Gender-Bias in the Movie-Industry: Classification of Movie-Lead Genders

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this study, a machine learning algorithm is trained on movie-data to classify the gender of the lead actor. The data-set includes number of words spoken by males/females and by the lead actor, gross profit, and year of production. The importance of three different features are studied to answer three main questions: Do men dominate the movie industry? Has this changed over time? Do male-lead movies make more money? The data included about 75 % male leads, and the choice of classification method was based on both overall accuracy, and the accuracy of identifying male/female leads. The method used was Gradient Boosting, with a cross-validated accuracy of around 87 % (70 % female, 94 % male). Words spoken by respective genders proved to be a critical feature, while gross profit and year of production made little impact. This shows of a critical gender inequality in the film-industry, which is not explainable by movie profitability nor is any significant improvement observed between 1970-2010.

1 Introduction

Many children, and adults, look up to their favourite movie characters. A gender imbalance in movie-leads likely has some effect on how gender inequalities propagate in society. Pinpointing sources of these inequalities in the movie-world could grant greater insight into what can be done to minimize them. Simply noting that something is a problem is the first step to any solution. Do men talk too much?

The article ”Film Dialogue” [1] looks at the amount male vs female actors speak to detect gender bias in the films. They found that male characters speak most, especially in children’s movies. Is this a possible source to the creation and development of gender structures in society? Has this changed over the past years? Do movies where men dominate make more money? These questions are here studied using various machine-learning algorithms with data-sets of parameters including words spoken, production year and gross profit, and labelled after the gender of the lead-actors’ gender.

This is an important topic to research, as it can give us insight to the gender inequalities in the film industry. Given this data, we attempt to classify the gender of movie-leads and study parameter importance. Notably, the data contains mainly data from men as movie-leads. As such, when developing a model, there will likely be a bias towards correctly classifying men over females. While historically it may be relevant to focus on correctly classifying men, as they were more frequently movie-leads, to device a model reinforcing this norm might not be desirable. Rather, we believe equality in accuracy has merit in itself. While fairness movie classification might not be as important as for example determining whether someone should be kept in prison. Still, fairness should be carefully considered in all machine learning projects, as the wider implications of its application can be hard to predict.

2 Methods

As mentioned in the introduction we have evaluated a few different methods for the binary classification of the gender of lead role. In the following section we explain the different methods and how they work.

(i) Logistic regression

Logistic regression [5] is a method in the family of parametric models. By using the logistic function

$$h(z) = \frac{e^z}{1 + e^z} \tag{1}$$

it is possible to define z as a linear regression model

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p = \theta^T \mathbf{x} \tag{2}$$

which squeezes the logistic function on the interval $[0, 1]$ (referred to as the *logit*). Since the model is predicting the outcome of the feature *male* or *female*, this implementation is a binary classification model. Instead of fitting to data, the sigmoid curve separates the data in the xy-plane with regards to a decision boundary. The model is trained over the data $\mathcal{T} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ by numerically solving

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ln \left(1 + e^{-y_i \theta^T \mathbf{x}_i} \right) \tag{3}$$

The parameter vector $\hat{\theta}$ is then applied in Function 1. When evaluating test data $\mathbf{x}_\star$, the logit outputs $\hat{y}(\mathbf{x}_\star) = 1$ if $h(\mathbf{x}_\star) > 0.5$ and $\hat{y}(\mathbf{x}_\star) = -1$ otherwise.

The *hyperparameter* is set to `C=1` [3] by default, but can be defined as an input for the `LogisticRegression` function to adjust for how the model should treat the weights. For large `C`, an obvious trade-off is overfitting since it applies large weights for the training data.

(ii) Discriminant analysis: LDA, QDA

Discriminant analysis uses the training observations to determine a boundary between response classes. The location of the boundary is determined by treating the observations of each class as samples from a multidimensional normal distribution.

Theoretically we can fit an n-dimensional normal distribution to the observations in each class. This involves calculating the mean vector and covariance matrix for each class. These determine the center and shape of the distribution.

A decision boundary is drawn where the density functions for the two distributions intersect. An implicit assumption for good performance is that the data is inherently separable; it will be hard to draw a suitable decision boundary for very mixed data.

For linear discriminant analysis (LDA) one assumes that every covariance is the same, and thus the boundary between the classes will be linear. LDA is a quick classification method, the calculations are few and simple. This function performs well when the covariance of the classes are actually the same and when the data points are not too mixed.

