NTNU
Kunnskap for en bedre verden

TTT4275 - Estimation, Detection and Classification

Music Genre Classification

Tarek El-Agroudi
Finn Gross Maurer

April 2022

Summary

This project investigated different methods for classifying songs and pieces of the GTZAN[1] data set into ten distinct music genres. Initially, the k-nearest neighbour classifier is implemented from scratch in python and its performance is improved gradually by considering separability, feature selection and feature extraction. The feature dimension was restricted to four, and the best performance of a 49% error rate was obtained by brute force feature selection in combination with k-fold cross validation. To arrive at a general classifer for all genres, the theory motivated the systematic tuning and training of a Gaussian mixture classifier and a multilayer perceptron from the python library Scikit-learn (sklearn) [17]. The Gaussian mixture classifer achieved an error rate of 46% and displayed signs of overfitting. The multilayer perceptron achieved the best performance of 23% and was, after robustifying it to overfitting through regularisation, concluded as the best general classifier the authors were able to implement.

Table of Contents

[1]http://marsyas.info/downloads/datasets.html

1 Introduction

In 2020, the global market for recorded music grew by 7.4 % [10]. This massive arsenal of songs and pieces is divided into many genres of music that help people find the music they like to listen to. But such labels are not necessarily placed on the music by the artist. Even if some particularly engaged people were tasked with the daunting task of classifying the worlds music, research [7] has found that humans classify music with an accuracy of only about 70%. This opens the door for classification algorithms, which are tailored to specifically this kind of classifying operation on large data sets!

This report documents the results of a student project on music classification in the course TTT4275 Estimation, Detection and Classifications at the Norwegian University of Science and Technology. As is indicated by the course name, classification is a third of the course and focus is placed on introducing the major techniques for classification. The project allows the authors to explore several of these techniques on the practical and highly relevant problem of classifying music genres. Specifically, 990 songs of the GTZAN database are used to train and test methods that classify songs into the ten music genres *pop, metal, disco, blues, reggae, classical, rock, hip hop, country* and *jazz*. Each song was represented as a set of 63 features, such as tempo.

Section 2 introduces the relevant theory needed to understand the different classifiers, performance metrics and other relevant elements of the tasks. The tasks themselves and any choices made in regards to them are discussed and justified in Section 3. The project is split into four tasks of increasing generality, and Section 3 is divided accordingly. The implementation details and results are presented and discussed in Section 4, were the partition of the four tasks is maintained. Finally, Section 5 presents a conclusion on the central results, successes and challenges faced throughout the project.

2 Theory

2.1 The KNN classifier

The k-nearest neighbours classifier classifies a sample by finding the k points in a training set that are “closest” to the sample, and selects the most abundant class from these k samples. This is a non-parametric approach to classification, as it directly uses the data points, and does not attempt to fit a statistical model to the data. However, a parameter that can be changed is the number of neighbours k.

There exist a number of algorithms for finding the k-nearest neighbours of a sample [1]. For reasonably sized training sets, a brute-force approach is very easily implemented. Here, one simply computes the distance between the sample and *each* point of the training set. Then, one selects the k shortest distances from these. It is clear that the number of computations here increases linearly with the number of points in the training set. Many of the more advanced algorithms [1] for finding the nearest neighbours do this more efficiently. This distinction is insignificant for smaller data sizes, but should be considered when data sizes approach tens of thousands of points.

It is clear that the concept of distance is crucial to the KNN classifier, as it looks for the ”closest” points to the sample. In cases where raw data are used, a natural choice is to use the euclidean norm. Other distance , such as the Mahlanobis distance [14], exist that make use of the variance of the data, which generally must be estimated from the data first. Section 2.2 discusses the importance of normalising data as a prerequisite for using the euclidean norm.

2.2 Normalisation

As outlined in Section 2.1, the KNN classifier is based on calculating distances, in the entire feature space, between the sample and the points of the training set. When features have different units or scales, as is natural to expect, using a simple euclidean distance measure will cause a bias towards the data with the largest scale. If differences are extreme, the distance measures approach simply a distance measure for the largest feature, effectively reducing the dimension of the feature space to one.

As a remedy, there are two main normalisation schemes discussed in the course [15]. *Min-max normalisation* normalises the data by, for each feature, replacing a sample x_l by its value relative to the minimum and maximum values:

$$x_l := \frac{x_l - \min x_l}{\max x_l - \min x_l} \tag{1}$$

A different normalisation approach, *z-score normalisation*, makes use of the expected value and variance of a feature, and normalises the data according to:

$$x_l := \frac{x_l - \mathbb{E}(x)}{\sqrt{\mathbb{V}ar(x)}} \tag{2}$$

2.3 Feature space and separability

The feature space is the space where all data points lie, whose dimension equals the number of features. Adding features is generally useful for classification because it often introduces *separability*. Separability is the property of a data set that points from different classes can be spatially separated from each other [19]. In other words, the average distance between points from different classes increases, making distance based classifiers, such as the k-nearest neighbours classifier, more likely to be correct.

However, increasing the number of features comes with a cost, commonly termed the "curse of dimensionality" [8]. This is the phenomenon that the sparsity of data increases with dimension of the feature space and hence the distance between all samples increases on average. This has the undesired effect that the differences between previously close-together and far-apart samples no longer is as significant as it was previously. In other words, the distance from a training sample to its closest neighbour is comparable to the average distance to any sample, such that one can be less confident in classifications based on "closeness".

One remedy to the curse of dimensionality is to gather more data, such that the sample density remains high. If this is infeasible, one should employ one of several dimensionality reduction techniques. The simplest is of course to just remove certain features, a technique called *feature selection*. This can be done visually by plotting feature histograms for different classes on top of each other and removing those features that have the most overlap. There also exist quantitative ways to perform feature selection. One such method relies on the analysis-of-variance or ANOVA method [6]. By, for all features, comparing the distance between the means of different classes with their variances, it can identify if certain classes have very similar means for a certain feature. If this is the case, the classes are hard to separate spatially for that feature, and it may be removed.

One does however not have to remove features to reduce dimensionality. Another group of methods, *feature extraction*, enables the extraction of a new set of features of a lower dimension from the entire feature space. A standard way of doing this is to perform a principal component analysis [2]. This is a technique that projects the samples onto the space spanned by the most significant eigenvectors, corresponding to the largest eigenvalues, of the covariance matrix of the samples. This is generally a better solution than simply removing features because it optimally fits a reduced feature space to the data while giving the same benefits with regards to the curse of dimensionality.

