Does Popularity Influence the Academy Awards?

Fojan Babaali, Maryam Honari Jahromi, Omar Elazhary, Ali Mashreghi
University of Victoria, Victoria, Canada

In this project we are going to predict whether a movie will win an Academy Award or not. Our main data is the Kaggels IMDB information on movies. However, we will add supplementary data about the history of Oscars' winners/nominees. We use the information after the movie was released since social media information -which is an important part of our features- only becomes stable a few months after release. We also intend to find the suitable algorithm to classify movies properly. Among the candidate algorithms are Naive bayes classifiers and linear regressors. We will also attempt to define a criteria by which movies are chosen.

Index Terms—**Oscars, Movies, Prediction, Classification, Machine Learning, Data Mining**

Highlights

1) In section *Features Used in Prediction*, we have explicitly mentioned the features that we use for prediction.
2) In the whole process of feature selection we only consider training data.
3) After the mid-term report we tried AdaBoost and Neural Networks on our data.
4) We applied a rather new technique based on modulo to generate test/train set.

I. Introduction

II. Related Work

III. Data Gathering and Description

As mentioned previously, our main dataset was a collection roughly 5000 movies from Kaggle created by *Mr. Chuan Sun* [12]. However, since we needed some features to be able to predict Oscar nominees and winners we started gathering historical data. To do this, we wrote a crawler in Java which went through the search results of the official Oscars' website (http://www.oscars.org) and collected all necessary information on winners and nominees from as early as 1916. Our crawler does that simply by parsing the html pages returned as search result. It also gathers the names of people who won a specific award.

Important features that we added were whether each movie in the dataset was nominated for each category or not. Also, we added features that tell us whether the movie has won a specific award or not (Figure 1). Based on this, we added a column which specifies the number of awards won by a movie. However, when we want to predict if a movie is going to be nominated for Best Picture we ignore all other features regarding nomination. The reason is that this kind of prediction is supposed to be used on the time before nominees' names are released. Similarly, when we aim at predicting if a movie is going to win an award or not, we ignore features regarding winning other awards. The reason again is that this classifier is supposed to be used before Oscars' ceremony. The same goes for predicting number of awards for which we ignore the columns that say whether the movie has won an award.

Note: Though we never use *Award Winning Features*, we still include them in the dataset to facilitate the process of analyzing our results. For example, when we want to see the information on some movie that is always classified wrong we may use these information to get a better understanding of our data.

A. Missing Movies

The dataset that we downloaded from Kaggle has mostly social media and popularity features (e.g. movie facebook likes, imdb score). However, our main goal here is to predict Oscar winners/nominees. With that in mind, there were near 250 movies which were nominated for the Best Picture Award over the past century but were absent in our dataset; hence, we collected the title of these movies using our Java crawler and also repeated the work of Chuan Sun (creator of the dataset) to add the missing 250 movies. To to this we used the information provided on this blog [12].

An important point is that unfortunately the way that IMDB links are retrieved in Chuan Sun's codes is a little bit problematic. It first searches the title of the movie in IMDB using IMDB's search API, and after that uses the first link in the result as the link for the movie. However, this does not always work fine since sometimes the order of the results is not as desired. So, we refined this part of code to consider the link whose related title matches the title that we have. This way we managed to reduce the number of mistakes.

Movies from 2016: We first retrieved the title of 100 good 2016 movies from rottentomatoes.com and used Chuan Sun's code to collect social and IMDB information on these movies. We used this dataset to predict potential nominees for the upcoming Oscar ceremony in 2017.

B. Some Data Statistics

We have nearly 5000 movie titles in our dataset. Some statistics about these movies are as follows:

1) **IMDB score:** IMDB score is an important feature in our dataset. Figure 2 shows a pie diagram on distribution of movies based on rating.
2) **Title year:** Naturally, in recent years the number of movies have grown rapidly and our dataset reflects this

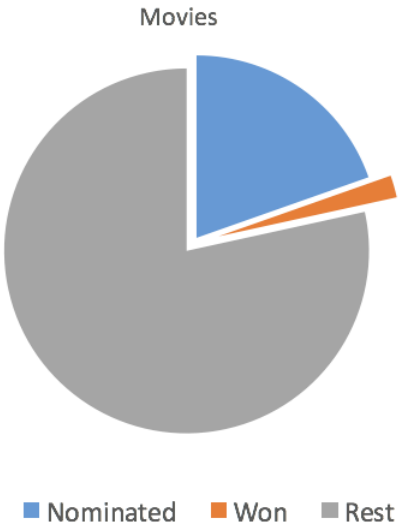

Fig. 1. Distribution of movies according to best picture nominated/winner statistics.