For quadratic discriminant analysis (QDA) one allows that different classes have different covariances. Here, the decision boundary can be quadratic, and is more flexible. QDA is still relatively quick even though it requires more memory and calculations to evaluate, store and invert the multiple covariance matrices. For QDA one can get bad estimations for the covariances if one does not have enough data, since the model will overfit to the training-data. If the data has no difference in covariance, this will perform worse than LDA. In cases with mid-sized datasets, as in this study, QDA will generally outperform LDA, due to its extra degree of adaptability.

(iii) Tree-based methods: classification trees, random forests, bagging

A classification tree taking in some f features and some training-data, partitions f-dimensional space into disjoint parts and in each part applies a simple model (e.g. a majority vote). The partitioning is often done in a way to minimize the misclassification rate in each step. The depth d of a classification tree is determined by the number of splits made.

One way of partitioning is using the so-called Gini-index as loss function. Qualitatively, it favours leaving large partitions of low quality, essentially assuming that it will be able to improve the quality of the big part in future splits. This is a commonly used method, and the one used here.

A deeper tree generally provides low bias but high variance. A deep tree runs the risk of seriously overfitting to training data. A method to reduce the variance of trees is to use a so-called ensemble method. Since tree-based methods can typically be computationally cheap to find, *bagging* (*b*ootstrapped *agg*regation) of the training data $\mathcal{T}$ can be used. Bagging is essentially drawing, with replacement, data from $\mathcal{T}$ into n new sets $\mathcal{T}_i$, possibly of the same size as $\mathcal{T}$. Now, each of these bagged datasets is trained using some simple model, e.g. a classification tree, and when a decision is to be made, the average output of all models is used. This way, bias is kept small, but the variance is reduced.

One limitation of bagging with decision trees is that the bootstrapped datasets $\mathcal{T}_i$ are correlated, so the decrease in variance is limited. One potential improvement to this is the random forest (first proposed by Leo Breima). The idea is essentially to perturb each tree in order to de-correlate them. This is done by only considering a random subset of the input data in each split. While this (slowly) increases the bias and increases the variance of each tree, the e-correlating effect is generally dominant, decreasing overall variance.

Overall, the random forest method is easy to run in parallel, the size of bagged datasets can be tuned depending on processing power, the random removal of selection points increases computation speed, and the method often works well without much tuning.

(iv) Boosting

A similar, yet to a large part opposite approach as *bagging* is *boosting*. The same as bagging, boosting is an ensemble model combining the efforts of multiple methods into a single model intended to obtain more accurate predictive behaviour.

The idea for boosting is to use a simple model with low variance, but which is unable to map any complex association between the input and output, therefore having large bias, and combine several instances of the simple model, thereby reducing the bias. The combination of these weak models then constitutes a single strong model, now with a lower bias, and also maintaining a low variance. The concept is applicable to, and able to improve, a large number of models within machine learning.

The combining of models is, unlike the use of bagging, done sequentially, using the result of the previous to establish the next. By re-weighting the data points depending on whether they were correctly predicted or not the next model provides a result accounting to a greater degree for the points missclassified by the previous. This process is repeated a predetermined number of times, each model building upon the previous ones. Depending on the error of each of the models they are given a coefficient determining their contribution to the one strong model.

Since boosting in itself is a very loose concept any choices of simple models, loss functions or determination of coefficients is left unspecified by the definition, although some cases appear more often. The usage of decision trees as the simple model is commonplace, and another aspect for comparison with bagging and random trees, and is the simple model chosen in this instance.

3 Implementations

For our implementation of the methods we used sklearn `LinearDiscriminantAnalysis` which implements LDA, `QuadraticDiscriminantAnalysis` which implements QDA, `LogisticRegression` which implements logistic regression, `RandomForestClassifier` which implements random forests, `GradientBoostingClassifier` for boosting and `kFold` which splits the training data for cross validation. We chose to use the built in sklearn implementations since it allows for simple comparison between models. Hyperparameters were initially tuned to optimize for accuracy by re-running the models many times and gradually tuning parameters, and later also tuned to optimize for approximately equal misclassification rates for both genders.

To evaluate our output we use cross validation and calculate the accuracy by calculating what percentage of the data was classified correctly of all our training data. The missclassification is the total number of male leads classified as females and female leads classified as male. These two represent false positives and false negatives.

