Maschinelles Lernen für Physiker

The Genre Factor

Henry Krämerkämper

henry.kraemerkaemper@tu-dortmund.de

July 29, 2023

TU Dortmund – Fakultät Physik

Contents

1 Introduction

The task of classifying the genre of a song is common in the digital music industry. Most services offering music listening present some information about each song, which often includes the genre. Some services might even use the information to suggest other songs to listen to, which requires accurate information about the genre (or the genres) that a song belongs to. Retrieving this information is not easy, since there are no clear definitions of a genres attributes. Additionally, most songs do not belong to only one genre. The genre itself might change over time as well, which further complicates the problem. While the classification task might be technically solvable by humans, it remains a non-trivial endeavour due to its inherent complexity. Given the immense size of most music libraries, a manual approach to classification becomes highly impractical, necessitating alternative, more efficient solutions.

With these factors in mind, the task is evidently predisposed to a solution via a machine learning approach. As such, this strategy has become prevalent in addressing this problem, with a plethora of diverse methods having been explored to date (see, for example [1]). In this study, we attempt to classify music genres using a dense neural network. For this, we use a dataset sourced from the website Kaggle [2] containing songs and their attributes taken from the services YouTube [3] and Spotify [4]. We compare the neural network with two other, less sophisticated machine learning techniques, namely support vector machines [5] and the k-nearest-neighbours-approach [6], to establish a baseline. We aim to find out whether employing more complex and labour-intensive techniques result in an improvement in the face of the limited information contained in the dataset.

The report is structured as follows; first, the utilized dataset and the applied preprocessing is described in detail. Subsequently, the architecture of the dense neural network is laid out and the results are presented. These findings are then compared to the results of the alternative approaches. Finally, we draw a conclusion based on our analysis.

2 The Utilized Dataset

2.1 Sourcing the Data

The dataset used in this project [7] contains 26 attributes about 18862 songs from 2079 unique artists. However, the genre of the song is not included in the dataset; we query wikidata for the corresponding genre of each song, using the python package `pywikibot` [8]. An example entry of the resulting dataset at this stage is shown in table 1. We do not keep all of these attributes; since the architecture of our neural network does not feature text embedding, the attributes `Track`, `Album`, `Title`, `Channel` and `Description` are dropped. The features `Uri` and `Url_youtube` are most likely random and do not contain useful information for our models to learn, therefore these are not used as well.

Feature	Example	Feature	Example
Artist	Gorillaz	Valence	0.772
Url_spotify	`https://open.spotify...`	Tempo	138.559
Track	Feel Good Inc.	Duration_ms	222640.0
Album	Demon Days	Url_youtube	`https://www.youtube...`
Album_type	album	Title	Gorillaz - Feel Good Inc. (Official...
Uri	spotify:track:0d28khcov6AiegS...	Channel	Gorillaz
Danceability	0.818	Views	693555221.0
Energy	0.705	Likes	6220896.0
Key	6.0	Comments	169907.0
Loudness	-6.679	Description	Official HD Video for Gorillaz'...
Speechiness	0.177	Licensed	True
Acousticness	0.00836	official_video	True
Instrumentalness	0.00233	Stream	1040234854.0
Liveness	0.613	Genre	Hip Hop

Table 1: The attributes contained in the dataset, shown for an example song.

2.2 Preprocessing

Subsequently, the dataset is cleaned; missing or erroneous values in numerical features are substituted by a value derived by a k-nearest-neighbours-approach. We implement this by using the `SimpleImputer` from the `Scikit-Learn` package [9]. The cardinal features are then scaled to a range of $[-1, 1]$ and transformed to follow a normal distribution to improve numerical stability and the convergence speed as well as preventing the 'Exploding/Vanishing-Gradient' problem [10]. These steps are implemented using `QuantileTransformer` and `MinMaxScaler` from `Scikit-Learn` [9]. The aforementioned wikidata query results in 397 different genres. As these are highly specific, these categories are consolidated into 26 broader genres to achieve a more streamlined dataset and increase the sample size per class. For example, the genre 'latin' contains 'salsa', 'bossa nova', 'samba' among others. Given the constraints of our datasets size, we only keep the top 6 genres with the most songs to further increase the sample size. The remaining genres are hip hop, rock, pop, electronic, metal and classic. The datasets class imbalance can be seen in figure 1.

