Classical classification vs. Deep Learning for Genre Recognition

Abstract—**The cutting edge research in genre recognition seems to heavily focus on neural networks and their variations. This is understandable since the genre of a musical piece is defined by features that are hard to define and codify, and therefore the success of deep learning is hardly surprising. However, deep learning is not an magical recipe for success, and often carefully tuned ”classical” methods can achieve similar, or even superior performance. Therefore we compare the performance of a neural network design found in research literature, with more classical methods such as classification by support vector machine and the Bayes classifier. In the end, a support vector machine combined with dimensionality reduction resulted in the best accuracy.**

I. Introduction

With the age of streaming music and video services, consumers have easy access to a larger and more varied amount of content than ever before. Faced with and unprecedented selection, new approaches to content discovery have developed.

An important feature of this is automatic recommendation of new content based on the personalized preferences of the individual consumer. Features such as Youtube’s *watch next*-list and Spotify’s recommended playlists allow users to find content similar to what they are used to listening. Better content recommendation should result in users consuming more content which is quite obviously desirable for any service provider.

Traditionally these recommendations have relied on collaborative filtering, with the idea to determine users’ preferences from historical usage data[1]. Videos or music that are listened to by the same group of users are likely to be similar and therefore can be recommended to a user that has already consumed a part of them.

The problem with the collaborative filtering approach is that it cannot recomment new songs or videos since there is no data on them. A new song from a user’s favourite artist is an obvious recommendation, but will not be picked up until there is sufficient data on the song to make the recommendation. From these problem we can see that if there was a way to detect similar songs or videos based on the content themselves, some of these problems could be remedied.

An interesting aspect of the genre recommendation is extracting usable features from the actual audio data. An average song sampled at the industry standard 44.1 kHz results in a data vector with length of nearly 10 million, far too much to be usable for any machine learning method. Transforming this raw data into features that represent the interesting features of the music require multidisciplinary knowledge of, amongst other things, audio engineering, music theory and the physiology of human hearing. For this project, the feature extraction has been performed in advance so it will be only discussed briefly, but the literature on the subject is broad and could warrant several reports in itself.

The question that this project aims to answer is whether these features contain enough information to accurately identify the genre of the song. One would anticipate that this task is difficult for a machine since even for humans not especially versed in music theory it can be hard to describe the features that give a song its genre. Some classes should have featues that make them intrinsically easier to distinguish from others, such as the rhythmic elements and intensity of electronic music. However for other classes, such as Blues and Jazz, are likely to be harder to distinguish from each other.

Also, the secondary question is whether classical methods can match neural networks in this task. Neural networks excel in identifying complex patterns from data and have much of the state of the art research in genre recommendation relies on them[3][4][5]. However, they are not magic key to success, and often finely tuned classical methods can also perfrom well[6].

II. Data analysis

As is always the case with statistical analysis and machine learning, the first thing to do is to investigate the data. Hopefully this will give us insights into the phenomenon we are modeling, and the distribution of the data that we have available

The prior distribution of the outputs $\mathbf{y}$ heavily influence the posterior probabilities of the predictions. Figure 1 shows the distribution of the labels for the training data set. We can see that the label 1 is by far the most common, and in fact almost exactly half of the training examples belong to that label. This skewed distribution means that the validation set used for evaluating the classifiers has to be large enough that there are sufficiently

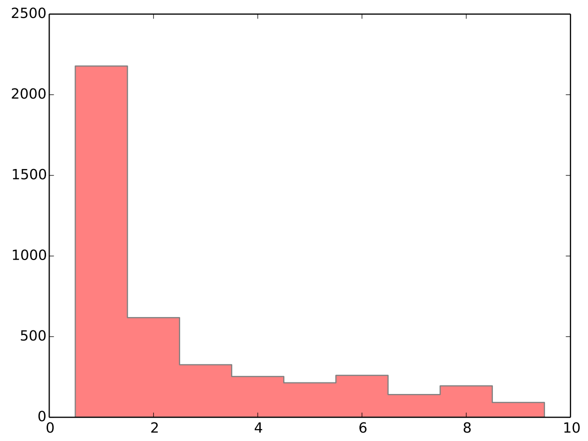

Fig. 1. Distribution of labels in training dataset

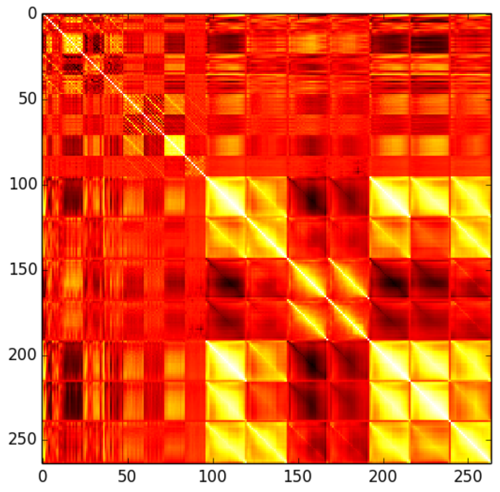

Fig. 2. Correlation between columns

many examples or each class label in the validation dataset. For example a validation set of size 50 would be appropriate in some instances, but is far from sufficient in this case.

