Do (wo)men talk too much in films?

Anonymous Author(s)
Affiliation
Address
email

1 Introduction

In a movie, there is almost always a main character that the rest of the film is in some way centered around. In some films the main character is not very nice, but most of the time, the audience is expected to sympathize with this character or even look up to, and be inspired by them. This is especially true in children's movies. Since it is often easier to be inspired by someone you can relate to, it is important to make sure that the distribution of main characters in movies at least approximates the distribution of the audience. If the main character is always a man, that makes it more difficult than necessary for girls and women to find someone to be inspired by and vice versa.

This paper investigates how the gender of the lead actor can be predicted using metrics such as the year when the movie was made and the age of said actor. Being able to make such predictions could give an insight into how the movie industry works with respect to the gender of the lead actor. This could provide clues about what areas to investigate further to understand why those connections exist and ultimately, make sure that everyone has a chance to be inspired by someone they can relate to.

2 Methods

We have chosen to focus on approaches using logistic regression, k-NN, LDA and QDA to classify the lead actor's gender.

In order to make the methods as comparable as possible, we have used a common set of transformations of the input variables for all tested methods.

To compare between families of models and between which tuning is better we chose to focus on two measures: accuracy (on average, of how often model makes a correct prediction) and AUC (area under ROC curve).

Several methods have a hyper-parameter λ that needs to be tuned. In order to find a value of λ that performs well on the data, cross-validation is used to find the optimal value in a finite set $\Lambda = \{\lambda_1, \ldots, \lambda_k\}$. Cross-validation works by splitting the data into n equally sized partitions and training the data separately on the n choices of $n-1$ partitions and testing on the partition that was left out. The test error E_{new} is estimated by the mean misclassification rate across the partitions. This procedure is repeated for each $\lambda_j \in \Lambda$ and the value resulting in the lowest estimated test error is chosen.

Since cross-validation is used to estimate the hyper-parameter λ, it cannot be used to estimate the test error of the whole procedure. Instead, the data has to be split into a training set and a testing set. Cross-validation is done on the training set, and the test error is estimated by evaluating performance on the testing set. This can yield significantly different estimates of the test error as only one split into training and testing data is considered. To get a better estimate of the actual test error, a bootstrap procedure is performed.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

The full dataset is an iid sample from some unknown distribution so the estimated test error $\hat{E}_{new}$ is a random variable. By drawing B independent splits into training and testing data with subsequent fitting and cross-validation, a bootstrap sample of $\hat{E}_{new}$ is obtained which can be used to obtain a better estimate of E_{new}. This is very computationally intensive if B is large.

2.1 Input transformations

In the given dataset, there are columns for the total number of words spoken as well as the number of words spoken by the lead, the co-lead etc. This could present a problem if we compare a movie where the lead says 10 out of 100 total words and another movie where the lead says 100 out of 1000 words. Most models would think that the lead speaks more in the second movie and miss the fact that the *proportion* of words spoken by the lead is the same. Hence, we have transformed several input variables to express a proportion instead of absolute numbers. We also believe it might be important to have a dummy variable indicating if the lead or the co-lead is oldest. All transformations are given in Table 1.

Table 1: Transformations of input variables.

Original column	New column	Transformation
Number of words lead	Proportion of words lead	$\frac{\text{Number of words lead}}{\text{Total words}}$
N/A	Proportion of words co-lead	$\frac{\text{Number of words lead - Difference in words lead and co-lead}}{\text{Total words}}$
Difference in words lead and co-lead	Ratio words co-lead lead	$\frac{\text{Proportion of words co-lead}}{\text{Proportion of words lead}}$
Number words female	Proportion of words female	$\frac{\text{Number words female}}{\text{Total words - Number of words lead}}$
Number of female actors	Proportion of female actors	$\frac{\text{Number of female actors}}{\text{Number of female actors + Number of male actos}}$
Number of male actors	Number of actors	Number of male actors + Number of female actors
N/A	Older lead	$\begin{cases} 1, \text{Age lead} > \text{Age Co-Lead} \\ 0, \text{else} \end{cases}$

Note that when determining 'Proportion of words female', this should only measure the words spoken by non-lead female actors so we have to subtract the lead's contribution to the total number of words. The column 'Number of male actors' was dropped since all necessary information in this column is contained in 'Proportion of female actors' together with 'Number of actors'. In order to improve regularization and k-NN, all remaining numerical input variables where centered and scaled by their standard deviation. This means that columns with proportions have values in the unit interval $[0, 1]$ and the other numerical variables have values that are of roughly the same magnitude. This scaling was not done for QDA as it is not necessary for that method.

