Spotify Track Features

ITMAL Slutprojekt - Gruppe 28

Jacob Odgaard Hausted - 201510912 Gill Lumer-Klabbers - 201607384
Kasper Gnutzmann Andersen - 201607263

5/12-2019

Contents

1 Problem Description

This project applies the the process for developing a Machine Learning (ML) Model described in the chaper "End-to-End Machine Learning Project"[1] with the goal of creating a ML model to predict genres and popularity of tracks from Spotify based on other features defined for each of these songs.

The core idea is to produce two pipelines in parallel, working independant of each other. One pipeline should end with a model to be used to predict the popularity of songs based on the given features, and the other to try and predict the genre of a given song. As such, this project will work both with regression and classification. It must be stated that in order for this process to work, whether it be prediction of a quantity or category, there must be some correlation/distinction between the features. Thus, the underlying assumption for the project, and decision to use the dataset, is that the audiofeatures generated by spotify to describe their songs are connected in some way - and our goal is find an algorithm that can use that connection for prediction, if it is there.

Since the dataset includes the target values of our predictions, these features will be seperated from the rest of the dataset, and will be used as the output ($\mathbf{y_{true}}$) labels/values in training the model. Hence, the training strategy will be supervised learning, for both the classification and regression problem, as the expected values are known.

The "End-to-End" process and supervised learning concepts will be further explained in the following sections.

1.1 End-to-End Machine Learning study process

Gennerally the chapter outlines the following key steps in developing ML-models:

- **Getting the Data and looking at the Big Picture:** Defining the problem we wish to apply a ML model to solve, along with finding and analyzing the dataset to get an overview and initial insight into any patterns and relationships between features in the data set.
- **Prepare the Data for Machine Learning Algorithms:** Based on the initial data analysis, the necessary data cleaning and feature scaling is planned and performed, to prepare the data for the selected model(s). Most ML models perform poorly on accidental null-data and features with a high variance. To this end we can drop bad samples or features and apply normalization or standardization to sanitize the data.
- **Selecting, Training and Fine-Tuning a Model:** When the data has been prepared, it is time to train an appropiate model and validate the result. This is done utilizing a iterative process, where the model is optimized according to a cost function, and then validated using a score metric.
- **Launching, Monitoring and Maintaining the System:** This is the eventual goal of the entire process. When the model has been optimized, then it is time to launch it. Hopefully it will perform well and make accuarate predictions. However it is important to keep monitoring and maintaining the model, as it generally tends to rot over time as the input data might change, and therefore retraining on fresh data might be necessary.

[1] Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurélien Géron, O'Reilly, 2017

1.2 Supervised learning explained

In this project, a supervised learning approach is used to train the selected models, see Figure 1 for a detailed map-view af this approach.

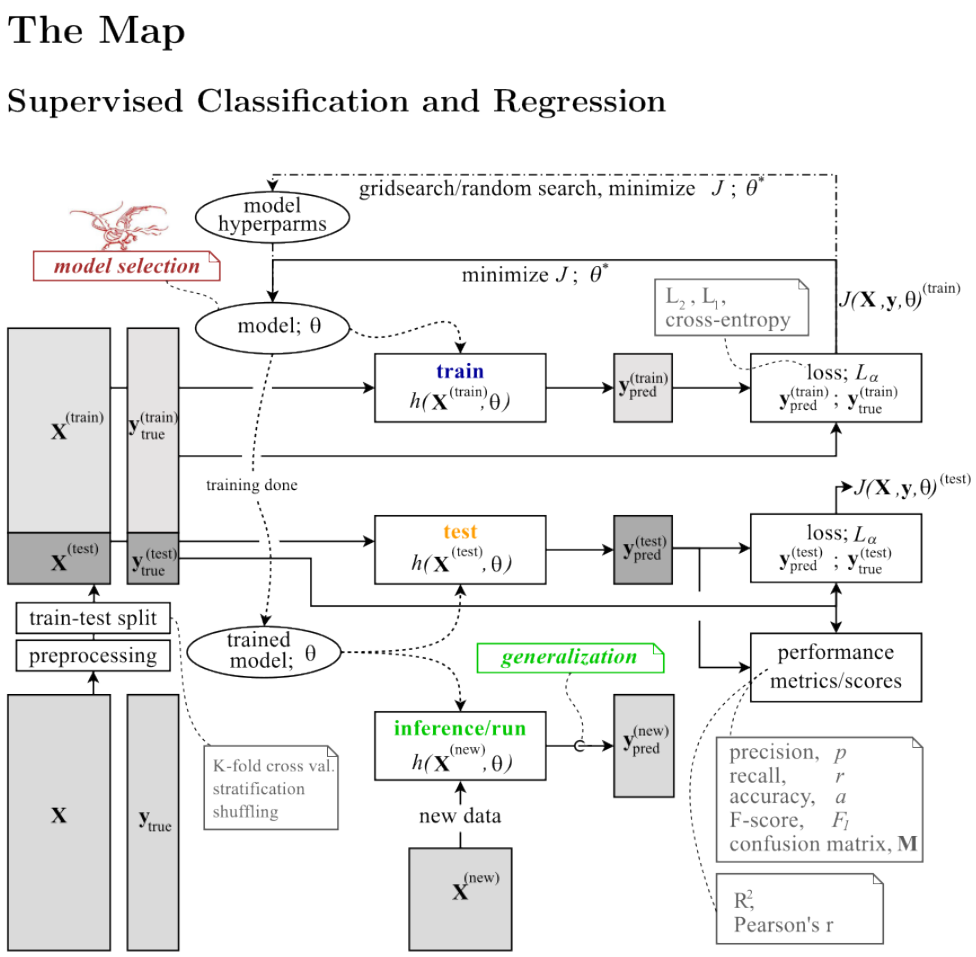

Figure 1: Map-overview of model-fitting using supervised learning

The collection of data samples containing all the information on all the different features denoted $\mathbf{X}$, is one of two components defining the entire set. The other denoted $\mathbf{y_{true}}$ is a list of true values, one corresponding to each sample in $\mathbf{X}$. $\mathbf{y_{true}}$ can either a be a label or an numerical value depending on whether the problem is classification or regression.

When the data have been aquired and preprocessed, it is split into two sets; one for training the model and another for performance validation. The components in these sets are denoted $\mathbf{X_{train}}$, $\mathbf{y_{train}}$, $\mathbf{X_{true}}$ and $\mathbf{y_{true}}$ for the training and validation (test) set respectively. Splitting the data into two sets is an important step, because in order for the validation to be meaningful, the model has to be applied to data it has not parsed before. This can also help to identify whether or not the models ability to generalize has been been compromised by over-fitting to the training data.
The next step after splitting the data is running the training loop. Here the model ($\mathbf{h}$) is fed the data and then tasked to make predictions ($\mathbf{y_{pred}}$) on each of the training samples. It then evaluates this result compared to the samples $\mathbf{y_{true}}$ using a cost function ($\mathbf{J}$). Often this function is chosen to be the least square solution. Training the model is an iterative process where the goal is to minimize the cost function for the entire set of training samples. In each iteration of the training loop the model parameters (θ) are tweaked, until the cost function reaches its minimum and the ideal model parameters have been found.

