Do (wo)men talk too much in films?

Authors: Jannes van Poppelen, Linus Falk, Steven Bijl, Niklas Kostrzewa

Abstract

In this work, we train different machine learning models using a data set of 2000 Hollywood movies in order to accurately predict the sex of the lead role of a movie, given a list of features corresponding to each movie. Even though the training data is heavily biased towards a male lead, all trained models end up outperforming a naive classifier that merely predicts a male lead. The generative QDA model was found to give the best performance after a 10-fold cross-validation, as out of all models it has the best mean prediction accuracy of 90% and a near-vanishing variance. Since the selected input features for this model were only briefly touched, a more thorough optimization for the input features could possibly yield an even more accurate model.

1 Introduction

A study of 2000 movies has recently shown that white men tend to dominate movie roles [1]. Attempting to confirm or deny this finding, multiple machine learning models will be trained using a relatively small data set of Hollywood movies, with the goal to predict the gender of the main actor in Hollywood screenplays. An additional focus will be put on analyzing gender bias for the lead- and co-lead actors. The provided data set has a wide variety of Hollywood movies, ranging over multiple decades and genres. Each entry consists of a total of 14 attributes. The 13 input parameters are the number of actors per gender, gross profit, the total number of words spoken, the number of words spoken per gender, the number of words spoken by lead, the difference in the number of words by lead and co-lead, lead- and co-lead age, and the mean age of male and female characters. The last data entry is the gender of the lead actor, which is the output that should be predicted.

2 Data Analysis

In order to get a better understanding of the gender bias within the data we can ask ourselves *which sex dominates speaking roles in Hollywood movies*. By summing and comparing the number of male- and female actors for all movies in the data set we can clearly see that there is male dominance in speaking roles, as is shown in figure 1. While this is a nice static observation, it makes more sense to ask the question in a dynamic context, by studying how this dominance has changed over time. To do this we start by calculating the percentage of female actors in each film year by year. Then, using NumPy's polyfit function, a straight line is fitted to the data. The resulting line shows a positive trend, indicating that the percentage of female actors per movie is increasing, but is still not close to 50 %. Next, we can examine economic differences between male- and female-dominated movies, where the most straightforward would be to analyze *if male-dominated movies generate more gross than female-dominated movies.* By sorting out all movies by the gender with the most words spoken and calculating the mean gross we can see that indeed male-dominated movies generate more gross, which is seen in the lower plot in figure 1. In order to get a bigger look at the data set and locate outliers, a histogram and pairwise scatter plot of the normalized features are made. These can also indicate obvious correlations between the features. The plot is found in the appendix in figure A.3, which does confirm the existence of some outliers. Since all features are numerical, there is no need for 'dummy' variables. Most features do not follow a normal distribution, which makes it difficult to exclude the outliers without transforming the data. Another way to analyze the

data would be to compute the covariance matrix, by calculating the covariance between all features $\text{cov}(X, Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y))$ [2], however, since the majority of the features does not follow a normal distribution, this would again be of limited use.

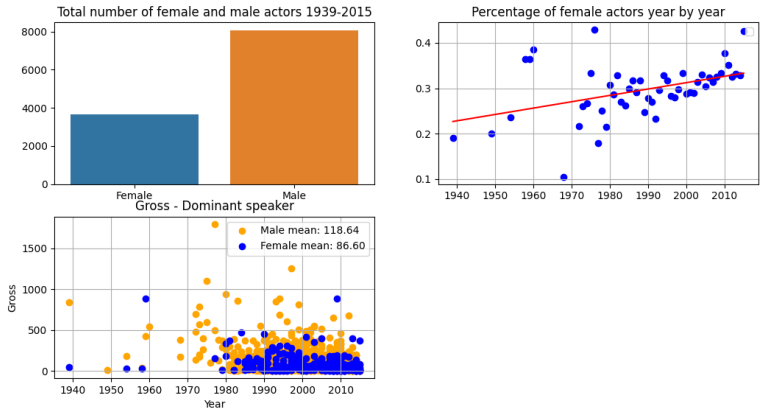

Figure 1: Various plots analyzing the gender bias in Hollywood movies.

3 Implementation of methods

There is an abundance of machine learning methods used for classification problems. This analysis, however, restricts itself to 5 different methods, these being logistic regression, discriminant analysis, k-nearest neighbour model, tree-based methods, and deep learning. The analysis of most of these methods relies heavily on the concept of cross-validation. This is a statistical technique used to test and train these models, by splitting the data set into training- and validation sets, whence the final error is obtained upon averaging the previously obtained errors.