2.4 Gaussian mixtures and expectation maximisation

In contrast to nonparametric classifiers such as the k-nearest neighbour classifier described in Section 2.1, parametric classifiers are based on estimating a statistical model of the data and using this model for classification. A popular approach to parametric estimation is to model the data in each class as a Gaussian mixture, and to use a maximum likelihood classification approach based on these models. The statistical model we estimate *for each class* ω_i is a mixture of Gaussians, each of dimension equal to that of the feature space, with a probability distribution:

$$p(x|\omega_i) = \sum_{j=1}^{J} \rho_j p(x|j), \tag{3}$$

where $x|j \sim \mathcal{N}(\mu_j; C_j)$ denotes a single Gaussian component with mean μ_j and covariance matrix C_j, J is the number of Gaussians in the mixture and ρ_j is a parameter that represents the probability that the j-th Gaussian is "active" [19]. Once the number of Gaussians, J, is decided, an expectation maximisation, or EM, algorithm is used to estimate ρ, μ and C for each Gaussian. Expectation maximisation [19] is an iterative algorithm that starts with an initial guess of the parameters of the Gaussians. It then assigns each sample of the training set to the Gaussian whose likelihood is the highest for the sample. Then a new set of Gaussians is estimated from the now grouped samples, and the process is repeated. Finally, once the algorithm has converged to a model, a maximum likelihood approach is used to select the class. If there is prior knowledge about the probabilities of classes, this approach falls under the category of Bayesian classification. Specifically, if it is known that the probability of each class is equal, the maximum likelihood classifier equals the maximum a-posteriori estimator or MAP [19], given by

$$\hat{\omega} = \underset{\omega_i}{\operatorname{argmax}}\, p(x|\omega_i) \tag{4}$$

A central tuning parameter for Gaussian mixture based classifiers is the number of Gaussians/clusters to use. If too few Gaussians are fitted, one may struggle to properly model the possibly complex distribution of samples. If too many are chosen, one encounters the issue of overfitting, the phenomenon that the model is so tailored to the training data that it no longer models new data samples correctly. One method for choosing the number of Gaussians that takes both these effects into account is the Bayesian Information Criterion, or BIC [5]. The BIC is a function that assigns a score to a training set (see Section 2.6) according to:

$$BIC = -2 \times LL + log(N) \times k \tag{5}$$

where LL denotes the log likelihood of the training set given the model, N is the number of samples in the training set and k is the number of model parameters. The best model according to this criterion is the one with the smallest BIC score, as this indicates a high likelihood and not too many model parameters, as to prevent overfitting.

2.5 Neural networks and backpropagation

Neural networks have gained increasing popularity in solving classification problems [19]. Many exist, but the simplest method is a feed forward neural network or multi-layer perceptron (MLP). In MLPs, a network is built by having a set of layers each containing a number of nodes. Each node has a value called its activation, which is computed from the activations of the nodes of the previous layer together with a set of weights and biases (Figure 1). A typical activation function is the logistic funtion (6) [18]. The activations in the first layer of the MLP are directly defined by the data.

$$f(x) = \frac{1}{1+e^{-x}} \tag{6}$$

Once a number of levels, or depth, and a number of nodes per level is selected, the problem of training the network reduces to tuning the weights and biases defining the activation functions.

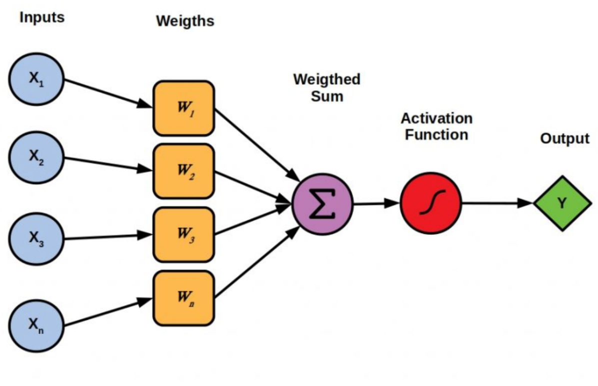

Figure 1: A single perceptron. Downloaded from starship-knowledge.com.

This is done by defining a cost function weighing the deviations from the last-layer activations to their target values. One popular cost function is the cross-entropy, or log-loss function, [9]:

$$C = -\sum_j d_j \log(y_j) + (1 - d_j) \log(1 - y_j) \tag{7}$$

where d_j is the desired value of the j-th last-layer activation and y_j is its actual value output by the network. d_j is typically 1 for the correct last-layer node and 0 for all others. For each sample of the training set (see Section 2.6), the weights and biases are adjusted as to minimise the cost function (7). The final adjustment to the weights and biases is simply the average adjustment over all training samples.

A common method of minimising (7) for a sample is to use a gradient descent algorithm. Gradient descent computes the gradient of the cost function and iteratively approaches its minimum by taking steps in the direction of the negative gradient [11]. A common and computationally straight forward way of finding the gradient is the method of back-propagation. For each activation error of each node of a layer, one determines how much the activation of the previous layer nodes should change. These desired changes are propagated to the layer before that, and so on, until the first layer is reached. This process is done for each of the last-layer nodes, and the desired activation adjustment for each node are averaged. This average is, due to the chain rule, the same as the gradient of the cost function [4].

Just as for the gaussian mixture classifier presented in Section 2.4, neural networks are also susceptible to overfitting [12]. One way to reduce overfitting is to use a regularisation method, of which the most common is called ride regression or L2 regularisation [12]. This augments (7) by adding an extra term $\alpha \times \|W\|_2^2$, where α is called the regularisation rate and $\|W\|_2$ is the euclidean norm of a vector of all weights of the network. Practically, this exploits the idea that large weights represent a more specialised network, while smaller weights represent more general networks.

A final problem that occurs is that the cost function can have multiple local minima for a given training sample. Which minimum the gradient descent algorithm converges to depends on the initial configuration of weights and biases. One must therefore be careful with concluding on an optimal amount of layers/nodes based off a single training session.

2.6 Training methods and performance evaluation

Development of classifiers generally requires a data set of labelled samples. It is common practice to partition this labelled data into data used for training the classifier, the “training set”, and data used to evaluate the classifier, the “test set”. Because a classifier is designed to function on new unlabelled data, it is advantageous that the data used for testing is not also used for training. Several techniques exist for such a partitioning of data. The simplest is to remove a certain ratio of samples from each class, train the classifier with the remaining samples and test it on the removed samples.

In order to make substantial conclusions on the performance of a classifier on such a test set, a quantitative way of representing and evaluating classifications is needed. A common way to represent the results of a classifier on a test set is to place each classified sample into a confusion matrix [19], where the row indicates the actual class of the sample and the column indicates what it has been classified as. It follows that samples that lie on the diagonal represent correct classifications while those that don't are misclassifications. Summing over all misclassifications and dividing by the total number of samples gives the *error rate* of the classifier on the test set, which can be used as a performance metric to compare classifiers.