When the training process is done, the model is tasked with making predictions on the test set (unseen samples). These predictions are compared to $\mathbf{y_{true}}$ but instead of applying a cost function, a score metric is used instead to get a meaningful representation of how well the model performs. The goal is to reach as high a score as possible in contrast to the finding a minimum for the cost function.

2 The Spotify Tracks dataset

The dataset for this project has been found on kaggle, and contains data gathered directly from spotify's API[2]. The dataset includes around ~233 thousand songs with 16 features. These features can be divided into two main categories, namely identification/general features and sound/feel features, as follows:

General		**Sound and feel**			
- Genre	*string*	- Acousticness	*0 - 1*	- Loudness	*< 0*
- Artist Name	*string*	- Danceability	*0 - 1*	- Mode	*string*
- Track Name	*string*	- Duration in ms	*> 0*	- Speechiness	*0 - 1*
- Track ID	*string*	- Energy	*0 - 1*	- Tempo	*string*
- Popularity	*0 - 100*	- Instrumentalness	*0 - 1*	- Time signature	*string*
		- Key	*string*	- Valence	*0 - 1*
		- Liveness	*0 - 1*		

It should be noted here that the data has already been processed to some degree by spotify before we attempt to do anything with it. For instance, the popularity feature is always a value between 0 and 100, and thus, must be generated from some algorithm translating the yearly/monthly/daily listens into some entity. With that said, some preprocessing must still be performed, be that cleaning or scaling, before the data is suitable for the models.
It should also be noted that the group has concerns about the size of the dataset, relative to the number of classes. With 26 unique genres, 233 thousand samples might not be sufficient for some models.

2.1 Visualising the data

As a first step in preprocessing the data, sufficient knowledge of the dataset and its features should be established. A special focus will be put on popularity and genre, since those are the features ($\mathbf{y_{true}}$) to be predicted by the models.

2.1.1 Feature distributions

Starting off with the popularity, a histogram is made using matplotlib to gain insight of its distribution.

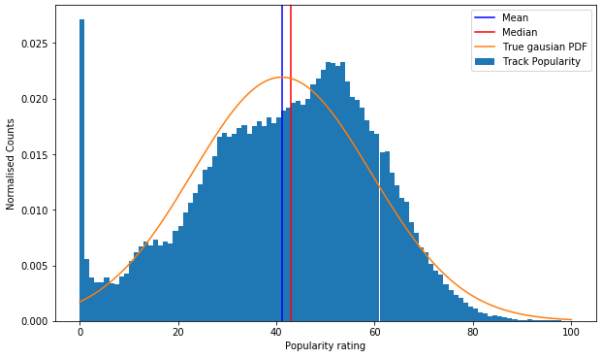

Figure 2: Histogram of the popularity feature, for the whole dataset

[2]https://developer.spotify.com/documentation/web-api/

It can be seen that the distribution resembles a normal distribution, but deviates significantly in some places. The most notable difference being the big spike in the first bin (a popularity rating of 0), which could have several explanations. It might be that there are simply just a lot of unpopular songs on the plaform, as the data would seem to indicate. Another explanation could be that the popularity score is not calculated by spotify before some criteria is met, e.g. a track has been on the platform for period of time. It depends on how and when the score is computed, when the tracks were put on the platform, etc - all information that the group does not currently have.

Continuing, since the dataset included multiple genres, an assumption of each genre having it's own distribution seems somewhat probable, and as such, the popularity across each genre is plotted.

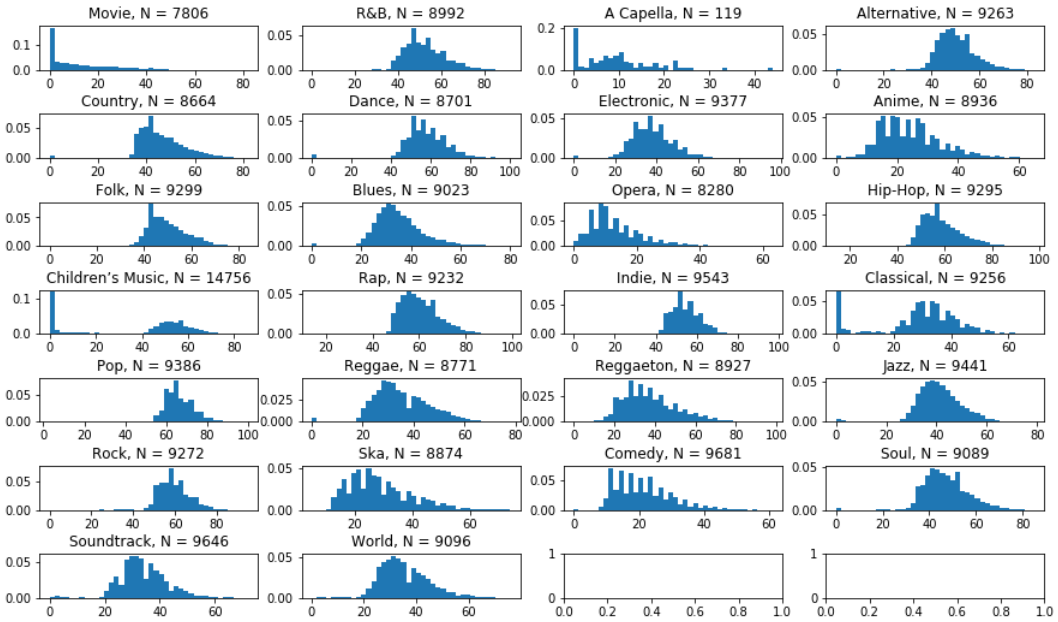

Figure 3: Histograms of popularity for each genre

From Figure 3, it is seen that the assumption holds somewhat true. Furthermore, a few key insights is gained, namely that the "A Capella" genre only has 119 samples, and only a few genres contains the outliers with a popularity rating of 0, which was seen in the general histogram. Indirectly, seeing that the feature distribution is somewhat different across genres, this could be examined further. Here, one could examine the means/variances across genres of different features and plotting these against each other, with an example below of the mean of the popularity for each genre.