2.2 Logistic Regression

Logistic regression is a *general linear model* (GLM), i.e. the relationship between the data $X \in \mathcal{X} \subseteq \mathbb{R}^p$ and the outcome Y is on the form

$$E(Y|X = x) = g^{-1}(x \cdot \beta) \tag{1}$$

where $\beta \in \mathbb{R}^p$ and g is the link function. In the case of logistic regression, $Y|(X = x) \sim Ber(p(x))$ and the canonical link function is the logit link $g(x) = \log\left(\frac{x}{1-x}\right)$ with $g^{-1}(x) = \frac{\exp(x)}{1+\exp(x)}$. Since $Y|(X = x) \sim Ber(p(x))$, we get $E(Y|X = x) = p(x) = g^{-1}(x \cdot \beta)$. In other words, $P(Y = 1|X = x) = g^{-1}(x \cdot \beta)$, which we can use to predict Y given data x.

To do the regression, we find $\hat{\beta} \in \arg\min_\beta \sum_{i=1}^n (y_i - \hat{y}(x_i; \beta))^2$ where $\hat{y}(x; \beta) = g^{-1}(x \cdot \beta)$. This is the MLE estimator of β This minimizes the mean squared error (MSE) loss function.

A potential problem with this approach is that there are no restrictions on the components of β and that can lead to overfitting, especially if n is not much larger than p. To address that issue, one can introduce regularization.

In general, regularization is done by adding a penalizing term to the loss function that restricts β in some way. If $L(\beta; x_i, y_i)$ is the loss function before regularization, we instead consider the new loss function $L(\beta; x_i, y_i) + \lambda R(\beta)$ and find $\hat{\beta}_{reg} \in \arg\min_\beta (L(\beta; x_i, y_i) + \lambda R(\beta))$. R is some penalizing function and λ is a hyper-parameter that can be tuned. The two most common forms of regularization is LASSO and Ridge regression.

LASSO regression uses L_1-regularization, meaning that $R_{LASSO}(\beta) = ||\beta||_1 = \sum_{i=1}^p |\beta_i|$ while Ridge regression uses L_2-regularization, $R_{Ridge}(\beta) = ||\beta||_2^2 = \sum_{i=1}^p \beta_i^2$.

2.3 k-Nearest Neighbors

The k-nearest neighbors (k-NN) method is based on the simple principle of finding the k closest neighboring points with respect to the input data $X \in \mathcal{X} \subseteq \mathbb{R}^p$. In the case of classification the outcome Y is then determined by a majority vote among the k nearest data points. The method is based on the idea that if a test data point is close to some training data point then the prediction should be that they have the same outcome Y.

The algorithm for k-NN can be implemented in a simple manner with a brute force algorithm for measuring the distance from the test data point $\boldsymbol{x_\star}$ to each training data point $\boldsymbol{x_i}$, where $i = 1, ..., n$ using some distance function $d(x, y)$. It is standard to use the Minkowski distance for a certain order p, depending on the problem, which is given by

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}, \text{ where } \boldsymbol{x} = (x_1, ..., x_p), \boldsymbol{y} = (y_1, ..., y_p) \in \mathbb{R}^p. \tag{2}$$

Note that $p = 1$ gives the Manhattan distance, and $p = 2$ gives the Euclidean distance. The brute force algorithm for k-NN is given by [Lindholm et al., 2021]

---

1. Calculate the distance $d(\boldsymbol{x_i}, \boldsymbol{x_\star})$ for each $i = 1, ..., n$
2. Set $\mathcal{N}_\star = \{\boldsymbol{x_i} : \textit{Where } \boldsymbol{x_i} \textit{ is one of the } k \textit{ nearest points}\}$
3. Return $\hat{y}(\boldsymbol{x_\star}) = \text{MajorityVote}\{y_j : j \in \mathcal{N}_\star\}$

A problem with the brute force algorithm is that for each point we need to calculate the distance to every other point, which is computationally demanding for larger datasets.