3.1 Logistic regression

Logistic regression is a classification model, while the name itself could suggest something else. Here we present a short description of the fundamentals behind the model, and how to implement and tune it for this specific problem. Assuming the reader has a basic understanding of linear regression we can start from there and build our logistic regression model: $z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p$. To fit our model to a prediction of probability $\text{p}(y = 1|x)$ we use the logistic function also known as the Sigmoid function, which is defined as $f(\text{z}) = \frac{e^z}{1+e^z} \in [0 \quad 1]$. Given the probability of one class in the case of a classification problem with 2 classes, we have the probability for the second class $\text{p}(y = -1|x)$= $1 - f(\text{z})$. By choosing 1 and -1 as the labels we can simplify our expressions such that we are left with $f(\text{z}) = \frac{e^z}{1+e^z} = 1 - f(\text{z}) = \frac{e^{\theta^T x}}{1+e^{\theta^T x}}$. Next, by using the training data, the goal is to find an optimal set of parameters $\hat{\theta}$ for the model. Using the maximum likelihood approach we find $\hat{\theta} = \arg\max \text{p}(\mathbf{y} \mid \mathbf{X};\theta) = \arg\max \sum_{i=1}^{n} \ln \text{p}(\text{y}_i|\mathbf{x}_i;\theta)$, which can be turned into a minimization problem by using the negative log-likelihood as the cost function. Since the labels are chosen in a clever way, we end up with the cost function: $J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y_i\theta^T x_i})$. Then, the optimal parameters for the logistic regression model is found by solving the minimization problem

$$\hat{\theta} = \arg\min \frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y_i\theta^T x_i}) \tag{1}$$

However, this modification to the regression model is not "perfect", as there exists no closed-form solution. To find the parameters $\hat{\theta}$, the minimization problem has to be solved numerically [2]. In the scikit-learn library, different numerical solvers for the logistic regression model can be chosen, all of which have their own pros and cons. This hyperparameter is one that can be tuned in logistic

regression. Others are the type of regularization and its strength.

Regularization is a method to decrease the flexibility, or in other words, to avoid overfitting the training data. To achieve this, a penalty term is added to the loss function. L1 regularization penalizes the parameter weights in proportion to the sum of the absolute value of the weights. This penalty can drive the weight for an irrelevant feature parameter to 0. An other option is L2 regularization, which penalizes the parameter weights in proportion to the sum of squares of the weights. The strength of the penalty is regulated with the coefficient: "Regularization strength" or C in the scikit-learn library.

Tuning the model further can be done by optimizing the features that are used. Since the features in the data set are all related to the problem and there is no simple way to exclude any of them One option is therefore to test all possible combinations of features by "brute forcing".

We start tuning the model by testing different solvers using the default settings for the other hyperparameters. First, a model is trained and tested using 10-fold cross-validation, after which the solver and set of features is picked out. Since some of the solvers have convergence problems with non-normalized data, the standardScaler and MinMaxScaler from the scikit-learn library are used to scale the input data. Even though all features are not normally distributed, it proved empirically better to use standardScaler rather than MinMaxScaler. The test resulted in the following solver: 'lbfgs' and set of features: 'Number words female', 'Number of words lead', 'Difference in words lead and co-lead', 'Number of male actors', 'Number of female actors', 'Number words male', 'Mean Age Male', 'Age Lead', 'Age Co-Lead'.

Next, the remaining optimal set hyperparameters to the logistic regression model, these being the type of regularization and its strength, are found by again testing all possible combinations and seeing which combinations perform best in a 10-fold cross-validation. A useful package from scikit-learn library is GridSearchCV, which allows us to set up lists of regularization methods and strengths and test all of them, after which it returns the best combination. The input for the grid search is Solver: 'lbfgs', penalty: ['l2'], C: np.logspace(-3,3,7), which returned the results below.

- **Model:** LogisticRegression(solver='lbfgs', penalty='l2', C=1.0)
- **Mean accuracy:** 87.5%

Comparing the result to a naive classifier that only predicts a male lead, which has an accuracy of around 75%, we can see an improvement of about 12%.

