****

**Fachbereich:** Data Science

**Lehrveranstaltung:** SE Advanced Data Challenge

**LV-Nummer:** 405.982

SS 2020

Projektarbeit zum Thema:

**„Hit or Miss? Ein Projekt zur Vorhersage der Popularität von Liedern mithilfe von Supervised Learning“**

**Vorgelegt von:**

Zivkovic Patrick (01422129)

Angerer Michael (01421983)

**Abgabedatum:**

31.07.2020

**Wortzahl:**

xxxx

[Abstract 1](#_Toc47090987)

[Introduction 2](#_Toc47090988)

[Original Dataset 2](#_Toc47090989)

[Spotify 2](#_Toc47090990)

[Loss of Data 3](#_Toc47090991)

[Genres 3](#_Toc47090992)

[Natural Language Processing 4](#_Toc47090993)

[Final Dataset 4](#_Toc47090994)

[Descriptives 5](#_Toc47090995)

[Modelling 7](#_Toc47090996)

[Baseline Model 8](#_Toc47090997)

[Generalized Linear Model 8](#_Toc47090998)

[Neural Network 9](#_Toc47090999)

[Random Forests 9](#_Toc47091000)

[Gradient Boosting 9](#_Toc47091001)

[Stacked Ensemble BOF 10](#_Toc47091002)

[Stacked Ensemble All 10](#_Toc47091003)

[All methods in comparison 10](#_Toc47091004)

[Conclusion and Future Work 11](#_Toc47091005)

# Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

# Introduction

Our original idea was to use the attached *song-lyrics* dataset (songdata.csv) to perform natural language processing techniques on it and hopefully extract some interesting information. We then decided to take an extra step and tried to use the acquired knowledge from the song-lyrics, among other variables, towards a prediction of the overall popularity of a song. The following work will describe all the steps and methods we used to be able to accomplish this goal.

# Original Dataset

The original dataset, that we found on Kaggle, is very simplistic. It lists the title of the song along with the artist and the associated lyrics, mostly without punctuation. Additionally, it provides information about where the lyrics of the songs were retrieved from in the form of a link. The amount of songs the dataset contains amounts to 57650.

