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

Anton Larsson **Oscar Hammarberg** **Oscar Jacobson** **Samuel Kristensen**

Abstract

In this project statistical machine learning classifiers are used to predict the (binary) gender of actors with lead roles in Hollywood movies. Predictions were made using a dataset of 1039 different movies containing 13 unique numerical features such as words spoken and gross profits. The performance of four machine learning classifiers were explored and tuned to the problem at hand. One was chosen to test its performance on a test set of 397 new films.

1 Introduction

Gender roles are one of the most debated and talked about topics in modern society. Debates are especially hot when it comes to appearance in highly sought after positions in society. Are males dominating every desirable field? In this project a study of gender balance in movies over the course of many years will be carried out. The aim is to gain knowledge about what parameters surrounding a movie can be used to predict the star or **lead** role in the movie.

Statistical machine learning methods were used on 13 different types of recorded data for each move. Examples of movie features studied are release year, number of words spoken by each gender and gross profits. Setting up what is called a binary classification problem and evaluating several methods of machine learning classification to see which one is best suited, which gives the best results and why?

2 Data analysis

The data studied in this project comes in the form of a .csv file with information of movies released between 1939 and 2015 and 14 defining features for each movie. The different features are all numerical except of the **lead** feature which is binary (Male or female) and to be classified by machine learning predictions.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

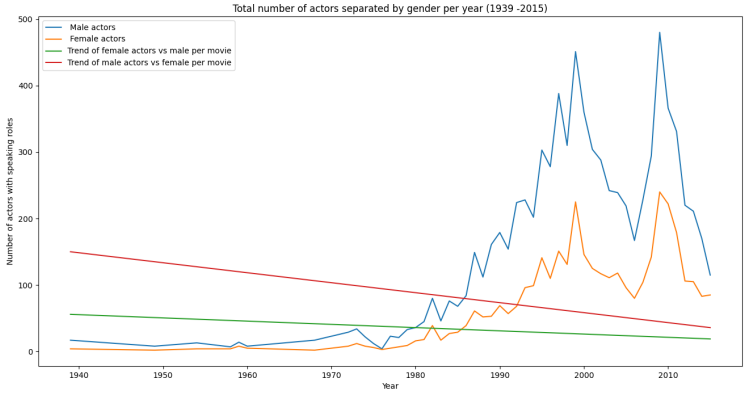

Figure 1: Plot of speaking role gender 1938-2015

In the given data, there are 776 movies with male leads, as opposed to 244 movies with female leads. Meaning movies with male leads make up approximately 75% of the movies analyzed. The average male dominated movie made approximately 37 % more gross profits than the average female dominated movie. There are 796 movies where male characters speak more than their female counterparts. Meaning there are only 243 female dominated movies. As seen in Figure 1 there seems to be a downward trend of words spoken, per movie, by male characters compared to their female counterparts, indicating a trend towards equality.

The worst case classifier which picks the same class every single time will perform reasonably well if it picks male, since movies with male leads make up 76% of movies, but will perform horribly if it picks female every time since movies with a female lead only makes up 24% of movies. It is very clear however that a mixture of movie features are required to predict the lead role with very high accuracy.

3 The different methods

3.1 Logistic Regression

Logistic regression is a machine learning method of solving classification problems like the one at hand, and the base idea of how it works is that the model uses the logistic function, or the sigmoid function as it is called in order to model the data accordingly. From this, in order to actually learn the logistic regression model the parameters θ which minimizes the logistic loss function needs to be calculated. [7]

After implementation and tuning, cross validation was carried out and a confusion matrix was used in order to get the accuracy-rate of which the model operated on. The correct predictions of the model are stored in the true positive and true negative cells, and any false predictions are stored in the false positive and false negative cells.

lead	Predict Female	Predict Male
True Female	105	23
True Male	86	525

Table 1: The confusion matrix for logistic regression

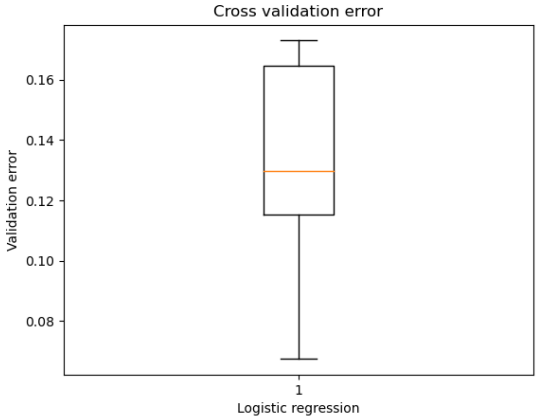

Figure 2: Cross validation error for logistic regression

From this the method is calculated to have an accuracy rate of 85.3%, which is not bad but an error rate of 15% is still not considered to be a good enough method. The problem with this model seems to be that it is biased towards male leads, which can be seen from the fact that the model predicted the lead role being a male 86 times when the lead role was actually a female.