The feature vectors are of length 264 for each sample divided into three sections. Values $1 - 48$ contain information about the *Mel Frequency Cepstral Coefficients* (MFCC), values $49 - 96$ contain information about the Chroma, and finally values $97 - 264$ contain information about the Rhythm patterns.

The MFCCs roughly describe the distribution of high and low frequencies in the music, adjusted to the sensitivity of the human ear[7]. Calculating the MFCC roughly means splitting the signal into short frames, calculating the frequency distribution of the signal in each frame, computing the Mel-spaced filterbanks by multiplying the frequency domain signal by specially designed filters, and finally taking the log and computing the discrete cosine transform of the resulting energies. The filters are designed so that they are more densely spaced at low frequencies since human ears are better able to distinguish between frequencies at the lower end of the spectrum. The log step is is performed due to the fact that humans do not perceive loudness on a linear scale, but rather at a logarithmic one. Doubling the perceived loudness requires inputting eight times as much energy. In processing the signal data for this project, 12 filterbands have been used, and for each of them 4 statistics have been calculated: the mean, standard deviation, max and min.

The chroma features describe the distribution of different musical notes in the audio. In western music, an octave is divided into 12 semitones, each a constant mutiple apart from each other. Two notes an octave apart mean that one of them will have exactly double the frequency of the other. To a person the same note an octave apart will sound similar, and therefore the distribution of the notes will give information about the song[8]. A song in a certain key will only use a subset of the notes, and the most common key for a particular genre will depend on the common instruments. For pop music, the most common key is G Major since it is easy to play on both guitar and piano[9]. On the other hand, genres utilizing horn instruments will commonly play in keys such as F and Eb.

Extracting the rhythm patterns is somewhat more complicated than the other features, but their interpretation is clear: they describe the amount of rhythmic elements in the music. From the sample figure shown in [10], one could assume that the rhythm features are quite highly correlated and could benefit from dimensionality reduction.

The correlation hypothesis can be confirmed from the data by computing the correlations between the columns. The resulting plot is shown in 2, where higher correlations are shown with brighter colours and lower correlation with darker colours. From the figure we see that indeed columns 97-264 containing the information about the rhythmic patterns are highly correlated, justifying the use of dimensionality reduction.

A. *Dimensionality Reduction*

Since the data only contains 4363 samples, each one of which has 264 features, it would seem likely that dimensionality reduction would be useful in this case, especially since some of the features are highly correlated.

Therefore we will investigate the effect of dimensionality reduction using PCA on the prediction accuracy of the classifiers.

Another method used for preprocessing the data is *Linear Discriminant Analysis*. LDA can be used by itself as a classifier, or as a preprocessing tool for other classifiers.

III. Methods and experiments

The Scikit-learn library is a popular machine learning library developed for the Python programming language [11]. It is based on the popular numpy and scipy python modules that offer numerical and scientific computing tools. Scikit-learn offers a large amount of common machine learning methods designed such that they all have the same interface. This means experimenting with different classifiers is very quick and easy. The library also offers other commonly required functionality such as standardization of the data and support for easy cross validation. All this lead to the choosing Scikit-learn for performing most of the experimentation.

A. Classical methods

The common steps for all the methods were importing and standardizing the data. Scikit-learn makes standardizing very easy with the command

```
preprocessing.StandardScaler()
```

For all the methods tested, standardizing the data improved accuracy by approximately five percentage points and the standardizer was therefore used for all methods.

Dividing the data into training- and validation sets is also simple using

```
X_train, X_test, y_train, y_test
 = train_test_split(X,
                    y,
                    test_size=0.2)
```

where the `test_size`-parameter specifies the fraction of the data to be used for the validation. For all the method, the value 0.2 was used. This gives a validation set of 872 samples.