3.2 Discriminant analysis: LDA, QDA

The Gaussian Mixture Model (GMM) is a generative model which attempts to model $\mathrm{p}(\mathbf{x}, y)$, where $\mathbf{x}$ is assumed to be numerical- and y a categorical variable. Here, we will look at two GMMs: Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA). These methods are considered when working with fully labeled data. Unlike the *discriminative* models, the GMM describes the joint distribution of both inputs and outputs. $\mathrm{p}(\mathbf{x}, y)$ states that, given input parameters $\mathbf{x}$, one finds a categorical output y with a certain probability. The GMM makes use of the following factorization of the joint probability density function: $\mathrm{p}(\mathbf{x}, y) = \mathrm{p}(\mathbf{x}|y)\mathrm{p}(y)$. Since y is categorical, the categorical distribution can be written as $\mathrm{p}(y = m) = \pi_m$, where the number of classifications ranges from 1 to m. Additionally, it is assumed that all probabilities follow a Gaussian distribution $\mathrm{p}(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mu_y, \Sigma_y)$, where μ_y is the mean vector of the parameter and Σ_y the covariance matrix. The GMM model can be trained using a supervised and semi-supervised approach. The goal of GMM is to determine the parameters $\theta = \{\mu_m, \Sigma_m, \pi_m\}_{m=1}^{M}$, which can be achieved using the log-likelihood function, $\hat{\theta} = \arg\max_{\mathrm{m}} \ln p(\mathcal{T}|\theta)$, where $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^{n}$ is the training data. This yields the following solutions: $\hat{\pi} = \frac{n_m}{n}$,$\hat{\mu}_m = \frac{1}{n_m}\sum_i \mathbf{x}_i$, and $\hat{\Sigma}_m = \frac{1}{n_m}\sum_i (\mathbf{x}_i - \hat{\mu}_m)(\mathbf{x}_i - \hat{\mu}_m)^T$, where n_m is the total number of data point in class m, and n the total number of data points [2].

The next step is to use the trained model to make predictions y_*, i.e., we want to find $p(y|\mathbf{x})$. Using Bayes' rule we get

$$\mathrm{p}(y|\mathbf{x}) = \frac{\mathrm{p}(\mathbf{x}, y)}{\mathrm{p}(\mathbf{x})} = \frac{\mathrm{p}(\mathbf{x}, y)}{\sum_{j=1}^{M} \mathrm{p}(\mathbf{x}, y = j)} \tag{2}$$

The final category is selected by using the largest probability, given the input $\mathbf{x}_*$

$$\hat{y} = \underset{\text{m}}{\text{argmax}} \ln p(y = m|\mathbf{x}_*), \tag{3}$$

which is known as the decision boundary. The equation above can be extended to the Quadratic Discriminant Analysis (QDA) method by taking the logarithm. Putting this all together, we get the final equation

$$\hat{y}_* = \underset{\text{m}}{\text{argmax}}\{\ln \hat{\pi}_m + ln\mathcal{N}(\mathbf{X}_*|\hat{\mu}_m, \hat{\Sigma}_m)\} \tag{4}$$

Taking the logarithm of the Gaussian distribution then gives rise to a quadratic decision boundary.

3.2.1 Training and evaluating LDA and QDA models

In order to complete this task, the scikit-learn library for supervised machine learning was used. This is a tool that makes it easy to implement the training and optimization of LDA and QDA methods. This library has functions for LDA and QDA, which creates the models and optimizes hyperparameters depending on the data that is worked with. The main hyperparameter for the LDA methods is the solver. There are three different solvers: "svd" (singular value decomposition), "lsqr" (least squares solution), and "eigen" (eigenvalue decomposition). When choosing lsqr or eigen, there is an additional parameter that can be tuned, which is the shrinkage parameter. The hyperparameter that was tuned for QDA was the regularization parameter, which ranges from 0 to 1. This parameter tries to combat overfitting and collinearity. Here it was found that the standard svd solver gives the highest accuracy for the LDA method, while a regularization paramater of 0.50 was found for the QDA method.

The next step is feature selection. This way optimized by training the models while withholding one feature from the dataset. Essentially, 13 different data sets were constructed, which in turn created 13 different models. This was tested using the svd solver for the LDA method and with regularization parameter of 0 and 0.50 for the QDA method. Using the LDA method, it was found that the highest misclassification accuracy was found when omitting the data from "Mean Age Female". The highest misclassification accuracy found with the LDA method was 13.5%. For the QDA it was found that the highest misclassification accuracy was obtained when using a regularization parameter of 0.5 and discarding the data from "Year". The highest misclassification accuracy obtained was 10.1%.