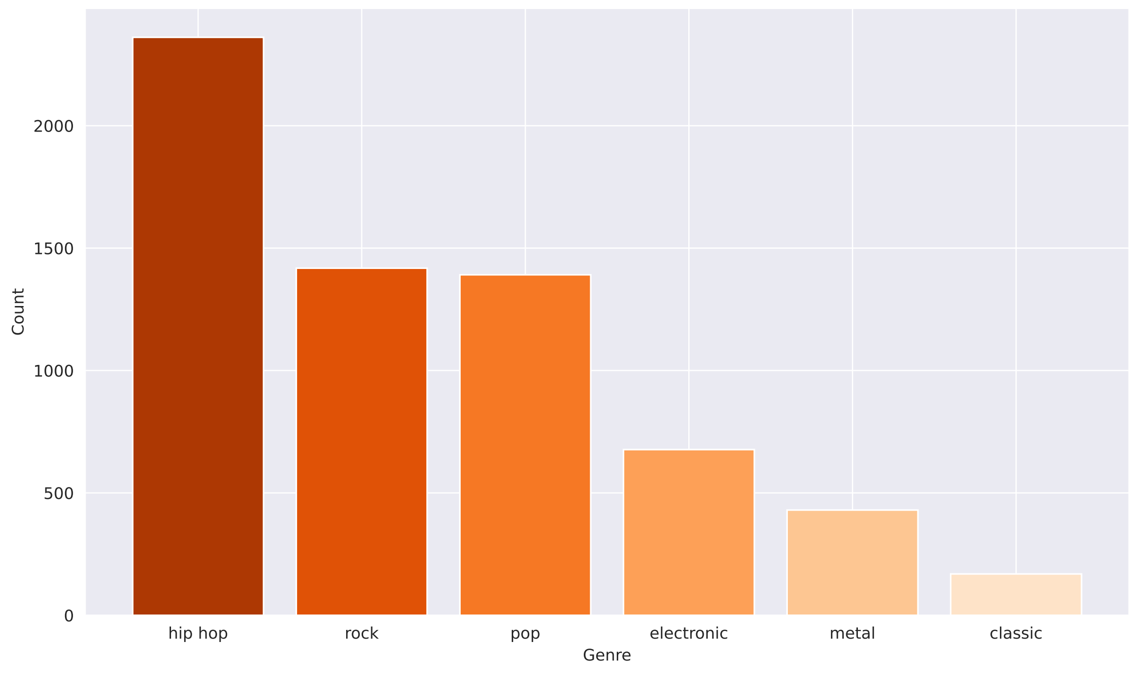

Figure 1: Class imbalance of the resulting dataset.

Upon application of the preprocessing steps, the dataset is reduced to 6446 entries. Given the high class imbalance in our dataset, the use of weights could potentially help mitigate this imbalance. However, in our specific case, we found that weighting did not improve model performance. Therefore, we decided not to use weighting in our final modelling approach. To enhance the sample size for the under-represented classes, synthesizing new entries could be a potential strategy for future work. After this step, the dataset is divided into a training-, test- and validation-set, with each containing 80 %, 16 % and 4 % of the data respectively. In this process, we use stratified sampling to ensure that the distribution of classes in each subset mirror the distribution in the original set.

3 The Neural Network Model

3.1 Architecture and Implementation

The sequential dense neural network employed to solve this task is composed of two hidden layers. Increasing the layer count further proved to result in no additional performance gains. The same holds true for using more neurons per layer, suggesting both strategies may lead to over-fitting. To combat over-fitting, we use dropouts between each dense layer as well as learning rate decay and early stopping. The later also reduces the needed computation time and annealing the learning rate increases the convergence speed. The output layer uses the activation function softmax, as it can be used to return a probability distribution for the classes. To measure the performance and loss, we use the accuracy and the categorical crossentropy respectively. The network architecture and the output shape of each layer can be seen in figure 2.