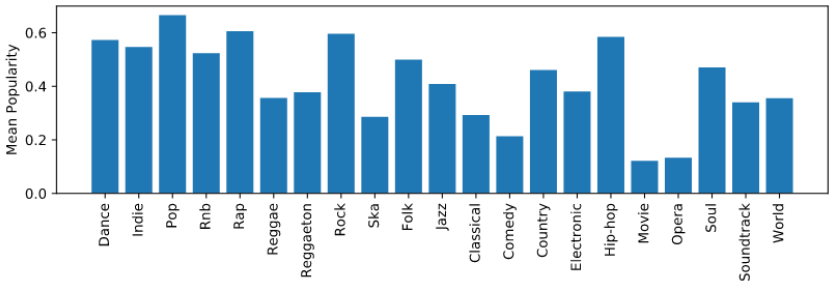

Figure 4: Popularity mean for each genre

This process however, would get rather tiresome to do for each feature. As a better alternative, a radar plot is used instead for each genre, with each plot containing multiple feature means. This type of plot has the

advantage of providing a more comprehensive overview of the differences and similarities across genres. It should be noted here, that the data has been scaled to value between 0 and 1, which is also done in order to improve our models efficiency later. The radar plots are shown below in Figure 5

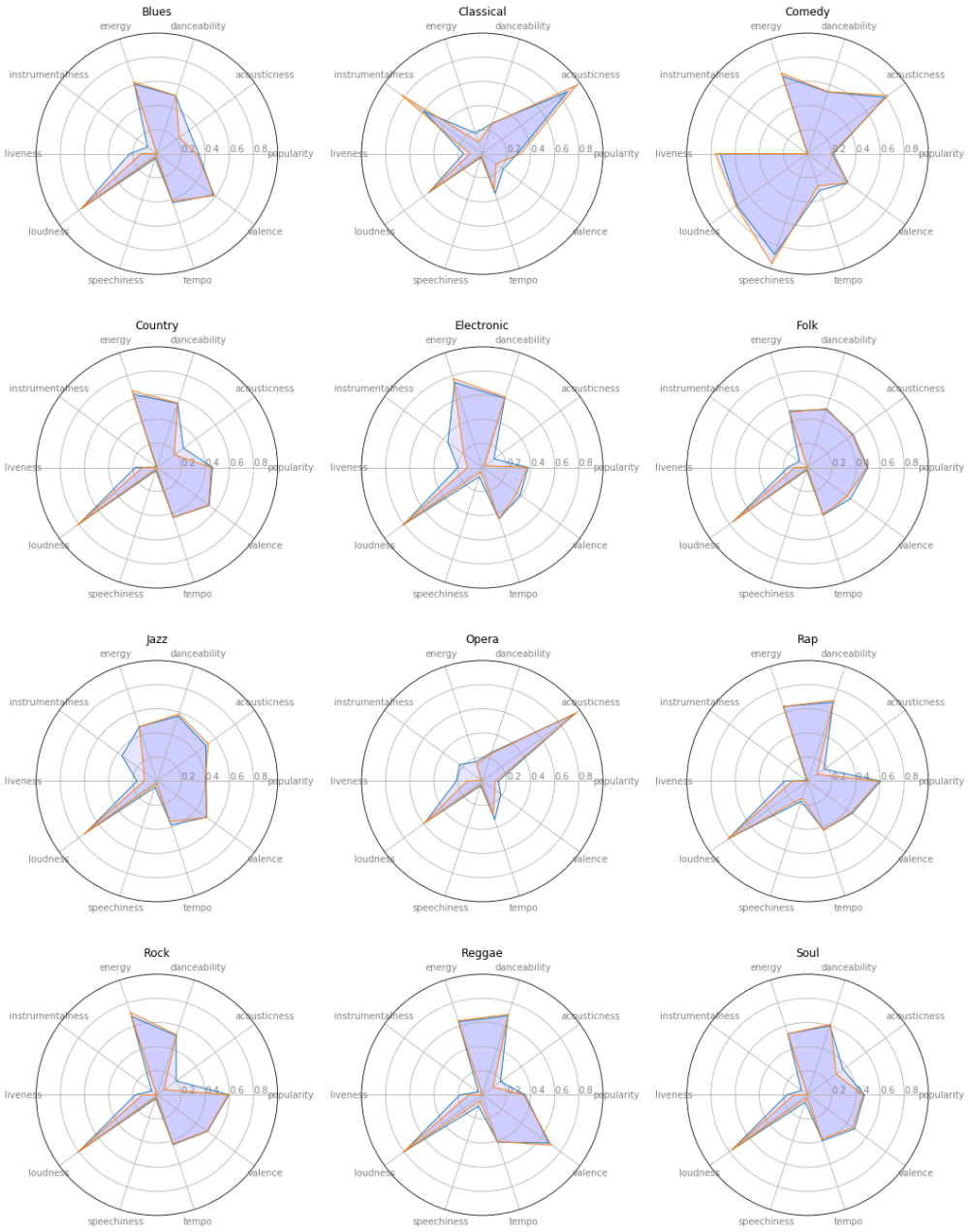

Figure 5: Mean and Median values for each numerical feature, pr genre visualised on radar plots

From Figure 5, a distinction between the shapes of the radarplots can be seen, which is good, because in order for classifier to distinguish between genres, their features should differ in some way. There are however some issues, which can be seen if comparing for example Rock, Raggae and Country. The shape in the radar plot of these genres is very similar, and as such, it must be expected that trying to classify tracks included in these genres will be harder. This thought will be expanded upon later in the report, by applying principal component analysis (PCA) to the data.

2.1.2 Correlation between features

Now, in moving back to the regressor to be used for predicting popularity, the correlations between popularity and other features should be analysed. Turning the beforementioned analysis around, where popularity was argued to be somewhat related to genre, the genre feature can be seen as an important feature to predict the popularity from - this leaves the rest of the features to be analysed. To easily visualize the results of these correlations, scikit learn's `scatter_matrix()` is used. Using only some of the features here as an example, the results of running the `scatter_matrix()` is seen below:

