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

**André Ramos Ekengren**

**Karl Rosengren**

**Martin Leino**

## Abstract

The purpose of this paper was to analyze data from the movie industry and implement classification methods to predict whether the lead role was held by a male or a female, and choose the model that performed the best. The three methods chosen were Logistic Regression, Random Forest Classifier and K-Nearest Neighbors. All three were implemented successfully but the most reliable one was the Logistic Regression model. This is also the one the group chose to be applied to a future dataset for examination. In the data analysis the group found that women are less represented as leading roles in movies according to the dataset. Which were plotted in a bar plot as well as with a fitted line through the data for visualization purposes. The importance of some specific variables in the dataset were discussed and analyzed alas the findings were quite problematic to analyze. This was because with the removal of a variable (all) the error decreased. This lead us to realize that the model used, Random Forest Classifier, was overfitted as the error decreased whilst the complexity decreased.

## 1 Introduction

The following paper has the intention of describing the implementation and analysis of three Machine Learning models applied to a dataset with variables relevant to the cinema industry with which the models are to predict the sex of the lead actor. The models to choose from were presented in a list from which we chose the following: Logistic Regression, K-nearest neighbors and Random Forest classifier. The goal is to tune the models and choose the one that performs the best.

The background for this study is the article "Film Dialogue from 2,000 screenplays Broken Down by Gender and Age" [1], in which the authors investigate relationships between the amount of speaking-time given to women and men in Hollywood movies. The authors come to the conclusion that men seem to speak more on average in the films that were considered.

## 2 Data analysis

Before starting to train the models, we set out to answer the following three questions about trends in the dataset:

- Do men or women dominate speaking roles in Hollywood movies?
- Has gender balance in speaking roles changed over time (i.e. years)?

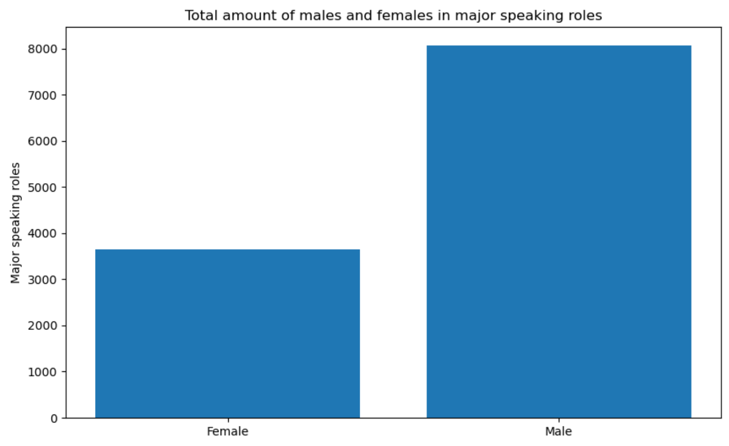

Figure 1: A bar plot of the total amount of male and female major speaking roles.

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

Inspecting Figure 1 shows that the total amount of men that have had major speaking roles are about twice as many as the women. If we assume that the amount of major speaking roles is a good metric for how dominant a gender is in Hollywood, then indeed the men are dominant.

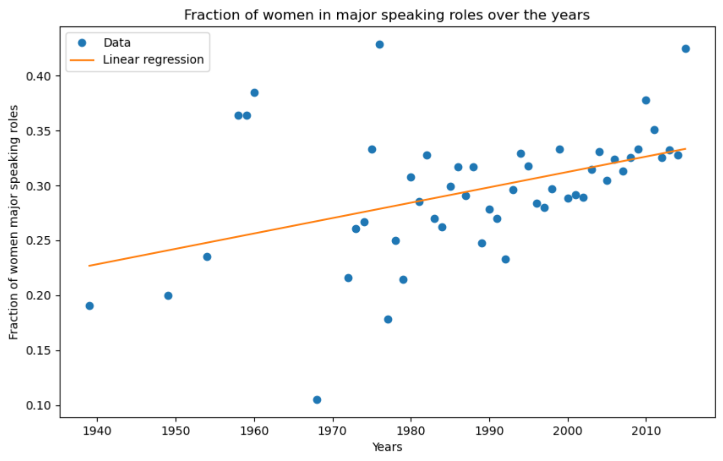

Figure 2: Plot of the fraction of women with major speaking roles over the years. A straight line is fitted to the data.

