NTNU

Kunnskap for en bedre verden

Department of Electronic Systems

TTT4275 - Estimation, Detection, and Classification

Can You Hear the Music?

Author:
Anton Tran, Vetle Rød Dahl

Figure 1: Oppenheimer could hear the music. Source: (Cinematerial 2023)

Date: 29th April 2025

Table of Contents

List of Figures

List of Tables

1 Abstract

In this study, we compare three supervised approaches for music genre classification - K-Nearest Neighbors (K-NN), a hybrid Gaussian Mixture Model (GMM) trained via Expectation-Maximization (EM), and as an additional exercise, a convolutional neural network (CNN) - on ten genres using data from the GTZAN dataset. We first evaluate a 5-NN classifier using four audio features under both Euclidean and Mahalanobis distance metrics, achieving 37.4% and 41.9% accuracy respectively. By examining feature distributions and consequently swapping tempo as a feature due to overlapping with MFCC 4 std, the K-NN classifier accuracy rises to 38.4% and 50.5%. A small boost is obtained by choosing $k = 7$ at 53.2%, still using the Mahalanobis distance. The GMM was implemented with EM as a hybrid supervised algorithm, yielding a classification accuracy of 66.7%. Further attempts at classification - now using a CNN - gave a 79.3% accuracy of classification. Our results demonstrate that the Mahalanobis distance and careful feature selection substantially improve performance of K-NN, but a probabilistic GMM can better capture overlapping, covariance-structured feature spaces and outperform nonparametric neighbors-based methods.

2 Introduction

Music plays a central role in human culture, communication, and entertainment. With the growing availability of digital music through streaming services, the need for automated and efficient methods for music categorization has increased. Automatic music genre classification can then be seen as important, as it provides valuable metadata for organizing and retrieving music content for aiding listeners in discovering music aligned with their preferences.

This project focuses on ten distinct music genres: Pop, Metal, Disco, Blues, Reggae, Classical, Rock, Hiphop, Country, and Jazz. The data for this analysis is based on the GTZAN dataset (Tzanetakis 2002), which is extracted from audio tracks by calculating statistical features of their spectrograms, obtained using the python library Librosa(McFee et al. 2015). Since the dataset includes labeled data, the study primarily focuses on supervised classifiers which are able to learn a mapping from inputs to outputs, allowing the model to predict the label of new data. Having the dataset labeled also assures that the classifiers can be tested against what the correct output values should be.

This paper explores the effectiveness of the Euclidean distance metric for the K-NN classifier, and further investigates the Mahalanobis distance as a more sophisticated distance metric.

Initially, the classification performance is evaluated using four audio features: Spectral rolloff mean, MFCC 1 mean, spectral centroid mean, and tempo. Subsequent analysis involves a comparison of feature distributions among the genres to better understand classification challenges, and guide feature selection. Finally, an enhanced classification accuracy is sought through the use of alternative classifiers with multiple selections of features.

The goal of this study is to assess the applicability and performance of K-NN classifiers for music genre classification and comparing it to other classifiers, highlighting the impact of feature selection and distance metrics, and contributing practical insights into music categorization systems.

3 Theory

This part presents the theoretical foundation necessary to implement the classification methods used in this study. The mathematical concepts introduced here are definitions of the covariance matrix and its relevance to the principles of K-Nearest Neighbor, the distance metrics, the Gaussian Mixture Model (GMM) and finally Long-Short-Term-Memory neural network (LSTM) used as our alternative classifier.

3.1 Covariance matrix

Let $\mathbf{X} = (X_1, X_2, ..., X_n)^\top$ be a random n-dimensional feature vector from a distribution P_k. The *covariance matrix* Σ_k of P_k is an $n \times n$ matrix defined as:

$$\Sigma_k = \text{Cov}(\mathbf{X}) = \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top\right]$$

Each element Σ_{ij} of the matrix represents the covariance between the feature X_i and X_j:

$$\Sigma_{ij} = \text{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])]$$

Intuitively, the covariance matrix highlights the variability and the correlation between features in a dataset. If the off-diagonal terms of Σ_k are zero, the features are uncorrelated; otherwise, they may vary together.

Having a baseline understanding of what a classifier is, sets a foundation for the topic of this task. A classifier is a predictive model/algorithm that assigns an input feature vector X to one of a finite set of labels. A linear classifier separates classes using decision boundaries defined by linear functions, where decisions are made by testing the sign of a weighted sum $w^\top X + b$. In contrast, an optimal classifier in a Bayesian sense is the one that minimizes overall misclassification by choosing, for each x, the label k with the highest posterior probability, in other words maximizing $p(k|x)$. When the class-conditional densities are known, $p(k|x)$ can be optimal. Our study has focused on supervised classifiers that learn from labeled examples, as introduced in the Introduction, but one could also employ unsupervised methods, which is typical for pure Gaussian Mixture Models when labels are unavailable. Finally, classifiers can be hard,making a single decision per input, or soft, producing a full vector of posterior probabilities (Duda et al. 2001).