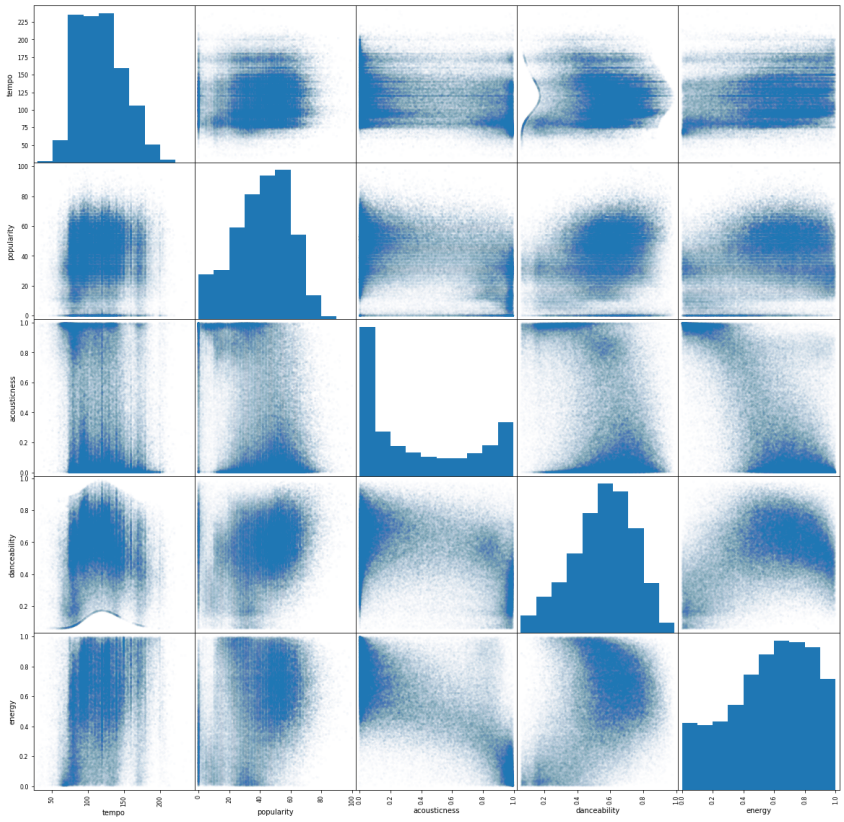

Figure 6: Scatter matrix of the popularity, tempo, acousticness, danceability and energy feature

Looking at Figure 6, the popularity does not seem to be correlated much with the other features, which is unfortunate. This result leaves the expectation for the regressor rather low - if there is no correlation, no mathmatical connection between the two can be made, and thus no predictions can be made. However, this method is limited to comparing two of the feature dimensions directly with each other, showing that one of the features cannot reliably predict the value of another. But some combination of multiple features might provide a better basis for predictions, and it is hoped that a regressor might uncover these if they exist.

2.2 Cleaning the data

From the visualization, a few key points was seen with regards to the data:

- The popularity feature contains a significant amount of samples with a rating of 0.
- The "A Capella" genre only contains 119 samples
- The duration feature is in a too high range, and contains significant outliers
- Both the genre, key, time signature and mode are all labeled traits

Since the group has no idea what causes a popularity rating of 0 - Is it a new song without any data yet? Is there a threshold of listens/checks that causes the popularity rating? Is it just unpopuler with no interrest? -

It has been decided to drop the samples with this rating from the dataset. This does however cause some implications, as our models ability to predict unpopular songs will be severely hindered. Futhermore, dropping samples from the already somewhat small dataset is not desireable, as it also hinders the models ability to gather information and improve itself.

The "A Capella" genre does not contain enough samples to provide valuable information about the genre itself, no model will be able to learn from 119 samples, and almost certainly not distinguish between other genres from it (outliers will have too much effect as no real distribution can be made).

The duration feature was difficult to tackle, as one could no simply scale it to a range of 0 - 1, since the outliers would cause too much of a skew in the range and thus, the feature would face truncation issues. From Figure 7 it is seen that each feature has at least 1 significant outlier for their duration feature, while all being distributed around a somewhat similar mean value.

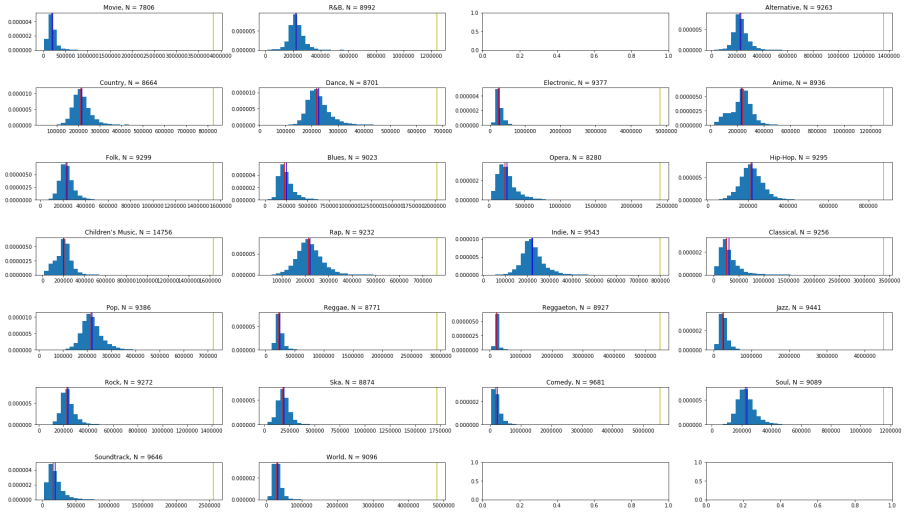

Figure 7: Showcasing the max value of outlier (yellow) of the duration feature pr genre

To overcome the outlier issue, a manual boundary could be set in place, which would discard the outliers, and remove them from the dataset. The boundary could be found from analysis of the distribution plots, but this solution would cause further reduction in the dataset. Therefore, since the correlation to popularity was already low, and the distribution mean and meadians across genres are so similar, the feature itself is removed instead of discarding samples.

For the labeled features, some form encoding must be applied, and it has been decided to use one hot encoding for this. This does not come without a price, as this effectively increases the amount of features used in the models, and the size of the dataset should be sufficient to still be able to learn properly. Looking at just the genre, we are looking to expand the dataset with an additional 24 features - this could be problematic.

The 'mode'-feature expands to 2 features, and 'time signature' to 3. Originally there were 5 distinct values for 'time signature' found in the dataset, 0/4, 1/4, 3/4, 4/4 and 5/4. But entries with 0/4 and 1/4 were removed from the dataset, as they are arguably meaningless in context of the dimension they are supposed describe, and it is unclear how they should be interpreted. In addition, there were few entries with 0/4 or 1/4, futher suggesting that they are anomalous artifacts of the procedure they were generated by, and their removal seems justified. The 'key'-feature would expand to an additonal 12, and there was not found any real correlation, thus the feature was decided to be dropped. The rest was encoded.

OBS: The genre has only been one hot encoded for the regressor, as alot of scikit learns classifiers contains *labelBinarizer()*[3] inside them - thus, they do not expect their y-values to encoded beforehand. In the case of them expecting it to, *labelBinarizer()* should be used for in order to satisfy the the models of scikit learn.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html#sklearn.preprocessing.LabelBinarizer

2.3 Feature scaling and PCA

After the data cleaning, the features should be scaled in order to improve the models efficiency. If a model uses the Euclidean Distance between points in the feature space of our datamatrix, the distance is governed by range of our features. As such, a feature with high range will contribute more than those with low range, which will skew the final distance, as the features will no longer contribute equally(which is what we want). This is especially seen if we are to use the K-nearest Neighbours model, as this is exactly based on the Euclidean Distance.

Scaling should be performed carefully, as stated before with the duration feature, truncation issues and outlier issues must be taken into account. If scaling is done properly, there should not be any downsides in form of performance of the algorithms, as those who works on non-scaled data does not generally suffer from receiving scaled input. Furthermore, in order to further investigate the the tendencies noted from the radarplots, a PCA decomposition into a 2D or 3D plane would be beneficial.