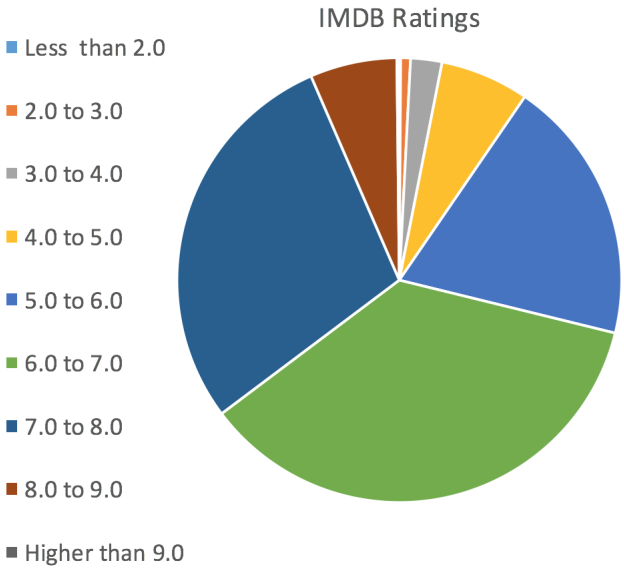

Fig. 2. Number of movies based on IMDB rating.

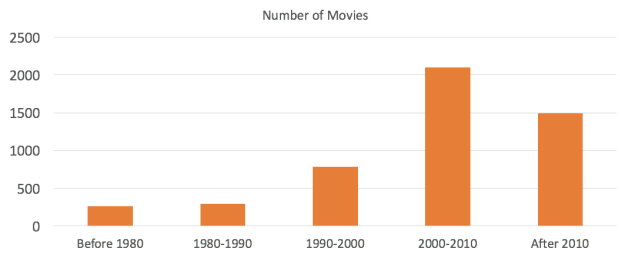

Fig. 3. Number of movies based on year.

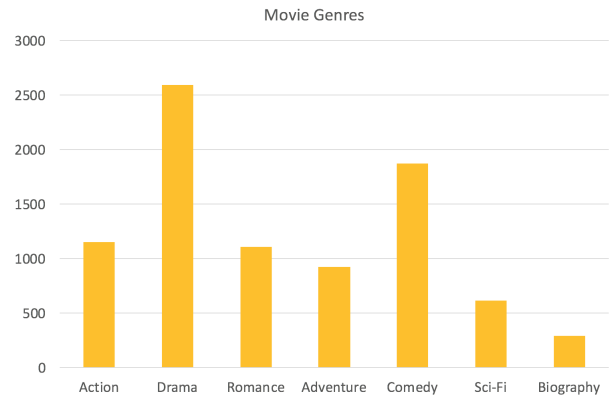

Fig. 4. Number of movies based on genres.

growth, as well. Overall, as illustrated in Figure 3, we have 261 movies earlier than 1980, 291 movies in 1980-1990, 786 movies in 1990-2000, 2100 movies from 2000 to 2010, and finally 1497 movies from 2010 to present time.

3) **Director:** There are 2399 directors involved in these movies. There are 36 directors who have each at least 10 movies. Top directors by the number movies they have in our dataset are: Woody Allen with 22 titles, Clint Eastwood with 20, Martin Scorsese with 20, Ridley Scott with 17, and Steven Soderbergh, Spike Lee and Tim Burton all with 16 movies. Also a total of 2363 directors have less than 10 and 1537 directors who have only one movie in our dataset.
4) **Actor/Actress:** There are 6258 actors/actresses involved in all these movies. There are 222 that have acted in at least 10 movies, and 6036 who have acted in less than 10 movies. Top actors/actress by the number of movies are: Robert De Niro with 54 movies, Morgan Freeman with 47 movies, Johnny Depp with 41 movies, Bruce Willis with 40 movies, Matt Damon with 38 movies, and Steve Buscemi with 37 movies. Also 3901 actors/actress are present in our dataset who have been involved in one movie only.
5) **Genres and plots:** Among the most frequent genres are Action, Drama, Romance, Adventure, Comedy, Sci-Fi and Biography. The most frequent genre is obviously Drama. And after that Comedy, Action and Romance follow. Figure 4 shows how many movies include each of these genres. Furthermore, plot keywords are another feature in our dataset. However, in order to be able to use them, we split keywords from genres and plots to different. Though this resulted in a huge matrix, we were able to get some important keywords such as *racism* and *suicidal thoughts* that help us in both classification and regression.
6) **Social networks** The only social network information that is present in this dataset is on facebook likes of movies, directors, actors/actresses and cast members overall. The highest movie facebook likes belongs to Interstellar with 349000 likes, then Django Unchained