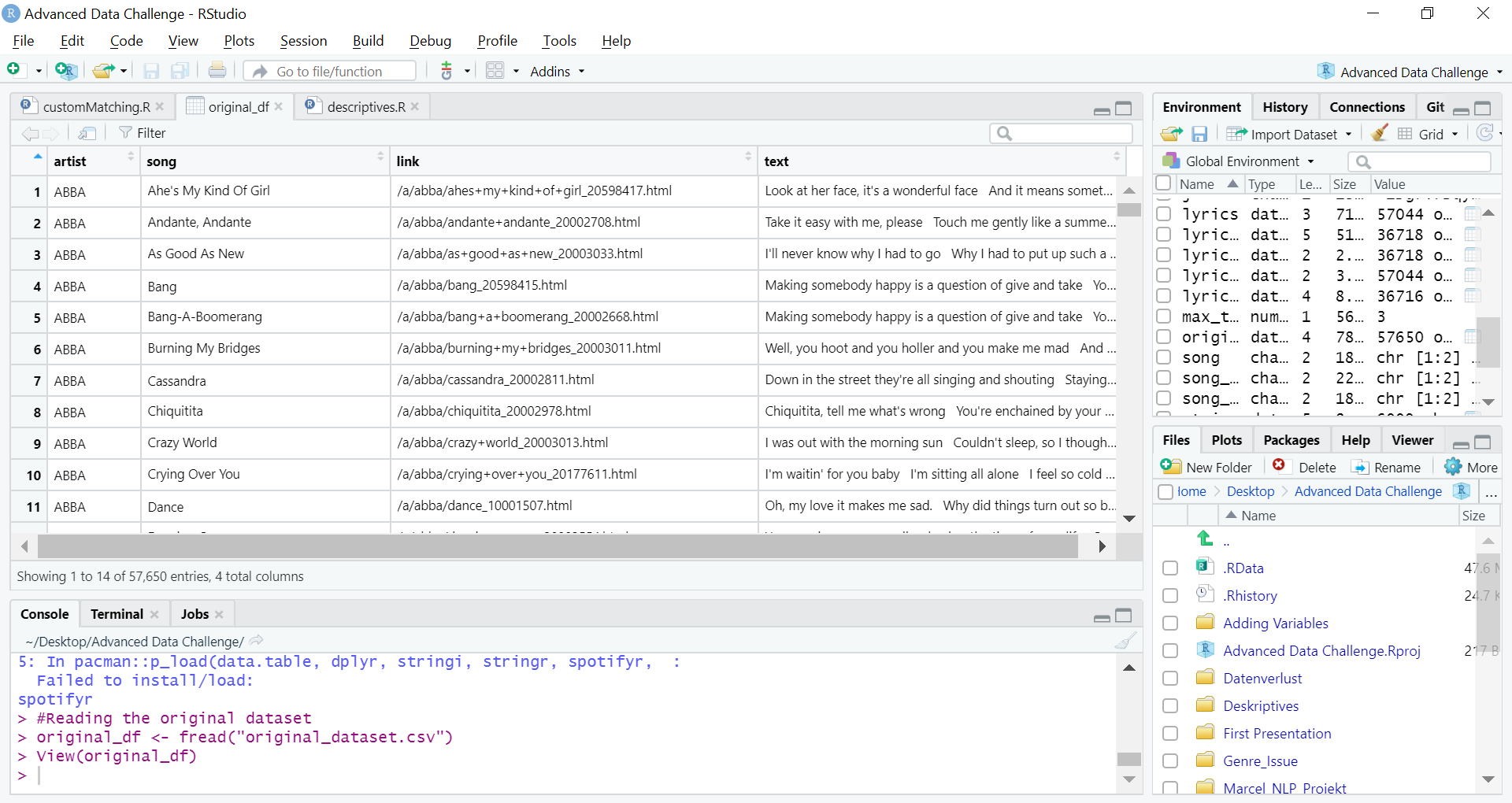


Table 1: Overview of the first few rows of the original dataset

# Spotify

As already mentioned, our main goal was to predict the overall popularity of the songs from our dataset, but to predict something you first need a measure of it. Spotify came right to our minds, because we know that Spotify provides an API for app developers and that they also store a lot of metadata for every song – including a popularity score of each song. From the massive amount of data Spotify stores, we decided to use “data that would theoretically be available without the Spotify API”, so one could later on predict the popularity of new songs with our models without dependencies of external data.

R provides a package (*spotifyr*) that allows you, after you register an app with Spotify to get a client key and a secret key, to connect to the Spotify database with R and extract metadata. Spotify itself stores different objects (Track, Album, Artist) with different kinds of metadata that are stored with unique keys. To extract to right data, you need to have unique keys for the artist id, the album ids, and track ids, which we used to enrich our original dataset (see fig.1).

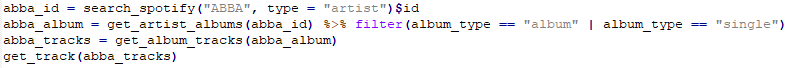


Figure 1: Simplified R-Code for extracting artist, album and song metadata from Spotify

We extracted the song lengths and the popularity of songs from the Track Objects, year of release from the Album Objects and artist genre and artist popularity from the Artist Objects. A challenge we encountered was that it seemed like that not every song was available in Spotify, but after manually searching for popular songs that seemed to be missing we encountered that some song names of our original datasets were misspelled.

E.g. The Beatles’ song “She’s My Kind Of Girl” with the right spelling in Spotify, turned out to be “Ahe’s My Kind Of Girl” in the original dataset. To solve this problem, we had to use a matching function in order to join the songs from our original dataset with the songs from the Spotify library.

The matching was performed with help of the R package *stringdist* and more specifically the function *amatch*. The integer parameter *maxDist* determines the value of how “similar” the song titles have to be in order to call them a match. The metric used to determine that similarity was the *restricted Damerau-Levenshtein distance.* What it does, in a nutshell, is counting the number of deletions, insertions and substitutions necessary to turn string b into string a.

# Loss of Data

During the intent of matching the song titles from our original dataset with the songs from the Spotify data base, a lot of titles were mismatched or could not find a corresponding match at all. All in all, our original sample size of 57650 songs shrank to a final number of 36889 (about 64% of the original dataset). Some reasons for the loss of data and some problems that led to mismatching are going to be listed in the following.

Initially we chose the *maxDist* parameter of our matching function to a fixed value of 2. That led to problems because all song titles were being treated in the same way disregarding their length. This especially led to mismatches with very short song titles, since the matching function allowed for a fixed amount of 2 differences between the strings. In one example a song named “Hope” was assigned to a song named “Home” and in another one the song title “God” was assigned to the song title “Gold”. This issue was solved by making the *maxDist* parameter dependent on the length of the song title, namely set it to a value equal to 15% (rounded down) of the amount of characters in the string. That way particularly small song titles would need to be exactly equal in order to propagate a match.

Our original dataset contained a comparatively small number of artists considering the large amount of songs. 598 artists on 57650 songs which averages to around 96 songs per artist. For that reason, most of the songs of the whole musical career of these artists were covered in the dataset. Some of those songs, however, were not featured in the Spotify library, so a lot of songs were lost due to a simple lack of availability.

By far the biggest reason why we experienced such a big data loss was due to appendages to the song titles in the Spotify library. There are a lot of songs that add additional information to the song title. For example, another Beatles song from the original dataset "All I’ve Got To Do” was only available on Spotify under the name "All I’ve Got To Do - Remastered 2009". Our matching function could, of course, not deal with such irregularities.