There are however more computationally efficient algorithms to find the k-NN compared to the brute force search such as ball-tree and k-d tree which are not explained in detail here. All three of these algorithms were tested and no significant difference in the results where noted thus the choice of algorithm was set to "auto" which chooses the best suited algorithm for a given problem, further described in the Scikit-learn documentation [skl, a].

For our problem we consider the Minkowski distance and let p and k be hyper-parameters which are to be tuned. This is done in an analogous manner to the case of finding the hyper-parameter λ in the regularization problem with logistic regression previously considered.

An alternative approach is to use weighted k-NN. The idea is to let the distance of the training data point to the test data point influence the strength of the vote. We compared *uniform weights* (standard k-NN, where all weights equal 1) and *distance weights* where the weights are given by

$$\frac{1}{d(\boldsymbol{x_\star}, \boldsymbol{x_i})}, \tag{3}$$

for each of the k-nearest neighbors. This reinforces the idea that proximity of test data points to training data points ought be a good predictor [skl, b].

2.4 LDA and QDA

For classification we construct a discriminative classifier from a generative model based on Bayes' theorem for the classes $m = 1, 2, .., M$ by

$$p(y = m \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid y = m)p(y = m)}{\sum_{i=1}^{M} p(\boldsymbol{x} \mid y = i)p(y = i)} \tag{4}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_n)$. We estimate the *uninformative prior probability* as $\hat{p}(y = m) = \frac{n_m}{n}$ where $n_m = \sum_{i=1}^{n} \mathbb{1}\{y_i = m\}$ and assume that $p(\boldsymbol{x} \mid y = m)$ is a normal density with expected value μ_m and covariance matrix Σ_m. The assumption that distinguishes LDA and QDA is that for LDA we assume that $\Sigma_1 = \Sigma_2 = ... = \Sigma_M$ but for QDA we make no such assumption, that is, we allow for the covariance matrices to differ. A consequence is that LDA is a special case of QDA, hence QDA is a model of higher complexity.

The estimates for the normal distribution parameters for each class is given by

$$\hat{\mu}_m = \frac{1}{n_m} \sum_{i:y_i=m} \boldsymbol{x_i} \text{ and } \hat{\Sigma}_m = \frac{1}{n_m - 1} \sum_{i:y_i=m} (\boldsymbol{x}_i - \hat{\mu}_m)(\boldsymbol{x}_i - \hat{\mu}_m)^T \tag{5}$$

derived from maximum likelihood estimation and adjusting $\hat{\Sigma}_m$ to make it unbiased. The *pooled covariance estimate* (weighted average of the covariance matrix estimates within each class) is given by

$$\hat{\Sigma} = \frac{\sum_{m=1}^{M}(n_m - 1)\hat{\Sigma}_m}{\sum_{m=1}^{M}(n_m - 1)} = \frac{1}{n - M} \sum_{m=1}^{M} \sum_{i:y_i=m} (\boldsymbol{x}_i - \hat{\mu}_m)(\boldsymbol{x}_i - \hat{\mu}_m)^T. \tag{6}$$

With these estimators we may express the discriminant analysis classifier as

$$\hat{p}(y = m \mid \boldsymbol{x}) = \frac{n_m \mathcal{N}(\boldsymbol{x} \mid \hat{\mu}_m, \hat{\Sigma})}{\sum_{i=1}^{M} n_i \mathcal{N}(\boldsymbol{x} \mid \hat{\mu}_m, \hat{\Sigma})} \tag{7}$$

where $\mathcal{N}(\boldsymbol{x} \mid \mu, \Sigma) = \frac{1}{(2\pi)^{M/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x} - \mu)^T \Sigma^{-1} (\boldsymbol{x} - \mu)\right]$ is the density for the normal distribution with mean μ and covariance matrix Σ.