Parameter	Value
Activation Function	LeakyReLU
Batch Size	512
Dropout Rate	0.4
Early Stopping Patience	20
Epochs	60
Neurons Layer (1)	256
Neurons Layer (2)	512
Neurons Layer (3)	512
Number of hidden Layers	2
Optimizer	Nadam
Reduce LR Factor	0.1
Min. LR Rate	0.0
Reduce LR Patience	5
Output Function	Softmax

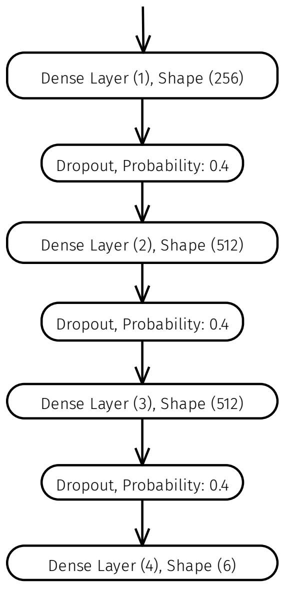
Dense Layer (1), Shape (256)
Dropout, Probability: 0.4
Dense Layer (2), Shape (512)
Dropout, Probability: 0.4
Dense Layer (3), Shape (512)
Dropout, Probability: 0.4
Dense Layer (4), Shape (6)

Figure 2: Overview of the neural networks' layers and parameters.

We implement the model using `tensorflow` [11] and `keras` [12].

3.2 Hyperparameter Optimization

The exact parameters given in figure 2 are the result of performing a hyperparameter optimization. For this, we carry out a grid search [13], implemented with `GridSearchCV` from `Scikit-Learn` [9], as it supports parallel computing. We use 3-fold cross-validation to mitigate overfitting and to make efficient use of the finite data. To be able to optimize a model created by `keras` with `Scikit-Learn` functions, we use `SciKeras` [14]. As we were limited in time and computational resources, only a subset of the parameter space has been explored; for most parameters, we tried between two and six values, but not all combinations of them. Still, 2033 models were tested, with 2 resulting in the highest performance. All tested models and their performance can be seen here[1]. Selecting the best model is a trade-off between accuracy on the

[1]`https://github.com/Henry-Kr15/TheGenreFactor/blob/main/HPO-data/HPO_combined.csv`

training set and overfitting, which results in lower performance on the validation set. Because of the limited size of the validation set (as it comprises of only $4\,\%$ of the data), relying on validation accuracy alone could prove inconsistent. We try to manually alter this trade-off to achieve higher generalisation than otherwise possible. For this, we combine the validation accuracy and training accuracy into a new metric which we call delta accuracy by taking the difference of the two. Then we calculate a score based on the delta and validation accuracy as follows:

$$\text{score} = \frac{1}{2} \cdot (w_1 \cdot (1 - \text{acc}_{\text{delta}}) + w_2 \cdot \text{acc}_{\text{val}}). \qquad (1)$$

The score minimizes $\text{acc}_{\text{delta}}$ whilst maximizing acc_{val}. Using the weights w_1 and w_2, the importance of each value can be specified. All models can be seen in figure 3. With $w_1 = 0.5$ and $w_2 = 1$ the model marked red has the best score.