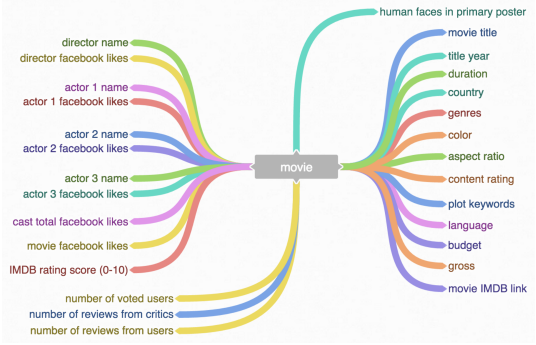

Fig. 5. Features in our dataset

with 199000 likes, Batman v Superman: Dawn of Justice with 197000 likes and Mad Max: Fury Road with 191000 likes. There are 749 movies that have less than 1000 likes on facebook.

C. Description of All Features

Now, we aim to describe each of the features present in our dataset. A simple diagram showing the name of all initial features of the Kaggle dataset downloaded from [12] can be found in Figure 5.

1) ***color:*** Shows the color of the movie. Its value can be either *Black/White* or *Color*.
2) ***director name:*** The name of the director as a string of characters.
3) ***director facebook likes:*** Number of likes on director's facebook page.
4) ***actor X name:*** X can be 1, 2 or 3. It is the name of the first, second or third most important actor/actress.
5) ***actor X facebook likes:*** The number of likes on the actor/actress's facebook page.
6) ***cast total facebook likes:*** The overall number of facebook likes on pages of all cast and crew.
7) ***movie facebook likes:*** Number of facebook likes of the movie.
8) ***imdb score:*** The rating of the movie on IMDB which is a number with one floating point between 0 and 10.
9) ***num voted users:*** Number of users who rated the movie on IMDB.
10) ***num user for reviews:*** Number of users who wrote a review for the movie on IMDB.
11) ***num critic for reviews:*** Number of critics who wrote a review for the movie on IMDB.
12) ***facenumber in poster:*** Number of human faces on the primary poster of the movie.
13) ***plot keywords:*** Keywords in the movie plot separated by pipe (|) character. We split these features into *keyword=* columns.
14) ***keyword=X:(Added by us)*** X is a word or collection of words. This feature is binary; 1 if the keyword is found in *plot keywords*, 0 otherwise.
15) ***genres:*** Genres of the movie separated by pipe (|) character. We split these features into *genre=* columns..
16) ***genre=X:(Added by us)*** X is a genre of movie. This feature is binary; 1 if the keyword is found in *genres* feature, 0 otherwise.
17) ***movie title:*** The name of the movie as a string.
18) ***title year:*** The year that the movie was released.
19) ***duration:*** Duration of the movie in minutes.
20) ***country:*** The country that produced the movie.
21) ***aspect ratio:*** Aspect ratio of the movie, for example if the screen is 16:9, the ratio will be 1.78.
22) ***content rating:*** Content rating of the movie, e.g. PG 13 which is not appropriate for children under 13.
23) ***language:*** Language of the movie.
24) ***budget:*** The budget of the movie.
25) ***gross:*** Gross income of the movie.
26) ***movie imdb link:*** The url of the movie on IMDB.
27) ***Nominated X: (Added by us)*** A binary feature indicating whether a the movie has been nominated for award category X. For example, *Nominated Art Direction*
28) ***Won X:(Added by us)*** A binary feature indicating whether a the movie has won a specific award X. For example, *Won Art Direction*
29) ***Num of Awards:(Added by us)*** The number of all awards won by that movie. This is bascilly the sum of *Won X* features for that movie.