3 Results

When comparing different models, it is important to have a baseline, or a null model to compare against. In this case, an obvious null model is the constant model that always predicts the same outcome regardless of input. The best null model is the one with highest accuracy, i.e. the constant model that predicts the most frequently occurring outcome. The model that always predicts a male lead has an accuracy of 0.756 and is thus chosen as the baseline.

3.1 Logistic Regression

For all logistic regression models fitted, the set of regularization parameters, Λ, consisted of 10 logarithmically spaced values between 10^{-4} and 10^{4}. This was the default value in the methods from scikit learn and having more densely packed values did not affect the model performance in any appreciable way. The number of folds used in cross-validation was also 10, no improvement was observed by increasing this value.

In Tables 2 and 3, the accuracy and AUC are estimated using the mean of 100 bootstrap samples in the case of LASSO regression and 400 in the case of Ridge regression. The reason for having different sample sizes is that computing the LASSO regression is much more computationally demanding.

We see that the regularization does not affect the model performance much. LASSO and Ridge regularization perform almost identically and yield at best around 0.003 extra accuracy but considering that the different splits of the data yielded estimated test errors in a range from 0.8 to 0.98, we cannot reject that regularization does not matter in this case.

Table 2: Accuracy and AUC for logistic regression models. 70% training data.

Input	Regularization	Accuracy	AUC
Before transformations	None	0.870	0.878
	LASSO	0.871	0.880
	Ridge	0.871	0.880
After transformations	None	0.893	0.920
	LASSO	0.895	0.921
	Ridge	0.894	0.921

Table 3: Accuracy and AUC for logistic regression models. 90% training data.

Input	Regularization	Accuracy	AUC
Before transformations	None	0.876	0.878
	LASSO	0.875	0.883
	Ridge	0.871	0.880
After transformations	None	0.895	0.924
	LASSO	0.897	0.924
	Ridge	0.898	0.923

3.2 k-Nearest Neighbors

When hyper-tuning k-NN the set of p-values and k-values were given by $\{1, 1.25, 1.5, ..., 4\}$ and $\{1, 2, 3, ..., 25\}$ respectively. The number of folds used in cross-validation was again set to 10. We found that $p = 2$ (Euclidean distance) and $k = 4$ performed best for our transformed data set. Using these parameters the k-NN algorithm was tested and performance was measured with the mean of 100 bootstraps samples, the size of the sample was chosen with regards to k-NN being computationally demanding. The results are summarized in tables 4 and 5.

Table 4: Accuracy and AUC using k-NN with 70% training data.

Input	Weighted k-NN	Accuracy	AUC
Before transformations	Uniform	0.745	0.675
	Distance	0.780	0.688
After transformations	Uniform	0.864	0.883
	Distance	0.872	0.888

Table 5: Accuracy and AUC using k-NN with 90% training data.

Input	Weighted k-NN	Accuracy	AUC
Before transformations	Uniform	0.750	0.678
	Distance	0.783	0.693
After transformations	Uniform	0.875	0.891
	Distance	0.882	0.901

It is obvious that k-NN is drastically improved by transforming the data, before transformations the model performance was even outperformed by, or just slightly better than the best null model. Weighted k-NN with the distance weight seemed to perform better than the uniform weight both before and after the transformation, the impact seems to be an increase of 0.007-0.008 in accuracy after transformation and an increase of almost 0.03 in accuracy before transformations.

3.3 LDA and QDA

The given dataset was bootstrapped 400 times and both DA models were tested before and after input transformations and dropping 'Year' and 'Gross'. From the Tables 6 and 7

Table 6: Accuracy and AUC for discriminant analysis models using bootstrap. 70% training data.

Input	Model	Accuracy	AUC
Before transformations	LDA	0.856	0.870
	QDA	0.818	0.849
After transformations	LDA	0.900	0.917
	QDA	0.945	0.984

Table 7: Accuracy and AUC for discriminant analysis models using bootstrap. 90% training data.