As for whether the gender balance has improved with time, Figure 2 shows how the fraction of the major speaking roles that have been women have changed over the years. Two things to note are that the trend seems to be positive, but also that women have always made up below 50 percent of all major speaking roles.

In Figure 3 the gross profit made by the films is plotted against the fraction of words spoken by males. The line fitted to the data has a slight positive slope, indicating that movies in which men speak more than women might make more profit.

Since the split between male and female lead in the data is about 3:1, the worst-classifier that picks only one class will either be right about 75% of the time if it picks male, and 25% of the time if it picks female. Therefore, we want an accuracy higher than 75% for our model to be better than this worst-classifier.

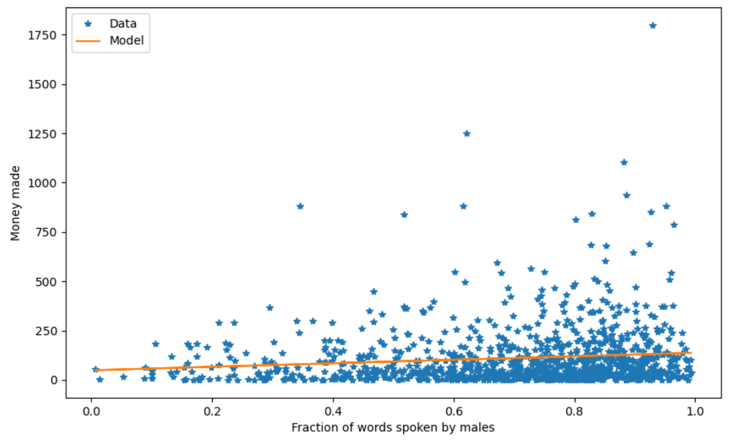

Figure 3: The fraction of words spoken by males plotted versus the profit for each film in the training dataset. A straight line is fitted to the data.

## 3 Methods

The classification methods that have been considered in this project are $k$-nearest neighbors ($k$-NN), logistic regression and random forests. There are Python packages available to implement these methods; all the details can be found in the Appendix. In the subsections following below, each of the three methods is described and some important aspects of the implementation are discussed. All the available independent variables were used in each classifier, and only the output (*Lead*) was considered categorical. In order to compare the performance of the three classifiers, 25 % of all the available data was withdrawn from the training data, allowing one to compare the accuracy of each method on data that had not been seen during training. This was done by using the Scikit-Learn function *train_test_split* to split the data into a training set and a test[1] set. The training set is the one the model is training on and later tests the performance on the test set. [7] The reason that the data was split instead of simply doing k-fold cross validation on the full data was so that the evaluation of the model would be as valid as possible while also choosing hyperparameters using k-fold cross validation on the training set. The same random state (42) was chosen in the train-test split for each classifier in order to get comparable results.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,y,test_size=0.25,
                     random_state=42)
```

When finding the optimal parameters for the $k$-NN and logistic regression models, 10-fold cross-validation was used. A choice of about 10 folds seems reasonable when the training can turn out to be computationally demanding [8, p. 62].

In the end, the classifier giving the highest accuracy on the test data was chosen for use in production on the actual test data.

### 3.1 $k$-nearest neighbors

Similar to other classification methods, $k$-nearest neighbors classifies data through a "majority vote" among training data points. More precisely, decision boundaries are drawn based on the most common class among the $k$ training data points that are closest to each location in the set in which the covariates take their values. (If two or more classes would happen to be equally common within a certain

[1]Note that the data referred to as "test data" in this section is not the actual test data, but merely a subset of the training data held out for model comparisons.

region, predictions for test data points falling in that region could be made by some randomization technique.)

In Python, a $k$-NN classifier can be created using the commands