Scikit-learn provides facilities to chain together preprocessing steps and the actual classifier using the `Pipeline`-object. Furthermore using the `GridSearchCV`-object it is possible to iterate over a range of parameters for all the objects in the pipeline simultaneously performing cross-validation to extract the best parameters for the classifier. This was used later on for fine-tuning the selected classifers.

PCA Dim	Validation Score
5	0.541809851088
10	0.546391752577
15	0.558991981672
40	0.54295532646
80	0.412371134021
150	0.406643757159
264	0.265750286369

TABLE I. Validation scores for Naive Bayes

1) Naive Bayes: The first of the classifiers tested was the Naive Bayes Classifier. The Scikit-learn implements the class `GaussianNB` that by default learns the prior probabilities of the classes from the data. One would assume that in this problem instance, the prior probability would play a big role, since it is so heavily skewed towards a certain class.

The Naive Bayes benefits greatly from dimensionality reduction, and the best results were given by using much fewer than the original 264 dimensions. Table I shows the accuracy of the Naive Bayes for several different PCA dimensions.

2) Nearest Neighbor: Scikit-learn implements the Nearest Neighbor-classifier with the class

```
neighbors.KNeighborsClassifier(k)
```

where the parameter k controls the number of neighbors used. As with the other classifiers, a number of different parameter combinations were tested. Values in the range $(10, 15)$ consistently provided the best validation scores. Unlike the Naive Bayes, dimensionality reduction with PCA provided very little benefits. The best validation scores were given by 100-150 dimensions, but the difference between unprocessed data was roughly one percentage point.

3) Support Vector Machine: The last of the classical methods is the Support Vector Machine. Scikit-learn provides two variants:

```
svm.SVC()
svm.NuSVC()
```

Both of the variants had similar performance, with the regular SVC performing slightly better when only a small number of PCA dimesions were used, and NuSVC performing better with a larger amount of dimensions. Both of the classifiers however performed best without the PCA preprocessing.

Of all the classical methods, the support vector machine provided the best validation scores. However, the disadvantage is that for multiclass classification, the support vector machine is very computationally intense compared to the other methods.

From initial tests, the Naive Bayes and SVM performed the best out of the classifiers considered. Therefore these were selected for further optimization to see if fine-tuning parameters would give increased performance.

Using grid search and cross validation to optimize the parameters did yield some improvements, however the default or automatically set parameters performed quite well also. In the end, the best validation and kaggle scores were obtained using linear discriminant analysis for preprocessing, and a support vector machine for classification. This yielded a cross validation scrore of 0.733 and our eventual kaggle score.

The hyperparameter with the most significant impact in the validation score was the shrinkage parameter of linear discriminant analysis. Also somewhat surprisingly the using the linear kernel was the best choice for the support vector machine.

B. Deep learning

As deep learning models have proven to achieve high accuracy in nonlinear problems, such as image classification problems, they may have great potential in learning music genre classification problems. However, previous research has not achieved major breakthrough in the music genre classification problem [3]. Feng [2] compared performance between fully connected (FC) neural network models and convolutional neural network (CNN) models in the music genre classification problem. His result showed that more complex CNN models do not outperform fully connected neural networks even though CNN models are nowadays standard solutions in image classification problems. In addition CNN models require far more training data than simpler FC neural network models. As our training data is limited, we focus only on FC neural network models in our experiments.

Neural networks consist of multiple nonlinear functions. These function are called neurons. Neurons are organized into layers. The layers are connected to each other sequentially. The size of the layers vary. Each neuron store its own weight and bias value in its memory and takes an input value from the output of the previous layers neuron output. With these values an output value is produced by non-linear function of a neuron. In general, neurons in the same layer use the same nonlinear function. Produced output values of a layer are sent to each neuron in next layer in the fully connected neural network structure. To achieve desired results, the bias and weight values of neurons are optimized by training data. Optimization methods play a major role when teaching a model. With the right method, weight and bias values converge to local minima that achieves high performance. Width of layers and depth of a neural neural network structure are defined based on the complexity of a problem.

Pytorch neural network framework is used in our experiments. The framework is widely used in both research and industry [12]. Hence all the newest optimizing methods can be found in the framework.

In classification problems, a model needs to produce values that determine the class of the input data. In our model, the last layer size is 10. Each individual neuron in the last layer define an individual music genre. The output of our model is a probability distribution over these 10 music genres. This can be achieved with a softmax function

$$f(y=j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{10} e^{x^T w_k}},$$

where w is the weight value of a neuron and x input values from a previous layer. Log likelihood loss function evaluates the prediction accuracy of a model and determines the direction in which to optimize the weight and bias parameters.