Note: The columns **Nominated Best Picture**, **Won Best Picture** and **Num of Awards** turn into our class labels to predict.

D. Preprocessing and Refining the Dataset

The first thing that we did for refining our data was to remove mistakes from the *year* feature. In the original dataset in many cases the years extracted from IMDB are wrong. However, we knew that a lot of movies in our dataset have been nominated for at least one Academy Award. Therefore, using the results of our crawler we could retrieve the correct year of the movies and replace them in the dataset.

Then we moved on to preprocessing the data. Firstly, the data is loaded from the csv file. Since everything that is read by python from csv is essentially a string type, each value is checked for its convertibility to a numeric type, and then converted accordingly. Empty values represented by " are converted to the standard numpy "NaN" value. The result of this process is two typed 2D arrays, the first containing features as specified by the preprocessor formatted as [<*number-of-samples*>, <*number-of-features*>], and the second contains class labels formatted as [<*number-of-labels*>, <*number-of-samples*>].
The second step is essentially dealing with all empty/missing values from the earlier step. This is accomplished by using scikit-learns "Imputer" functionality. The "Imputer" function is used on a per-feature basis (labels are assumed not to hold empty values). As a filling strategy, the *most-frequent* mode is used. The reasoning behind this is quite simple. These are mostly categorical data, i.e. director names, actor names, and others. The mean, in this case, would make no sense. The median could be used, as it represents a single value,

however, it does not accurately give a sense of how the data is distributed. Neither does the mode, for that matter, however, it is the most logical choice among the three. Ideally, the dataset would be complete, or need to be completed by hand in order to fill in the gaps. So, this could be considered a weakness.
The third step is converting all categorical data into numerical data capable of being processed by scikit-learns methods. This is done using scikit-learns "LabelEncoder" functionality, which results in string values being mapped to numbers (usually their indices in an array). These are then substituted into the data. Another alternative was to pipe the result of the "LabelEncoder" through a "OneHotEncoder" to apply the binary technique discussed in class. This, however, would result in a larger matrix. And given the larger number of features that had been initially identified (roughly 8000), increasing matrix size would be considered disadvantageous, and was abandoned in favor of just using the "LabelEncoder".

E. Feature Selection and Dimensionality Reduction

Two of the features already given in the dataset consist of delimited genres and plot keywords. These were expanded into unique values per genre and unique values per keyword. Each additional feature would be encoded as a boolean flag, whereby if a movie is assigned a particular genre/keyword, its corresponding flag would be switched on (1), otherwise it would be switched off (0). This resulted in a large amount of extra features.
Due to the large amount of features and the large time consumed by processing them, a feature selection and dimensionality reduction strategy was considered. Pearsons correlation coefficient was calculated for each feature and each label. The feature-label combinations whose absolute correlation was greater than 0.1 (i.e. strength of the correlation, regardless of its nature) were the ones considered later for classification/regression. Variations based on team-member judgement on what would improve classification quality were later used. This is discussed later with each individual algorithm applied.
Note: As Dr. Fyshe pointed out, we were careful to select important features only using our training data, since including test data in feature correlation results in overfitting.

F. Features Used in Prediction

Basically, when we want to predict each of the class labels (i.e. Best Picture Nominee, Winner and Number of Awards) we use all the features whose correlation value the desired class label is above 0.1. (This selection of features is base only on the Training Data.) The reason for this threshold is that except for ***Nominated X*** features most of our correlation values are below 0.2 and if we do consider a rather high threshold there will not be enough features for the classifier/regressor to use. Here is the list of features that we use for prediction. Note that we only mention some of the ***keyword=X*** and ***genre=*** features as they are quite a few. Also, we mention any feature whose correlation result with respect to *at least* on class label is above 0.1.