However, a problem occurs when we want to optimise a classifier based on the error rate achieved for a test set. Effectively, doing this will fit the classifier specifically to the test set, such that it is no longer "unbiased", and looses its power in performance evaluation. A way to still use error rates in classifier tuning that does not pollute the test set is the method of k-fold cross validation [3]. In this framework, the training set is split into k partitions. One of theses partitions is called the validation set. The classifier is trained on the other $k - 1$ partitions, and the error rate is computed for the validation set. This process is done once for each partition as a validation set, and the error rate used for tuning is the average.

3 Task outline

3.1 Simple KNN classifier

In this part of the project, a k-nearest neighbour classifier was implemented to classify all ten music genres. The implementation was made from scratch such as to demonstrate the underlying principles, and the code can be found in the attached folder in the delivery. To restrict the scope, a fixed $k = 5$ was given by the project task. Furthermore, the feature space was restricted to the four features *tempo, spectral centroid mean, spectral rolloff mean* and *mfcc 1 mean.* The classifier was implemented in python, and the 30 second music segments were used for the training and testing.

The central implementation details of the KNN classifier is given in section Section 4.1. When calling the classifier, the user may choose to normalise the data using min-max normalisation or z-score normalisation, as discussed in section Section 2.2. This was done to facilitate the investigation of the effects of normalisation on the performance of the classifier.

In order to evaluate performance, the confusion matrices and error rates on the test set were computed. These are first analysed holistically, then compared with regards to the type of normalisation used. In order to compare the performance of the classifier with a more seasoned implementation, the python library sklearn [17] and its integrated KNN functions were used on the same data, and compared with our classifier.

Finally, the confusion matrices were studied in detail and observations were contextualised through the authors understanding of music. To gain even further understanding, some of the misclasified songs were listened to in order to glimpse some *musical* reasons for the mistakes.

3.2 Linear separability

In the second part of the project, linear separability was investigated by reducing the feature space from part 1 of the project and systematically removing one feature at a time. For the four genres *pop, disco, metal* and *classical*, the four features from part 1 were plotted in histograms. By studying the overlap between classes for each feature, the features are ranked from most separable to least separable. Features are removed one by one in order of increasing separability and confusion matrices and error rates are computed. The results and implementation-details are given in

Section 4.2.

As an extension to the simple feature selection required by the task, the feature-extraction technique of principle component analysis, introduced in Section 2.3, was applied to the same feature set. This extension of the task is motivated by the theory that suggests the potential for better performance, and thus better separability, if three artificial "features" are extracted from the four dimensional feature space.

3.3 Feature selection

This part of the project focuses on how the choice of features affects the performance of the KNN classifier designed in part 1 of the project. All ten genres are to be classified, and exactly four features are to be chosen, where at least three must be selected from: *spectral rolloff mean, mfcc 1 mean, spectral centroid mean,* and *tempo.*

Because only four features are used, and the training set is of a reasonable size, a possible method for determining what feature to replace by which feature was done by running *all* possible combinations of four features, where 3 are selected from the four features provided. Then the chosen configuration was simply that which, for the validation sets (see Section 2.6), provided the lowest error rate. It is worth noting that such a "brute-force" approach is infeasible for larger feature dimensions and larger training sets. In order to test a more sophisticated technique of feature selection, the analysis-of-variance or ANOVA method, introduced in Section 2.3, was applied to the same problem. This was done by using the function `SelectKBest` from sklearn.

3.4 Designing a general genre classifier

The task in this part of the project was to design a general classifier for all ten genres of music. From the analysis of the KNN classifer from the first parts of the project, it is clear that we are nowhere near the benchmark on human performance of a 30% error rate [7]. Thus, an alternative classification method is desirable. Briefly put, three other general classification methods have been studied in the course TTT4275 Estimation Detection and Classification: Bayesian classifiers, Linear classifiers and Neural networks.

The decision in this task was to focus on a Bayesian Gaussian-mixture based classifier (Section 2.4) and a Neural Network (Section 2.5). Linear classifiers were omitted because the investigation of linear separability in Section 4.2 indicated that the data is not easily linearly separable in all classes. However, the histograms analysed in Section 4.2 indicated that a Gaussian or mixture of Gaussians seems to apply well to most features, motivating the investigation of Bayesian classifiers. Finally, the vast popularity and success of neural networks in classification problems has motivated the testing of a network on all features and genres.

To limit the scope of the project, the expectation maximisation algorithm for fitting Gaussian mixtures, which is described in Section 2.4, as well as the back-propagation algorithm for tuning neural network weights described in Section 2.5, were implemented using the algorithms provided by sklearn. For the MLP, sklearn uses the cross entropy cost function introduced in Section 2.5. Results are motivated, analysed and compared in Section 4.4.

As for the length of data sets, previous research [13] has shown that the best performance in music classification is gained from longer data sets. Hence, the full 30 second data segments were chosen to train and test the general classifier.

4 Implementation and Results

This section presents the implementation and results of the tasks introduced in Section 3. Performance is generally analysed via confusion matrices and error rates, introduced in Section 2.6.

4.1 KNN classifier

The KNN classifier was implemented as class in python. This allowed the implementation of option selection by the user, such as the type of normalisation. In order to compare the designed classifer with the integrated KNN classifier from sklearn, a python class responsible for calling the sklearn KNN was also implemented.

The error rates of the KNN classifier when tested on the test set provided by the task are given in Table 1, for all types of normalisations. Figures 2 and 3 show the confusion matrices for no normalisation and min-max normalisation respectively.

Table 1: Error rates for the KNN classifiers for different normalisation types.

	Error rate			
	Our		Sklearn	
Normalisation	Train	Test	Train	Test
without	0.33	0.63	0.48	0.63
z-score	0.35	0.60	0.48	0.60
min-max	0.35	0.59	0.48	0.61

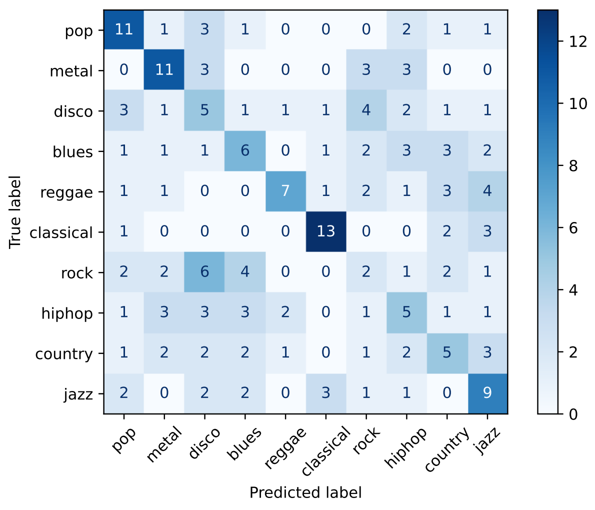

Figure 2: Confusion matrix for our KNN classifier, where no normalisation is used. Error rate = 0.63.