We decided not to deal with this problem in the scope of this project, because we found that our dataset with 36889 songs was still big enough for the later modelling.

# Genres

One problem with using Spotify to merge metadata is, that it doesn’t store genres for individual songs but only for its artists. Also, Spotify uses a very broad definition of genres, for example “mellow gold” for soft rock and folk rock with a strong correlation to album-oriented classic rock of the ‘60s, ‘70s, and ‘80s. So this music, which could narrowly be defined as “rock” is classified as mellow gold by Spotify. For generalization purposes on our rather small sample of 36889 songs with originally over 300 different genres (see fig. 2) we decided that a narrower definition of the genres is more fitting to predict the popularity of a song in a particular overlying genre (rock, pop, metal, country, etc.).

We performed web scraping on <https://en.wikipedia.org/wiki/List_of_popular_music_genres> and <https://www.musicgenreslist.com/> using the r-package “*rvest*” to extract to overlying genres for every underlying genre and matched them with the extracted genres from Spotify. For even more specific genres that weren’t included but contained buzzwords, we performed buzzword matching to assign them to an overlying genre (“heartland rock” -> rock, “ post-teen pop” -> pop). With this method we managed to reduce the amount of genres to 23 (see fig. 2).

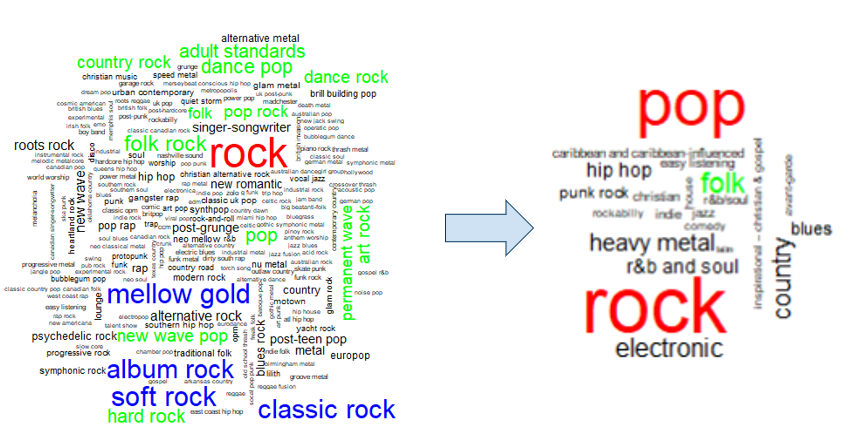


Figure 2: Visiualization of genre reduction through matching with overlying genres

To ensure a moderately fair classification of the genre of a song while only using the artist genres, we assigned every song of an artist the overlying genre that he matched the most. So, if an artist had 3 underlying genres belonging to rock and 2 underlying genres belonging to metal, every song of the artist would be classified as rock.

# Natural Language Processing

To extract information out of the provided song lyrics of our original dataset we used Natural Language Processing (NLP). Our goal was to get an indicator for the polarity of the songs (e.g. are they sad, happy, etc.) and a readability score for every song lyric so that our models can learn if there may be a way to success by writing specific lyrics in a certain way, either by influencing their polarity or by making it hard (or easy) to understand.

We decided to try 2 separate polarity scores using the *tidytext*-package. Our first approach included the NRC-lexicon (<http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>) that the tidytext-package provides. It groups every word into one (or multiple) of 10 basic emotions namely joy, surprise, trust, sadness, anger, fear, anticipation, disgust, negative and postive. We created bag of words from the lyrics and matched the included emotion words with the basic emotions from the NRC-lexicon, counted the basic emotions the song contained and scaled them by dividing the amount of one basic emotion through all emotion words found in the assessed song lyric.

Our second approach stemmed of the idea that some emotion words have more sentimental value than others (e.g. wonderful is a more positive word than good) and that all emotion words can coarsely be defined as either positive, neutral or negative and that a song can be ambivalent or have almost no emotional value in its lyrics and still be a good song. For this approach we used the afinn-lexicon (<http://corpustext.com/reference/sentiment_afinn.html>) which rates the valence of an emotion word between integer -5 and 5. To better define if an emotion word is inherently negative, positive or neutral, we joined the Loughran-lexicon (<https://emilhvitfeldt.github.io/textdata/reference/lexicon_loughran.html>) where the emotion word is classified as either positive or negative. We created bag of words from the lyrics and matched the included emotion words with the resulting lexicon of combining afinn and Loughran and summed the values of the emotion words and grouped them by positive, negative or neutral.