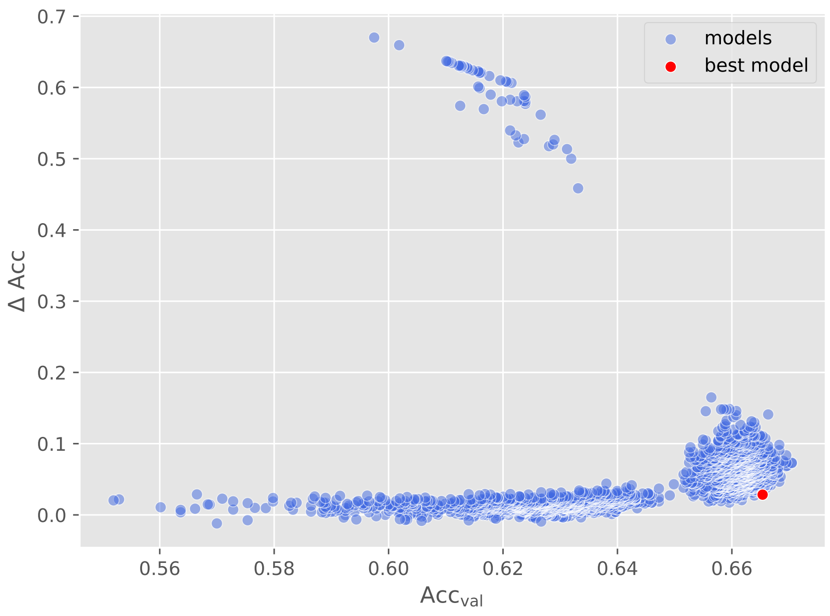

Figure 3: Comparison of training and delta accuracy for all models.

3.3 Performance and Results

The model achieves $64.43\,\%$ accuracy on the test data set. Accuracy and loss are shown in figure 4 and figure 5 respectively.

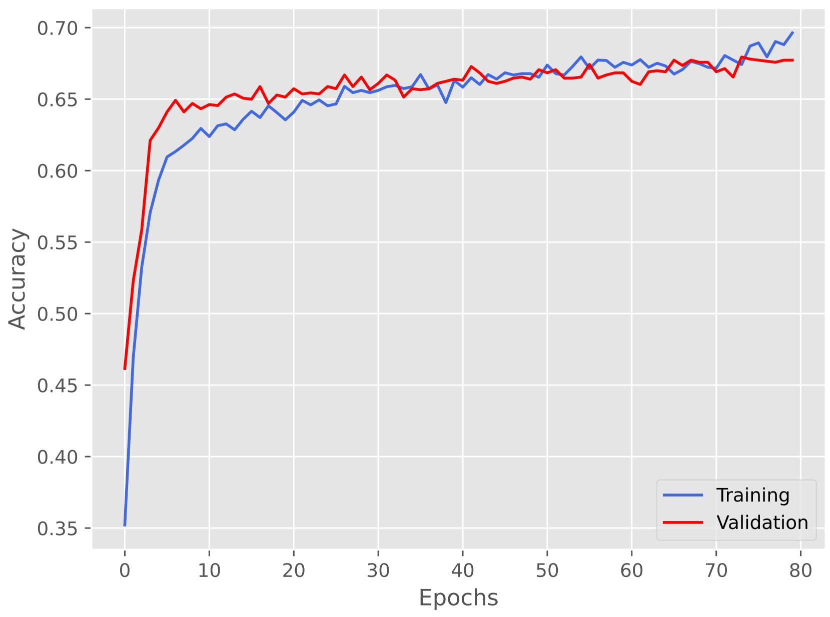

Figure 4: Plot of the accuracy of the neural network on the training and validation dataset.