3.3 k-nearest neighbour

From a conceptual point of view, the k-nearest neighbour (k-NN) algorithm is arguably the most straightforward one. The core idea of the algorithm is to predict similar outputs for those inputs that are in close proximity to each other in parameter space. The k-NN algorithm belongs to the class of distance-based machine learning methods, and, despite its simplicity, can be applied to both regression and classification problems. In more detail, suppose we have a set of training data $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ and an input $\mathbf{x}_\star$ whose output we want to predict. Then, one starts by finding the k closest data points $\{\mathbf{x}_i\}_{i=1}^k$ to $\mathbf{x}_\star$, which gives rise to the name k-NN, by computing the distance $||\mathbf{x}_\star - \mathbf{x}_i||$ between them. Note that the metric need not necessarily be induced from the ℓ^2 norm. In fact, every norm induces a metric, which can be used to compute the distance between the input and data points, whose physical interpretation heavily depends on the type of metric that is used. Norms that are frequently used are the ℓ^1-, ℓ^2- and the more general ℓ^p norm, whose metrics are more commonly known as the Manhattan-, Euclidean-, and Minkowski distance, respectively. Moreover, the inclusion of different norms to compute the distances extends the regular k-NN algorithm to the kernel k-NN algorithm, where the similarity of data points can be specified according to the used kernel [3]. Next, once the k-NNs are found, a prediction is made depending on the type of problem (regression/classification). For classification with y classes, the prediction then takes the form of $y_\star = \max_y \sum_{(\mathbf{x}_i, y_i) \in \mathcal{T}} \omega_i \mathbb{1}_{\{y=y_i\}}$, where $\omega_i \in [0, 1]$ are so-called weights and $\mathbb{1}$ denotes the indicator function. For unweighted classification, all weights are equal to 1, but in the case where one would like to further decrease the importance of the data points furthest away from $\mathbf{x}_\star$, their respective weights can be scaled down.

The k-NN algorithm for classification is quite rigid, as it only leaves room for the hyperparameter k to be varied, as well as deciding whether or not to classify using weights. While the latter is

merely done by setting a flag to yes or no, the former is a bit more flexible and needs to be properly investigated. Furthermore, it is important to note that the k-NN model behaves differently for each k depending on which features are used in the input. This investigation concerns therefore both the input features and k, simultaneously. While there exist roughly 8200 possible combinations of input features, here, without loss of generality, we restrict ourselves only to combinations of 10, 11, 12, and 13 input features, where a minimum 10 features is set to avoid underfitting the model.

Pipelines from the Scikit-learn python library are used to determine the best combination of input features and k, by doing a 10-fold cross-validation for all of the aforementioned combinations of features, while simultaneously varying k to include 1 to 50 nearest neighbours. Here, every input is normalized so that scaling does not play a role, which is a known issue for the k-NN algorithm. The results for various norms and the inclusion/exclusion of weighted classification are presented in table 1. For visual clarity and brevity, the exact input features are omitted. Three of the models reach the

Model	Accuracy	k
Unweighted ℓ^2	0.864	14
Weighted ℓ^2	0.883	12
Unweighted ℓ^1	0.883	8
Weighted ℓ^1	0.883	11
Unweighted ℓ^3	0.874	8

Table 1: Cross-validation accuracy, the optimal value of k, and optimal number input features for different models.

same maximum accuracy. A bigger data set could have allowed for tiny differences in performance, which allows a single optimal model to be found. However, this is not an option, so the weighted ℓ^2 model has been selected as the best model due to its larger optimal value of k, as this allows for more flexibility than the other models. For this model, the accuracy per k is plotted in figure A.1.

Evident from the figure is that the number of nearest neighbours reaches its optimum very quickly. Moreover, the accuracy quickly drops due to overfitting, as is indicated by the long "tail" for large values of k. Finally, the optimal k-NN model for the given data uses the input features 'Number words female', 'Total words', 'Difference in words lead and co-lead', 'Number of male actors', 'Year', 'Number of female actors', 'Number words male', 'Mean Age Male', 'Age Lead', 'Age Co-Lead', together with $k = 12$.