3.1 Preprocessing

For some methods, such as logistic regression, preprocessing of training data is needed in order to perform well. There exists numerous ways of linearly transforming the training data. One method is passing the data through linear combinations of functions such as $\sin$, $\cos$ and $\ln$ but in this case the `StandardScaler` is used to reduce the magnitude of the dataset by letting each column sum to zero.

3.2 Rebalancing data

A method used to rebalance data was the function `SMOTE` (Synthetic Minority Over-sampling TEchnique) from the package `imblearn`.[4] As the name implies, it randomly re-samples the minority class until they are of equal size. This will introduce some bias, so should be used with some caution.

3.3 Cross validation

Cross validation is a method of evaluating and comparing how well a trained classification method performs, k-fold is the simplest and most used cross validation technique. It works by dividing the shuffled training data set into n segments and running the learning algorithm on all data except one segment and analyzing the performance (testing) on the last segment. This is then done for all combinations of choosing one from n, and thus all data will be tested on once but trained on $n-1$ times. Here, the `sklearn` built-in function `kFold` was used with 10 folds for our cross-validation.

3.4 Feature importance

To test feature-importance, a function to generate data-sets without all combinations of some select features is used. The various models are then trained and cross-validated on all combinations. To compare the models, all models were merged into one script to run sequentially.

4 Results

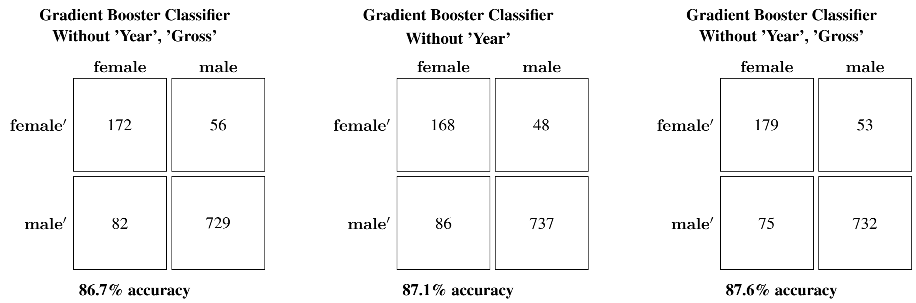

Figure 1: Confusion matrices from Gradient Booster Classifier. The predicted classes are denoted with an apostrophe.

The data in table 1 is the average of 10 times boosting for each method.

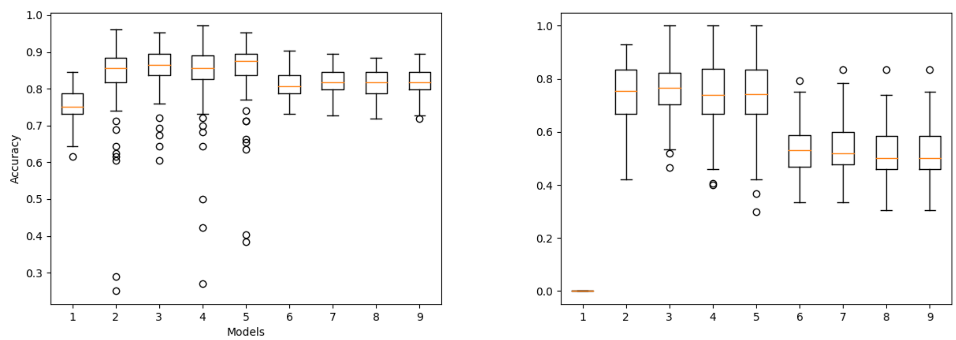

Figure 2: Accuracy and accuracy for females for 11 runs with cross validation (n=10) of an "all-male-guess" as well as 8 instances of QDA with some parameters dropped. The parameters dropped from left to right are: (2) None, (3) Year, (4) Gross, (5) Year and Gross, (6) Number words female and Number words male, (7) Year, Number words female and Number words male, (8) Gross, Number words female and Number words male, (9) Year, gross and number words female and number words male.