3.2 Discriminant analysis

A discriminative model describes how the output y is generated, i.e $p(y|x)$ There are two models we have worked with in this project, linear and quadratic discriminant analysis. They are both derived from the Gaussian mixture model (GMM) when trained in a supervised way, from fully labeled data results in the LDA and QDA models. The Gaussian mixture model makes use of the factorization

$$p(x, y) = p(x|y)p(y) \tag{1}$$

of the joint probability density function. The second factor is the marginal distribution of y. Since y is categorical, and thereby takes values in the set $\{1, ..., M\}$, this is given by a categorical distribution with M parameters $\{\pi_m\}_{m=1}^{M}$

To complete the model we need to assume that each $p(x|y)$ is a Gaussian distribution for the GMM[7]

$$p(x|y) = \mathcal{N}(x|\mu_y, \sum_y) \tag{2}$$

with class-dependent mean vector μ_y and covariance matrix $\sum_y$ This is the basics for the GMM model. The key insight for using a generative model $p(x, y)$ to make predictions is to realize that predicting the output y for a known value x amounts to computing the conditional distribution $p(y|x)$. From probability theory we have

$$p(y|x) = \frac{p(x, y)}{p(x)} = \frac{p(x, y)}{\sum_{j=1}^{M} p(x, y = j)} \tag{3}$$

The left hand side is the predictive distribution and the right hand sides expressions are defined from the GMM above, this combined gives us:

$$p(y = m|x_*) = \frac{\hat{\pi}_m \mathcal{N}(x_*|\hat{\mu}_m \hat{\sum}_m)}{\sum_{j=1}^{M} \hat{\pi}_j \mathcal{N}(x_*|\hat{\mu}_j \hat{\sum}_j)} \tag{4}$$

We can obtain "hard" predictions $\hat{y}_*$ by selecting the class which is predicted to be the most probable and then compute corresponding decision boundaries. Taking the logarithm and noting that only the numerator depends on m we can write this as

$$\hat{y}_* = argmax_m p(y = m|x_*) = argmax_m \{ln\hat{\pi}_m + ln\mathcal{N}(x_*|\hat{\mu}_m \hat{\sum}_m)\} \tag{5}$$

Since the logarithm of the Gaussian density function is a quadratic function in x, the decision boundary for this classifier is also quadratic and the method is therefore refered to as quadratic discriminant analysis (QDA). This method arises natuarally from the GMM, if we however make the additional simplifying assumption that the covariance matrix is equal for all classes we obtain the linear discriminant analysis (LDA) [1]

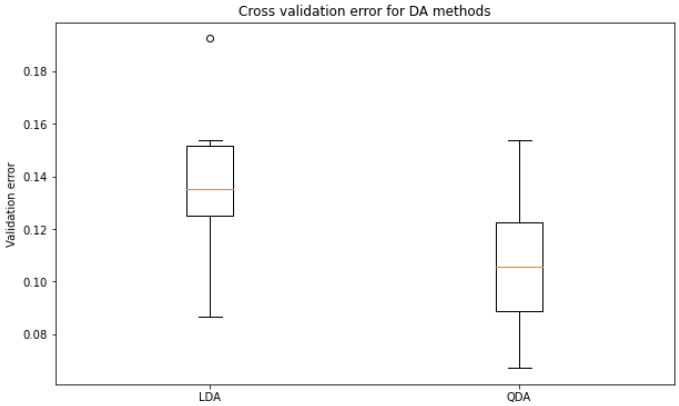

Figure 3: Boxplot for the validation error of LDA and QDA

The missclassification rate with the tuned parameters and 10-fold crossvalidation were 0.137 for LDA and 0.106 for QDA

3.3 k-NN

We have two sets of data ,the training set and the test set, for which we want to learn k- nearest neighbor classification. This is done by going through the test inputs and for each x calculating the euclidean distance between all the training data points and the test input. After this, the k training points closest to x are chosen and the output class is decided as a majority vote between the k neighbors. In our case we have a binary classification problem and the majority vote is done by looking at how many of the k neighbors belongs to the class $['Male']$ and comparing that to the number of points belonging to $['Female']$. Thereafter, the output is $['Male']$ if $nr_{Male} > nr_{female}$ and $['Female']$ if $nr_{Male} < nr_{Female}$. If there are an equal number of votes sklearn.neighbors chooses the class that happens to appear first in the list of neighbors–that is, since sklearn stores all the neighbors in a list of some sort, the tie will be decided by the class that happens to appear first in the list–this is one of many possible ways to handle a tie in k-NN classification.