Input	Model	Accuracy	AUC
Before transformations	LDA	0.866	0.877
	QDA	0.840	0.869
After transformations	LDA	0.900	0.918
	QDA	0.947	0.984

we conclude that QDA seems to be more apt for this problem for the final inputs. It is unclear which method performs better for the original inputs. For QDA on the original inputs, the variance in accuracy is high which can be explained by the fact that the original inputs are close to being colinear resulting in inaccurate matrix inversion of Σ. For example, the standard deviation of accuracy and AUC of the QDA classifier, on the original inputs, with 400 bootstrapped datasets and 90% training data were 0.076 and 0.081 respectively, compared to 0.021 and 0.019 with the final inputs, ceteris paribus. Fortunately, the final inputs are not colinear.

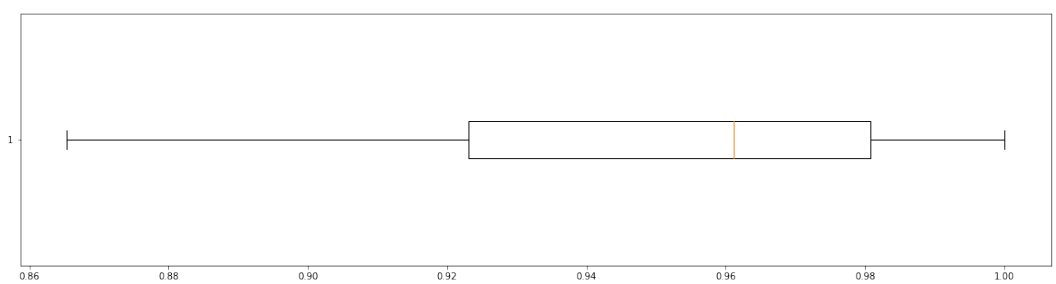

Figure 1: Accuracy estimation using cross validation with 20 folds.

Cross validation was carried out to estimate the accuracy using 20 folds resulting in an estimated accuracy of 0.949 using 70% training data. As can be seen in Figure 1, there is a noticeable variance in accuracy for the different training sets suggesting that there might be outliers in the data that the model has problems accounting for. Increasing the number of folds also increases the minimum accuracy that can be found in the corresponding box-plot, as to be expected. Also using cross validation to compare the effects of input transformations we see a 0.090 increase in accuracy using the final inputs compared to the original inputs. Adding the variables 'Years' and 'Gross' back into the inputs we see a small decrease in accuracy hence they are left out.

4 Conclusions

Based on the results we draw the conclusion that k-NN is the worst performing model of the ones considered. Nevertheless, it is worth noting the drastic effect that data transformations has on the k-NN model. Before transformations the model performed at the level of the baseline model, after the transformations we saw a accuracy of 0.882 and a AUC of 0.901 which is found in Table 5. The logistic regression model slightly outperformed k-NN. Looking at the best performing setup we see that logistic regression had an accuracy of 0.898 and AUC 0.923 which is seen in Table 3. The LDA model had an accuracy of 0.903 which is only a 0.005 increase from the logistic regression model. Taking variance into account, we draw the conclusion that we can not say with any confidence which model performs better. The QDA model performs best among all considered models by a margin with an accuracy of 0.945 and 0.984 AUC according to the bootstrap test and accuracy 0.949 according to the cross validation.

We answer the following questions assuming that the given data set is representative of Hollywood movies in general.

1. **Do men or women dominate speaking roles in Hollywood?** In the given data we see that in 75.6% of movies there is a male in the lead. Further, the mean of 'Proportion of words female' is $34,6\%$ indicating that Hollywood movies consist of almost 65% male speaking.
2. **Has gender balance in speaking roles changed over time (i.e. years)?** None of the models considered suffered performance loss from removing the input variable 'Year'. However, the correlation in the given data set between 'Year' and all of 'Numbers of words female', 'Number of female actors' and 'Mean Age female' is positive and not trivially small (0.032, 0.134 and 0.170 respectively). We conclude that this indicates a slight change in the gender balance over time but that the model does not necessarily use this relationship to make predictions.
3. **Do films in which men do more speaking make a lot more money than films in which women speak more?** None of the models considered suffered performance loss from removing the input variable 'Gross' and the QDA model even improved a bit indicating that 'Gross' may be a bad input variable for this model. This might be because over the years, the change in 'Gross' for men and women may have changed at different rates and because of inflation; this makes 'Gross' a potentially bad input variable for future predictions.