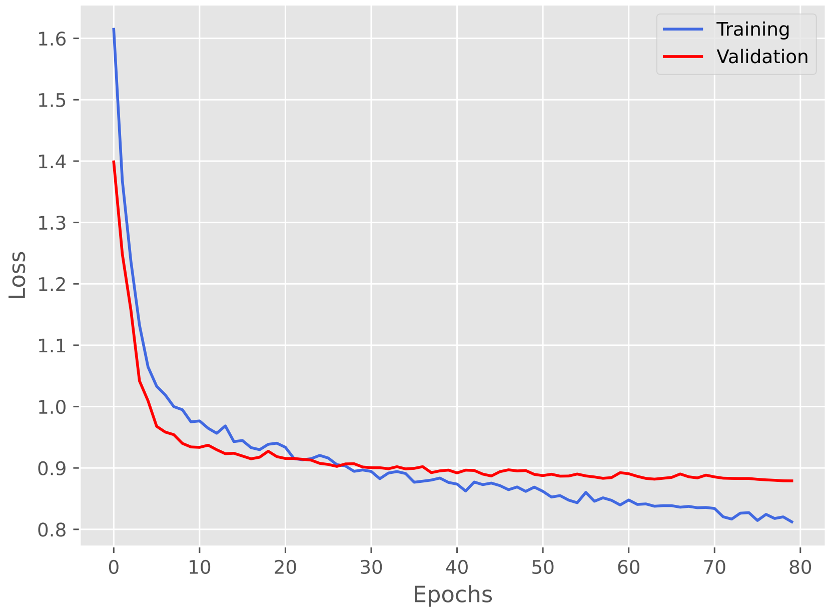

Figure 5: Plot of the loss curve of the neural network on the training and validation dataset.

The accuracy curve shows a satisfactory trend, indicating that the model is performing well on both the training and validation datasets. The loss curve however shows a slight indication of overfitting after approximately 40 epochs. The confusion matrix can be seen in figure 6.

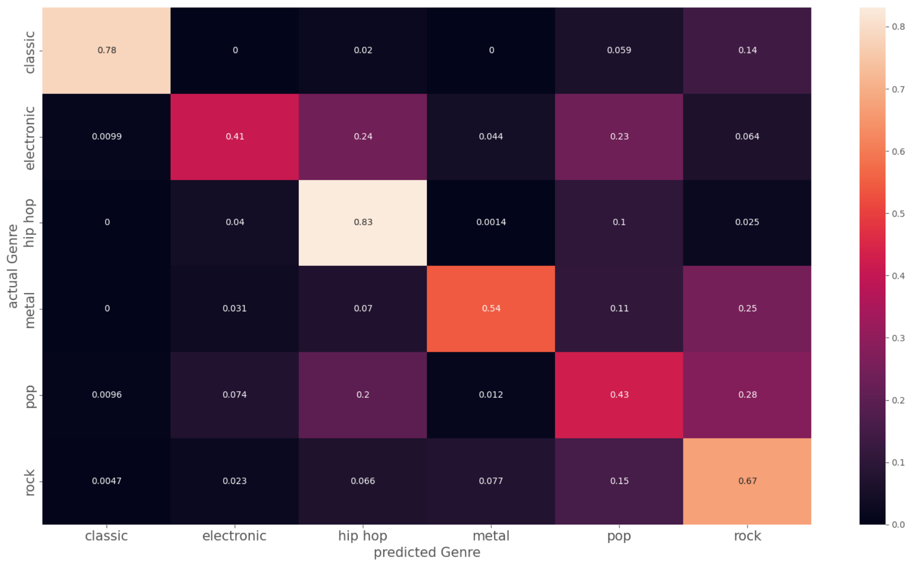

Figure 6: Confusion matrix of the neural network on the test dataset.

The confusion matrix shows, that although the models performance is for the most part adequate, problems arise for genres which consist of a broader spectrum with less clear defined characteristics, such as pop or electronic. The precision-recall curve, depicted in figure 7, shows a comparable result.