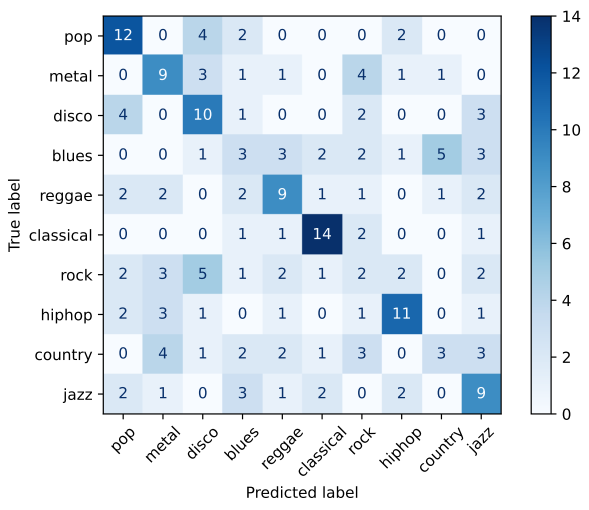

Figure 3: Confusion matrix for our KNN classifier, where min-max normalisation is used. Error rate = 0.59.

An initial observation from the error rates is that they all are at around 60 %, which is significantly worse than the human performance of 30 %, which was for all 10 genres. This can be attributed to the fact that the k-nearest neighbour classifier is heavily reliant on the features being spatially separable, as discussed in Section 2.3, which they very well may not be. Furthermore, the four features selected may not be the optimal features with regards to separability. Both these problems are investigated further in Section 4.2 and Section 4.3.

With regards to the normalisation scheme used, it is evident that differences are not very significant. However, min-max normalisation performs slightly better than z-score normalisation, and both normalisation methods perform better than no normalisation at all. The latter is expected and can be attributed to the use of the usage of the euclidean distance measure, as discussed in Section 2.2. Min-max normalisation is used for the rest of the tasks.

When comparing the KNN classifier with that of sklearn, it is seen that the resulting error rate on the test set is similar. However, an interesting observation is that the difference between the error rate for the training and test sets is much smaller for the sklearn classifier. This is likely caused by an implementational difference to our classifier. Specifically, when there is a tie in the k nearest neighbours, sklearn chooses a class by picking the sample with the smallest class id. However, the author's implementation chooses the class of the sample that is the closest, likely causing the smaller error rates.

Looking at the confusion matrices in greater detail, it is interesting to analyse how the KNN performs on certain genres compared to others. It is evident that, for instance, the classifier is very good at classifying classical music, where only 5 songs are misclasified out of a total of 19. In contrast, the classifier is terrible at classifying rock songs, where most are classified as disco. The superiority of classical music can seem natural from a human perspective. Combinations and harmonies of instruments such as the violin, cello, clarinet and so on are almost exclusively used in classical music, and this fact likely contributes to separating the genre from the others in the feature space.

Furthermore, insight may be gained by listening to some of the misclassifications. Particularly, many rock songs are classified as disco. One such song is "Sweet Dream" by Jethro Tull[2]. While being a rock song, it is technically progressive rock, which is a rather broad rock genre. For the authors as listeners, the distinct synthesiser melody reminds of the start of Earth Wind & Fire's "Boogie Wonderland", which is a classic disco song. Another common misclassification was classifying blues as country. One such example was "Train carried my girl from town" by Kelly Joe Phelps[3]. The initial guitar melody almost sounds like a banjo to the authors, and has the classic fast repetitive finger picking melody that is common in many country songs, like The Dead South's "Banjo Odyssey".

4.2 Linear separability

Figures 4, 5, 6 and 7 show histograms for the four features and four genres specified in Section 3.2. As described in Section 2.3, separability, which is essential for good KNN classifier performance, is identified by being able to spatially separate the classes. This is clearly hardest to do for tempo, where all the classes are almost completely superposed. This seems quite natural to the authors, as the distinction between pop, disco, metal and classical music is not in their tempo but in their instruments, dynamics and texture of the sound. Tempo is therefore removed first from the feature space.

Selecting the next feature to remove is harder, but it looks like mfcc 1 mean has the most overlap, and it is therefore the second feature removed. For the same reason, the spectral centroid mean is removed, leaving the spectral rolloff mean as the last feature left.

The error rates and confusion matrices for all four experiments are given in Table 2. A significant improvement is observed when tempo is removed. This can be attributed to the fact that almost no improvement with regards to separability is achieved by including it, but that the inclusion occurs at the expense of the curse of dimensionality, described in Section 2.3. Comparing the corresponding confusion matrix in Figure 9 with that of the full feature space in Figure 8 shows that the biggest performance increase is gained when classifying metal music. This is can be explained by studying the histograms, from which is clear that the metal distribution is the only one that does not, for at least one feature, separates itself from the other classes. Hence, it is very sensitive to the sparsity introduced by an increase in the feature space dimension.

After this, the error rate actually increases when removing features. This is likely because, for three or less dimensions, the data density is high enough for the curse of dimensionality not be a problem anymore. However, the loss of separability due to the loss of features effectively clumps the samples so much together that the KNN becomes less efficient, as motivated in Section 2.3.

An interesting observation from all four confusion matrices is that classification of classical music is completely unaffected by the feature reduction. This is again easily justified by the histograms, where the classical genre is easily separated from the others in *at least* one of the features each

[2]https://www.youtube.com/watch?v=Jp9yu8mTm9w
[3]https://www.youtube.com/watch?v=o35VJMtuu7Q

experiment.

Table 2: Error rates of our KNN classifier for different feature configurations and four genres.

Features	Error rate	Confusion Matrix
spectral rolloff mean, spectral centroid mean, mfcc 1 mean, tempo	0.30	Figure 8
spectral rolloff mean, spectral centroid mean, mfcc 1 mean	0.25	Figure 9
spectral rolloff mean, spectral centroid mean	0.28	Figure 10
spectral rolloff mean	0.39	Figure 11
PCA, 3 components	0.14	Figure 12

Finally, a principle component analysis was performed with three components on the full four dimensional feature space. The resulting error rate of 14 % is i a massive improvement and shows the superiority of feature extraction to feature selection in this case. The corresponding confusion matrix is given in Figure 12. The classifier is now perfect for metal and classical, and very good for pop music. However, it is sub-optimal for disco music. This is harder to explain because the "features" now no longer are specific physical characteristics, but rather eigenvectors of the feature space. This is one of the disadvantages of PCA. However, the new feature space can be visualised in a scatter plot, which is done in Figure 13. Here we can visually see that disco has the most overlap with other genres, explaining the worse performance!