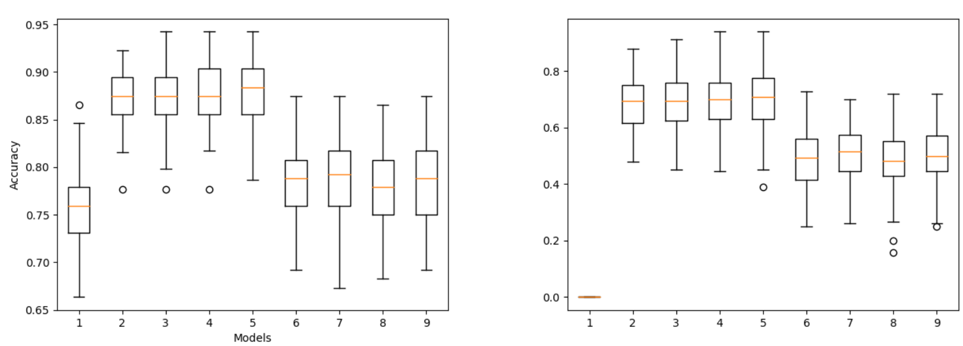

Figure 3: Accuracy and accuracy for females for 11 runs with cross validation (n=10) of an "all-male-guess" as well as 8 instances of GradientBoostingClassificatier with some parameters dropped. The parameters dropped from left to right are: (2) None, (3) Year, (4) Gross, (5) Year and Gross, (6) Number words female and Number words male, (7) Year, Number words female and Number words male, (8) Gross, Number words female and Number words male, (9) Year, gross and number words female and number words male.

Table 1: Performance data of every method when excluding ’Gross’ and ’Year’

	Accuracy [0,1]		
Method	Total	Female	Male
Log Reg	0.86910	0.61811	0.95032
QDA	0.85467	0.79528	0.87389
Tree	0.82676	0.68110	0.87389
Grad Boost	0.87680	0.70472	0.93758

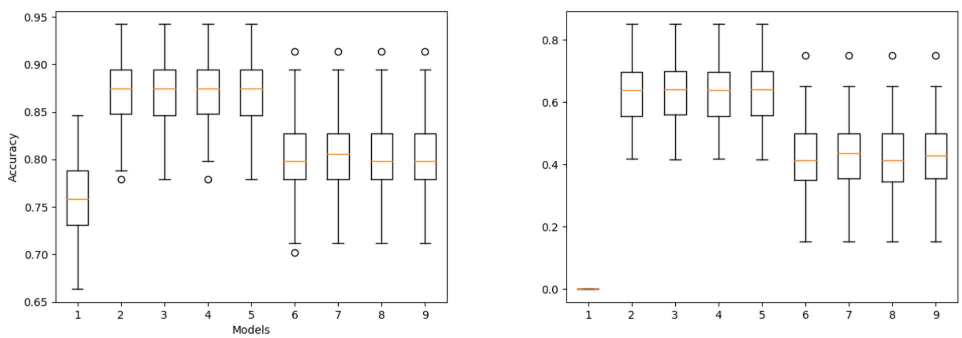

Figure 4: Accuracy and accuracy for females for 11 runs with cross validation (n=10) of an "all-male-guess" as well as 8 instances of LogisticRegression with some parameters dropped. The parameters dropped from left to right are: (2) None, (3) Year, (4) Gross, (5) Year and Gross, (6) Number words female and Number words male, (7) Year, Number words female and Number words male, (8) Gross, Number words female and Number words male, (9) Year, gross and number words female and number words male.

5 Discussion

For the purpose of this discussion, the data set provided as training data is assumed to be representative of Hollywood-film as a whole, and that assumptions made about this particular subset is applicable the movie industry as a whole. Whether this is true is part of a discussion not pertinent to this project.

The reality is that the majority of men have lead roles in Hollywood movies, and this leads to a bias for our model. In our given training data of 1039 movies 785 have male lead roles. When we choose a model, we have to balance the overall performance on the data where men dominate the amount of lead roles, and the accuracy balance between classifying male vs female lead roles correctly. If we want our model to perform best overall on our given training data, we can choose a model that performs really well on classifying males and rather bad on classifying female leads which gives us a quite good model based on the fact that our training data has a majority of male leads.

However, as we do not know the distribution in the final test set, we have to make a choice when selecting a final model. We want to make sure the model performs overall well, and does not act unreasonably unfair, and therefore we have chosen to use a model that performs better on both classifying male and female leads, even if this means a slight drop in overall accuracy on classifying specifically our training set.

5.1 Rebalancing data

During testing, re-balancing of data-sets using SMOTE proved inefficient in improving our model. Further tuning could likely have yielded more even performances for F/M classification, but the re-sampling proved to introduce significant bias for the random forest which significantly over-performed during testing, though likely just due to the heavy re-sampling.