To apply this method to the problem we first have to decide which k to use. A good way to pursue this venture is to use an n-fold cross-validation for a relatively small n in a for loop that tests a number of different k-values. After that you can get a rather good idea of the interval of k-values that have good performance by plotting the validation error contra the k-values. Then perform an n-fold cross-validation for the given interval, and by the same procedure it will be easier to locate the optimal k-value. By this procedure we find that k = 10 is the best performing k-value for the data we have been given. A 100 fold cross-validation is performed to evaluate the performance of 10-NN the results of which are illustrated in figure 5. Furthermore, the confusion matrix is also presented to aid in the evaluation of performance.

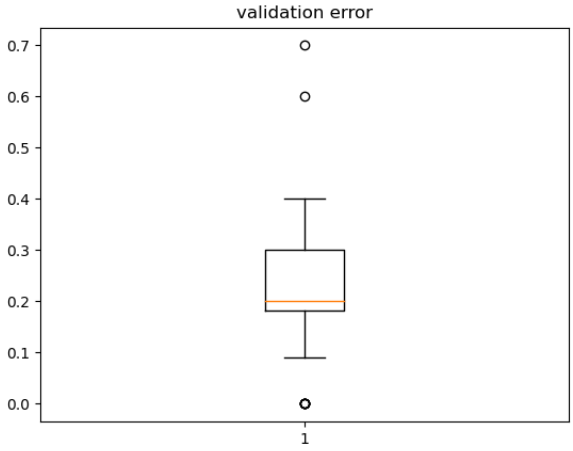

Figure 4: Boxplot for the validation error of 10-NN

lead	Predict Female	Predict Male
True Female	7	2
True Male	184	546
Accuracy	0.748	

Table 2: The confusion matrix for 10-NN

The model has an accuracy of $\approx 75\%$ which makes it slightly worse than just picking male every time. As such, the model is not doing great in regards to the data it operates on. The main problem seems to be that it is biased towards the male output, but not biased enough that it picks male every time. Therefore, the model is not good enough to put into production, at least not without first lowering the bias with a method such as boosting.

3.4 Tree based methods

A tree based method is a subset of the rule-based models. One example of these is the k-NN model described above and will have a lot of the same properties. The essence of a tree based method is that the defining rules can be organized as a binary tree, hence the name.

When constructing a binary tree the computational power used grows combinatorically with tree size which is never desired. The first way of making a model like this computationally feasible is to make the model greedy, that is, to make the model only care about the current formation of the tree and not care about future iterations. The greedy implementation creates a bias in the model which can be mitigated using great tree-depth. But a very tall(or deep) tree will unfortunately cause over fitting to training data. The model therefore has to be implemented with this trade-off in mind.

In this project, to solve the trade-off problem, ensamble methods were used. An ensamble method makes use of averaging and/or combining multiple trees to improve performance. An ensamble method, *Bagging*[3], takes multiple tall trees which will have a low bias but high variance, and average the fitted models out. This will keep the low bias of the initial models but reduce the variance greatly. The one problem with this method is that all the trees are based on the same training data and will be correlated. A *Random forest* method solves this problem by using random perturbation of training data. When the trees are created only some uniformly randomly selected training data points are considered, thus creating de-correlated trees. The random forest method does cause some bias to be reintroduced in the model but will often produce better overall results compared to normal bagging.

The classification model which will define how we generate the binary splits in the tree is based on a majority vote in data points. There are a lot of ways to define these splitting criterions but is most simply defined as the *missclassification rate*:[7, p.31] The classic *missclassification rate* has a linear loss function and will not favor pure nodes (Nodes that are almost entirely made up of one variable) which can be a disadvantage. Other splitting criterion with curved loss functions are the *Gini index* and the *Entropy criterion*. These methods are more suited to compensate for what we give up in greedy model implementation used in this project.(see [7, p.31])

4 Implementation

The methods were all implemented using the SciKit learning - toolbox [5],[2],[6] for python. For Linear Regression, LDA,QDA and Tree based methods. A parameter search was then carried out using SciKit learnings *RandomizedSearchCV*[4] to find the best performing hyper parameters to use in a ten fold cross validation. Randomized search finds the best combination of parameters for the model by trying every single possible combination of predefined parameter values.
For Linear Regression different types of solvers, penalties and C-values were to be tested. According to the random search, the best combination of parameters for the model are LogisticRegression(penalty='l2', C=2.0674811789304073, solver='liblinear'). Note that only 3 different solvers were tested, this is because there are 5 different solvers for the LogisticRegression() function, and the computing time for 3 solvers was already long enough. [6]
The parameters for LDA were 'solver' and 'shrinkage' and for QDA 'reg_param'. The program also test for different decision threshholds, 'r'. The best parameters for the models were the default for LDA and 'reg_param' = 0.2 for QDA. The best threshold for the two models were 'r' = 0.470 for LDA and 'r' = 0.475 for QDA.
The parameters used for Tree Based models were: Maximum tree depth, Minimum split, number of estimators and the use of either the 'gini index' or 'entropy' loss functions. The most optimal random forest method managed to perform an average of 0.831% accuracy in ten fold cross validation.
When using k-NN one does not need to implement randomized search since the only parameter to be tuned is the k. The optimal K was found using cross validation and iterating over different values of K.