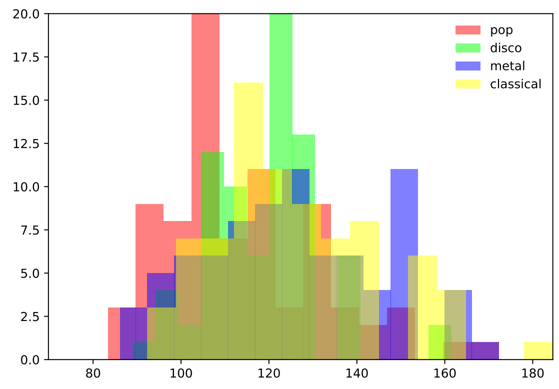

Figure 4: Genre distribution for tempo.

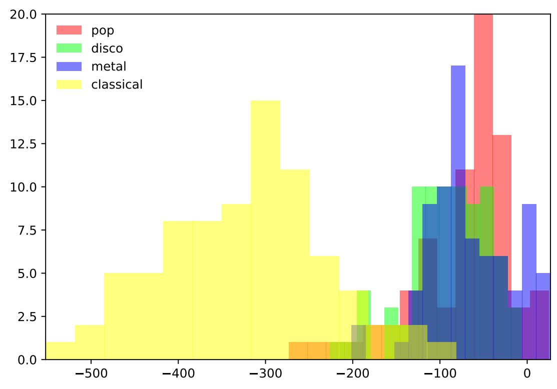

Figure 5: Genre distribution for mfcc 1 mean.

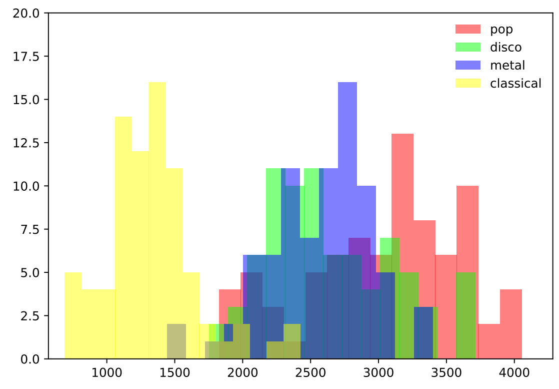

Figure 6: Genre distribution for spectral centroid mean.

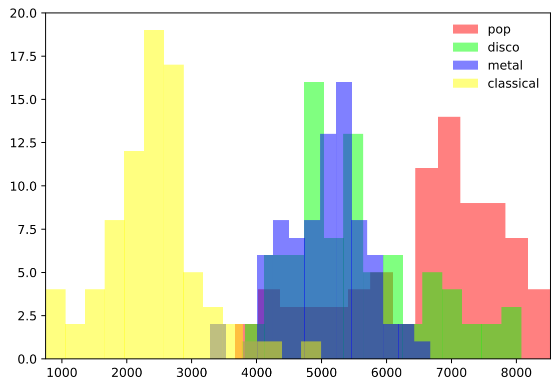

Figure 7: Genre distribution for spectral rolloff mean.

4.3 Feature selection

The attached folder in the delivery contains the implementation of the two methods of feature selection outlined in Section 3.3. The analysis-of-variance method was implemented through the sklearn function `SelectKBest`. The k-fold cross validation scheme was implemented by nested for loops were all different feature combinations fulfilled the requirements of Section 3.3. k-fold cross validation, as described in Section 2.6, was then used to decide the best combination of features.

Table 3 gives the selected features and corresponding error rates for both methods. It is evident that the brute force approach utilising k-fold cross validation described in 2.6 is superior with an error rate of 49 % compared to 57 % for the analysis-of-variance based method. This is quite natural because brute force by definition picks the feature that gives the minimum error rate over the validation set. However, as discussed in Section 4.3, brute force becomes infeasible for larger data sets and feature spaces, such that the analysis-of-variance method or a similar method becomes the only viable alternative.

Looking at the selection of features specifically, it is noticeable that both methods identify spectral rolloff mean, spectral centroid mean and mfcc 1 mean as the three features to keep, throwing away

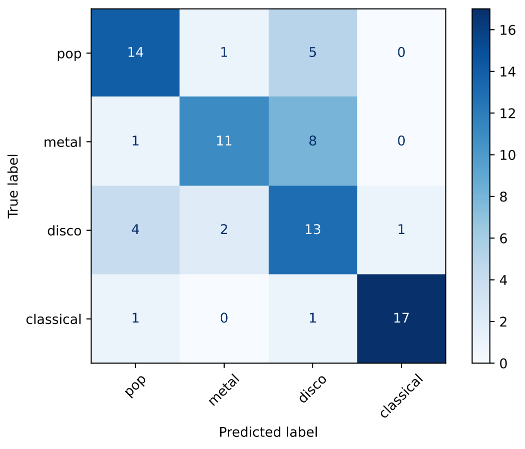

Figure 8: Confusion matrix for four genres with the features: spectral rolloff mean, spectral centroid mean, mfcc 1 mean and tempo. Error rate = 0.30.

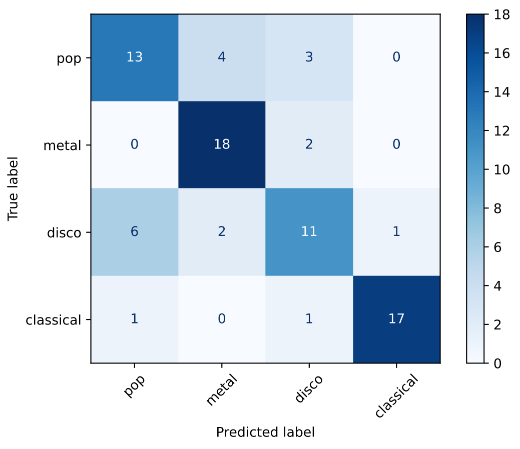

Figure 9: Confusion matrix for four genres with the features: spectral rolloff mean, spectral centroid mean and mfcc 1 mean. Error rate = 0.25.

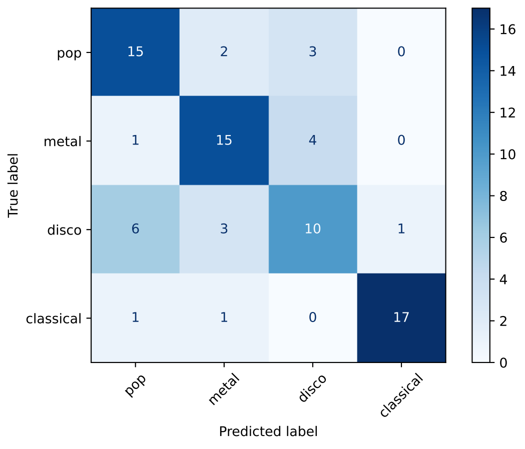

Figure 10: Confusion matrix for four genres with the features: spectral rolloff mean and spectral centroid mean. Error rate = 0.28.

