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 (who plays the main character), can be predicted using metrics such as the year when the movie was made and the age of the lead 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 ROC/AUC.

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 since 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. For that reason we have transformed several input variables to express a proportion instead of absolute numbers. We also believe it might be important to have a

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

dummy variable indicating if the lead or the co-lead is oldest. All transformations are given in Table 2.1.

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	$\begin{cases} 1, \text{Age lead} > \text{Age Co-Lead} \\ 0, \text{else} \end{cases}$

Table 1: Transformations of input variables.

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.

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 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, $R_{Ridge}(\beta) = ||\beta||_2^2 = \sum_{i=1}^{p} \beta_i^2$.

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, this method cannot be used to estimate the test error of the whole procedure. Instead, the dataset has to be split into a training set and a testing set with a specified fraction of the total data in each set. The whole procedure above is done on the training set and the test error is estimated by evaluating the performance of the model on the testing set. However, this can yield significantly different estimates of the test error since only one split into training and testing data is considered. To get a better estimate of the actual testing error, a bootstrap procedure is performed.

Since the full dataset is an iid sample from some unknown distribution, the estimated test error $\hat{E}_{new}$ is a random variable. By repeating the whole procedure B times (i.e. B independent splits into training and testing data and subsequent fitting and cross-validation), a bootstrap sample of $\hat{E}_{new}$ is obtained which can be used to estimate the distribution (or at least properties thereof) of $\hat{E}_{new}$. This is very computationally intensive but gives a much clearer view of the variability of the test error compared to just computing it for one split.

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 closely related to 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 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 normal to use the Minkowski distance of order p which is given by

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

Where $p = 1$ is the Manhattan distance, and when $p = 2$ we have the Euclidean distance of course any distance function could be used. The brute force algorithm for k-NN is given by

---

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 all the training data has to be stored and each distance has to be calculated which can be rather computer intensive. There are however algorithms as the ball-tree and k-d tree which speeds up these calculations. The general principle is still the same, exactly how these algorithms are performed are thus left out of the report.

In this case we let the Minkowski distance be our distance function and we let p and k be hyper-parameters which are to be tuned. This is done in an analogous manner as in the case of finding the hyper-parameter λ in the Logistic regression case above.

Weighted k-NN is an alternative approach to the normal k-NN where the k nearest neigbors also are weighted based on how far or close from the test data point they actually are effecting the majority vote such that for example closer points have "stronger" vote. In our case we tested between *uniform* weights (Standard k-NN) and *distance* weights where the weight points equals

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

for each of the k-nearest neighbors, this results in giving closer neighbors a stronger influence. **??**

2.4 LDA and QDA

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

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

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 assumes 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

$$\hat{\mu}_m = \frac{1}{n_m} \sum_{i:y_i=m} \boldsymbol{x_i}, \tag{5}$$

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

The *pooled covariance estimate* (weighted average of the covariance matrix estimates within each class) is given by

$$\hat{\Sigma} = \frac{\sum_{m=1}^{M}(n_m - 1)\hat{\Sigma}_m}{\sum_{m=1}^{M}(n_m - 1)} = \frac{1}{n - M} \sum_{m=1}^{M} \sum_{i:y_i=m} (\boldsymbol{x}_i - \hat{\mu}_m)(\boldsymbol{x}_i - \hat{\mu}_m)^T. \tag{7}$$

Finally we may express the discriminant analysis classifier as

$$\hat{p}(y = m \mid \boldsymbol{x}) = \frac{n_m \mathcal{N}(\boldsymbol{x} \mid \hat{\mu}_m, \hat{\Sigma})}{\sum_{i=1}^{M} n_i \mathcal{N}(\boldsymbol{x} \mid \hat{\mu}_m, \hat{\Sigma})} \tag{8}$$

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

3 Results

3.1 Logistic Regression

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.

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.

The model performance was measured by accuracy (1 - misclassification rate) and AUC (area under ROC curve). 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.

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

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

We see that the regularization does not affect the model performance much. LASSO and Ridge regularization perform almost identically and yield at best around 0.3% 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.