Now, there are several feature scaling methods to choose from, but for the groups project, only standardized scaling has been used:

$$z = \frac{x - \mu}{\sigma}$$

The group decided on this scaling method due to the desire to use PCA on the dataset. The advantage for using standarized scaling in comparisson to just normalizing the data is that the PCA maximises the variance of the feature projections onto it's component axes. In order to ensure an equal weight of each feature, min max scaling(normalization) should not be used, as the outlier effect of this scaling method would degrade the performance of the PCA, in the sense that min max scaling ensures no guarantee that the variances of the features is dimensioned equally.

With the scaling done, the relation tendencies from the radarplots can be further investigated using PCA, plotting both in 2 dimensions and 3:

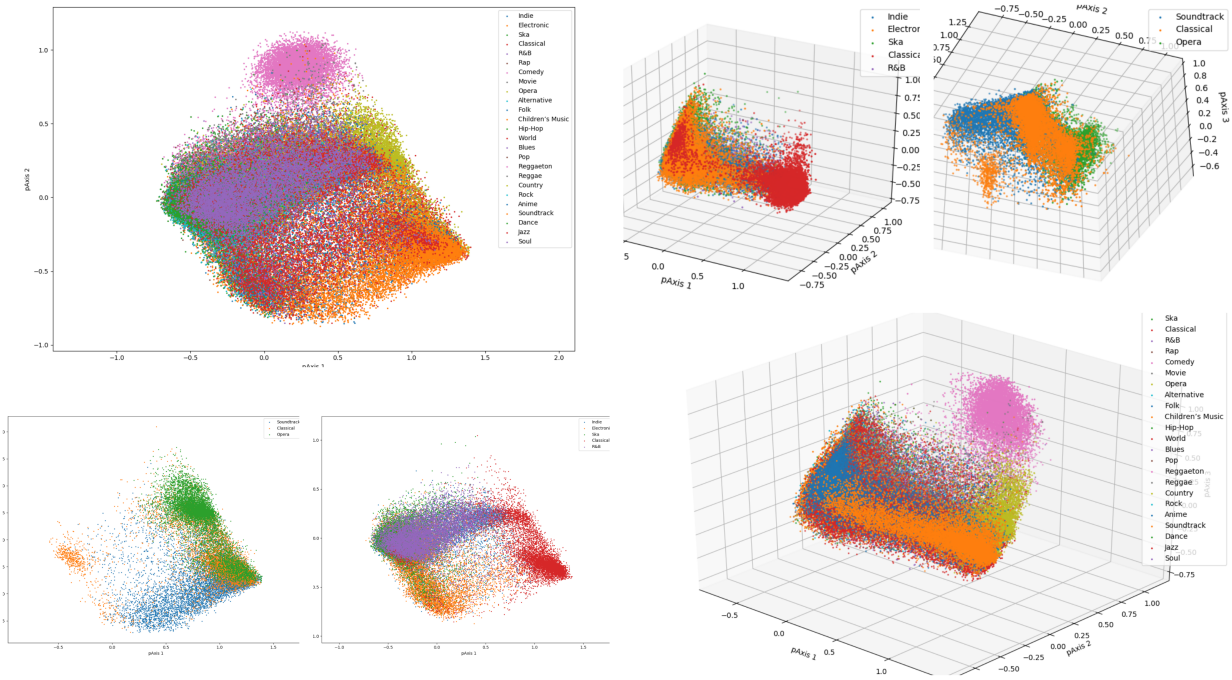

Figure 8: PCA plot using matplotlibs scatter, both in 2D and 3D. Only a sample of the results used is shown here. Both 2D and 3D contains the same samples, but reduced differently (Plots cropped to keep size at acceptable level). *Left*: 2D Plots, *Right*: 3D Plots.

From the PCA plots, the hypothesis of some genres overlapping more than others is verified, and as such, it must be expected that the classifier will have varying results, being better at distinguishing some genres than others. Some interresting things to note from the plots is the fact that alot of the more "*mainstream*" genres, such as *Rock, Pop, Rap, Hip-Hop* are all grouped together. Likewise, both *Classical, Opera, Soundtrack* and especially *Comedy* are somewhat distinguishable. *Classical, Opera, Soundtrack* are grouped somewhat together, as they all overlap in one place, which from intuition makes somewhat sense, as *Soundtracks* often uses *Classic* pieces, and *Opera* and *Classical* often use elements from each other (Mozarts pieces are all classified as classical on spotify, but "*Die Zauberföte*" is a mix of opera and classic).

3 Genre Classification

One of the main goals of setting up the ML-pipeline with this dataset, is to investigate the prospect of predicting a genre, based on the audiofeatures of a track. As mentioned, this is a supervised classification task, as the desired output is one destinct category of a predefined set of possible options.

3.0.1 Algorithm Selection

Before training can begin, an algorithm, or set of algorithms, must be selected for the job. The choice of algorithm has been affected by which algorithms the group has had previous experience with, and the primary basis of comparison has been their success at performing similar classification tasks. Due to the very good results with the K-Nearest Neighbors (KNN) algorithm in the MNIST search quest exercise for O3, this was the initial choice for the genre-classification task.

In addition, the group has chosen a second type of model, fully connected Neural Networks (NN), for comparison. This algorithm seems like a good complementary alternative to the KNN algorithm, primarily due to its flexibility and scalability. Since there are more ways to configure a Neural Network, it could also prove more difficult and time-comsuming to find the optimal initial configuration of the network. But this also means that it has a good chance of being able to cover any areas where the KNN is insufficient, making it a good fall-back candidate

3.0.2 Data processing and structure

The input for the classification models consists of all the scaled audiofeatures, along with the one-hot encoded 'mode' and 'time signature' features discussed in Section 2.2. This gives the model a total of 15 features to use a basis for assigning a genre to each input. The classes themselves are created from the "genre" column of the dataset, which is encoded through a LabelEncoder, assigning an integer to each possible value, replacing the original string. This means that the classes will be represented by integer values ranging from 0-24, and this data will be the y-values for our model.

Before the model is trained, the data is split into training and validation sets. The model is fitted to the training-set, and afterwards the validation-set is used to test how the model responds to previously unknown samples. The proportion of the dataset assigned to the testset was selected to be ≈ 0.3.

3.1 K-Nearest Neighbors

As mentioned, the KNN classifier has been selected as the primary candidate for genre-prediction. The configuration of the model is described below, followed by the results.

3.1.1 Hyperparameter Search

In order to get the best possible performance, the model is initialized multiple times with different combinations of hyperparameters, so that the best values for each hyperparameter can be found by evaluating the performance of each combination. The search performed on the KNN classifier considers the following hyperparameters and values:

```
n_neighbors = [3, 4, 5, 10, 20, 50, 100]
    weights = ['uniform', 'distance']
  algorithm = ['ball_tree', 'kd_tree', 'brute']
          p = [2, 3, 4]
```

After searching through the selected hyperparameter space, the best hyperparameters were found to be:

```
n_neighbors = 100
    weights = 'uniform'
  algorithm = 'ball_tree'
          p = 2
```

These are the final parameters given to the K-Nearest Neigbor model.