All of the models fitted in this project focus on prediction instead of inference, hence it is not possible to use them to answer the above questions directly.

5 Feature Importance

In order to determine which of the features 'Year', 'Gross' or 'Number of female actors' is most important, we fitted models excluding one or more of the features. Using a logistic regression model with LASSO (L1) regularization, we found that the model performance in terms of prediction accuracy was completely unaffected by removing either or both of the features 'Year' and 'Gross'. On the other hand, any model that excluded the variable 'Proportion of words female' (which contains all information about how much male and female actors speak), had its performance reduced drastically. As seen in Table 3, the model including all features had an accuracy of 0.9. This dropped to 0.8 when removing the 'Proportion of words female'. Comparing this to the null accuracy of 0.756, we conclude that the 'Proportion of words female' is a very important feature in the model. Interestingly, this 'Proportion of words female' seems very important for all models, but especially so for QDA; this suggests that QDA may depend more heavily on this input for predictions. The same results hold for both QDA and k-NN as well, the models are completely unaffected by removing either 'Year' or 'Gross' (or both), while suffering 0.08 accuracy loss for k-NN and 0.12 loss for QDA. Further, not only accuracy is affected but AUC is affected in the same way.

Appendix A: Code

Transformations of input variables and various common functions.

```
import numpy as np
import pandas as pd
import sklearn.preprocessing as skl_pre

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

cols_to_norm = [
       'Total words',
       'Year',
       'Gross',
       'Mean Age Male',
       'Mean Age Female',
       'Age Lead',
       'Age Co-Lead',
       'Number of actors'
]

def pre_process(raw_data, cols_to_norm):
       data = raw_data.copy()

       data['Lead'] = pd.get_dummies(data['Lead'])
       data['Number of words co-lead'] = data['Number of words lead'] -
           data['Difference in words lead and co-lead']
       data['Proportion of words lead'] = data['Number of words lead']/data
           ['Total words']
       data['Proportion of words co-lead'] = data['Number of words co-lead'
           ]/data['Total words']
       data['Ratio words co-lead lead'] = data['Number of words co-lead']/
           data['Number of words lead']
       data['Proportion of words female'] = data['Number words female']/(
           data['Total words'] - data['Number of words lead'])
       data['Number of actors'] = data['Number of male actors'] + data['
           Number of female actors']
       data['Proportion of female actors'] = data['Number of female actors'
           ]/data['Number of actors']
       data['Older lead'] = data['Age Lead'] < data['Age Co-Lead']
       data['Older lead'] = pd.get_dummies(data['Older lead'])

       scaler = skl_pre.StandardScaler()
       data[cols_to_norm] = scaler.fit_transform(data[cols_to_norm])

       return data

data = pre_process(rawData, cols_to_norm)

def fit_and_test(classifier, train, test, features, target, suppress_output
     = False):
       classifier.fit(train[features], train[target])
       if not suppress_output:
              skl_met.plot_roc_curve(classifier, test[features], test[
                  target])
              print('accuracy: ' + str(classifier.score(test[features],
                  test[target])))
              print(' auc: ' + str(skl_met.roc_auc_score(test[target],
                  classifier.predict_proba(test[features])[:,1])) + '\n')
```

```
                print(skl_met.classification_report(test[target], classifier.
                    predict(test[features])))
        return classifier

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

featureSet1 = [
        'Year',
        'Gross',
        'Number of actors',
        'Proportion of female actors',
        'Mean Age Male',
        'Mean Age Female',
        'Age Lead',
        'Age Co-Lead',
        'Total words',
        'Proportion of words lead',
        'Proportion of words co-lead',
        'Ratio words co-lead lead',
        'Proportion of words female',
        'Older lead'
]


print('Null accuracy: ' + str(max([np.mean(data[target]), 1 - np.mean(data[
    target])])))
```

