuracy))*100:.1f}%")

#                    Boosting: AdaBoost tree
# Evaluate optimal AdaBoost tree model using k-fold cross-validation where k=14
estimator= tree.DecisionTreeClassifier(min_samples_split=200, max_depth=5)
adaboost_model = skl_en.AdaBoostClassifier(estimator=estimator, n_estimators=1000, random_state=None,\
learning_rate=1.0)
Accuracy = skl_ms.cross_val_score(adaboost_model, X, y, scoring='accuracy', cv=cross_validation)
print(f"Accuracy using AdaBoost Tree & k-fold cross-validation with k=14:\
{(np.mean(Accuracy))*100:.1f}%")

#                    k-Nearest-Neighbor (kNN)
# Evaluate optiaml kNN with k=5 using k-fold cross-validation where k=14
kNN_model = skl_nb.KNeighborsClassifier(n_neighbors=5)
Accuracy = skl_ms.cross_val_score(kNN_model, X, y, scoring='accuracy', cv=cross_validation)
print(f"Accuracy using kNN with k=5 & k-fold cross-validation with k=14:\
{(np.mean(Accuracy))*100:.1f}%")

#    Naive Classifier that always predicts lead gender as male
y_validation = y
Accuracy = []
y_prediction = naive_classifier(train)
Accuracy.append(accuracy(y_prediction, y_validation))
Accuracy_percentage = np.mean(Accuracy)*100
print(f"Accuracy of the Naive Classifier that predicts lead gender as male: {Accuracy_percentage:.1f}%")
```

Plotting misclassifications for implemented methods

```
from sklearn.preprocessing import LabelEncoder
from sklearn import tree

# Implementation:
estimator= tree.DecisionTreeClassifier(min_samples_split=200,
    max_depth=100)

# k-fold cross-validation for all chosen methods (except Naive
    Classifier) using k=14
models = [skl_da.LinearDiscriminantAnalysis(solver='svd', tol
    =0.001, shrinkage=None),\
     skl_da.QuadraticDiscriminantAnalysis(reg_param=0.2, tol
        =0.001,\
         store_covariance=True, priors=[0.25, 0.75]),\
     skl_lm.LogisticRegression(solver='newton-cholesky',
         tol=0.0001, penalty='l2',\
     multi_class='ovr', fit_intercept=False, C=1.0, max_iter
        =10000,\
         random_state=0),\
     skl_en.RandomForestClassifier(n_estimators=1000,
         min_samples_split=5,\
         max_depth=None, max_samples=None, bootstrap=False,\
         criterion='entropy', min_samples_leaf=5, random_state=0),\
     skl_en.AdaBoostClassifier(estimator=estimator,
         n_estimators=1000, random_state=None,\
         learning_rate=1.0),\
     skl_nb.KNeighborsClassifier(n_neighbors=5)]

missclassification = np.zeros((14, len(models)))

for i, [train_index, validation_index] in enumerate(
    cross_validation.split(X)):
     X_train, X_validation = X.iloc[train_index], X.iloc[
        validation_index]
     y_train, y_validation = y.iloc[train_index], y.iloc[
        validation_index]

     for m in np.arange(0, 6):
         model = models[m]
         model.fit(X_train, y_train)
         predicition = model.predict(X_validation)
         missclassification[i, m] = np.mean(predicition !=
             y_validation)

plt.figure(6)
plt.boxplot(missclassification)
plt.title('k-fold␣cross␣validation␣for␣the␣impelmented␣methods␣
    with␣k=14')
plt.xticks(np.arange(6)+1, ('LDA', 'QDA', 'LogReg', 'R.Forest', '
    AdaBoost', 'kNN'))
plt.ylabel('Validation␣Error')
plt.show()
```

Predicting Lead using chosen model

```
# Implementation:

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
X_train = train.copy().drop(columns=['Total␣words', 'Lead'])
X_test = test.copy().drop(columns=['Total␣words'])
y = train['Lead']
QDA_model = skl_da.QuadraticDiscriminantAnalysis(reg_param=0.2,
     tol=0.001,\
     store_covariance=True, priors=[0.25, 0.75])
QDA_model.fit(X_train, y)
QDA_predictions_categocial = QDA_model.predict(X_test)

# Converting categorical classes to numerical
convert_element = {'Female': 1, 'Male': 0}
QDA_predictions_numerical = [convert_element[x] for x in
    QDA_predictions_categocial]

# Prints the lead gender predictions using QDA as model & compare
    with categorical predictions
#print("Predictions of Lead gender using QDA where 'Female'=1 & '
    Male=0")
#print("\n")
#print(QDA_predictions_numerical)
#print("\n")
#print(QDA_predictions_categocial)

# Save the predictions as a single row of comma seperated ones and
     zeroes
QDA_predictions_numerical = np.array(QDA_predictions_numerical).
    reshape(1, -1)
np.savetxt('QDA_predictions.csv', QDA_predictions_numerical,
    delimiter=',', fmt='%d')
```