For the readability indices we chose two different formulas, the Coleman-Liau-Index and the SMOG (Simple Measure of Gobbledygook)-Index. Because readability indices use sentence lengths in their computation and song lyrics tend to miss the necessary punctuation to infer sentences we tried to clean the punctuation of the song lyrics to make the formulas work. Luckily our lyrics still contained upper case letters at the beginning of new sentences. While controlling for capital “I” and proper nouns by filtering for very small sentence lengths when splitting sentences before upper case letters, we inserted punctuation at points where upper case letters occurred. Figure 3 shows an example of the text before and after cleaning for punctuation.

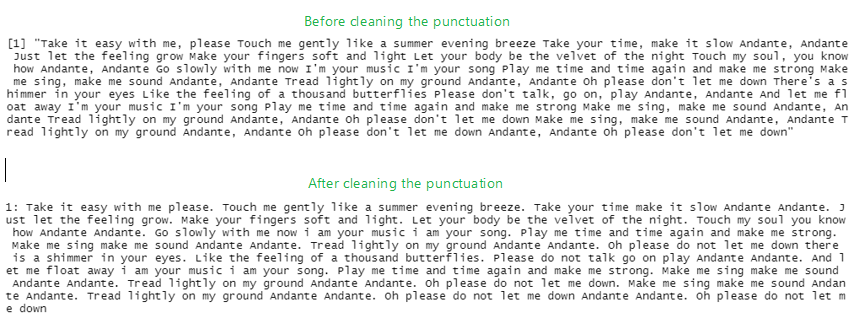


Figure 3: Example of song lyrics before and after cleaning the punctuation

Both readability indices put out the approximate US. grade level needed to comprehend the text, but compute them differently. Coleman-Liau was designed to be easily calculated mechanically from samples of hard-copy text and therefore uses the amount of characters (letters) for its computation, while most other readability indices, like the SMOG-index, use a form of syllable counting.

The coleman-liau-index uses to following formula

while the SMOG-index uses

While the SMOG-index was normed on a sample of 30 sentences and the statistical validity is questionable on text with fewer than 30 sentences, the coleman-liau index is useable without restrictions.

We computed the indices on the cleaned song lyrics with the help of the *qdap*-package and its functions *SMOG* and *coleman\_liau*. Figure 4 shows how the Coleman Liau and the SMOG-index interact. While the American school system has 13 grades (including Kindergarten) all computed indices for both readability indices should be between 0 and 13, every other value should be invalid and negative values shouldn’t appear. The valid SMOG-values are bounded by the red lines and the valid Coleman Liau-values are bounded by the blue lines. As figure 4 shows there are only two points that are not valid for either the Coleman Liau-score or for the SMOG-score, every other readability value is for at least one of the indices valid, so they complement each other well on our data.

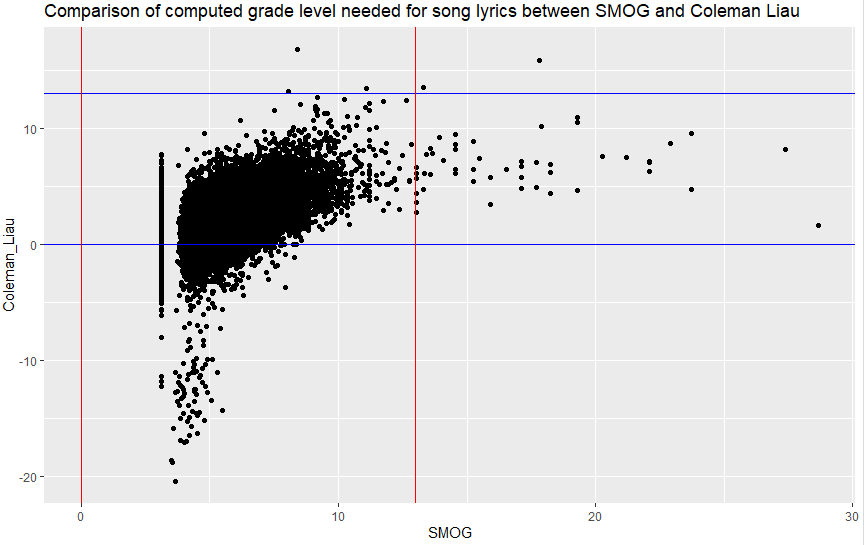


Figure 4: Scatterplot of Coleman Liau vs SMOG

# Final Dataset

The following table gives an overview on our final dataset after all feature collection and extraction steps mentioned above were completed:

|  |  |  |
| --- | --- | --- |
| **variable** | **description** | **example** |
| artist | The artist’s name | ABBA |
| songname | The title of the song | She’s My Kind Of Girl |
| year | The year the song was released in | 1973 |
| popularity | The popularity of the song (0-100) | 20 |
| duration\_s | The duration of the song in seconds | 165 |
| artist\_popularity | The popularity of the artist (0-100) | 80 |
| genre | An overgenre of the artist | pop |
| Coleman\_Liau | Readability index of the lyrics (mean = 2.39) | 1.97 |
| SMOG | Readability index of the lyrics (mean = 5.68) | 4.66 |
| trust | Sentiment indices (ranging from 0 to 1) | 0.23 |
| fear | 0 |
| negative | 0 |
| sadness | 0.08 |
| anger | 0 |
| surprise | 0.08 |
| positive | 0.31 |
| disgust | 0 |
| joy | 0.27 |
| anticipation | 0.04 |
| neg\_sent | How positive/neutral/negative is the song in terms of the sentiment-words and their sentiment value? (starting from 0) | 0 |
| neutral | 8 |
| pos\_sent | 13 |

# Descriptives

In the following we are going to present some interesting descriptives of our dataset, mostly focusing on the variables, that are most relevant for the building of our models in the next chapter.