5 Feature importance task

In this task additional analysis of four specific aspects of the movie data was carried out to gain deeper understanding of the history of the movie industry. For the feature importance task the quadratic discriminant analysis method was selected. It was chosen for the methods flexibility and possibility for low bias. Low bias will hopefully produce clearly different results for the different combinations of training data where a linear method might blur the lines because of the high bias. Using cross validation and fitting the QDA model to data while rotating different combinations of *All data*, *Female/Male words spoken*, *Year* and *Gross* the following information was obtained:

From Figure 6 we can see a clear increase in cross validation error when the data for total words spoken by each gender is ignored. Implying that the amount of words spoken by either gender has a significant impact of whether the lead role is male of female. When looking at the confusion matrices in table 3, for the same data, it is clear that the method over estimate the amount of male leads, compared to the other implementations. The exclusion of male and female words spoken will therefore be detrimental to the performance of the model.
Looking at the predictions made from single variable training QDA. We can see that all iterations score almost the same accuracy at around 75%. Looking at the confusion matrices in Table 4 we can see that the lead role is predicted to be male almost 100% of the time. Training methods to one single variable is therefore not significantly more effective than guessing male 100% of the time.

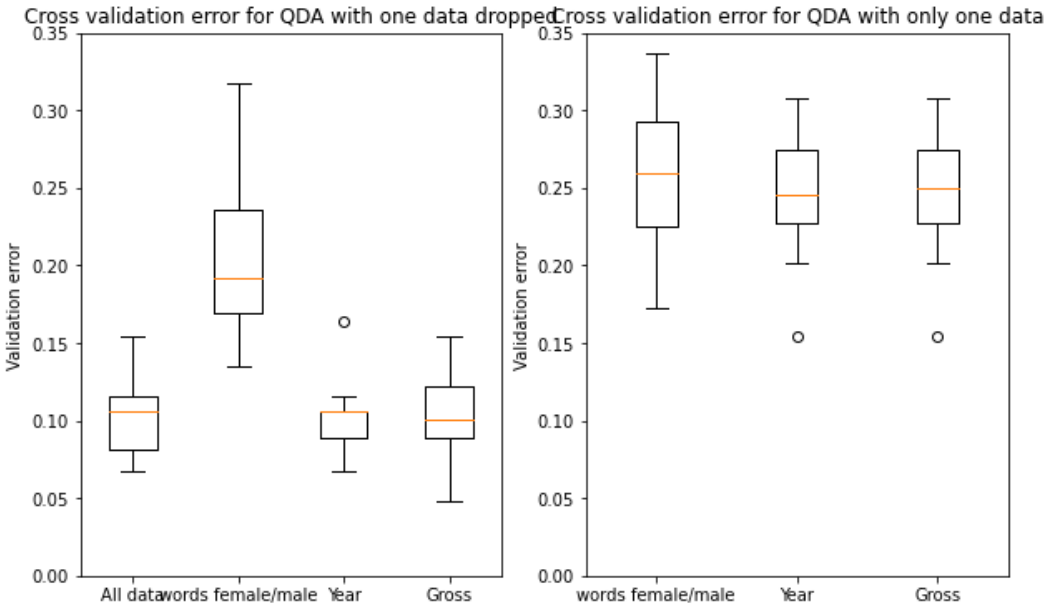

Figure 5:
On the left: implementations of QDA using all **except** the labeled data.
On the right: implementations of QDA using **only** the labeled data.

All data		
lead	Predict Female	Predict Male
True Female	114	54
True Male	20	351
Accuracy	0.887	

Without female/male words		
lead	Predict Female	Predict Male
True Female	75	48
True Male	59	357
Accuracy	0.814	

Without Year		
lead	Predict Female	Predict Male
True Female	101	37
True Male	33	368
Accuracy	0.885	

Without Gross		
lead	Predict Female	Predict Male
True Female	114	57
True Male	20	352
Accuracy	0.899	

Table 3: Confusion matrix for QDA using all **except** the labeled data.

Only female/male words		
lead	Predict Female	Predict Male
True Female	1	6
True Male	133	399
Accuracy	0.742	

Only Year		
lead	Predict Female	Predict Male
True Female	0	1
True Male	134	404
Accuracy	0.750	

Only Gross		
lead	Predict Female	Predict Male
True Female	0	0
True Male	134	405
Accuracy	0.751	

Table 4: Confusion matrix for QDA using **only** the labeled data.