3.4 Tree-based methods: classification trees, random forests, bagging

The basic idea behind tree methods is to split the input space into regions. The splitting is done according to rules, which essentially turns it into a binary tree. At each internal node, a rule $x_j < s$ with input parameter x_j and threshold s decides which branch will be chosen for the given input x. The leaf nodes then decide which class is chosen. Each leaf node corresponds to one region. The deeper the classification tree, the finer the partition [2]. For this project, three methods will be explained and tested. Classification trees are the most basic form. Bagging is an algorithm to improve the variance of the classification tree, and Random Forest uses a random subset of multiple trees to even further decrease the variance of the classification tree.

Classification Tree: To find the best partition, which means minimizing the training error, a greedy recursive binary splitting is used. It splits regions recursively in two half-spaces until a stopping criterion is met. The class prediction in each region can be expressed by $\mathrm{p}(y = m|x_\star) \approx \sum_{l=1}^{L} \hat{\pi}_{lm} \mathbb{I}\{x_\star \in R_l\}$, with the proportion of data points in region l of class m described by $\hat{\pi}_{lm} = \frac{1}{n_l} \sum_{i:x_i \in R_l} \mathbb{I}\{y_i = m\}$. At each internal node, the input variable and threshold with a minimal error are chosen. Common error measurements are Log Loss, Entropy, and Gini index [2].

Bagging: Bootstrap Aggregation or Bagging tries to lower the variance without raising the bias. To get a low bias, a tree has to grow deep, which can result in overfitting, yielding a high variance. Multiple slightly different classification trees are trained with different training sets in order to avoid this. Bootstrapping is a technique to create multiple datasets of size n from one separate dataset of size n. To obtain multiple data sets from one training set, one samples n times from the training set

and repeat this B times to get B data sets. After that, one can learn B models whereafter averaging over them results in a low variance.

Random Forest: Since the individual models used in Bagging are correlated, the variance reduction is limited. The Random Forest approach tries to fix this issue by using only a random subset of inputs as splitting variables at each internal node. This results in more different individual trees since dominant input variables are not picked all the time. Computationally this is also better, as it allows for calculations to be done in parallel, on top of using smaller subsets of variables.

Implementation The scikit-learn library provides implementations to all the methods, as well as a gridsearch implementation. The feature selection was done per the 'brute force' method, for each of the three methods, individually. All possible combinations of input features were used to fit the model and the accuracy was tested using 10-fold cross-validation. To tune the hyperparameter of each method, the gridsearch is performed with each method and different parameters. The resulting accuracy is used to determine the best parameters. The following final models are calculated: **Classification Tree** with input features ['Number words female', 'Number of words lead', 'Difference in words lead and co-lead', 'Number of male actors', 'Number of female actors', 'Number words male', 'Age Lead'] and parameter (max_depth=6, min_samples_split=3, min_samples_leaf=5, max_features='log2', splitter='best', citerion='gini'), **Random Forest** with ['Number words female', 'Number of words lead', 'Difference in words lead and co-lead', 'Number of male actors', 'Number of female actors', 'Number words male', 'Age Co-Lead'] and (n_estimators=5, max_depth=9, min_samples_split=5, min_samples_leaf=2, max_features=None, criterion='entropy') and **Bagging** with ['Number words female', 'Number of words lead', 'Difference in words lead and co-lead', 'Number of male actors', 'Number of female actors', 'Number words male', 'Mean Age Female']. Since Bagging performed better without hyperparameter tuning, no gridsearch was done, and instead default parameters were used. The final accuracies are for the basic tree is 0.837, while Random Forest achieves an accuracy of 0.877 and Bagging reaches 0.861.

3.5 Deep learning

Deep Learning (DL) can be considered a subset of Machine Learning. DL deals with hierarchical machine learning models and can describe more complex relations between input and output. A Deep Neural Net (DNN) consists of three regions: the input layer, the hidden layer, and the output layer. In this case, the input layer and output layer are the same as with the statistical machine learning methods. It is in the hidden layers where DL differentiates itself from standard machine learning. The hidden layer can be understood as many independent layers of linear regression models with non-linear activation functions.