3.1.2 Results

Initializing a model with the found hyperparameters and fitting it to the training data, has yielded the results shown in Table 2 and Table 3

3.1.2.1 Performance metrics The performance metrics for the KNN model are taken from sklearn's classification report, which contains the following four metrics:

- *Precision* - The fraction of positives that were true positive, defined as $\frac{tp}{tp+fp}$
- *Recall* - The fraction of positive samples the model was able to find, defined as $\frac{tp}{tp+fn}$
- *F1-score* - The weighted mean of *recall* and *precision* (both are equally weighted in this case)
- *Support* - The number of samples in the validation set

In combination, these four metrics give a good picture of the models overall performance.

Table 2: Classification results (pr. class/genre)

Class/Genre	Precision	Recall	F1-score	Support
Alternative	0.23	0.20	0.21	2687
Anime	0.51	0.38	0.44	2660
Blues	0.36	0.30	0.32	2640
Children's Music	0.26	0.19	0.22	3289
Classical	0.53	0.51	0.52	2423
Comedy	0.95	0.93	0.94	2708
Country	0.25	0.42	0.31	2563
Dance	0.20	0.17	0.18	2595
Electronic	0.48	0.48	0.48	2736
Folk	0.24	0.31	0.27	2692
Hip-Hop	0.27	0.40	0.32	2783
Indie	0.16	0.11	0.13	2835
Jazz	0.36	0.33	0.34	2813
Movie	0.57	0.27	0.37	1609
Opera	0.67	0.86	0.75	2389
Pop	0.30	0.40	0.34	2861
R&B	0.20	0.17	0.18	2718
Rap	0.24	0.18	0.21	2793
Reggae	0.40	0.38	0.39	2588
Reggaeton	0.45	0.58	0.51	2636
Rock	0.25	0.32	0.28	2744
Soul	0.21	0.11	0.14	2739
Soundtrack	0.53	0.69	0.60	2821
World	0.37	0.36	0.36	2694

Table 3: Classification results (summary)

	Precision	Recall	F1-score	Support
Accuracy			0.38	66778
Macro avg.	0.38	0.38	0.37	66778
Weighted avg.	0.37	0.38	0.37	66778

3.1.2.2 Distribution of predictions pr. class The models performance is examined further by plotting how the models predictions for the test data is distributed for each genre (true and false positives).

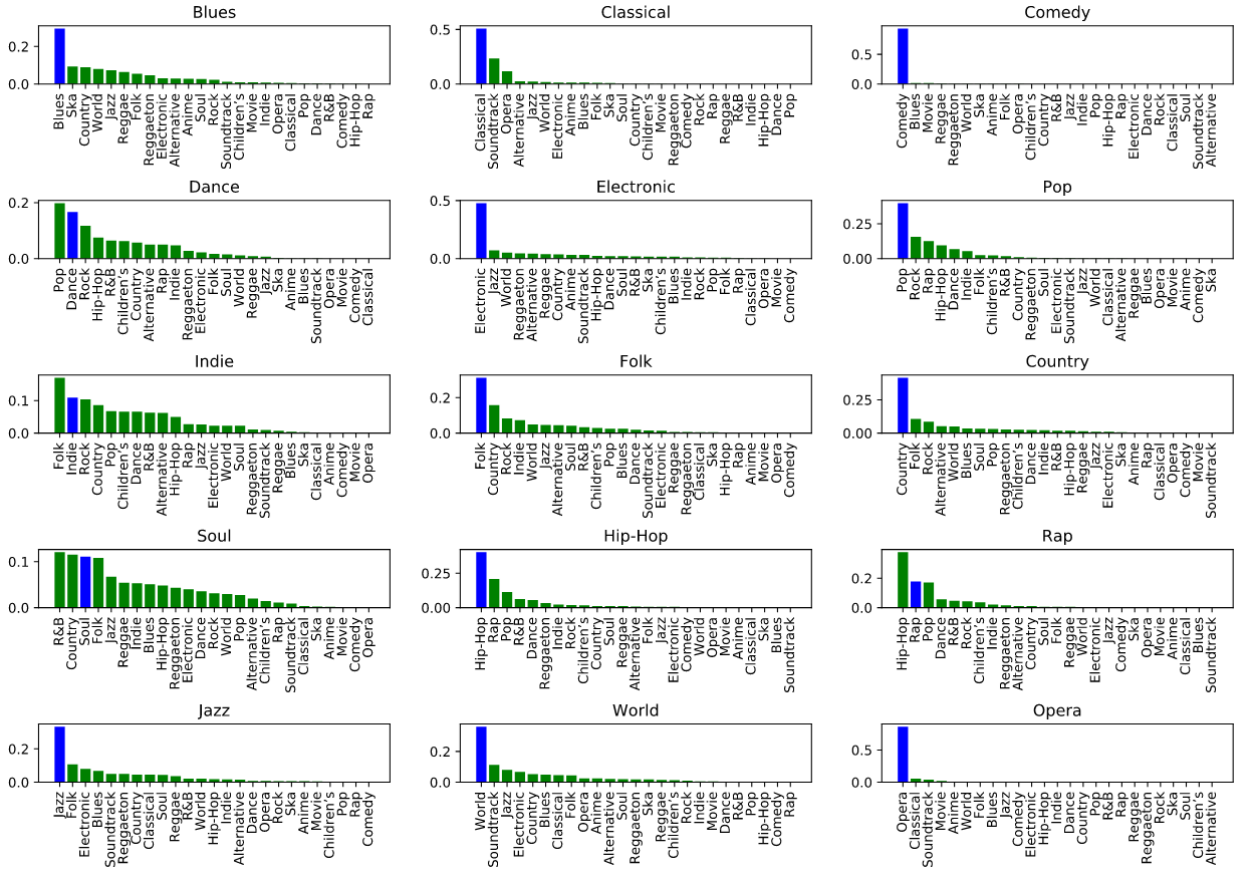

Figure 9: Distribution of correct and incorrect positive predictions pr genre

3.2 Neural Network

The second model chosen for genre-prediction is the fully connected neural network, in order to have something to compare the KNeighbor classifier to. From previous iterations, no other classifier from the scikit learn "cheat-sheet"[4] came close to the KNeighbor classifier in terms of precision.

Now, like previously, the pipeline should be adopted to the NN, automating the process of the finding the optimal model hyperparameters. This task was found to be harder than expected, as the core idea was to implement scikit learns search algorithms (focusing on a gridsearch) with the keras NN api, which caused some issues. In the end, the group was unsucessful in implementing such a feature for the NN, as both issues on the GPU cluster (gridsearch memory issues? Scikit learn not supporting GPU usage, while the underlying

[4]https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

keras model attempts to use it?) and memory bounds on the groups laptop caused the idea to be unfeasible. Given that the group does not have an infinite amount of time to pour into the project, it was decided to perform a "manual" gridsearch, checking each hyperparameter by itself. This obviously goes against the core idea of the pipeline automating the process, but, in order to have something to hold the KNeighbor regressor up against, a NN was required.