```
import sklearn.neighbors as skl_nb
kNN_model = skl_nb.KNeighborsClassifier(n_neighbors=k)
```

where $k$ needs to be specified. Then, before fitting the training data to the model, it is wise to rescale the data. Namely, since covariates of consideration can vary greatly in their size of magnitude, one should rescale the variables to similar magnitudes in order for the $k$-NN classifier not to become too dependent on just one or a few covariates when distances[2] are measured (here, the Euclidean norm was used). In the data set at hand, one can note that, for example, the variable *Total words* has values of several thousand for each film, whereas the variables *Number of female actors* and *Number of male actors* range between significantly smaller values. Since it was not desired to give the former variable a greater importance than the two latter ones, rescaling was made here. In Python, rescaling was made by using the commands

```
import sklearn.preprocessing as skl_pre
scaler = skl_pre.StandardScaler().fit(X_train)
```

and, when fitting the classifier to the data,

```
kNN_model.fit(scaler.transform(X_train), y_train)
```

A *StandardScaler* object scales the data by from each variable subtracting the mean and dividing by the standard deviation; that is, the scaled variables are given mean 0 and unit variance [6].

The optimal value of $k$ was decided through 10-fold cross-validation. This value was found to be $k = 7$, and when used on the "held-out" data mentioned before, the accuracy obtained was 0.7577.

## 3.2 Logistic regression

In this binary classification problem, a logistic regression model aims to estimate the class conditional probabilites $p(y = 1 \mid \mathbf{x})$ and $p(y = 0 \mid \mathbf{x})$, and making a prediction of the class given the probability. Here, the positive class '1' corresponds to a movie having a female lead and '0' as having a male lead. The class conditional probability $p(y = 1 \mid \mathbf{x})$ is modeled using the logistic function, resulting in

$$g(\mathbf{x}; \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}^T \mathbf{x}}}{1 + e^{\boldsymbol{\theta}^T \mathbf{x}}}$$

[8, pp. 44-45]. The parameter vector $\boldsymbol{\theta}$ is then chosen such that

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} -\frac{1}{n}\sum_{i=1}^{n} y_i \ln(g(\mathbf{x}; \boldsymbol{\theta})) + (1 - y_i)\ln(1 - g(\mathbf{x}; \boldsymbol{\theta}))$$

[8, p. 46], and the model will make predictions given some data according to

$$\hat{y}(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}; \hat{\boldsymbol{\theta}}) > 0.5 \\ 0 & \text{if } g(\mathbf{x}; \hat{\boldsymbol{\theta}}) \leq 0.5 \end{cases}$$

[8, p.47]. In the implementation of the model, the training data and test data were first scaled using `sklearn.preprocessing.StandardScaler()` [6]. The model was then initialized using the logistic regression class from Scikit-Learn. Afterwards, a grid search [4] was done to tune the hyperparameters to maximize accuracy, which is done by 10-fold cross validation. The hyperparameters that were considered were the ridge regularization constant and the class weight. The latter was considered because of the imbalance between male and female in the data, as passing 'balance', it will try to correct by calculating weights for the classes. Once finished, the best estimator of the training data was given by `LogisticRegression(C=35.622478902624444, class_weight=None)`. Testing this model on the test data gave an accuracy of about 0.88, a ROC-AUC score of about 0.81 and an f1-score of about 0.75.

[2]Commonly, but not necessarily, measured in the Euclidean norm.

### 3.3 Random Forest

Random forests is a forest because the algorithm is made up from many decision trees, thus forming a forest. The random forest is a Machine Learning algorithm that tries to fit several classifiers on sub-sets of the data and obtains an average to control the performance and overfitting. The average in the performance part of classification is done through the "average result" of the splits, the majority with a certain decision is the decision that is in the output.

The implementation of the random forest classifier was coded in JupyterLab and was done as follows:

```
import pandas as pd
df = pd.read_csv("train.csv", encoding = "utf-8")
```

Here *LabelEncoder* is imported to encode the variable *Lead* which defines the leading role of the movie with the sex of the actor. The machine can not interpret strings in the way we need it to, as such we change the encoding from female/male to 0/1. [5]

```
from sklearn.preprocessing import LabelEncoder
encode = LabelEncoder()
df['Lead'] = encode.fit_transform(df['Lead'])
```

The next step is to import our classifier *RandomForestClassifier* from Sklearn. The random forest produces a forest based on the parameter *n_estimators* which acts as the number of trees in the forests, an increase in the number of trees in the forest will increase the performance drastically for lower numbers but the increase becomes less as the number of trees reaches a certain limit. [2]

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10, random_state = 42)
model.fit(X_train,Y_train)
model_prediction = model.predict(X_test)
```

The final step is to rate the model's performance and this is done through the *Accuracy_score* and *confusion_matrix*. Where *Accuracy_score* returns a float representing the percentage of correct "guesses". The *confusion_matrix* shows us the model's guesses and from that we can tell where the model has correctly predicted the sex of the lead or where it has gotten it wrong. [3]

```
from sklearn.metrics import accuracy_score, confusion_matrix
print("Accuracy Score : ", accuracy_score(Y_test, model_prediction))
print("Misclassification error: ", 1-accuracy_score(Y_test,model_prediction))
confusion_matrix(Y_test, model_prediction)
```

### 3.4 Choice of model

The summary of the performance of the different models is described in Table 1. This performance was what made us take the decision we did, which was to choose logistic regression as the model to represent our findings. The decision to only compare accuracy and not include other metrics is due to the fact that none of the models performed equally well based on accuracy. If two or more models performed in ways that were difficult to distinguish, then metrics such as false positives and false negatives might have been considered in the choice. The logistic regression also had a ROC-AUC score of 0.81 and an f1-score of 0.75, which is not completely terrible.

Table 1: The accuracies for the three classifiers when applied to the test data.

| Classifier | Accuracy |
|---|---|
| $k$-NN | 0.7577 |
| Logistic regression | 0.8846 |
| Random forests | 0.8423 |

## 4 Feature importance

For the analysis of this question we used Random Forest Classification method, insted of Logistical Regression which performed better, on the dataset whilst changing the dataset to perform the classification without certain variables as described in the assignment. The reason why Random Forest Classification was chosen for this problem was that we were particularly interested in how the number of covariates influenced the performance while keeping the maximum tree depth constant.

The importance of the following four variables was studied here: *Gross*, *Number of words female*, *Number of words male* and *Year*. The performance of the different models explored, measured in terms of their accuracy[3] and displayed in confusion matrices, is found in the Appendix as well in Table 2 and 3 below.

First of all, the accuracy for the whole dataset was about 0.785, with a false positive (female) of 15 and a false negative of 41, which acts as our reference points for our experiments. By excluding the different variables from the dataset one by one and training the model, the accuracy was increased in all the cases. When all the four variables were excluded from the dataset, however, the accuracy decreased.

Also, the data was fitted to models in which only one of the four variables mentioned above was used at the time as a covariate. Out of these four models, the one having *Year* as its single covariate performed best in terms of its accuracy. Considering the fact that the performance increased the most when dropping this variable from the "full" model, this is a rather unexpected result as most, or at least some of, these variables seem to be of value for a good prediction. Seeing this behaviour in regards of the misclassification error dropping while the complexity is decreasing we could make a statement that the model is currently overfitted, while the behaviour seen when dropping all the variables suddenly giving less accuracy might be that the model then became too simple. [8, p. 64] Whilst having a good performance in a high complexity environment is positive, it is not positive if the performance is false due to overfitting.