Figure 5 shows how the songs of our dataset are spread out over the decades. The number of songs gradually increases peaking in the 2000s with just above 10000 songs.

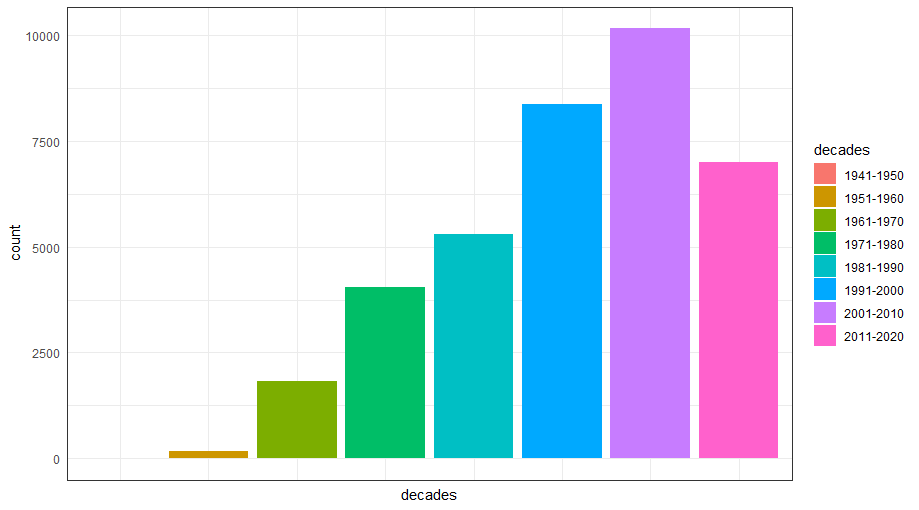
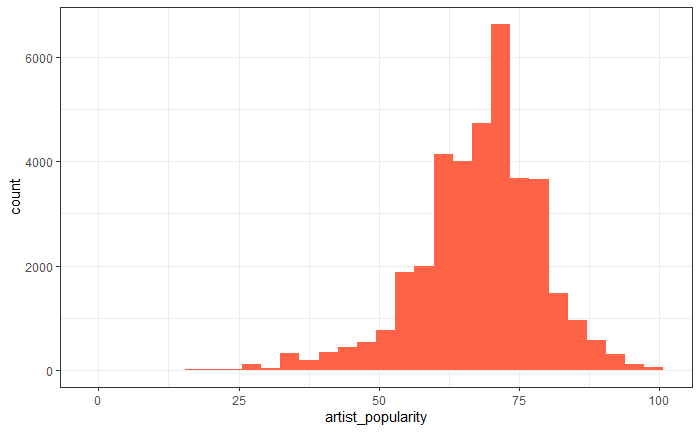
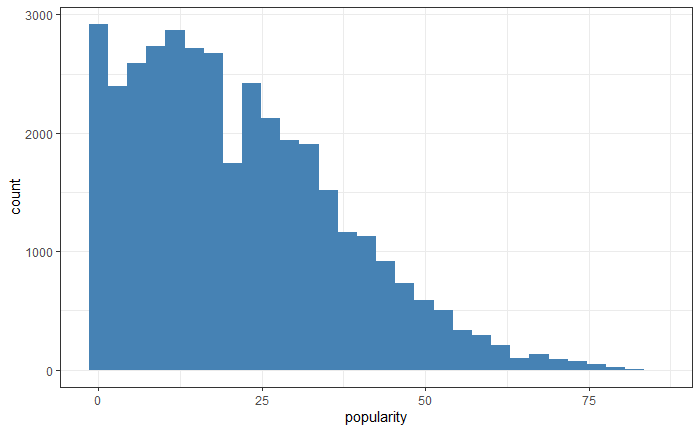
*Figure 5: Number of songs by decade*

Figure 6 and Figure 7 show the scatter of the artist popularity and the song popularity. It is immediately evident, that the song popularity has a distribution that is strongly skewed to the left, meaning that most of the songs of our dataset have a popularity that ranges between 0 and 50 although the variable generally ranges between 0 and 100.