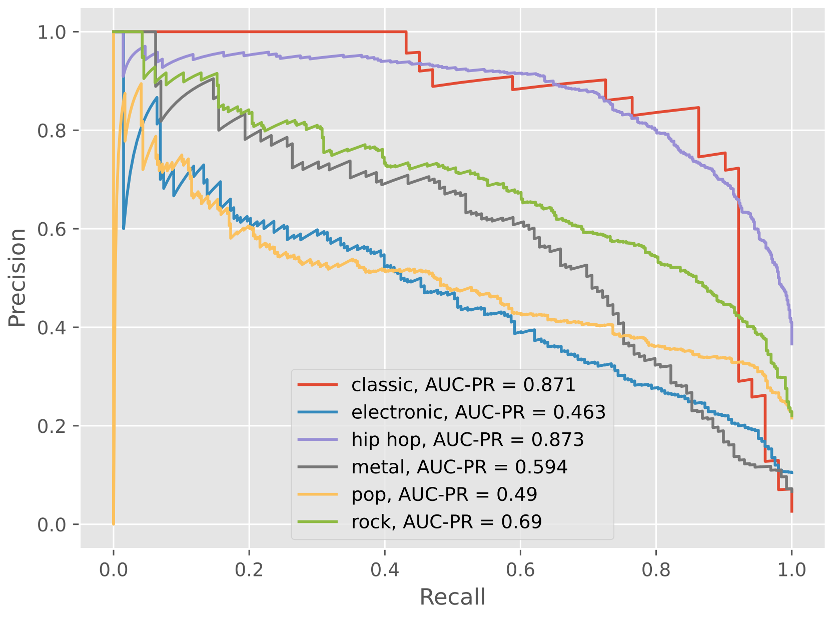

Figure 7: Precision-recall curve for each genre on the test dataset.

Differentiating rock from metal and pop seems to be more difficult as well; this conforms to the areas where humans would also struggle. Astonishingly, the neural network performs the best for classic, which has the lowest sample size of any genre in the dataset. Classic and hip hop seem to get differentiated well, while pop and electronic show suboptimal performance.

4 Alternative Approaches to the Problem

4.1 Exploration of alternative Methods: K-Nearest Neighbours

In addition to the neural network, we also explored other machine learning methods for comparison. One such method is the K-Nearest Neighbors (KNN) algorithm, a simpler method which can be used for classification as well. This section provides an overview of our implementation and evaluation of the KNN approach. The `Scikit-Learn`-package [9] provides an implementation of a Knn algorithm, the `KNeighborsClassifier`. We use the 12 nearest neighbours for the classification, as this results in the highest performance. The model achieves an accuracy of $60.62\,\%$ on the test dataset. The resulting precision-recall curves for each genre are depicted in figure 8.

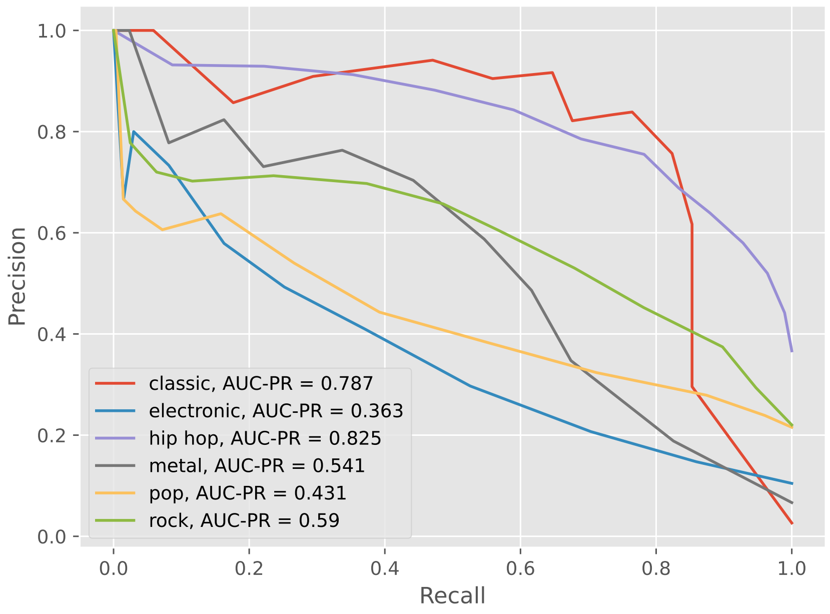

Figure 8: Precison-Recall curves of the Knn-algorithm for all genres.

The precision-recall curves show the limitations of the Knn-algorithm regarding more complex problems as the curves are significantly worse than the ones from the neural network. Although the accuracy is not far behind compared to the neural network (as they are in 10 percentage points range of each other) the differences in handling complexity between the models become apparent. The confusion matrix, depicted in figure 9, further highlights these differences.