One problem to take into consideration, which we have not handled in practice however, is the correlation between the independent variables. It might therefore be possible to improve our models further by the use of interaction terms (see, for example, [9]).

Summing up, the procedure of dropping variables as described above illustrates the trade-off on the model-complexity scale: a more complex model might be better at fitting to the training data, but in our situation the less complex models given by dropping one variable seem to generalize better to new data. Perhaps an even better result would be seen if more variables were dropped.

Table 2: The accuracies without specific variables of the dataset.

| Without 'Variable' | Accuracy |
|---|---|
| Number Words Male | 0.823 |
| Number Words Female | 0.7961 |
| Year | 0.8576 |
| Gross | 0.8192 |
| All variable | 0.7807 |

Table 3: The accuracies with ONLY specific variables of the dataset.

| Only 'Variable' | Accuracy |
|---|---|
| Number Words Male | 0.5923 |
| Number Words Female | 0.65 |
| Year | 0.7192 |
| Gross | 0.6653 |

[3]Which is 1 minus the missclassification error.

## 5 Conclusions

Analysis of the dataset shows that males are represented more than females in the movie industry, and that male dominant movies make more profit. Over time, however, the difference in representation seems to decrease (see Figure 2). After training and testing, the model with the highest accuracy was the logistic regression model. For this reason, the logistic regression model was chosen to be the model to put in production.

One important conclusion from the project is that the model does not necessarily make the most correct predictions on test data when all possible independent variables are included. This suggests that one should perhaps leave out some of the available variables when fitting a model as higher complexity does not always lead to a more accurate model.

Reflecting on the way in which we chose to split the data in the beginning (making a 75/25 % split into training/hold-out validation data for comparisons between the models later on), one can conclude that this could have been done differently. Maybe it would have been better to use $k$-fold cross-validation as a means of comparing the performance of the models. This is an approach that we would consider for similar projects in the future.

## References

[1] Daniels M. Anderson, H. Film dialogue from 2,000 screenplays broken down by gender and age. https://pudding.cool/2017/03/film-dialogue/, 2016.

[2] Scikit Learn. sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[3] Scikit Learn. sklearn.metrics.accuracy_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

[4] Scikit Learn. sklearn.model_selection.GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[5] Scikit Learn. sklearn.preprocessing.LabelEncoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

[6] Scikit Learn. sklearn.preprocessing.StandardScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[7] Scikit Learn. sklearn.model_selection.train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[8] Wahlström N. Lindsten F. Schön T.B. Lindholm, A. *Draft (April 30, 2021) of Machine Learning – A First Course for Engineers and Scientists*. http://smlbook.org, 2021.

[9] Wikipedia. Interaction (statistics). https://en.wikipedia.org/wiki/Interaction_(statistics).

## Appendix

Here the code from the project is presented.

### Figure 1 and 2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.linear_model as skl_lm

import statsmodels.api as sm

#from IPython.display import set_matplotlib_formats
#set_matplotlib_formats('png')
from IPython.core.pylabtools import figsize
figsize(10, 6) # Width and hight
#plt.style.use('seaborn-white')

moviedata = pd.read_csv("train.csv")

# Fraction of female major speaking roles over time

actors_with_time = moviedata[["Year", "Number of female actors",
            "Number of male actors"]].groupby("Year").sum()
actors_with_time[["Fraction of women"]] = actors_with_time["Number of female actors"] /
                                        (actors_with_time["Number of female actors"] +
                                        actors_with_time["Number of male actors"])


Years = np.sort(moviedata["Year"].unique()).reshape(-1, 1)
model = skl_lm.LinearRegression()
model.fit(Years, actors_with_time["Fraction of women"])
actors_predict = model.predict(Years)
print(model.intercept_)
print(model.coef_)