There are many types of neural networks, here a fully connected neural network (FCN) is used. The general structure of a FCN looks as follows:

$$y_{jk}(x) = \sum_{i=1}^{N} \sigma(W_{jk} x_i + b_j). \tag{5}$$

Here the final output y_{jk} is given by the matrix multiplication of N hidden layers. Each layer consists of the weight matrix W and the bias b, and the number of nodes in each layer is represented by x_i. The σ denotes the activation function of each layer. It is standard to use the rectified linear activation function (ReLU) for each hidden layer, and a separate activation function for the final layer, depending on how the output looks (i.e. Sigmoid activation function in the case of a binary output). As can be seen in the equation above, the number of layers and nodes in each layer gives rise to a large sequence of matrix multiplication in order for us to find the output. Thus, computation time and training of the DNN heavily depend on the size of your network [2].

Once the network parameters are initialized (weights and biases), DNN can start to optimize its parameters, which is referred to as training. There is a multitude of ways in which a DNN can optimize its parameters. This is done by using a loss function $L(\mathbf{x}_i, \mathbf{y}_i, \theta)$ and a cost function $J(\theta)$. Additionally, one has to choose an optimizer, which is the algorithm for the gradient descent of the parameters. This often comes in the form of standard packages that are available. They use a variety of methods to descend toward a global minimum for the parameters. This process is repeated until one terminates the training when a criterion is fulfilled.

The network was tested using multiple layers with multiple different numbers of nodes in each layer. The best parameter settings were found with three hidden layers, with a number of nodes of 128, 64, and 32, for each layer, respectively, using the ReLU activation function. A dropout layer showed to have a negative effect and this option was discarded. The output layer consists of a single neuron with a Sigmoid activation function. For the optimizer, the "Adam" optimizer was used, as, in the industry, this optimizer is considered to be the best-performing optimizer. The optimal learning rate was found to be 0.001. The network was trained with 100 epochs with a batch size of one. The final accuracy with this model was found to be $87.9 \pm 3.4\%$.

3.6 Model selection

In order to determine the best model, a 10-fold cross-validation is done for each of the optimized models, as well as the naive classifier. For the logistic regression and $k-$NN models, the inputs are scaled as they were trained with scaled data. The results of the cross-validation are summarized in a box plot in figure A.2. The performance of each of the models is measured using the misclassification accuracy, where a lower indicates a better performance. On top of a low misclassification accuracy, a good model also should be unbiased and have a low variance. It turns out that the QDA model performs best, as it reaches the lowest misclassification accuracy, whilst also having an almost vanishing variance. Next, logistic regression and random forest share a similar performance, as they have a slightly higher misclassification accuracy than QDA, but maintain a low variance. As expected, random forest performs better than the other tree methods, as it improves them. The Bagging model has a moderate variance, which is not expected, indicating that something possibly went wrong in the implementation. The $k-$NN model has a low variance, however, its misclassification error is quite poor. Lastly, the LDA and DL methods perform similarly but are not quite as good as the QDA model.

In light of the analysis above, we have decided to put the QDA model 'in production', as its performance is unmatched by all other models.

4 Conclusion

In this work, we train different machine learning models using a data set of 2000 Hollywood movies in order to accurately predict the sex of the lead role of a movie, given a list of features corresponding to each movie. Here we test and cross-validate five different machine learning models: logistic regression, linear and quadratic discriminant analysis, k-nearest neighbour, deep learning, and tree-based methods: classification trees, random forests, and bagging.

The data provided was heavily biased towards male actors, both in the number of lead actors and the total number of female and male speaking roles. This imbalance was also tested when comparing the percentage of female actors over time. It turns out that, even though a male bias persists, it has gradually improved over the years, getting closer to a more even and fairer split. There was also an economic component included in the provided data. When looking at the economic differences between male- and female-dominated movies, it was found that male-dominated movies generate more gross than female-dominated movies. All in all, the data provided was heavily biased towards male leads and that had to be taken into account when constructing the machine learning models.

It was found that the generative QDA model gives the best performance after a 10-fold cross-validation, as out of all models it has the best mean prediction accuracy of 90% and a near-vanishing variance. Surprisingly, the model performed best when discarding the 'year' feature. Even though we found a strong correlation between features and the percentage of female actors over the years. When looking at the hyperparameters of the QDA model, we found they have the highest accuracy with a regularization parameter of 0.50. Further improvements could be made by brute forcing all combinations of input features, and by creating new features from old ones.