Logistic Regression

```
trainRatio = config['Train Ratio'][0]
seed = config['Random Seed'][0]
train, test = skl_ms.train_test_split(data, train_size=trainRatio)


features = featureSet1.copy()
#features.remove('Proportion of words female')
#features.remove('Year')
#features.remove('Gross')
#features = ['Proportion of words lead']
target = 'Lead'

# No regularization

B = 100
accuracies = []
aucs = []
```

```
for i in range(B):
        train, test = skl_ms.train_test_split(data, train_size=trainRatio)
        logReg = fit_and_test(skl_lm.LogisticRegression(penalty='none',
            solver='newton-cg'), train, test, features, target,
            suppress_output=True)
        accuracies.append(logReg.score(test[features], test[target]))
        aucs.append(skl_met.roc_auc_score(test[target], logReg.predict_proba
            (test[features])[:,1]))

# LASSO

B = 100
accuracies = []
aucs = []
for i in range(B):
        train, test = skl_ms.train_test_split(data, train_size=trainRatio)
        logRegLasso = fit_and_test(skl_lm.LogisticRegressionCV(Cs=10, cv=10,
             penalty='l1', solver='liblinear', n_jobs=10), train, test,
            features, target, suppress_output=True)
        accuracies.append(logRegLasso.score(test[features], test[target]))
        aucs.append(skl_met.roc_auc_score(test[target], logRegLasso.
            predict_proba(test[features])[:,1]))

# Ridge

B = 400
accuracies = []
aucs = []
for i in range(B):
        train, test = skl_ms.train_test_split(data, train_size=trainRatio)
        logRegLasso = fit_and_test(skl_lm.LogisticRegressionCV(Cs=10, cv=10,
             penalty='l2', solver='liblinear', n_jobs=10), train, test,
            features, target, suppress_output=True)
        accuracies.append(logRegLasso.score(test[features], test[target]))
        aucs.append(skl_met.roc_auc_score(test[target], logRegLasso.
            predict_proba(test[features])[:,1]))
```