6 Conclusions

The problem of predicting the binary lead role in a movie is clearly very complex. When implementing machine learning methods this problem can be simplified to the tuning of just a few parameters while still giving great performance. The feature importance task brought valuable information about choice of data and data relevance in relation to the desired result. There seems to be a positive correlation between including the data for spoken words for both genders in movies and the performance of the model. But this data alone could not be used to create a well performing model, barely surpassing the "all male - predicting model", highlighting the importance of model complexity and the bias-variance tradeoff.

The method chosen to put into production was the QDA method. The reasons being: that the methods high flexibility succeeded in giving real and valuable information in the feature importance task as well as it performing better than the other explored methods including its other family member LDA.

7 Appendix

7.1 Data Analysis

```
from cProfile import label
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import csv
from IPython . core . pylabtools import figsize
figsize(16, 8)

data = pd.read_csv('train.csv')

year = data.groupby("Year")

male = (year["Number of male actors"]).sum()
female = (year["Number of female actors"]).sum()
male.plot(label = " Male actors ")
female.plot(label = " Female actors ")
plt.ylabel(" Number of actors with speaking roles ")
plt.title(" Total number of actors separated by gender per year (1939 ...
    -2015) ")









with open('train.csv') as csvfile:
    datatypes = ["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","Lead"]
    reader = csv.DictReader(csvfile)
    Years = {}
    MaleDomMovies = []
    FemaleDomMovies = []
    MeanM = [0 for x in range(0,2015)]
    MeanF = [0 for x in range(0,2015)]
    i = 0
    m = [0 for x in range(0,2015)]
    f = [0 for x in range(0,2015)]
    TotWF = [0 for x in range(0,2015)]
    TotWM = [0 for x in range(0,2015)]
    TotW = [0 for x in range(0,2015)]
    TotM = [0 for x in range(0,2015)]
    Cash = [0 for x in range(0,2015)]


    for row in reader:
        if row["Year"] in Years:
            pass
        else:
            Years[row["Year"]] = {"m": 0,"f": 0,"TotWF":0,"TotWM":0, ...
                "TotW":0, "TotM":0, "Cash":0, "Mov":0}



        if row[datatypes[-1]] == "Male":
            Years[row["Year"]]["m"] += 1
```

```
54          else:
55              Years[row["Year"]]["f"] += 1
56
57
58
59          if int(row["Number words male"]) > int(row["Number words ...
                female"]):
60              MaleDomMovies.append(float(row["Gross"]))
61          else:
62              FemaleDomMovies.append(float(row["Gross"]))
63
64
65
66          Years[row["Year"]]["TotWF"] += int(row["Number words female"])
67          Years[row["Year"]]["TotWM"] += int(row["Number words male"])
68          Years[row["Year"]]["TotW"] += int(row["Number of female actors"])
69          Years[row["Year"]]["TotM"] += int(row["Number of male actors"])
70          Years[row["Year"]]["Cash"] += float(row["Gross"])
71          Years[row["Year"]]["Mov"] += 1
72          i += 1
73      datasize = i
74
75      # Y =[Year for Year in Years]
76      # print(max(Y), min(Y))
77
78      for i in range(1939,2015,1):
79          if str(i) in Years:
80              # print(Years[str(i)])
81              f[i] = Years[str(i)]['f']
82              m[i] = Years[str(i)]['m']
83              TotWF[i] = Years[str(i)]['TotWF']
84              TotWM[i] = Years[str(i)]['TotWM']
85              TotW[i] = Years[str(i)]['TotW']
86              TotM[i] = Years[str(i)]['TotM']
87              MeanM[i] = ...
                    Years[str(i)]['TotM']/Years[str(i)]['Mov']/(Years[str(i)]['TotM']+Years[str(i)
88              MeanF[i] = ...
                    Years[str(i)]['TotW']/Years[str(i)]['Mov']/(Years[str(i)]['TotM']+Years[str(i)
89              Cash[i] = Years[str(i)]['Cash']
90          else:
91              pass
92
93      FG = sum(FemaleDomMovies)/len(FemaleDomMovies)
94      MG = sum(MaleDomMovies)/len(MaleDomMovies)
95
96      x = [x for x in range(1939,2015)]
97
98
99      xn = np.linspace(1939, 2015, 1000)
100     N = np.polyfit(x, MeanF[1939:2015], 1)
101     N2 = np.polyfit(x, MeanM[1939:2015], 1)
102     avgF = np.polyval(N*1000, xn)
103     avgM = np.polyval(N2*1000, xn)
104     print(N)
105     print(N2)
106     plt.plot(xn, avgF, label='Trend of female actors vs male per movie')
107     plt.plot(xn, avgM, label='Trend of male actors vs female per movie')
108
109     print(f"Movies with male leads = {sum(m)}\nMovies with female ...
            leads = {sum(f)}")
110     print(f"Male dominated movies = {len(MaleDomMovies)}\nFemale ...
            dominated movies = {len(FemaleDomMovies)}")
111     print(f"Average gross profits of male dominated movies = ...
            {MG}\nAverage gross profits of female dominated movies = {FG}")
```

```
    print(f"Male dominated movies make approximately ...
        {int((MG-FG)/FG*100)}% more gross profits on average")


plt.legend()
plt.show()
```