*Figure 6: The distribution of the variable artist popularity*

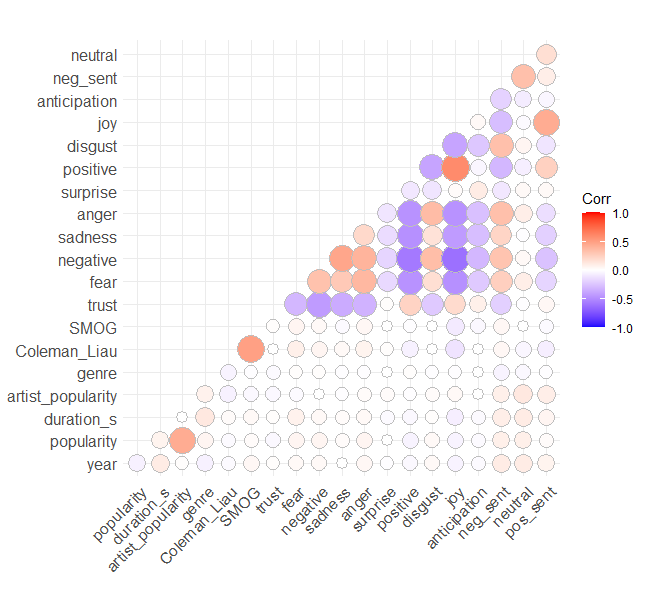


*Figure 7: The distribution of the values of the song popularity*

# Modelling

In this chapter we are going to report the results we achieved trying to predict the popularity of a song with all the variables at our disposal. First, we are going to take a look at a baseline model to set a benchmark for the more complex models that will follow later.

Figure 8 shows a correlation plot illustrating all interdependencies among the variables of our dataset. Strong correlations are found between the variables popularity and artist popularity, the two readability indices Coleman-Liau and SMOG and in general between variables of the same general sentiment.



*Figure 8: Correlation plot with all the variables of the dataset*

For all methods described below, a 10-fold cross-validation was performed in order to get an accurate measure of how well the particular method is suited for this kind of problem. The mean absolute error was used as a metric to determine the quality of the prediction performed on the test set. It was calculated for every single run and then averaged over the 10 different runs for a result, that is comparable with the other methods used.

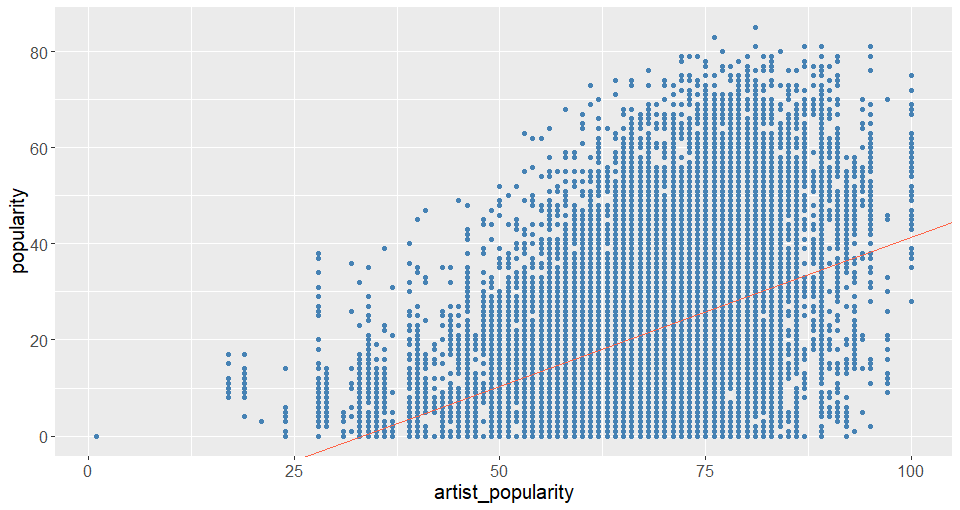
## Baseline Model

For some baseline results, we simply calculated the mean value of the song popularity for a training set and predicted the same value for all the songs of the test set. The resulting average mean absolute error was **12.758** with a standard deviation of 0.089 across the 10 runs of the cross-validation. This result already appears to be pretty good, since it suggests that the popularity of any given song can be predicted with an accuracy of ±12.758 in a space that ranges between 0 and 100. On closer inspection we can see that the predictions are only accurate in the subspace where most values are populated, which is between 0 and 50 (see: Figure 7).

## Generalized Linear Model

As our first method, that also took our independent variables into account, we chose a simple multivariate regression. We performed a forward selection in order to determine which variables have the largest impact on our dependent variable song popularity and to incorporate them into our model. Judging by the adjusted R-squared values, the four most important variables were artist popularity, the sentiment variable joy, the release year and the duration of the song. Figure 9 shows that the strongest independent variable artist popularity is highly correlated with the song popularity (r = 0.427) and contributes the most towards good predictability. This does not come as a surprise, since already popular artists tend to release songs that become popular as well.