The input data was standardized in the same way as with the classical machine learning methods. First we experimented prediction accuracy of models with different depth and width. We chose to use Adam method to optimize weight and bias parameters, because it is widely used in the newest deep learning methods [13]. Adam method is more sophisticated than Stochastic gradient descent by its feature to adjust learning parameters by itself [14]. Here defining a learning rate is convenient. Our results show that a with number of layers higher than 2 the prediction error increases. The reason is most likely that critical features of data are lost with deeper structures. As the feature size of the input data is 284, the best prediction performance is achieved with layer widths of 80. Hence the optimal neural network structure is two layers that that both have 80 neurons. Appendix A represents the built neural network model with Pytorch.

IV. Results

Table II shows the final Kaggle results for the neural network model. The achieved accuracy with the generated validation data was 0.69. Kaggle results show that the size of the validation data was too small or data distribution of the Kaggle data differs from the training data. Table III shows the final Kaggle results for the best performing support vector machine. The SVM outperforms the built neural network model.

Metric	Score
Accuracy	0.612
Log-loss	0.2237

TABLE II. KAGGLE SCORES FOR NEURAL NETWORK IMPLEMENTATION.

Metric	Score
Accuracy	0.647
Log-loss	0.2055

TABLE III. KAGGLE SCORES FOR SVM IMPLEMENTATION.

V. CONCLUSION

Figure 3 shows the *logarithmic loss* performance of the built neural network model. With limited amount of data compared to the complexity of the problem, prediction error with training data is high. In addition the figure shows that the model starts overfitting training data fast. Hence increasing iterations would not increase the performance of the model. Figure 4 shows the prediction accuracy distribution with the training data. The model predicts well class 0 (*Pop Rock*) data samples, but the model is not able to recognize classes that have few samples in training data. The model minimizes the risk of predicting the class 0 data incorrectly. As the dataset is unbalanced, better results are achieved without predicting values that occur unlikely such as class 8 (*Reggae*) or 9 (*Blues*) data. Figure 5 illustrates the prediction accuracy with the generated validation data. The model only achieves to predict *Pop Rock*, *Electronic*, and *Rap* data somewhat correctly.

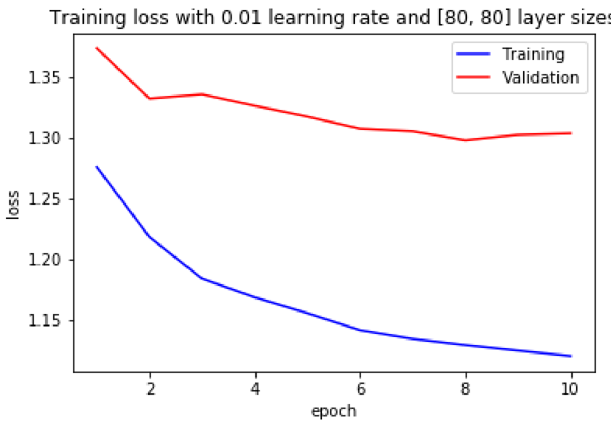

Fig. 3. *Logarithmic loss* performance of the neural netwrok model

As the table IV shows the support vector machine predicts more broadly genres. This shows that the neural network model requires in general more traning data of all the classes. For example the support vector machine predicts the *Rap* genre well even thought the training data has few of *Rap* songs. Still the performance is not optimal as the results show. *Pop Rock* data is often predicted inaccurately. Reason might be that *Pop Rock* music is often a combination of other genres.