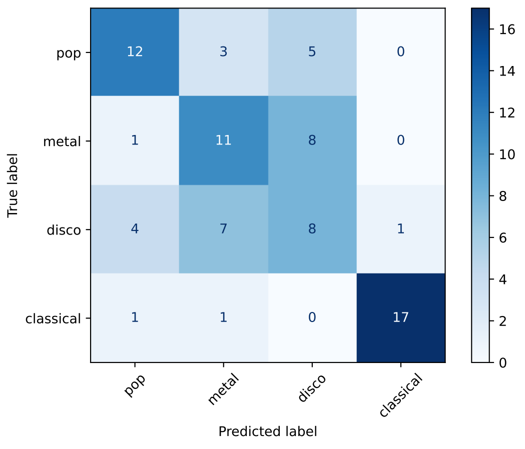

Figure 11: Confusion matrix for four genres with the feature: spectral rolloff mean. Error rate = 0.39.

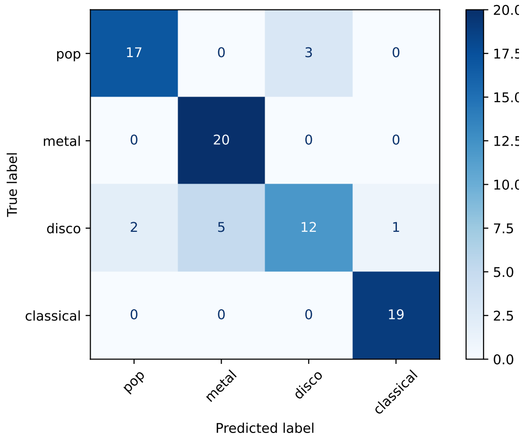

Figure 12: Confusion matrix for four genres with 3 principal components used. Error rate = 0.14.

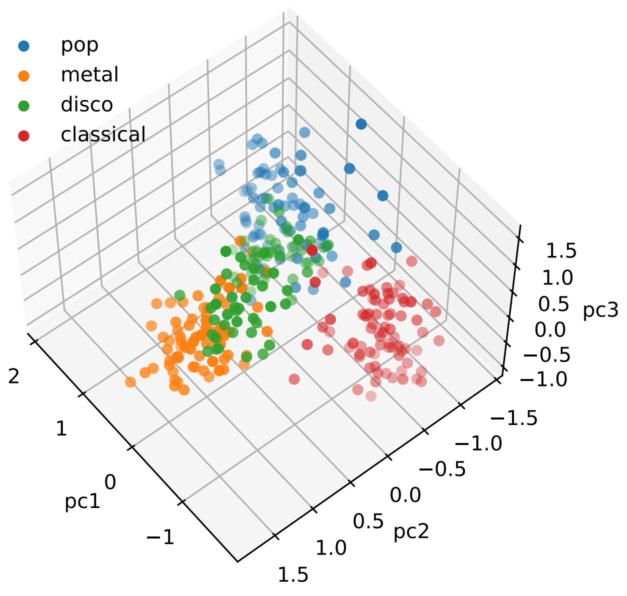

Figure 13: Genre distribution, principle component analysis.

Table 3: Error rates for feature selection with cross validation brute force and analysis of variance

Method	Chosen features	Error rate	Confusion Matrix
Cross validation brute force	spectral rolloff mean, spectral centroid mean, mfcc 1 mean, rmse var	0.49	Figure 15
ANOVA	spectral rolloff mean, spectral centroid mean, mfcc 1 mean, spectral bandwidth mean	0.57	Figure 14

tempo. This is motivated by the experimental results of Section 4.2, were the inferiority of tempo as a classification feature was demonstrated. However, the methods differ in the feature chosen as the replacement for tempo. While brute force gives the rmse variance, analysis-of-variance gives the spectral bandwidth mean. This can be explained through the fact that studying validation set error rates is a significantly different metric than studying the variability of feature means, as outlined in 2.3.

4.4 Designing a general genre classifier

The python implementation of the training procedures and evaluations of the gaussian mixture classifier and multi layer perceptron is presented in the attached code folder.

The Gaussian mixture classifier is considered first. The first task was to use the training set to select the specific features to use, as well as the number of components in the Gaussian mixture models, as outlined in Section 2.4. Implementation-wise, this was done by iterating over all possible numbers of features, and using the ANOVA method through the sklearn function `SelectKBest` for each. The training data was then split into a smaller training set and a validation set, according to the procedure outlined in Section 2.6. For each feature combination, the best suited Gaussian mixture was found by fitting a Gaussian mixture model with number of components reaching

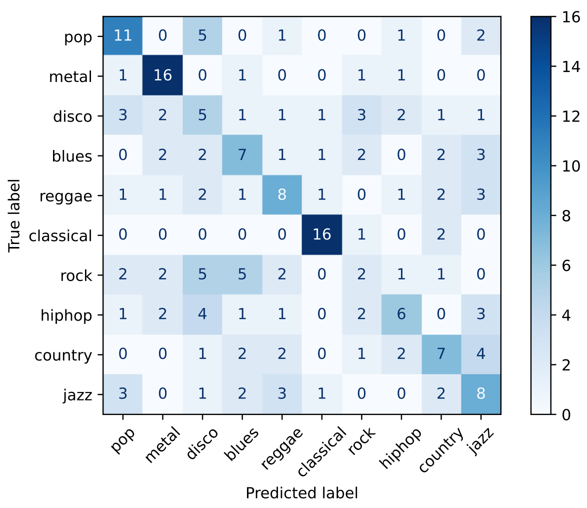

Figure 14: Confusion matrix for our KNN classifier, where features are selected by ANOVA. Error rate = 0.57.

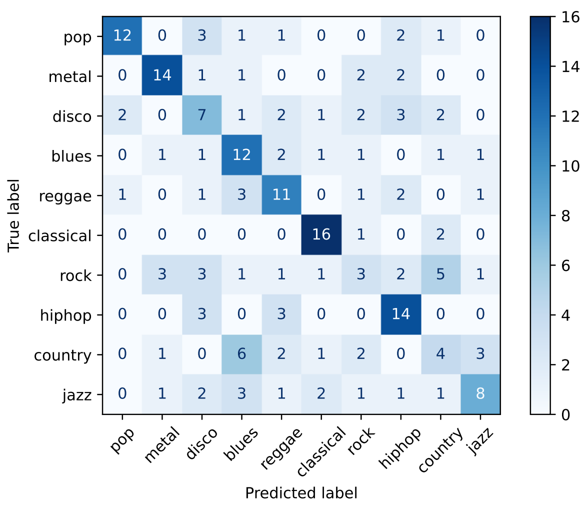

Figure 15: Confusion matrix for our KNN classifier, with features selected by k-fold cross validation. Error rate = 0.49.

from 0 to 10, and choosing the one with lowest the lowest Bayesian information criterion (see Section 2.4). The error rate on the validation set was then calculated. The test set was evaluated with the feature combination and models with the best validation error rate.