The cross-validation showed that the model with the best result was the one that used a total number of 12 variables of the 19 variables our dataset contains. The average mean absolute error for that model turned out to be **11.301** with a standard deviation of 0.029.



*Figure 9: Scatterplot showing the relationship between the artist popularity and the song popularity*

## Neural Network

Next, we trained a fully-connected neural network in the Keras environment in Python. We set it up by building two hidden layers with 6 hidden neurons in total, because that number seemed to produce the best results for this particular problem. We were training the model for 50 epochs, using the technique of early stopping if the validation error did not substantially change for 10 epochs.

Figure 10, however, shows that a neural network approach is not suitable for this problem, since the test error does not considerably decrease. The result of the approach shows almost no improvement compared to the linear regression having an average mean absolute error of **11.066** and a standard deviation of 0.69.

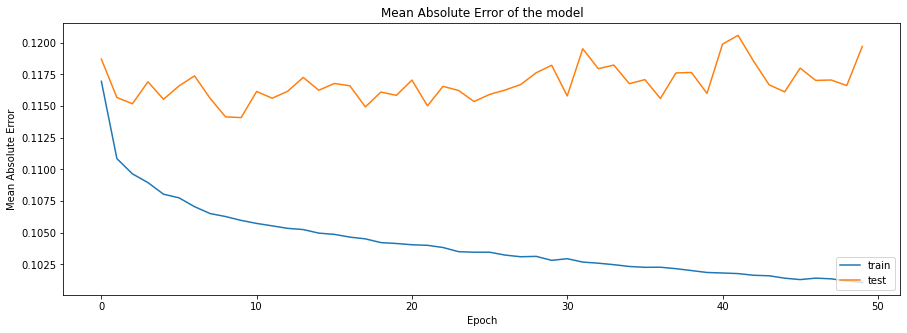


Figure 10: Change of MAE over 50 epochs of training on the test and on the trainingsdata

All the following models were trained in R with the *h2o*-package. H2o itself is an open source Machine Learning platform with its core code written in java. The h2o-package accesses the h2o-rest-API and translates the R-code into java code and executes it on the local Java Virtual Machine. We decided to train models with 3 different types of algorithms: one (Distributed) Random Forest, one Gradient Boosting Machine and one Auto-Machine Learning-algorithm that creates Stacked Ensembles by combining different types of algorithms with a GLM-regression. For the modelling itself the data was split into 60% training data and 40% test data.

## Random Forests

For the training of the Distributed Random Forest (DRF)-model, we performed a hyperparameter grid search to find the optimal maximal depth (between 12 and 20) and the optimal amount of trees (between 10 and 30) to reach the best performance (measured by mean absolute error) on the cross-validated data. The best performing DRF-model included 27 trees with a consistent depth of 18 and a mean of 7755 leaves. It lead to an averaged mean absolute error of **9.701** with a standard deviation of 0.105 on the cross validated training data and mean absolute error of 9.71 on the test data.

## Gradient Boosting

The training of the Gradient Boosting Machine (GBM)-model was performed similar to the DRF-model. We performed a hyperparameter grid search to find the optimal maximal depth (between 5 and 15), the optimal amount of trees (between 5 and 20), the optimal learning rate (between .05 and .15), the optimal column and row sample rate (between .5 and 1) and the fewest amount of allowed observations in a leaf (between 5 and 20). The best performing GBM-model on the cross validated data (measured by mean absolute error) included 19 trees with a consistent depth of 15, fewest amount of allowed observations in a leaf of 6, a learn rate of 0.15 and a column and row sample rate of 1 respectively. It lead to an averaged mean absolute error of **9.178** with a standard deviation of 0.104 on the cross validated training data and test data a mean absolute error of 8.916 on the test data. Figure 11 shows the most influential predictors in the GBM-model with artist popularity, year of release and genre as the most important predictors.

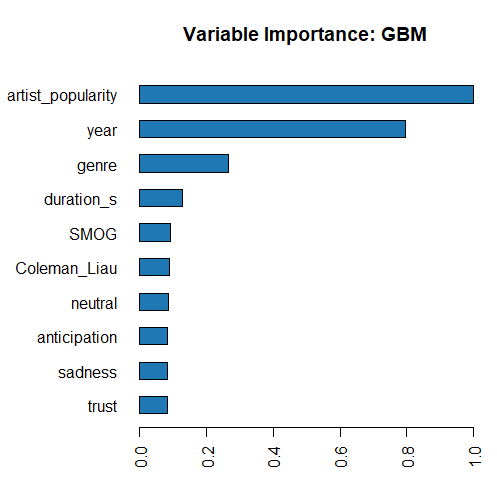


Figure 11: Variable Importance of the predictors in the GBM-model

## Auto-ML

Auto-ML is an algorithm provided by h2o that automatically trains and cross validates DRF-, GMB-, Extremely Randomized Forest (XRT)- , Deep Learning (DL)- and Generalized Linear Model models with randomly initialized hyperparameters. It than creates a Stacked Ensemble by using a GLM-Regression on the resulting predictions on the cross validated training data and so weights the influence that particular models have on the final prediction. Auto-ML uses two different models for the stacking process: The first Ensemble is very big and uses every of the (here) 223 models created (Stacked Ensemble All). The second one (Stacked Ensemble Best of Family (BOF)) uses only the best performing model on the cross validated training data of every family (the best DRF, GBM, XRT and DL).