k-NN

```
import sklearn.neighbors as skl_nb
X = data[featureSet1]
y = 'Lead'

## Using gridsearch to find the optimal parameter
knn2 = skl_nb.KNeighborsClassifier()
param_grid = {'n_neighbors':np.arange(1,25),
               'p':np.linspace(1,4,13)}

knn_gscv = GridSearchCV(knn2, param_grid, cv =10)

knn_gscv.fit(X,y)


## k-NN Distance weight(swap 'distance' to 'uniform' for standard distance)
     after transformations(swap featureSet1 to rawFeatures for before
    transformations)

B = 100
accuracies = []
aucs = []
```

```
knn = skl_nb.KNeighborsClassifier(n_neighbors = 4, p=2, weights='distance')
features = featureSet1
for i in range(B):
    train, test = train_test_split(data, test_size=0.3)
    KNN_gscv = fit_and_test(knn,train, test, features, 'Lead' )
    accuracies.append(KNN_gscv.score(test[features], test['Lead']))
    aucs.append(skl_met.roc_auc_score(test['Lead'], KNN_gscv.predict_proba(
        test[features])[:,1]))

## k-NN Uniform weight(swap 'distance' to 'uniform' for standard distance)
    after transformations(swap featureSet1 to rawFeatures for before
    transformations)


B = 100
accuracies = []
aucs = []
knn = skl_nb.KNeighborsClassifier(n_neighbors = 4, p=2, weights='uniform')
features = featureSet1
for i in range(B):
    train, test = train_test_split(data, test_size=0.3)
    KNN_gscv = fit_and_test(knn,train, test, features, 'Lead' )
    accuracies.append(KNN_gscv.score(test[features], test['Lead']))
    aucs.append(skl_met.roc_auc_score(test['Lead'], KNN_gscv.predict_proba(
        test[features])[:,1]))



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
from IPython.core.pylabtools import figsize
import sklearn.model_selection as skl_ms
import sklearn.metrics as skl_met
import itertools
import math

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

## Accuracy estimation using cross validation
##
#X = dataset[dataset.columns[1:14]]
X = pre_process(dataset).drop(columns=['Lead'])
X=X[{
    'Year',
    'Gross',
    'Number of actors',
    'Proportion of female actors',
    'Mean Age Male',
    'Mean Age Female',
    'Age Lead',
    'Age Co-Lead',
    'Total words',
```

```
    'Proportion of words lead',
    'Proportion of words co-lead',
    'Proportion of words female',
    'Older lead'}]

Y = dataset['Lead']
# Split randomized data into training and validation (30% validation)
X_train, X_val, Y_train, Y_val = skl_ms.train_test_split(X,Y, test_size
    =0.1)
# List of models
models = []
#models.append(skl_lm.LogisticRegression(solver='liblinear'))
#models.append(skl_da.LinearDiscriminantAnalysis())
models.append(skl_da.QuadraticDiscriminantAnalysis())

n_fold=20
accuracy = np.zeros((n_fold, len(models)))
model_accuracy = np.zeros(len(models))
cv = skl_ms.KFold(n_splits=n_fold, shuffle=True)

for i, (train_index, val_index) in enumerate(cv.split(X)):
    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
    Y_train, Y_val = Y.iloc[train_index], Y.iloc[val_index]

    for m in range(np.shape(models)[0]):
        model = models[m]
        model.fit(X_train, Y_train)
        prediction = model.predict(X_val)
        accuracy[i,m] = np.mean(prediction == Y_val)
        model_accuracy[m] += accuracy[i,m]
for m in range(np.shape(models)[0]):
    print('Model accuracy for model ' + str(models[m]) + ' is ')
    print(model_accuracy[m]/n_fold)

plot = plt.figure(figsize=(20,5))
ax = plot.add_subplot(111)
bp = ax.boxplot(accuracy, vert=False)
plot.show()

## Bootstrapping
##
classifier = skl_da.LinearDiscriminantAnalysis()
features = [
    'Number of actors',
    'Proportion of female actors',
    'Mean Age Male',
    'Mean Age Female',
    'Age Lead',
    'Age Co-Lead',
    'Total words',
    'Proportion of words lead',
    'Proportion of words co-lead',
    'Proportion of words female',
    'Older lead',
]
target = 'Lead'
X = pre_process(dataset)
X=X[{
    'Number of actors',
```

```
    'Proportion of female actors',
    'Mean Age Male',
    'Mean Age Female',
    'Age Lead',
    'Age Co-Lead',
    'Total words',
    'Proportion of words lead',
    'Proportion of words co-lead',
    'Proportion of words female',
    'Older lead', 'Lead'}]
B = 400 # number of training sets to sample
accuracies = []
aucs = []
for i in range(B):
    train, test = skl_ms.train_test_split(X, test_size=0.3)
    QDA = fit_and_test(classifier, train, test, features, target,
        suppress_output=True)
    accuracies.append(QDA.score(test[features], test[target]))
    aucs.append(skl_met.roc_auc_score(test[target], QDA.predict_proba(test[
        features])[:,1]))

print('mean accuracy: ' + str(np.mean(accuracies)))
print(' mean auc: ' + str(np.mean(aucs)))

print(' std auc: ' + str(np.std(accuracies)))
print(' std auc: ' + str(np.std(aucs)))
```

References

Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, and Thomas B. Schön. *Supervised Machine Learning*. Pre-pub:Cambridge university Press, draft version: january 12, 2021 edition, 2021.

Scikit-learn, documentation - nearest neigbors. `https://scikit-learn.org/stable/modules/neighbors.html`, a. Accessed: 2021-02-23.

Scikit-learn, documentation - sklearn.neighbors.kneighborsclassifier. `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn-neighbors-kneighborsclassifier`, b. Accessed: 2021-02-23.