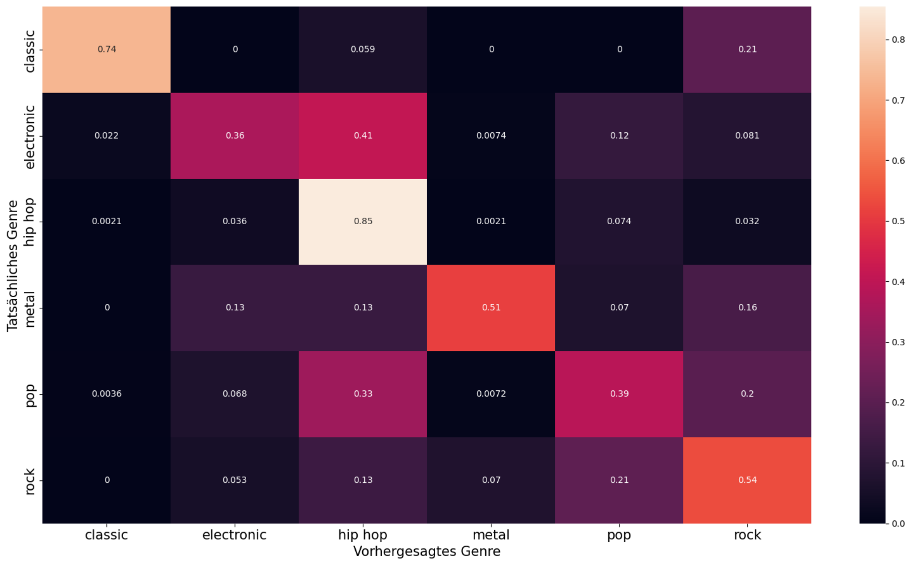

Figure 9: Confusion matrix of the Knn-algorithm on the test dataset.

The values on the diagonal show adequate performance, but the spread is substantially higher compared to the neural network.

4.2 Exploration of alternative Methods: Support Vector Machines

Another alternative method we investigated is the Support Vector Machine (SVM) approach. Through combining multiple SVMs, using them for classification tasks with multiple classes is possible. This section discusses our implementation and performance of the SVM method in the context of our project. In standard form, support vector machines can only be used for binary classification. Here, we use an ’One-vs-One’ approach: for every pair of genres, one support vector machine is trained. The multiclass classification is then performed by majority vote of the SVMs. The `Skicit-Learn`-package [9] provides the `svm.SVC` method, which we use with the radial basis function as a kernel function. This model achieves an accuracy of $63.88\,\%$ on the test dataset. The resulting precision-recall curves for each class are depicted in figure 10.

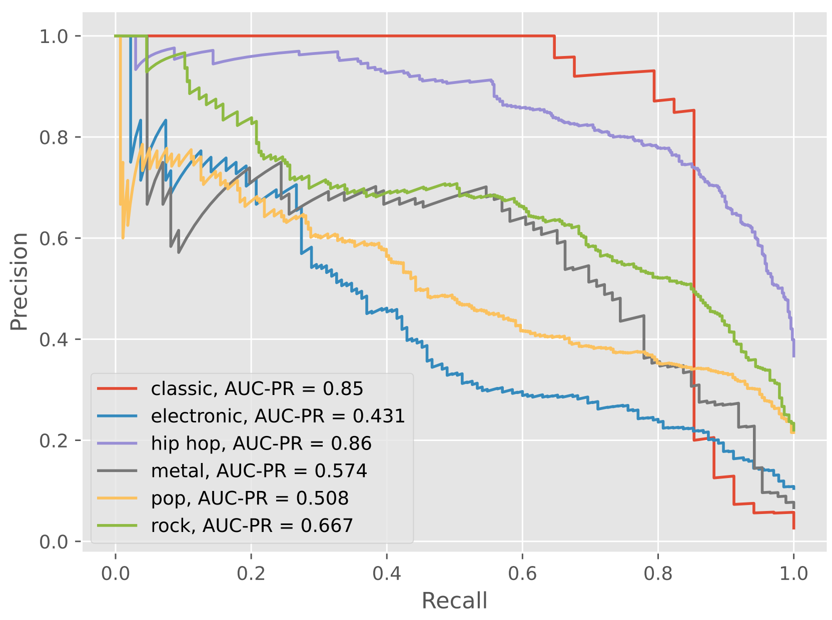

Figure 10: Precision-recall curves for all classes of the SVMs on the test dataset.