Figure 12 shows the results of the GLM of the **Stacked Ensemble All**. It shows that of the 223 models only 11 models influence the final prediction of the Ensemble. Out of all the models we trained, this Ensemble shows the best performance on the cross validated training data with an averaged mean absolute error of **8.721** and a standard deviation of 0.0542 and a mean absolute error of 8.489 on the test data.

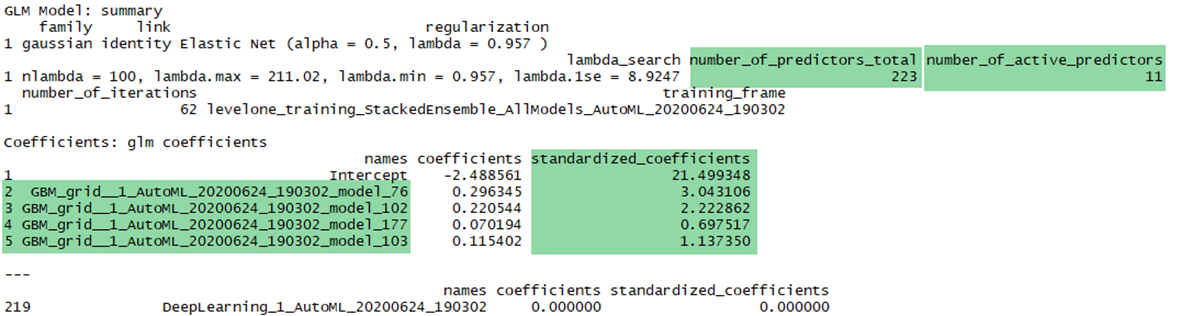


Figure 12: Influence of the individual models in the Stacked Ensemble All

Figure 13 shows the results of the GLM of the **Stacked Ensemble BOF**. It uses only the best model of each family, but as can be seen, the Deep Learning- and the Generalized Linear Model-model have no influence on the final prediction of the Ensemble. The model with the highest influence is by far the GBM-model, the DRF and the XRT model only have a small influence. Out of all the models we trained, this Ensemble has the second best performance on the cross validated training data with an averaged mean absolute error of **8.927** and a standard deviation of 0.0191 and a mean absolute error of 8.688 on the test data.

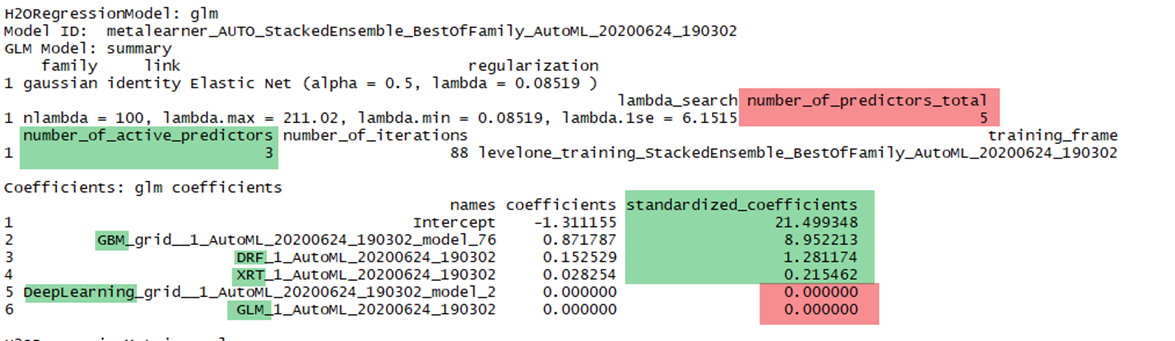


Figure 13: Influence of the individual models in the Stacked Ensemble All

## All methods in comparison

|  |  |  |
| --- | --- | --- |
| **Model** | **Mean Absolute Error (averaged over 10 folds)** | **Standard  Deviation** |
| Baseline Model | 12.758 | 0.089 |
| GLM | 11.301 | 0.029 |
| Neural Network | 11.066 | 0.69 |
| Random Forests | 9.701 | 0.105 |
| Gradient Boosting | 9.178 | 0.104 |
| Stacked Ensemble BOF | 8.927 | 0.019 |
| **Stacked Ensemble All** | **8.721** | **0.054** |

# Conclusion and Future Work