7.2 Logistic Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.linear_model as skl_lm
import sklearn.model_selection as skl_ms
from scipy.stats import loguniform
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
import csv

# Read the csv file
data = pd.read_csv("train.csv")
# data.info()

# First we need to split the data into a training and testing data ...
    set at random
np.random.seed(1)
train_i = np.random.choice(data.shape[0], size=300, replace=False)
Index = data.index.isin(train_i)
train = data.iloc[Index]
test = data.iloc[~Index]

train_y = train['Lead']  # The output
train_x = train[['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']]

test_y = test['Lead']
test_x = test[['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']]

model = skl_lm.LogisticRegression(penalty='l2', C=2.0674811789304073, ...
    solver='liblinear')  # Here we create the solver model and the ...
    prediction
model.fit(train_x, train_y)

# Prediction
Female = 1
Male = 0
prediction_probability = model.predict_proba(test_x)
print(model.classes_, '\n')

prediction_label = np.empty(len(test_x))
prediction_label = np.where(prediction_probability[:, 0] >= 0.5, ...
    'Female', 'Male')

```

```
print(prediction_label[0:10])

X = data.drop(columns=['Lead'])
y = data['Lead']
n_fold = 10
missclassification = np.zeros((n_fold, 1))
cross_validation = skl_ms.KFold(n_splits=n_fold, random_state=1, ...
    shuffle=True)
for i, (Index, val_index) in enumerate(cross_validation.split(X)):
    train_x, x_val = X.iloc[Index], X.iloc[val_index]
    train_y, y_val = y.iloc[Index], y.iloc[val_index]
    model.fit = (train_x, train_y)
    prediction = model.predict(x_val)
    missclassification[int(i)] = np.mean(prediction != y_val)

plt.boxplot(missclassification)
plt.title('Cross validation error')
plt.ylabel('Validation error')
plt.xlabel("Logistic regression")
plt.show()

# Now we just need the confusion-matrix, and the accuracy
print('Confusion matrix: \n')
print(pd.crosstab(prediction_label, test_y), '\n')

print(f'Accuracy: {np.mean(prediction_label == test_y):.3f}')

# Here we tune the model with Random Search

tuningModel = skl_lm.LogisticRegression()
# Here we define evaluation
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# Here we define the search space
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear']  # Here are the ...
    different parameters that will be evaluated
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']  # The 4 types ...
    of penalties
space['C'] = loguniform(1e-5, 100)  # And a bunch of C-values
# Here we define the search
search = RandomizedSearchCV(model, space, n_iter=500, ...
    scoring='accuracy', n_jobs=-1, cv=cv, random_state=1)
# execute the search
result = search.fit(train_x, train_y)
# and lastly summarize the result
print('Best result: %s' % result.best_score_)  # Here the best result ...
    prints
print('Best Hyperparameters: %s' % result.best_params_)  # Here the ...
    hyperparameters that achieved the best result prints
```

7.3 Discriminant analysis

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

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

import sklearn.discriminant_analysis as skl_da
import sklearn.model_selection as skl_ms