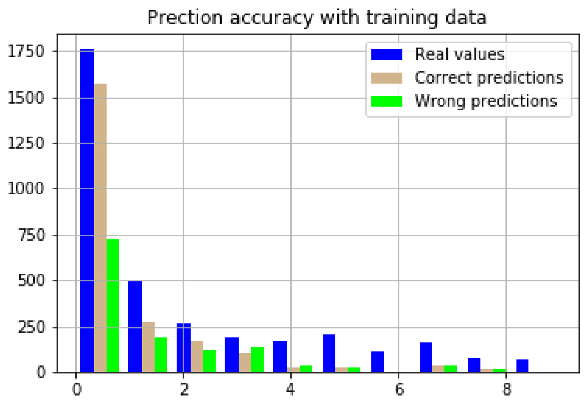

Fig. 4. The neural network's prediction accuracy with validation data

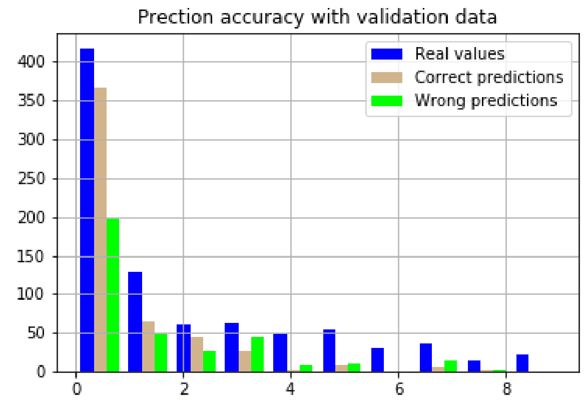

Fig. 5. The neural network's prediction accuracy with validation data

Imbalanced data, and limited amount of data have influence on that the classical machine learning method outperforms the neural network model. In general neural networks models need a large amount of training data in order to be able to achieve good performance. Nevertheless for this problem the classical machine learning

1982	85	19	20	11	31	11	11	4	4
147	426	19	11	1	9	2	0	2	1
37	12	261	1	0	9	1	0	5	0
52	17	3	163	5	8	1	2	1	1
99	8	6	8	69	11	6	2	3	2
74	11	13	11	8	131	2	2	4	4
61	9	4	10	6	2	39	5	3	2
100	1	1	3	1	3	0	84	0	2
9	4	8	0	2	3	1	1	64	0
44	0	2	7	1	2	1	3	0	26

TABLE IV. CONFUSION MATRIX

methods are better because these methods are faster to take into use than are flexible to achieve proper results with limited data.

However accuracy of music genre classification is not enough for industrial usage. As the previous research has shown better results have not been achieved with just music data [2]. It would be interesting to see if combining different sources of data that is often available in music genre classification problem could improve the results. For example utilizing music lyrics might improve the classification problem.

References

[1] Dieleman, S., *Recommending music on Spotify with deep learning*, Retrieved from: http://benanne.github.io/2014/08/05/spotify-cnns.html

[2] Feng, T. *Deep learning for music genre classification*,

[3] Van den Oord, A., Dieleman, S., Schrauwen, B. *Deep content-based music recommendation.* In Advances in neural information processing systems (pp. 2643-2651). 2013

[4] Wang, X., Wang, Y. (2014, November). *Improving content-based and hybrid music recommendation using deep learning.* In Proceedings of the 22nd ACM international conference on Multimedia (pp. 627-636). ACM.

[5] Wang, H., Wang, N., Yeung, D. Y. (2015, August). *Collaborative deep learning for recommender systems.* In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1235-1244). ACM.

[6] Mandel, M. I., Ellis, D. *Song-Level Features and Support Vector Machines for Music Classification.* In ISMIR (Vol. 2005, pp. 594-599).

[7] *Mel Frequency Cepstral Coefficient (MFCC) tutorial*, Retrieved from: http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[8] Ellis, D., *Chroma Feature Analysis and Synthesis*, Retrieved from: https://labrosa.ee.columbia.edu/matlab/chroma-ansyn/

[9] Van Buskirk, E., *The Most Popular Keys of All Music on Spotify*, Retrieved from: https://insights.spotify.com/us/2015/05/06/most-popular-keys-on-spotify/

[10] *Audio Feature Extraction*, Retrieved from: http://ifs.tuwien.ac.at/mir/audiofeatureextraction.html

[11] Pedregosa *et al.*, *Scikit-learn: Machine Learning in Python*, JMLR 12, pp. 2825-2830, 2011

[12] Chintala, S., Gross, S., Paszke, A., *Pytorch github* from: https://github.com/pytorch

[13] Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Le, Q.V. and Ng, A.Y., *On optimization methods for deep learning.* InProceedings of the 28th international conference on machine learning (ICML-11)(pp. 265-272).

[14] Goodfellow, I., Bengio Y., Courville, A., *Deep Learning*, MIT Press, 2016 from:http://www.deeplearningbook.org

Appendix

A. Built neural network model

```python
class MultiFC(torch.nn.Module):

    def __init__(self, in_width, layers, out_width, activation, useNLL):
        assert(1 < len(layers))
        super(MultiFC, self).__init__()
        self.functions = []
        in_layer = in_width
        for out_layer in layers:
            self.functions.append(nn.Linear(in_layer, out_layer))
            in_layer = out_layer

        self.output_function = nn.Linear(in_layer, out_width)
        self.activation = activation
        self.useNLL = useNLL

    def forward(self, x):

        for function in self.functions:
            x = self.activation(function(x))

        if(self.useNLL):
            x = F.log_softmax(self.output_function(x))
        else:
            x = self.activation(self.output_function(x))
        return x
```