Given the implications from above, the work on the NN quickly becomes tiresome, and as such, the pipeline must be said to be non-ideal.

3.2.1 Hyperparameter search

Before the search began, it was decided to use a NN with only one hidden layer. Afterwards, like stated, a manual search has been performed attempting to identify the optimal; `batch_size`, number of `neurons`, hidden layer `activation` function, kernel initializer (`init_mode`) and `optimizer`:

```
batch_size = [10, 20, 40, 60, 80, 100, 200, 400, 600, 800, 1000]
   neurons = [1, 2, 3, 4, 5 , 6 , 7, 8, 9, 10, 11, 12, 13, 14, 15,
              16, 17, 18, 19, 20, 21, 22, 23, 25, 26, 50, 100, 150]
activation = ['softmax', 'softplus', 'softsign',
              'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']
 init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero',
              'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']
 optimizer = [SGD(lr=0.1), RMSprop(lr=0.1), Adagrad(lr=0.1),
              Adadelta(lr=0.1), Adam(lr=0.1), Adamax(lr=0.1), Nadam(lr=0.1)]
```

From this, a NN model with the following hyperparameters was found to perform the best:

```
def createModel():
    model = Sequential()
    model.add(Dense(input_dim=15, units=50, activation="sigmoid",
    kernel_initializer="he_uniform"))
    model.add(Dense(units=25, activation="sigmoid"))
    model.compile(loss='categorical_crossentropy',
                  optimizer=Adam(lr=0.1),
                  metrics=['acc', f1_m, precision_m, recall_m])
    return model
```

3.2.2 Results

Initializing a model with the found hyperparameters and fitting it to the training data, has yielded the results shown in Table 4:

3.2.2.1 Performance metrics The results of the NN has been processed to match the shape used for Scikit learn's scoring metrics. The NN output differs from the KNeighbor classifier in that the output is still binarized, and each label contains the weight assigned by the model (How sure the model is of it's guess), similar to the output structure of sklearns `predict_proba()`. However, as the previous KNN results were generated from the regular `predict()`, the NN output has been processed to use one hot encoded outputs, only flagging the label which the model had assigned the heighest weight. The results is thus directly comparable to that of the KNeighbor classifier:

Table 4: Classification results (summary)

	Precision	Recall	F1-score	Support
Accuracy			0.40	NaN
Macro avg.	0.41	0.40	0.39	NaN
Weighted avg.	0.40	0.40	0.39	NaN

3.2.2.2 Distribution of predictions pr. class The models performance is examined further by plotting how the models predictions for the test data is distributed for each genre (true and false positives).

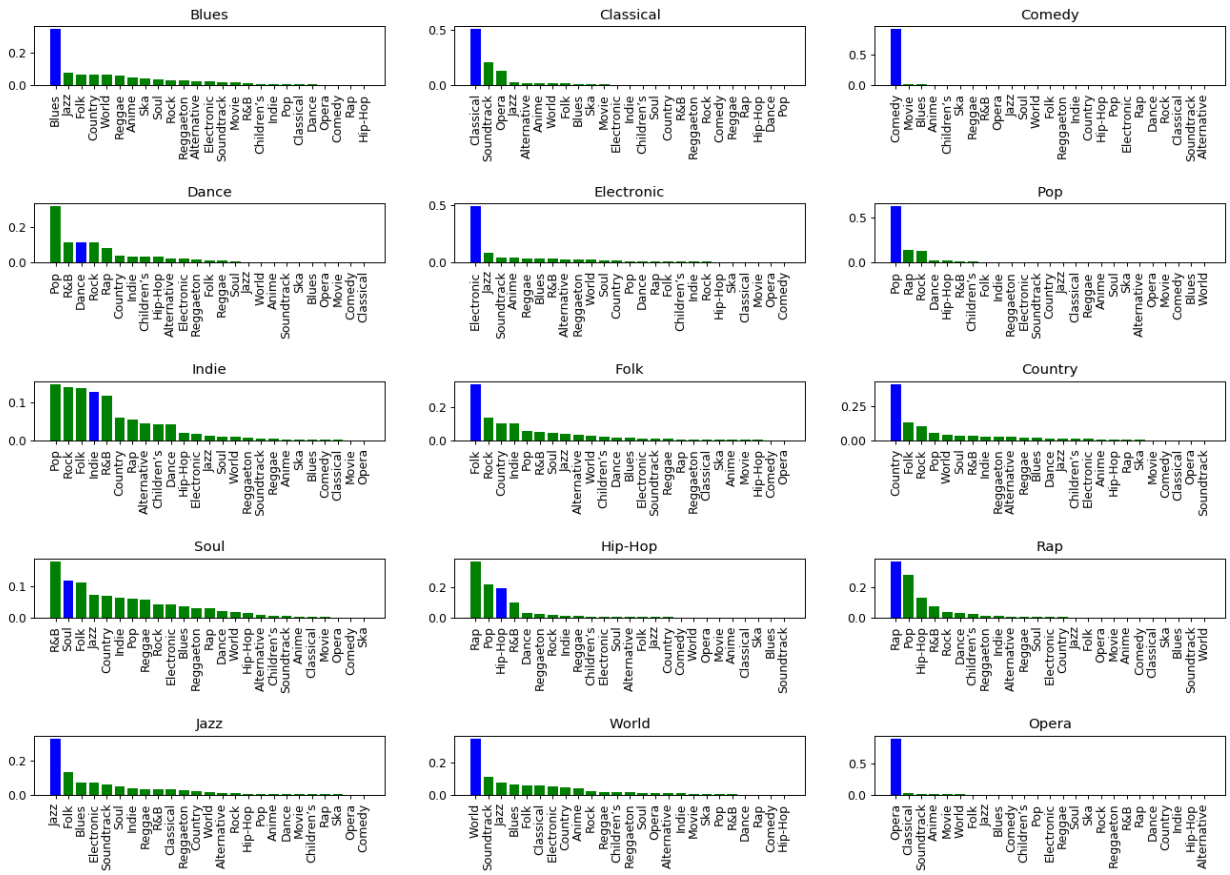

Figure 10: Distribution of correct and incorrect positive predictions pr genre

4 The popularity estimator

For the popularity estimation, a simple randomized search was performed using the "scikit learn cheatsheet"[5]. Here, the "Stochastic gradient descent" algorithm was found to be the best estimator. The search evaluation was based on the "*R2*" score, of which is an indicator of the "goodness of fit" for the regressor.