A Appendix

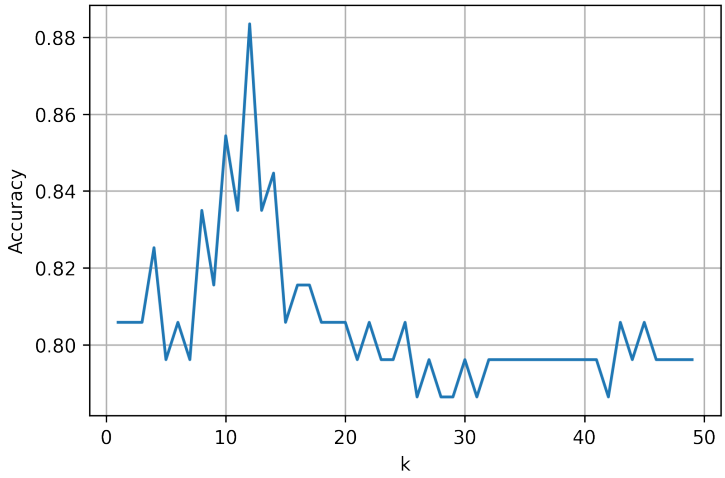

Figure A.1: Accuracy of the k-NN model, which uses the ℓ^2 norm to do a weighted classification, for the number of nearest neighbours k.

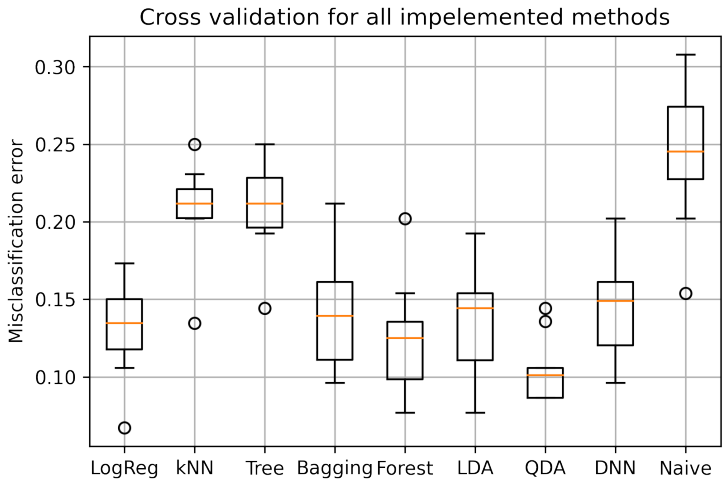

Figure A.2: Box plot of performance of all optimized models.