X2 = sm.add_constant(Years)
est = sm.OLS(actors_with_time["Fraction of women"], X2)
est2 = est.fit()
print(est2.summary())
plt.plot(Years, actors_with_time["Fraction of women"], 'o', label="Data")
plt.plot(Years, actors_predict, label="Linear regression")
plt.title("Fraction of women in major speaking roles over the years")
plt.xlabel("Years")
plt.ylabel("Fraction of women major speaking roles")
plt.legend()
plt.show()

# Comparison of total major speaking roles
actors_female = sum(moviedata["Number of female actors"])
actors_male = sum(moviedata["Number of male actors"])
plt.bar(["Female", "Male"], [actors_female, actors_male])
plt.title("Total amount of males and females in major speaking roles")
plt.ylabel("Major speaking roles")
plt.show()
```

**Figure 3**

```
# Mini-project: data analysis task
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

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

import statsmodels.api as sm

plt.close('all')

film_data = pd.read_csv('train.csv', na_values='?').dropna()

year = film_data['Year']
female_actors_msp = film_data['Number of female actors']
male_actors_msp = film_data['Number of male actors']
gross = film_data['Gross']
words_female = film_data['Number words female']
words_male = film_data['Number words male']
words_lead = film_data['Number of words lead']
total_words = film_data['Total words']
lead = film_data['Lead']

plt.plot(words_female, words_male, '*')
plt.xlabel('Number of words, female')
plt.ylabel('Number of words, male')
plt.show()

plt.figure()

plt.plot(female_actors_msp, male_actors_msp, '*')
plt.plot([0,30],[0,30],'r')
plt.xlabel('Number of female actors with major speaking roles')
plt.ylabel('Number of male actors with major speaking roles')
plt.show()

more_men = 0
more_women = 0

for i in range(0,len(film_data)):
    if male_actors_msp.iloc[i] > female_actors_msp.iloc[i]:
        more_men += 1
    elif male_actors_msp.iloc[i] < female_actors_msp.iloc[i]:
        more_women += 1

fraction_more_women = more_women/len(film_data)
fraction_more_men = more_men/len(film_data)

print('Fraction of films with more women in major speaking roles:',
      round(fraction_more_women,2), '.')
print('Fraction of films with more men in major speaking roles:',
      round(fraction_more_men,2), '.')

"""
```

```
Has gender balance in speaking roles changed over time (i.e. years)?
"""

fraction_males_msp = []
for i in range(0,len(film_data)):
    fraction_male = male_actors_msp.iloc[i]/(female_actors_msp.iloc[i] +
                    male_actors_msp.iloc[i])
    fraction_males_msp.append(fraction_male)

plt.figure()
plt.plot(year,fraction_males_msp,'*')
plt.xlabel('Year')
plt.ylabel('Fraction of males in major speaking roles')
plt.show()


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

frac_words = np.array([])

for i in range(0,len(film_data)):
    if lead.iloc[i] == 'Male':
        words_spoken_male = words_male.iloc[i] + words_lead.iloc[i]
    else:
        words_spoken_male = words_male.iloc[i]

    frac_words_male = words_spoken_male/total_words.iloc[i]
    frac_words = np.append(frac_words, frac_words_male)

money_regression = skl_lm.LinearRegression()
money_regression.fit(X = np.column_stack([frac_words]), y = gross)
money_predictions = money_regression.predict(X = np.column_stack([frac_words]))
slope = money_regression.coef_
print('The slope of the line is', slope, '.')