Original Features Used in Prediction:
color, duration, gross, num_voted_users, facenumber_in_poster, num_user_for_reviews, country, title_year, imdb_score , movie_facebook_likes

Nominated X Features Used in Prediction:
Leading Actor/Actress, Supporting Actor/Actress, Art Direction, Cinematography, Costume Design, Directing, Film Editing, Makeup, Music Scoring, Music Song, Sound, Sound Editing, Visual Effects, Writing
(Notice that features such as Best Animated Movie is ignored)

Some of the Used Genres and Plot Keywords:
genre=Romance, genre=Action, genre=War, genre=Thriller, genre=Biography, genre=History, genre=Drama, keyword=jazz age, keyword=broadway play, keyword=gallantry, keyword=republic, keyword=casablanca morocco, keyword=visa, keyword=boer war, keyword=longshoreman , keyword=rose, keyword=dangerous job, keyword=coin toss

G. A New Technique for Generating Test and Train Sets

There are a number of ways to generate test/train sets. However, not all of them seem reasonable. For example, generating test/train sets using randomness does not seem logical because the number of Best Picture nominees/winners is small compared to the number of movies in our dataset. So, if we are unlucky we might not preserve the same distribution in our test/train sets as the original dataset.
Having that in mind, and based on what some works in the literature, splitting test and train using ***title year*** feature is a good idea. On simple way is, for example, to including movies after 2000 in our test set and movies before 2000 in our train set. However, we thought that since ***title year*** is a crucially important feature (as proved with the results of the Pearson's correlation), it is better to have representative of each *period of time* in our test set. In particular, out of every 4 years we put all the movies from one year into our test and the rest into our train set. This is done by computing the year modulo 4; if the remainder is 0 that movie goes to test set, otherwise, it goes into the train set. As a result, our test set contains nearly 25% of the data.

IV. Methods

This section discusses the methods we used in order to examine whether or not the gathered data were adequate to predict nominees and winners. The idea behind this was the following; we assume that data pulled from various sites (social media and others) regarding movies and their related features should be enough to predict whether or not a movie would get nominated or win. As such, we would use that data and feed it into various classifiers and regressors, then check to see if we achieved a reasonable performance level.

A. Classification

1) Support Vector Machines

One of our promising candidates for classification and regression is SVM. Are data is perfectly labeled and we can

use this supervised learning algorithm to first predict wether a movie is going to be nominated for the Best Picture Award and if so then we can try to see whether it will win the award. We can also use SVM as a regressor to predict the number of awards that a movie is going to win.
Our implementation of SVMs relies on the scikit-learn library which provides us with both classifiers (svm.SVC) and regressors (svm.SVR). Once we preprocessed our dataset we can feed it to scikit-learn classifiers.

2) Difficulties faced using SVMs

We faced a number of expected/unexpected problems while working with SVMs:

1) **Tuning parameters** SVMs have a number of parameters. Some of the important ones are *kernel* and *C* which specify type of kernel to be used and penalty parameter in SVM, respectively. In order to overcome this problem we used an exhaustive search and tried a variation of combination of parameters to see which one works better. Based on experiments we found a *poly kernel* or degree 3 is the best fit for our dataset. In particular we use the following settings for SVC and SVR:
   svm.SVR(C=0.005,kernel='poly',degree=3)
   svm.SVC(C=1.5,kernel='poly',degree=3)
2) **Scaling features** During our research towards optimization of SVM parameters we found out that SVMs are not scale invariant [10]. It is recommended that we scale features to the range $[-1, +1]$, with $\mu = 0$ and $\sigma = 1$, before fitting the classifier since the algorithm might not converge if features are not scaled. Also, before doing this we were getting very weird results in terms of accuracy when we used SVM as a regressor.
3) **Class weights** We compared the predictions of our classifiers with the true class labels for instances in validation set. In most cases, our classifiers have a tendency to predict more 0s than 1s, meaning that they most likely guess that something would not be nominated or would not win best picture award. This might be something natural coming from the distribution of our training set. However, we suppose that it is possible to get slightly better results by manipulating class weights in our classifier.
   Note: We did not manipulate class weights since our results were already acceptable and we thought doing so might result in an overfitted model.

3) Logistic Regression

4) Gaussian Naive Bayes

5) AdaBoost

Adaptive boosting is also one of our best learners that we have tried. What basically this classifier/regressor does is that it takes advantage of a number of weak learners to create a strong learner. As mentioned in [13], unlike support vector machines and neural nets, the training phase of AdaBoost only selects features which have predictive power; hence, faster running time even when a lot of features are around. But what we were surprised about was the fact that unlike SVMs we did not spend a significant amount of time and trial/error to tune the classifier. In fact, in our very first attempt we got results that were clearly better than that of the SVMs. (We will elaborate on the results later). For regression/classification we used the following setting:
AdaBoostRegressor(DecisionTreeRegressor(max_depth=20), n_estimators=300,random_state=None)
AdaBoostClassifier(DecisionTreeClassifier(max_depth=1), algorithm="SAMME",n_estimators=200)

In fact, the base estimator that is used here is a Decision Tree. Moreover, we are using the discrete version of the AdaBoost (SAMME) as we are interested in getting class labels as predictions rather than class probabilities.