The precision-recall curves display a better performance than the Knn-algorithm but are inferior to the neural network, which is also resembled in their differences in their accuracy score. The confusion matrix is depicted in figure 11.

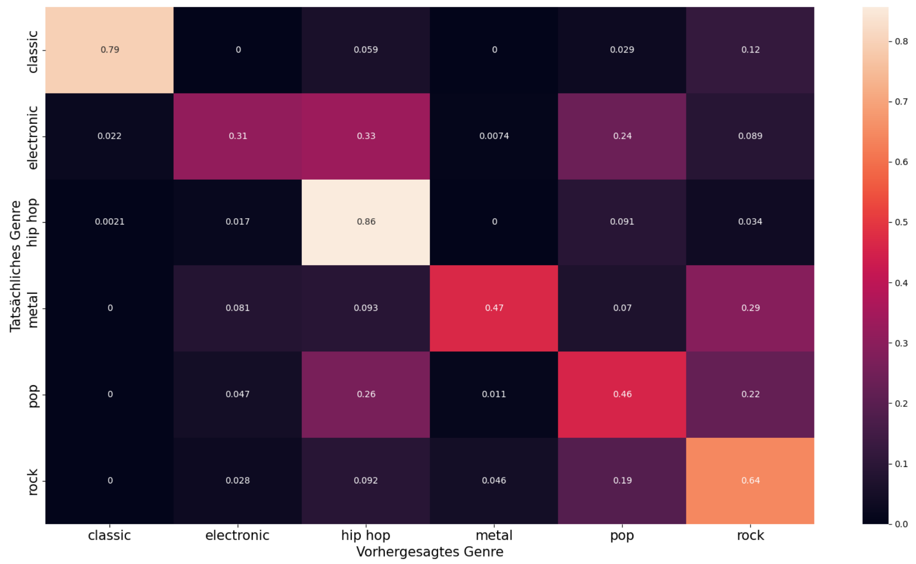

Figure 11: Confusion matrix of the SVMs on the test dataset.

5 Discussion and Insights

References

[1] Zhouyu Fu et al. "A survey of audio-based music classification and annotation". English. In: *IEEE Transactions on Multimedia* 13.2 (2011), pp. 303–319. ISSN: 1520-9210. DOI: 10.1109/TMM.2010.2098858.

[2] D. Sculley. *Kaggle: Level up with the largest AI and ML community*. URL: https://www.kaggle.com/ (visited on 07/26/2023).

[3] *YouTube*. URL: https://www.youtube.com/ (visited on 07/26/2023).

[4] *Spotify*. URL: https://open.spotify.com/ (visited on 07/26/2023).

[5] Corinna Cortes and Vladimir Naumovich Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995), pp. 273–297.

[6] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[7] Salvatore Rastelli, Marco Guarisco, and Marco Sallustio. *Spotify and Youtube: Statistics for the Top 10 songs of various spotify artists and their youtube-video*. URL: https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube (visited on 07/26/2023).

[8] *Pywikibot: Python MediaWiki Bot Framework*. URL: https://pypi.org/project/pywikibot/ (visited on 07/26/2023).

[9] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[10] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras and Tensorflow*. Third Edition. O'Reilly Media, Inc., 2022.

[11] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[12] François Chollet et al. *Keras*. https://keras.io. 2015.

[13] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[14] Adrian Garcia Badaracco. *Scikeras: Scikit-Learn compatible wrappers for Keras Models*. URL: https://github.com/adriangb/scikeras.