4.1 Algorithm Selection

Though it was not obvious to us during the data visualizion it seems that a Linear Regression performs best in terms of predicting poularity values. In this case the Stochastic Gradient Descent Regressor is a recommended choice for a model as our data set contains above 10.000 samples.[6]

4.2 Data processing, structure and hyperparameter search

The input for the regression model is very similar to that of the classification problem, so for more detail on the preprocessing, please see Section 3.0.2 The key difference being that genre values are not isolated as the y-values instead in make up part of the feature set. Instead the popularity values are selected as the y-values

[5]https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
[6]https://scikit-learn.org/stable/modules/sgd.html#regression

for this set. These values are represented as a float value between 0 and 1. - Similar to the pipeline for the classification model, the data set is split into two partitions for training and validation.

In the gridsearch to identify the best model, different algorithms were selected and tested with individual sets of hyperparameters. The tested models include: Ridge, Lasso, SGDRegressor (Stochastic Gradient Descent Regressor), SVR (Support Vector Machine for Regression). In the end it was the SGDRegressor which performed best, with the following options included in the hyperparameter gridsearch for this model:

```
         loss = ['squared_loss', 'huber', 'epsilon_insensitive',
                 'squared_epsilon_insensitive']
      penalty = ['none', 'l2', 'l1', 'elasticnet']
        alpha = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.0006, 0.0007, 0.0008,
                 0.0009, 0.001, 0.0011, 0.0012, 0.0013, 0.0014, 0.0015]
fit_intercept = [True, False]
```

It should be stated that since our regressor boiled down to "*how close can we get to the true popularity*" value, the results are very limited in regards to presentation, as the easiest way to measure the performance is by the mean deviation, either squared or absolute.

4.3 Results

The results of the SGD regressor is measured by it's MSE and MAE on the test set, of which the following scores was found using scikit learns build in MSE and MAE functions:

Method	Score
MAE:	~0.075
MSE:	~0.010

Since the popularity feature has already been scaled to a value between 0 - 1, the score of the MAE translated directly to er percentage score; 7,5% deviation from the true popularity.

5 Discussion and Conclusion

In this section, project and its results are discussed and reflected upon. The classification model/pipeline and regression model/pipeline will be discussed seperately.

5.1 A note on underfitting and overfitting

Before discussing the results of the models chosen in this project, one should stop and ask whether the models are any good at all. Do they have the means to perform the task? These questions relate to whether the models are underfitted or overfitted - and how can one be sure they are either/neither?

If we are underfitting, it can be hard to tell whether a model has been so. The issue here is that there is a risk that the chosen model does not have a high enough complexity to make a satisfactory "goodness of fit" for the dataset. The issue here lies mainly with bias. Increasing the number of features alleviates this problem as it icreases the complexity. Likewise, overfitting is when the model has too high of a complexity for the dataset. So, there seems to be a midpoint that is "just-right" for model complexity, how does one determine this?

For underfitting, although not done explicitly for this project, the *"bias-variance-tradeoff"*[7] could be used. When performing our searches, it is key that we are not only looking at an accuracy score, as this has no relation to under/overfitting. Instead, by using the F1 score (classification), and R2 (regression), we are able to somewhat measure the goodness of fit while also tuning the penalizing factors (minimizing variance = reduces overfitting). Furthermore, by using cross-validation we also introduce an early way to identify the

[7] https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff#k-nearest_neighbors

overfitting issue as this is an indicator of how well the model will react to unseen data. Lastly, if the models training scores and test scores varies by a big amount, this might be an indicator of overfitting aswell.

Feature reduction (or the posibility of it) could be done in order to minimize bias, but has unfortunately not been implemented in the pipeline.

5.2 Classification

For the classification models, the group is very satisfied with the results, even though a f1 score of 37% and 40% does not seem like much at first glance. If compared to the random case, or the case of dummy identifier, the probability of predicting correcly is only $\frac{1}{25}$. Furthermore, the plots of the correct guesses pr. genre, supports the theory that was discussed during the PCA analysis - that the some genres, comedy especially, was easier to predict correctly than those more closely grouped together. Another aspect to consider is that genres are inheretently subjective (earlier in the report the group stated that *"Die Zauberflöte"* was opera, but this is just the groups opinion), and as such, their labels do not give reason to hard divisions between every genre. Assuming the beforementionend is true, no classifier should be able to predict genres with certainty, as the truth would be that the same songs could be classified as more than one genre.

Looking at the dataset with the intention of genre classification, it must be noted that a lot of features were found to be uncorrelated to the desired prediction. This was mainly from the fact that the dataset in reality contained a lot of labelled features - some of which were Unique (and therefore had to be removed), and some which had almost 0 correlation with the genres (e.g *"Mode"*).

5.3 Regression

As one of the original ideas with the dataset was to only produce a regressor able to predict the popularity of songs, the project has definitely given the group some experience in regards to the reality of the ML world. Even though no correlation could immeadiately be made by the group in the preprocessing between popularity and other features, the results after the search was no less than suprising. A mean error of around 7,5% was much less than expected. One theory here, is that removing the samples with a popularity of 0, increased the correlation in some way not obvious to the group.

The low expectations were primarily formed by the initial visualizations and correlations, which only compared a few features/dimensions at a time. While the plotting (and our intuition) is limited to 2-3 dimensions at a time, computers do not have this limitation, as demonstrated by the regressor.

5.4 General for the project

Looking at the roadmap of getting to the models, the group must conclude that the pipeline is not as automated as first hoped. Likewise, the data processing took a lot longer than originally anticipated, but in turn, yielded a great deal more knowledge of the dataset. Seeing as this is the first time the group is attempting to implement an *"End-to-end ML"* project, it must be concluded that a lot of first time errors was made - e.g trying to include that dataprocessing part of the pipeline with the visualization just made the code bloated and unreadable. Although not entirely succesfull in the implementation of an automated pipeline, the group is satisfied with the results.

It should be noted that for the dataset, a lot of samples were removed when sorting out the samples which had a popularity rating of 0. The effect of this is not entirely clear, as one could guess that the (espicially the regressor) would have issues estimating unpopular songs. Likewise, it could be expected that by excluding these samples, we are actually hindering the effectiveness of the classifier, as popularity was seen to be somewhat correlated with the genre.

The actual effect of many of the groups descisions to remove certain samples and features, remain unknown at this point. If time permitted, it would have been informative to re-train the models with differently preprocessed data, to determine if these descisions actually had the desired effect of improving the performance of the models.

The project could definitely be further developed as it holds some aspects which has not been explored by the group in this project. Since the data is gathered from the spotify API, the pipeline would be expected to greatly benefit from an added part that would gather more songs. This would lend itself to the use case that when new songs were added to the spotify platform, the models could be used to predict either their popularity or genre.

Another aspect is the underlying *"Timbre vectors"* of the data. From Spotify's API documentation it seems that the features worked upon in this project is extracted from these, which in essence holds the digital signal processing made on the songs when they are added to spotify. This however would require much more work than done for this project, which is why the group early on decided not to explore this aspect, although interresting.