6) Perceptrons

Another approach we tried was Perceptrons. Ideally, perceptrons would be able to separate the data linearly [14], into nominees and non-nominees, and then winners and losers. After performing 10-fold cross-validation on our training set, we were able to tune the parameters to the following:

- For nominations, the classifier was regularized using *l2* with the default value of 0.0001 in order to emphasize the more significant features.
- For wins, the classifier was regularized using *l1* with a much smaller value 0.00001 than the default.

These two settings showed the most promising results during cross-validation. These were the best tuning parameters possible given the data's non-linear distribution.

7) Neural Networks (MLPs)

The last approach in this section was neural nets [15] (or multi-layered perceptrons) as they are called in *scikit-learn* [16]. These were also run through cross-validation (10-fold), and tuned for the best possible performance.

- Net Structure:
  - For nominations, the network consists of 1 hidden layer of 200 nodes. This yielded the best results during cross-validation.
  - For wins, the network size was increased to accommodate two hidden layers of 200 and 250 nodes respectively.
- Activation Function:
  The most effective activation function after cross-validation was found to be the default function *relu* for both nominations and wins.
- Solver:
  - For nominations, the most effective solver was found to be the stochastic gradient descent solver *sgd*.
  - For wins, the most effective solver was found to be the default solver *adam*.
- Regularization:
  - For nominations, by keeping the regularization term *alpha* at the default, we obtained the best average F1-score.
  - For wins, we obtained the best results by increasing the value of the regularization term from its default 0.0001 to 0.001.

B. Regression

1) Support Vector Regressors (SVRs)

Covered in IV-A2.

2) AdaBoost

Covered in IV-A5.

3) Linear Regression

For linear regression, and after experimenting with all possible variations of the tuning parameters, the best possible fit was with its default settings, i.e. $fit_intercept = True$, and $normalize = False$. The $normalize$ setting was set to $False$ mainly due to the fact that the dataset was already scaled earlier on the the data preprocessing step. Linear regression, much like the perceptron, would try to estimate a line that runs closest to as much as possible of the points in the set, and use the equation from that estimated line in order to predict the dependent variable (in this case, the number of awards).

V. Results

VI. Discussion

VII. Contributions

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] http://www.economist.com/blogs/economist-explains/2015/01/economist-explains-14, Accessed on 10/02/16

[3] https://www.kaggle.com/datasets?sortBy=hottest&group=featured&search=imdb, Accessed on 10/02/16

[4] https://www.youtube.com/watch?v=qhfxo8xPNGU, Accessed on 10/03/16

[5] http://insights.principa.co.za/data-scientists-predict-oscar-winners (February 2016)

[6] Krauss, Jonas, Stefan Nann, Daniel Simon, Peter A. Gloor, and Kai Fischbach."Predicting Movie Success and Academy Awards through Sentiment and Social Network Analysis." In ECIS, pp. 2026-2037. 2008.

[7] http://maxmelnick.com/2016/05/18/predicting-oscar-nominations.html (May 2016)

[8] Barber, Stephen, Kasey Le, and Sean ODonnell."Predicting the 85th Academy Awards." (2012).

[9] https://blog.nycdatascience.com/student-works/machine-learning/movie-rating-prediction/, Accessed on 10/03/16

[10] http://scikit-learn.org/stable/modules/svm.html

[11] Kessy,Agnan,AlexLewin,andKorbinianStrimmer. "Optimalwhiteninganddecorrelation."arXivpreprintarXiv:1512.00809(2015).

[12] https://blog.nycdatascience.com/student-works/machine-learning/movie-rating-prediction/

[13] https://en.wikipedia.org/wiki/AdaBoost

[14] Han, J., Pei, J. and Kamber, M., 2011. *Data mining: concepts and techniques*. Elsevier.

[15] https://en.wikipedia.org/wiki/Artificial_neural_network accessed on 12/01/2016

[16] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html accessed on 12/01/2016