During the for loop outlined above, the validation error rate was tracked for the different number of features selected through the ANOVA method. The resulting plot is shown in Figure 18. The error rate drastically decreases until about 14 features, after which it is relatively constant. Since too-high dimensional Gaussians can be problematic with regards to the overfitting and the curse of dimensionality phenomena discussed in Section 2, the configuration of 14 features chosen by the ANOVA method was chosen. For this selection the resulting Gaussian mixture is given in Table 4. It is clear that for most classes, one or two Gaussians are optimal with regards to the Bayesian information criterion.

Table 4: The number of components for the Gaussian mixture for each genre.

pop	metal	disco	blues	reggae	classical	rock	hiphop	country	jazz
1	1	8	2	1	1	4	6	1	3

The resulting error rate and confusion matrix for the Gaussian mixture classifier is given in Table 5 and Figure 16. Clearly, the error rate of 46 % is significantly better than the KNN classifier, which never performed better than 59 % for all genres.

Table 5: Error rates for the Gaussian mixture classifier and the multilayer perceptron classifier.

Classifier	Train	Test
ML with GM model	0.10	0.46
MLP	0.00	0.23

Looking at the confusion matrix for the Gaussian mixture classifier in Figure 16 reveals an interesting observation. Instead of mediocre performance across classes, the classifier seems to generally be either very good or very poor. For instance, there are only two misclassifications for metal, while the classifier does not classify a single song correctly for disco! Remembering the mixture of Table 4, one can observe that disco is also the class to which the most Gaussians were fitted. More generally, the classes that perform the worst are all classes to which the algorithm fitted more than three Gaussians. This can be explained by phenomonenon of overfitting. The more Gaussians are fitted to the mixture, the more likely it is that the it models training-set specific noise as well. Returning to the error rates, we see that they are significantly lower for the training set than for the test set, indicating that we may indeed have overfitted. This also begs the question of whether the Bayesian information criterion introduced in Section 2.4 really is a good way of selecting the number of mixture components.

Having tuned and investigated the Gaussian mixture classifier, focus is shifted to whether even better performance is achievable through the multilayer perceptron. Here the first step was to determine the number of layers and nodes of the network. This was again solved with an iterative solution, where some rules of thumb [16] were used to specify iteration limits. The best model was again chosen through k-fold cross validation and consisted of a single hidden layer with 40 neurons.

The resulting error rate and confusion matrix for the selected multi layer perceptron is given in Table 5 and Figure 17. The error rate of 23 % is the best encountered so far. Specifically, it is even better than the quoted human performance of 30 %, which is quite remarkable for the simplest form of neural network! This vast improvement can be attributed to the sheer size and complexity of the neural network, such that it can pick up trends in the features that other classifiers don't. The vagueness of this explanation is one of the disadvantages of neural networks, in that it is very hard to motivate its inner workings.

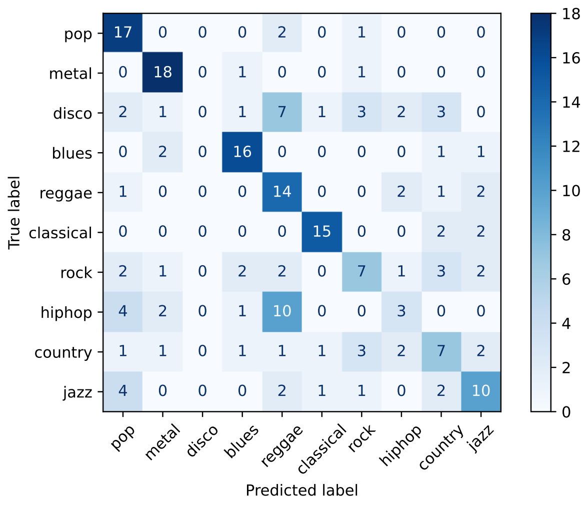

Figure 16: Confusion matrix for Gaussian mixture classifier. Error rate = 0.46.

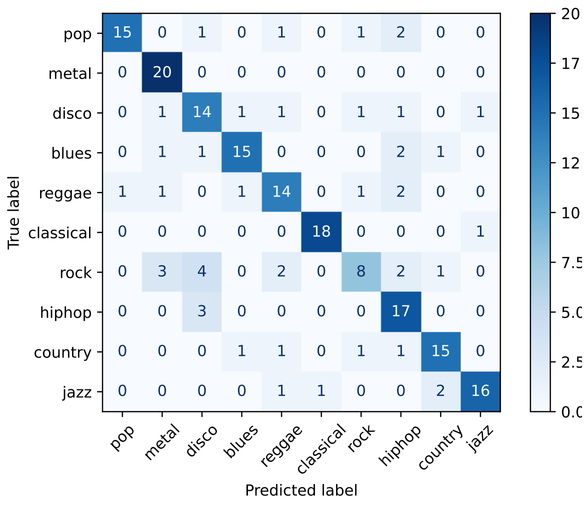

Figure 17: Confusion matrix for multilayer perceptron. Error rate = 0.23.

Looking at the confusion matrices for the multilayer perceptron in detail, it generally performs very well on all classes. Except for rock, there are few "standout" classes or extreme differences such as those observed for the Gaussian mixture classifier. It can be argued that this is a more desirable trait for a classifier, as it has less performance-wise bias depending on the class. However, one would have to analyse different case studies in order to identify if this difference holds generally.

Returning to the error rates for the multilayer perceptron of Table 5, they look like an extreme case of overfitting. The error rate for the training set is rounded to 0 %, while that of the test set is 23 %! As discussed in Section 2.5, a popular remedy for overfitting in neural networks is L2-regularisation. In sklearn, this is incorporated by the giving the regularisation rate α as an input argument to the classifier. This was tested for a large range of rates, and the error rates for both the training and test set were plotted against α in Figure 19. The difference between training and test set error rates approaches zero as α increases, indicating less overfitting as expected. However, a general increase in error rate is observed for increasing α, which is undesirable. A possible trade-off between these effects could be chosen as $\alpha \approx 1$, since this gives the least overfitting before the error rate skyrockets.

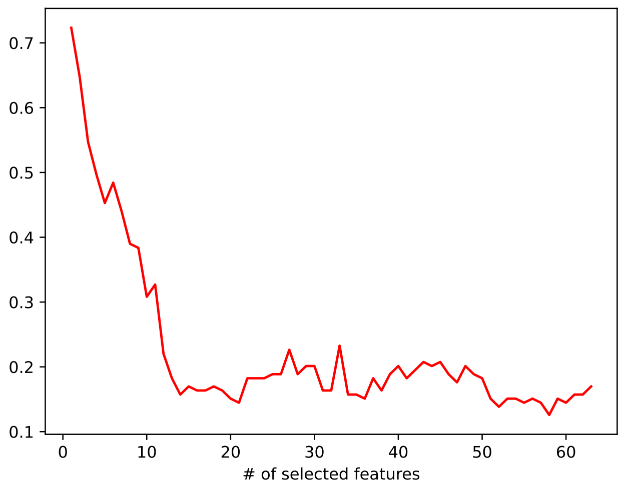

Figure 18: Error rates for different number of features, using a Gaussian mixture classifier.