3.2 k-Nearest Neighbors

When hyper-tuning the algorithm 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. It was found that $p = 2$ (Euclidean distance) and $k = 4$ performed best for our data set. Using these parameters the k-NN algorithm was tested and performance was measured with the mean of 100 bootstraps samples as before, the size of the sample was chosen with regards to k-NN being computationally demanding. The results are summarized in Table 4 and Table 5 below.

It is obvious that k-NN is drastically improved by transforming the data, before transformations the model performance was even outperformed 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 $0.7 - 0.8\%$ extra accuracy after transformation and almost 3% extra accuracy before transformations.

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 4: Accuracy and AUC for k-NN models. 70% 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

Table 5: Accuracy and AUC for k-NN models. 90% training data.

3.3 LDA and QDA

The given dataset was bootstrapped 400 times and both DA models were tested before and after input transformations. From the Tables 6 and 7 we can draw the conclusion that QDA

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

Input	DA 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

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

seems to be more apt for this problem after transformations. It is unclear which method performs better before the transformation. For QDA the variance in accuracy is high which can be explained by the fact that the original inputs are close to being colinear making Σ close to being singular which in turn results in an inaccurate matrix inversion. For example, the standard deviation of accuracy and AUC of the QDA classifier, before transformations, on 400 bootstrapped datasets with 90% training data were 0.076 and 0.081 respectively compared to 0.021 and 0.019 after transformations ceteris paribus.

Cross validation was carried out to estimate the accuracy using 150 folds resulting in an estimated accuracy of 0.948 using 70% training data. As can be seen in table 8 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 transformed inputs compared to the original inputs. Adding the variables ’Years’ and ’Gross’ back into the inputs we see a decrease in accuracy of 0.007 hence they are left out.

project/tex/QDAboxplot.png

Table 8: Accuracy estimation using cross validation with 150 folds.

4 Conclusions

It seems like k-NN was the worst performing of the models. Worth noting is the drastic effect that data transformations has on the k-NN model, performing at the same level as the baseline model before any transformations. The best results for the k-NN were a mean accuracy score of 0.882 and a mean AUC Score of 0.901 which is found in table 5.

Logistic regression slightly outperformed k-NN. Looking at the best performing setup we see that logistic regression had a mean accuracy score of 0.898 and a mean AUC score of 0.923 which is seen in table 3.

LDA had a mean accuracy score of 0.903 which is only a 0.5% increase from the logistic regression case, considering that both the results from logistic regression and LDA had some variance we cannot reject that LDA and logistic regression performance is similar.

QDA however had the best performance among all the models performing at best with a mean accuracy score of 0.942 and a mean AUC score of 0.977(!). Almost a 4% increase in accuracy compared to the second best method. One problem with the QDA model was however the outliers seen in the boxplot of 8. Similar outliers were found in the case of k-NN and logistic regression these could probably be explained by the occurrence of a "bad split" where for example many movies deviating from the average movie end up in the test portion resulting in a bad performance. However the model performed well enough overall such that QDA was chosen for production.

1. Do men or women dominate speaking roles in Hollywood?

   Based on the data it seems like males are in the lead. In 75.6% of the cases the lead is male which was found by looking at the baseline model. Also the mean for "Proportion of words female" was calculated to $34,6\%$ indicating that Hollywood movies consist of almost 65% male speaking.

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

   Since none of the models we used seem to be effected by removing the year variable we cannot say that this is the case.

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

   Since none of the models seem to be effected by removing the gross variable we cannot say that gross is a good indicator for determining whether there is more male speaking than female in a given movie. That is films with more male speaking cannot be said to make more money than a film with more female speaking.

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 90%. This dropped to 80% when removing the Proportion of words female. Comparing this to the null accuracy of 75.6%, we conclude that the Proportion of words female is a very important feature in the model.

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 8% accuracy loss for k-NN and 12% loss for QDA. Further, not only accuracy is affected but AUC is affected in the same way, showing that Year and Gross are more or less redundant features in the model, while Proportion of words female is incredibly important for predicting whether the lead is male or female.

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

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)

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'
]

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]))
```