5.2 Feature importance

For the different models it can be observed in figure 2, 3, and 4 respectively how well the model performs when trained on data including and excluding combinations of three features. We see that including the number of words spoken by female and male non-lead roles increases the performance of our model, and thus a strong association can be derived between the gender of the lead of a movie and by which genders all the non-lead words were spoken. For the other two features, both the feature 'Gross' and 'Year', the result closely resembles that of which when no features are excluded and is therefore seen to have a very minor effect on the prediction.

Along with changes in society over the past decades it could be expected that gender bias in the film industry would have come to see notable changes over the years. This is a correlation not found in the process of this project. The simple fact that the release 'Year' feature is of little to no significance when it comes to learning an accurately predicting the gender of the lead actor is largely equivalent to saying that the gender bias is equal for any given year and has therefore not changed over time.

The income of a film is to be considered a strong incentive for the making of a movie and evaluating the factors that may impact the films gross-product would be part of the decision making within the film industry. It has been noted that the feature 'Gross' is largely insignificant to the prediction of whether the lead actor is male or female and an assumption that female-lead films are associated with a lesser box-office is as such erroneous. It should however be noted that year and gross product are correlated but since the exclusion of both still yield the an unaffectedly good model they can still both be considered insignificant.

5.3 Conclusions and wider reflections

Selecting a "best" method has not proven trivial. On average, the various methods show similar performance: overall accuracy lying between $\sim$83-88%. However, when considering the variance of some methods (i.e. not averaging over cross-validation results), some models show outliers with rather bad performance. Specifically QDA displays this behaviour. This signifies a higher variance and as such a higher degree of susceptibility to unique test sets which to greater extent differ form the original training set. Another critical benchmark for choosing model is accuracy per gender. Most models performed significantly better on male-leads than on female ones; e.g. the chosen model, compares 70/94% F/M accuracy, compared to QDA 79/90%. This, combined with a high average accuracy, would be a strong argument for QDA, but the great variance displayed in figure 2 is deterring. Comparing to figure 3, a great reduction in outliers is noted.

To conclude, the study of feature importance shows that words spoken by respective genders is a critical feature for determining lead gender: Males dominate movies where they lead. Gross profit and year of production on the other hand made very little impact and sometimes removing them even improved performance slightly. This displays that not only is there a gender equality issue in the Hollywood film industry, but any notion that movies starring female lead actresses implies a lower gross profit is erroneous. Also incorrect is the idea that this is an issue already being resolved; while the data is not all too recent (most recent data 2010), no improvement appears detectable at all in the frequency of female leads.

References

[1] Hanah Anderson and Matt Daniels (April 2016) *Film Dialogue* url: `https://pudding.cool/2017/03/film-dialogue/`

[2] Ling Liu, M. Tamer Özsu (2009) *Encyclopedia of Database Systems* url: `https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_565`

[3] Scikit-learn developers (BSD License) (2021) *sklearn.base: Base classes and utility functions* url: `https://scikit-learn.org/stable/modules/classes.html#module-sklearn`

[4] The imbalanced-learn developers, *SMOTE* (2021), url: `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html`, retrieved 2021-12-06.

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

Appendix

Code for testing the various ML-algorithms. All code is also available on GitHub: `https://github.com/benjaminverbeek/Project-Statistical-ML`

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

import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn import tree
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,
↪ GradientBoostingClassifier
# For balancing trainingdata. "Synthetic Minority Over-sampling TEchnique"
from imblearn.over_sampling import SMOTE

def crossVal(model, X, y, print_accuracy=True, run_all_models=False, dropCols=[],
↪ random_state=1):
    print(f"model is {model}")

    # Split index for the folds
    kf = KFold(n_splits = 10, shuffle = True, random_state = random_state)
    testIndicies = []

    # Initiate cumulative sum variables
    all_predictions = []
    all_ys = []
    data = {'Female':[0, 0],
            'Male':[0, 0]}
    tot_crosstab = pd.DataFrame(data, index=['Female', 'Male'])