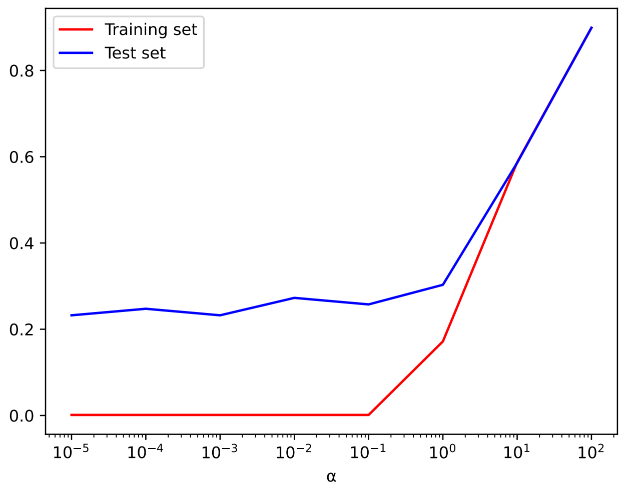

Figure 19: Cross-validated error rates for a range of regularisation rates

5 Conclusion

This project has presented a series of popular classifiers and used them in order to classify songs of the GTZAN data-set into 10 different genres. The k-nearest neighbour classifier was implemented and tested first. Performances of around 60% error rate were achieved on the initial feature selection provided by the task. Some misclassifications were studied by listening to the tracks in question, and the authors found it easy to sympathise with the classifier. The second part of the project revealed that performance could be improved by removing tempo from the feature space, resulting in an error rate of 25% for classification of four genres. An even better error rate of 14% was achieved when the dimension reduction was performed through a principal component analysis, demonstrating the power of feature extraction compared with feature selection.

In the third part of the project, a brute force approach in combination with k-fold cross validation was compared with an analysis-of-variance approach in order to replace one of four given features with an arbitrary feature. The brute force approach proved superior with an error rate of 49% for all ten genres - the best performance achieved with the KNN classifier. However, due to the computational requirements of brute force approaches, the analysis-of-variance approach to feature selection was considered a more realistic alternative, and was used in the remainder of the project.

In the final part of the project, a general classifier for all ten genres was sought with the aim of beating the quoted human performance of 30% error rate. First, a Gaussian mixture classifier on 14 features improved the error rate to 46%. While increasing the performance significantly from the KNN classifier, its performance depended heavily on which genre it was tested on, with the possible reason being overfitting. This suggested that the Bayesian information criterion is sub-optimal for tuning the number of components of Gaussian mixtures.

Finally, a multilayer perceptron was trained and tested. A single-layer 40 node network achieved an error rate of 23% on all ten genres, beating the quoted human performance significantly. The network classified songs consistently well across genres. Since the error rates revealed possible significant overfitting, L2 regularisation was used to identify a regularisation rate of $\alpha \approx 1$ for the network. This gives similar performance, but reduces the error rate mismatch between the training and test set such that the network can be considered more robust to future test data.

This project has demonstrated that multiple classifiers can be used for music classification. It has allowed the authors to investigate the effects of normalisation, separability, feature selection, feature extraction and regularisation on classifier performance. It has left them with a practical understanding of the course material, and an eagerness to explore the topic of classification further.

Bibliography

[1] *1.6. Nearest Neighbors.* URL: https://scikit-learn.org/stable/modules/neighbors.html#neighbors (visited on 25th Apr. 2022).

[2] *2.5.1. Principal component analysis (PCA).* URL: https://scikit-learn.org/stable/modules/decomposition.html#pca (visited on 25th Apr. 2022).

[3] *3.1. Cross-validation: evaluating estimator performance.* URL: https://scikit-learn.org/0.17/modules/cross_validation.html#cross-validation (visited on 25th Apr. 2022).

[4] *Backpropagation.* URL: https://brilliant.org/wiki/backpropagation/ (visited on 25th Apr. 2022).

[5] *BAYESIAN INFORMATION CRITERION.* URL: https://stanfordphd.com/BIC.html (visited on 25th Apr. 2022).

[6] Jason Brownlee. *How to Perform Feature Selection With Numerical Input Data.* URL: https://machinelearningmastery.com/feature-selection-with-numerical-input-data/ (visited on 25th Apr. 2022).

[7] Mingwen Dong. 'Convolutional Neural Network Achieves Human-level Accuracy in Music Genre Classification'. In: *arXiv e-prints*, arXiv:1802.09697 (Feb. 2018), arXiv:1802.09697. arXiv: 1802.09697 [cs.SD].

[8] Peter Grant. *k-Nearest Neighbors and the Curse of Dimensionality.* URL: https://towardsdatascience.com/k-nearest-neighbors-and-the-curse-of-dimensionality-e39d10a6105d (visited on 25th Apr. 2022).

[9] Geoffrey E Hinton. *Connectionist learning procedures.* Artificial intelligence 40.1, 1989.

[10] IFPI. *IFPI issues Global Music Report 2021.* URL: https://www.ifpi.org/ifpi-issues-annual-global-music-report-2021/ (visited on 25th Apr. 2022).

[11] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* 2e. New York, NY, USA: Springer, 2006.

[12] Artem Oppermann. *Regularization in Deep Learning — L1, L2, and Dropout.* URL: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036 (visited on 25th Apr. 2022).

[13] Yannis Panagakis, C. Kotropoulos and Gonzalo Arce. 'Music genre classification via sparse representations of auditory temporal modulations'. In: *European Signal Processing Conference* (Jan. 2009).

[14] Pierluigi Salvo Rossi. *TTK4275 Lecture 14-15: Performance Evaluation.*

[15] Pierluigi Salvo Rossi. *TTK4275 Lecture 18: Performance Evaluation.*

[16] Harpreet Singh Sachdev. *Choosing number of Hidden Layers and number of hidden neurons in Neural Networks.* URL: https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev/ (visited on 25th Apr. 2022).

[17] *scikit-learn - Machine Learning in Python.* URL: https://scikit-learn.org/0.17/index.html.

[18] *sklearn.neural_network.MLPClassifier.* URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (visited on 25th Apr. 2022).

[19] Magne H. Johnsen Tor A. Myrvoll Stefan Werner. *Estimation, detection and classification theory.*