# If in addition we want the P-value for the estimated slope:
X2 = sm.add_constant(frac_words)
est = sm.OLS(gross, X2)
est2 = est.fit()
print(est2.summary())

plt.figure()
plt.plot(frac_words,gross,'*',label='Data')
plt.plot(frac_words,money_predictions,label='Model')
plt.xlabel('Fraction of words spoken by males')
plt.ylabel('Money made')
plt.legend()
plt.show()
```

**k-NN classifier**

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

plt.close('all')

# Read data from file
film_data = pd.read_csv('train.csv', na_values='?').dropna()


# Create variables
X = film_data.drop('Lead', axis=1)
y = film_data['Lead']

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,
                                                    random_state=42)

n = len(film_data)
training_errors = np.array([])
k_vector = np.arange(1,200)

# Scale data
scaler = skl_pre.StandardScaler().fit(X_train)

# Fit kNN to the training data for different k
for k in k_vector:
    kNN_model = skl_nb.KNeighborsClassifier(n_neighbors=k)
    kNN_model.fit(scaler.transform(X_train), y_train)
    pred_kNN = kNN_model.predict(scaler.transform(X_train))

    # Calculate E_train
    no_missclassified = 0
    for i in range(0,len(pred_kNN)):
        if pred_kNN[i] != y_train.iloc[i]:
            no_missclassified += 1
    E_train = no_missclassified/n
    # Save E_train in a vector
    training_errors = np.append(training_errors, E_train)

# Plot the missclassification errors versus k
plt.plot(k_vector, training_errors, 'r')
plt.xlabel('k')
plt.ylabel('E_train')
plt.title('k-NN: Errors when all data used as training data')
plt.show()

"""
10-fold cross-validation
"""
X = X_train
y = y_train
# indices = np.arange(0,len(X_train))
# X.set_index(indices)

# Random permutation of data
n_fold = 10
cv = skl_ms.KFold(n_splits=n_fold, shuffle=True, random_state=2)
missclassification = np.zeros(len(k_vector))

# Loop over different training/validation datasets
```

```
for train_index, val_index in cv.split(X):
    X_tr, X_val = X.iloc[train_index], X.iloc[val_index]
    y_tr, y_val = y.iloc[train_index], y.iloc[val_index]

    # Fit model to training data, evaluate on validation data
    for j, k in enumerate(k_vector):
        model = skl_nb.KNeighborsClassifier(n_neighbors=k)
        model.fit(scaler.transform(X_tr), y_tr)
        prediction = model.predict(scaler.transform(X_val))
        missclassification[j] += np.mean(prediction != y_val)

missclassification /= n_fold

# Plot cross-validation errors
plt.figure()
plt.plot(k_vector, missclassification)
plt.title('Cross-validaton error for kNN')
plt.xlabel('k')
plt.ylabel('Validation error')
plt.show()

# Find best hyperparameter k
E_val_min = min(missclassification)
best_k = np.where(missclassification == E_val_min) + np.array([1])
print('The best k on validation data is', best_k, 'with a validation error of',
      E_val_min)

# Evaluate on "test" data held out for comparison with other methods
k = input('Write best_k as an int: ')
final_kNN = skl_nb.KNeighborsClassifier(n_neighbors=k)
final_kNN.fit(X_train,y_train)
final_pred = final_kNN.predict(scaler.transform(X_test))
final_error = np.mean(final_pred != y_test)
print('The estimated E_new for k-NN (k = ', best_k, ') is',
      round(final_error,4), '.')
```

### Logistic Regression Classifier

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

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.model_selection as skl_ms
import sklearn.metrics as skl_mtr


df = pd.read_csv("train.csv").dropna()

# Imbalance in the data.
# print(df["Lead"].value_counts())
# Male: 785
# Female: 254
# Choose female as positive class.

df["Lead"].replace(["Female", "Male"], [1, 0], inplace=True)

X = df.drop("Lead", axis=1)
```

```
Y = df["Lead"]
# Split data
X_train, X_test, y_train, y_test = skl_ms.train_test_split(X, Y,
                                    test_size=0.25, random_state = 42)

# Scale the data
scaler = skl_pre.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = skl_lm.LogisticRegression()
param_grid = {'C': np.logspace(-5, 5, 30),
        'class_weight' : ["None", "balanced"]}