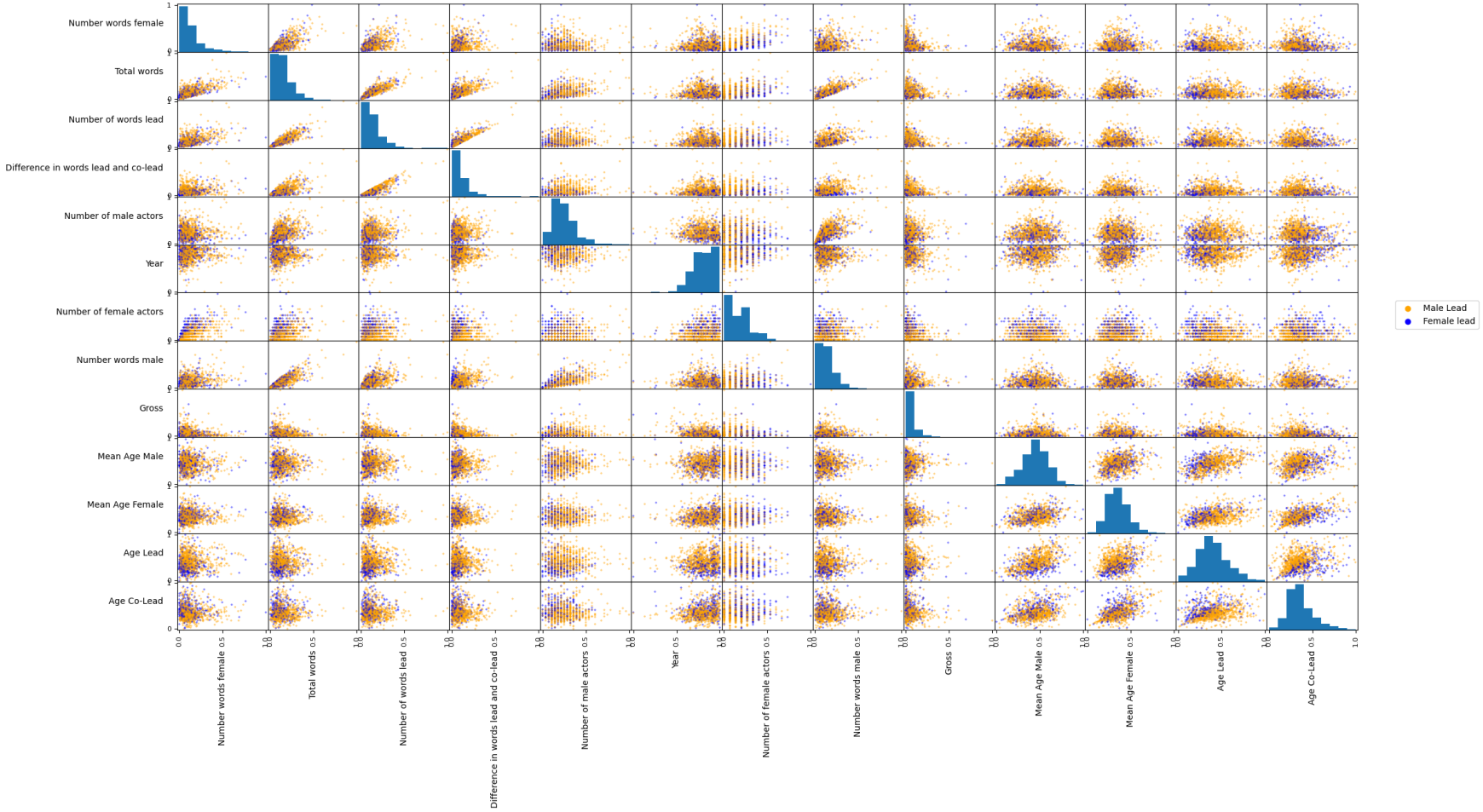

Figure A.3: Histogram and scatter plot.

Supplementary code for the reproduction of results. A lot of the code is inspired by or taken from the exercise sheets of the Statistical Machine Learning course [4]. These, in turn, use a lot of functions from external python libraries [5–18].

References

[1] The Pudding, "MS Windows NT kernel description." `https://pudding.cool/2017/03/film-dialogue/`.

[2] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[3] K. Yu, L. Ji, and X. Zhang, "Kernel nearest-neighbor algorithm," *Neural Processing Letters*, vol. 15, no. 2, pp. 147–156, 2002.

[4] S. M. L. team, "Statistical machine learning exercise sheets.," *Available through course page on Studium*, 2022.

[5] Contributors to Wikimedia projects, "Covariance - Wikipedia," Nov. 2022. [Online; accessed 7. Dec. 2022].

[6] Scikit-learn, "sklearn.ensemble.BaggingClassifier," Dec. 2022. [Online; accessed 22. Dec. 2022].

[7] Scikit-learn, "sklearn.model_selection.cross_validate," Dec. 2022. [Online; accessed 22. Dec. 2022].

[8] Scikit-learn, "sklearn.tree.DecisionTreeClassifier," Dec. 2022. [Online; accessed 22. Dec. 2022].

[9] Scikit-learn, "Scikit-learn: GridSearchCV," Nov. 2022. [Online; accessed 7. Dec. 2022].

[10] Scikit-learn, "sklearn.model_selection.KFold," Dec. 2022. [Online; accessed 22. Dec. 2022].

[11] Scikit-learn, "Scikit-learn: LogisticRegression," Nov. 2022. [Online; accessed 7. Dec. 2022].

[12] Matplotlib, "matplotlib.pyplot.boxplot — Matplotlib 3.6.2 documentation," Nov. 2022. [Online; accessed 22. Dec. 2022].

[13] Scikit-learn, "Scikit-learn: MinMaxScaler," Nov. 2022. [Online; accessed 7. Dec. 2022].

[14] Numpy, "Numpy: Polyfit," Nov. 2022. [Online; accessed 7. Dec. 2022].

[15] Scikit-learn, "sklearn.ensemble.RandomForestClassifier," Dec. 2022. [Online; accessed 22. Dec. 2022].

[16] Google, "Google Machine Learning Education: Regularization," Nov. 2022. [Online; accessed 7. Dec. 2022].

[17] Pandas, "Pandas: Scatter matrix," Nov. 2022. [Online; accessed 7. Dec. 2022].

[18] Scikit-learn, "Scikit-learn: StandardScaler," Nov. 2022. [Online; accessed 7. Dec. 2022].