    # Iterate over all k-folds, fit model and sum to cumulative confusion matrix
    for train_index, test_index in kf.split(X):
        testIndicies.append(test_index)
        X_train, X_test = X.iloc[train_index,: ], X.iloc[test_index,: ]
        y_train, y_test = y[train_index], y[test_index]

        model.fit(X_train, y_train)

        predict_prob = model.predict_proba(X_test)

        prediction = np.empty(len(X_test), dtype=object)
        prediction = np.where(predict_prob[:,0]>0.5, 'Female', 'Male')

        conf_mat = pd.crosstab(prediction, y_test)

        tot_crosstab = tot_crosstab + conf_mat
```

```python

    print(tot_crosstab)
    # Statistics:
    acc = (tot_crosstab['Female'][0] + tot_crosstab['Male'][1]) / tot_crosstab.values.sum()
    nFem = tot_crosstab['Female'].values.sum()
    nMale = tot_crosstab['Male'].values.sum()
    accFem = tot_crosstab['Female'][0] / nFem
    accMale = tot_crosstab['Male'][1] / nMale
    percentMale = nMale / (nMale + nFem)

    if print_accuracy:
        print(f'Accuracy: {acc:.5f}')
        print(f'Accuracy Female / Male:\t {accFem:.5f} / {accMale:.5f} \t (testdata contains {percentMale*100:.5f} % males)')

    if run_all_models:
        results.append((acc, accFem, accMale, model, dropCols))


def modelDropParams(model, X, y, dropCols=[], run_all_models=False, random_state=1):
    """Function running model dropping some X-params. With cross-validation."""

    print(f"\nResults without {dropCols}")
    X = X.copy().drop(columns=dropCols)

    crossVal(model, X, y, run_all_models=run_all_models, dropCols=dropCols, random_state=random_state)

    #print(f'Accuracy tree: \t\t {np.mean(y_predict == y_test):.2f}')
    #allMale = y_test.copy().replace(["Female"],"Male") # make a copy with all Male.

    print("-----------------")

def allCombos(lst):
    """Takes in a list of lists and returns a list of all combinations of list elements."""
    combos = []
    for i in range(2**len(lst)):
        a=i
        params = []
        for j in range(len(lst)):
            if a%2 == 1:
                params += lst[j]
            a = a//2
        combos.append(params)
    return combos

def rescaleDataFrame(df):
    scaler = StandardScaler()
    scaled_input = scaler.fit_transform(df.values)
    scaled_df = pd.DataFrame(scaled_input, index=df.index, columns=df.columns)
    return scaled_df
```

```python
######

# Read the files into data frames
practiseTrain = pd.read_csv("train.csv")
practiceTest = pd.read_csv("test.csv")

# Split data into two frames, X and y
X = practiseTrain.copy().drop(columns=["Lead"])      # target
y = practiseTrain["Lead"]
#sm = SMOTE(random_state=42)
#X, y = sm.fit_resample(X, y)

# Rescale dataframe, can be commented to test if it gives better results or not
X = rescaleDataFrame(X)

# Dict with models
models = {
        'boosting': GradientBoostingClassifier(n_estimators=500, learning_rate=1.0, min_samples_split=0.5),
        'LDA': skl_da.LinearDiscriminantAnalysis(),
        'QDA': skl_da.QuadraticDiscriminantAnalysis(),
        'random-forest': RandomForestClassifier(max_depth=5, min_samples_leaf=1, class_weight="balanced"),
        'logreg': skl_lm.LogisticRegression(solver='lbfgs', C=12, random_state=0)
    }
# 'tree': tree.DecisionTreeClassifier(max_depth=4, min_samples_leaf=1)
# 'boosting': GradientBoostingClassifier(n_estimators=500, learning_rate=0.4, min_samples_split=0.5),

model = models['random-forest']
run_all_models = True

# Declare parameters to evaluate and extract all combos
testParams = [["Year"], ["Gross"], ["Number words female", "Number words male"]]
combos = allCombos(testParams)
print(f"Generated {len(combos)} combinations.")
print("Running ML-algo. for all combos.")

# TODO: possibly add output to excel for easier report-writing? Or all just take their model and write.
# OR save results to a dict and find max accuracy.
# Iterate over all combos


if run_all_models:
    results = []
    for model in models.values():
        print(f'----- RUNNING MODEL: {model} -----')
        for c in combos:
            modelDropParams(model, X, y, dropCols=c, run_all_models=True, random_state=random.randint(0,42))
    print('\n#### FINAL RESULTS ####')
    print(f'Top 5 by total accuracy: \n')
    print(*list(reversed(sorted(results)[-5:])), sep='\n')  # prints line-by-line
```

```
    print('###########')
    print(f'Worst 5 by total accuracy: \n')
    print(*sorted(results)[:5], sep='\n')
else:
    for c in combos:
        modelDropParams(model, X, y, dropCols=c)
```