```

```
from IPython.core.pylabtools import figsize
figsize(10, 6)

import warnings
warnings.filterwarnings("ignore")

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

#sampling indices for training
np.random.seed(1)

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

params = {'LDA' : {'solver':['lsqr', 'eigen'], 'shrinkage':
                   [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]},
          'LDA_svd' : {'solver':['svd']},
          'QDA': {'reg_param':[0, 0.2,0.4, 0.6, 0.8, 1]}}

models = {'LDA':skl_da.LinearDiscriminantAnalysis(),
          'LDA_svd':skl_da.LinearDiscriminantAnalysis(),
          'QDA':skl_da.QuadraticDiscriminantAnalysis()}

for mod in models:
    rand = skl_ms.RandomizedSearchCV(models[mod], params[mod])
    rand.fit(X , y)
    print(f'Best score for {mod}: {rand.best_score_:.3f}')
    print(f'Best parameters for {mod}: {rand.best_params_}')

print()
#Cross validation for both models
threshold = np.linspace(0, 1, 201)
n_runs =10
models={'LDA':skl_da.LinearDiscriminantAnalysis(),
        'QDA':skl_da.QuadraticDiscriminantAnalysis(priors=None, ...
            reg_param=0.2,
                                                   store_covariance=False)}

missclassifications = np.zeros((n_runs, len(models)))
scores = {'LDA': np.zeros((n_runs, len(threshold))),
          'QDA':np.zeros((n_runs , len(threshold)))}

#Split into 10 sets and use 1 as validation 9 as training and rotate
cv = skl_ms.KFold(n_splits=n_runs, random_state=1, shuffle=True)
for p, (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]

    i=0
    predict_prob = {}
    #Make the crossvalidation for both models
    for m in models:
        model = models[m]
        model.fit(X_train, y_train)
        prediction = model.predict(X_val)
        missclassifications[p, i] = np.mean(prediction != y_val)

        i = i+1
        predict_prob[m] = models[m].predict_proba(X_val)
        for j, r in enumerate(threshold):
            predict_class = np.where(predict_prob[m][:, 0] >= r,
                                     'Female', 'Male')
```

```
            scores[m][p, j] = np.mean(y_val == predict_class)

best_r = {}
avg_score = {}
for mod in scores:
    mean_scores = np.mean(scores[mod], axis=0)
    avg_score[mod] = np.amax(mean_scores)
    max_acc_ind = np.where(mean_scores == np.amax(mean_scores))
    best_r[mod] = threshold[max_acc_ind][0]

print(f'The average score of the models with {n_runs}-fold ...
    crossvalidation are:')
for mod in avg_score:
    print(f'{mod} (with r = {best_r[mod]:.3f}) average score: ...
        {round(avg_score[mod], 3)}')

#Boxplot of both models
print(f'Misclassification rate for LDA is: {np.mean([item[0] for item ...
    in missclassifications]):.3f}')
print(f'Misclassification rate for QDA is: {np.mean([item[1] for item ...
    in missclassifications]):.3f}')
plt.boxplot(missclassifications)
plt.title('Cross validation error for DA methods')
plt.xticks(np.arange(2)+1, ('LDA', 'QDA'))
plt.ylabel('Validation error')
plt.show()
```

7.4 k-NN

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.model_selection as skl_ms
import sklearn.neighbors as skl_nb


data = pd.read_csv("train.csv")
np.random.seed(1)
train_i = np.random.choice(data.shape[0], size=300, replace=False)
Index = data.index.isin(train_i)
train = data.iloc[Index]
test = data.iloc[~Index]

train_y = train['Lead']  # The output
train_x = train[['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']]

test_y = test['Lead']
test_x = test[['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_fold = 10

cross_v = skl_ms.KFold(n_splits=n_fold, random_state=2, shuffle=True)
```

```
K = np.arange(1, 100)
missclassification = np.zeros(len(K))
X = data.drop(columns=['Lead'])
Y = data['Lead']
for train_index, val_index in cross_v.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, k in enumerate(K):
        model1 = skl_nb.KNeighborsClassifier(n_neighbors=k)
        model1.fit(X_train, Y_train)
        prediction1 = model1.predict(X_val)
        missclassification[j] += np.mean(prediction1 != Y_val)


missclassification /= n_fold
plt.plot(K, missclassification)
plt.title('Cross validation error for kNN')
plt.xlabel('k')
plt.ylabel('error')
plt.show()
n_fold = 10
K2 = np.arange(4, 20)
for train_index, val_index in cross_v.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, k in enumerate(K2):
        model2 = skl_nb.KNeighborsClassifier(n_neighbors=k)
        model2.fit(X_train, Y_train)
        prediction2 = model2.predict(X_val)
        missclassification[j] += np.mean(prediction2 != Y_val)
missclassification /= n_fold
plt.plot(K, missclassification)
plt.title('Cross validation error for kNN')
plt.xlabel('k')
plt.ylabel('error')
plt.show()
k = 10
model3 = skl_nb.KNeighborsClassifier(n_neighbors=k)
model3.fit(train_x, train_y)
prediction3 = model3.predict(test_x)
print('Confusion matrix: \n')
print(pd.crosstab(prediction3, test_y), '\n')
print(f"Accuracy:{np.mean(prediction3 == test_y):.3f}")
n_fold = 100

cross_v = skl_ms.KFold(n_splits=n_fold, random_state=2, shuffle=True)
Y = data['Lead']
X = data.drop(columns=['Lead'])
missclassification3 = []


for train_index2, val_index2 in cross_v.split(X):
    X_train2, X_val2 = X.iloc[train_index2], X.iloc[val_index2]
    Y_train2, Y_val2 = Y.iloc[train_index2], Y.iloc[val_index2]
    model3.fit(X_train2, Y_train2)
    prediction3 = model3.predict(X_val2)
    missclassification3.append(np.mean(prediction3 != Y_val2))

plt.boxplot(missclassification3)
plt.title('validation error')
plt.show()
```

7.5 Tree based methods