# Do gridsearch for optimal regularization and class_weight.

clf = skl_ms.GridSearchCV(model, param_grid, scoring ="accuracy",
                          cv = 10)  # Maybe use f1 as scoring?
clf.fit(X_train_scaled, y_train)
optimal_model = clf.best_estimator_
print(clf.best_estimator_)
optimal_model.fit(X_train_scaled, y_train)
#print(clf1.best_estimator_)
#print(clf1.best_params_)
#print(clf1.best_score_)

test_prediction = optimal_model.predict(X_test_scaled)
print(f"ROC-AUC score: {skl_mtr.roc_auc_score(y_test, test_prediction)}")
# Should maybe use area under precision-recall curve
print(f"Accuracy on test data: {skl_mtr.accuracy_score(y_test, test_prediction)}")
print(f"F1 score on test data: {skl_mtr.f1_score(y_test, test_prediction)} ")
print(skl_mtr.confusion_matrix(y_test, test_prediction))
```

## 5.1 Random Forest Classifier

```
#Read data and create Pandas Dataframe
import pandas as pd
df = pd.read_csv("train.csv", encoding = "utf-8")

#Encode Lead from Female/Male to 0/1
from sklearn.preprocessing import LabelEncoder
encode = LabelEncoder()
df['Lead'] = encode.fit_transform(df['Lead'])

#Set X and Y as the dataframe and the target output as lead
X = df
X = df.drop(['Lead'],axis=1)
y = df['Lead']

#Import Train_test_split and RandomForestClassifier from sklearn as well as Sklearn.preprocessing
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size=0.25, random_state=42)

import sklearn.preprocessing as skl_pre
scaler = skl_pre.StandardScaler().fit(X_train)

model.fit(scaler.transform(X_train), Y_train)
```

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators = 10)
model.fit(X_train,Y_train)
model_prediction = model.predict(X_test)


from sklearn.metrics import accuracy_score, confusion_matrix

print("Accuracy Score : ", accuracy_score(Y_test, model_prediction))
print("Misclassification error: ", 1-accuracy_score(Y_test,model_prediction))
confusion_matrix(Y_test, model_prediction)
```

**Results from the feature importance task**

**Using all independent variables**

Accuracy Score : 0.7846153846153846 Misclassification error: 0.2153846153846154 array([[ 31, 41], [ 15, 173]], dtype=int64)

**Without *female words***

Accuracy Score : 0.7961538461538461 Misclassification error: 0.2038461538461539 array([[ 33, 39], [ 14, 174]], dtype=int64)

**Without *male words***

Accuracy Score : 0.823076923076923 Misclassification error: 0.17692307692307696 array([[ 34, 38], [ 8, 180]], dtype=int64)

**Without *Year***

Accuracy Score : 0.8576923076923076 Misclassification error: 0.14230769230769236 array([[ 43, 29], [ 8, 180]], dtype=int64)

**Without *Gross***

Accuracy Score : 0.8192307692307692 Misclassification error: 0.1807692307692308 array([[ 41, 31], [ 16, 172]], dtype=int64)

**Without all the four variables above**

Accuracy Score : 0.7807692307692308 Misclassification error: 0.21923076923076923 array([[ 31, 41], [ 16, 172]], dtype=int64)

**Only *Number words female***

Accuracy Score : 0.65 Misclassification error: 0.35 array([[ 17, 55], [ 36, 152]], dtype=int64)

**Only *Number words male***

Accuracy Score : 0.5923076923076923 Misclassification error: 0.4076923076923077 array([[ 17, 55], [ 51, 137]], dtype=int64)

**Only *Year***

Accuracy Score : 0.7192307692307692 Misclassification error: 0.28076923076923077 array([[ 3, 69], [ 4, 184]], dtype=int64)

**Only *Gross***

Accuracy Score : 0.6653846153846154 Misclassification error: 0.33461538461538465 array([[ 13, 59], [ 28, 160]], dtype=int64)