```
1
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6
7
8  import sklearn.model_selection as skl_ms
9
10 from sklearn import tree
11 from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
12
13 import graphviz
14 from sklearn.utils import column_or_1d
15
16
17 S = 600
18
19 LEAD = pd.read_csv('train.csv')
20 np.random.seed(1)
21
22 #Data val av test/tr ningsdata
23 trainIndex = np.random.choice(LEAD.shape[0], size = S, replace=False)
24 train = LEAD.iloc[trainIndex]
25 test = LEAD.iloc[~LEAD.index.isin(trainIndex)]
26 print(f'{len(LEAD)} datapoints {S} used as trainingdata and ...
       {len(LEAD)-S} used as validation data')
27
28 #Anpassa modell efetr tr ningsdata
29 X_train = train.copy().drop(columns=['Lead'])
30 y_train = train['Lead']
31 model = tree.DecisionTreeClassifier(max_depth=2)
32 model.fit(X=X_train, y=y_train)
33
34 #F r att visa tr d
35 #dot_data = tree.export_graphviz(model, out_file=None, feature_names= ...
       X_train.columns, class_names = model.classes_, filled= True, ...
       rounded=True, leaves_parallel=True, proportion=True)
36 #graph = graphviz.Source(dot_data)
37
38 #Testa modell ge accuracy och confusion matrix
39 X_test = test.copy().drop(columns=['Lead'])
40 y_test = test['Lead']
41 y_predict = model.predict(X_test)
42 print(f'Normal accuracy is {np.mean(y_predict == y_test):.3f} %')
43 cross = pd.crosstab(y_predict, y_test)
44 print(cross)
45
46 #Bootstrap calssifier
47 model = BaggingClassifier()
48 model.fit(X_train,y_train)
49 error = np.mean(model.predict(X_test) == y_test)
50 print(f'Bagging classifier accuracy is {error:.3f}')
51
52
53 #Random forest classifier
54 model = RandomForestClassifier(n_estimators=100)
55 model.fit(X_train, y_train)
56 error = np.mean(model.predict(X_test) == y_test)
57 print(f'Random forest classifier accuracy is {error:.3f}')
58
59
60 #Cross validation part
61 print('Param search:')
```

```
n_runs = 10
models = []
criterions = ['gini', 'entropy']
max_d = [1,5,10,12,13,14,15,16,17,18,19,20]
min_split = [3,4,5]
estim = [100,125,150,175,200,225,250,300]
#d = [15,15]
#s = [4,3]
#e = [100,200]
#c = ['gini','entropy']
#for c in criterions:
#    for d in max_d:
#        for s in min_split:
#            for e in estim:
#                models.append(RandomForestClassifier(n_estimators= ...
    e, criterion = c, max_depth = d, min_samples_split=s))
#models.append(RandomForestClassifier(n_estimators= e[0], criterion = ...
    c[0], max_depth = d[0], min_samples_split=s[0]))
#models.append(RandomForestClassifier(n_estimators= e[1], criterion = ...
    c[1], max_depth = d[1], min_samples_split=s[1]))

X = LEAD.drop(columns= ['Lead'])
y = LEAD['Lead']
params = {'n_estimators':estim,'min_samples_split': min_split, ...
    'criterion':criterions, 'max_depth':max_d}
model = RandomForestClassifier()
rand = skl_ms.RandomizedSearchCV(model, params)
rand.fit(X,y)
print(f'Best score: {rand.best_score_}')
print(f'Best params: {rand.best_params_}')



model = ...
    RandomForestClassifier(n_estimators=150,min_samples_split=5,max_depth=13,criterion='gini')
missclassifications = np.zeros((n_runs, 1))


print('Cross validation:')
#Split into 10 sets and use 1 as validation 9 as training and rotate
cv = skl_ms.KFold(n_splits=n_runs, 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]

    model.fit(X_train, y_train)
    prediction = model.predict(X_val)
    missclassifications[i] = np.mean(prediction != y_val)

plt.boxplot(missclassifications)
plt.title('Cross validation error for Tree methods')
#plt.xticks(np.arange(1), ('estimators = 150, splits = 4, crit = ...
    gini, depth = 13'))
plt.ylabel('Validation error')
plt.xlabel('estimators = 150, splits = 4, crit = gini, depth = 13')
plt.show()
```

References

[1] S mair lecture 4 - discriminant analysis, k-nearest neighbors. https://uppsala.instructure.com/courses/46077/pages/lectures used 2022-03-02.

[2] Scikit learn - discriminant analysis.
https://scikit-learn.org/stable/modules/lda_qda.html
used 2022-03-02.

[3] Scikit learn - ensamble.
https://scikit-learn.org/stable/modules/ensemble.html used 2022-03-02.

[4] Scikit learn - randomizedsearchcv.
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
used 2022-03-02.

[5] Scikit learn - tree.
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
used 2022-03-02.

[6] sklearn.linear_model.logisticregression. used 2022-03-02.

[7] F. Lindsten T. B. Schön A. Lindholm, N. Wahlström. *Machine learning - A first course for engineers and scientists*. Cambridge University Press, 2021.