3.2 K-Nearest Neighbor Classifier (K-NN)

The Nearest Neighbor classifier is a supervised template-based classifier. It classifies an unknown data point by comparing and matching the item to a template of known, labeled data points (for future reference called classes). The data point is then assigned to the class of its closest neighbor in the feature space. The K-NN generalizes this idea by considering the k closest data points instead of a singular one, and assigning the majority class among them.

This method relies on a distance metric to determine the closeness of the unknown data point and the surrounding classes. When classes are clearly separated, K-NN can be highly effective in assigning unknown data points their label. However, performance depends on choosing an appropriate value for k and a distance metric that reflects the structure or pattern of the data. The distance metric chosen can also heavily affect performance.

Supervised methods like K-NN require labeled training data to make accurate predictions.

3.3 Euclidean distance

A commonly used distance metric in KNN classification is the Euclidean distance, also called the 2-norm of the vector between the data points. In other words, the Euclidean distance is the geometric distance between two points in n-dimensional space. Assume two points $X_1, X_2 \in \mathbb{R}^n$. The Euclidean distance between X_1 and X_2 is then defined as

$$d_{euclidean} = \sqrt{(X_1 - X_2)^\top (X_1 - X_2)}. \tag{1}$$

It assumes that all features contribute equally and independently to the distance, which can be a limitation when the feature data are correlated but not distributed spherically.

3.4 Mahalanobis distance

Data points from a joint distribution are rarely distributed as perfect spherical clusters in n-dimensional space. Therefore, the Euclidean distance will not always be the best metric. The Mahalanobis distance is an attempt to include the covariance of the data's distribution in the distance calculation. This is done by introducing the inverse of the covariance matrix for the data features.

Assume $X_1, X_2 \sim P_k^n$ with the corresponding covariance matrix Σ_k for the distribution. The Mahalanobis distance between X_1 and X_2 is then defined as (2)

$$d_{mahalanobis} = \sqrt{(X_1 - X_2)^\top \Sigma_k^{-1} (X_1 - X_2)} \tag{2}$$

For this distance to be used effectively, one must assume the data distribution is linear, or Gaussian, and that the calculated covariance matrix is a good representation of the real distribution's covariance.

A common, specific case of the Mahalanobis distance is instead calculating the distance between a point X P_k^n and the mean μ_k of their shared distribution, as shown in Equation 3

$$d_{mahalanobis} = \sqrt{(X_1 - \mu_2)^\top \Sigma_k^{-1} (X_1 - \mu_2)} \tag{3}$$

3.5 Gaussian Mixture Model (GMM)

A Gaussian Mixture Model is a probabilistic model that represents the data as a mixture of several normal distributions, also called Gaussian distributions. Typically, GMMs are used in an unsupervised setting to determine the probability that a given data point belongs to a cluster. Each distribution represents a group or cluster of data. In this study, the audio data from each genre could be modeled as a Gaussian distribution characterized by a mean and a covariance matrix.

GMMs combine these Gaussian distributions using weights, or mixing coefficients, to describe the overall distribution of the data. Given that we have M_i distributions for each class ω_i, we assume that each data point x is generated by one of the M distributions. The GMM of a data point x is given by:

$$p(x/\omega_i) = \sum_{k=1}^{M_i} c_{ik} \mathcal{N}(\mu_{ik}, \Sigma_{ik}) = \sum_{k=1}^{M_i} \frac{c_{ik}}{\sqrt{2\pi^D |\Sigma_{ik}|}} exp[-\frac{1}{2} d_{mahalanobis}], \tag{4}$$

where μ_{ik} is the mean, Σ_{ik} is the covariance, and c_{ik} are the mixing weights from Equation 10, which have the constraint $\sum_k c_{ik} = 1$ (Johnsen 2017, p. 9).

As each genre can be categorized into single Gaussians with a known set of x values, it would be optimal to maximize the log-likelihood of Equation 4, resulting in

$$log[p(x_{ik}/\Lambda)] = -\frac{D}{2} log(2\pi) - log|\Sigma_{ik}| - \frac{1}{2} d_{mahalanobis}. \tag{5}$$

Maximizing this log-likelihood can be shown to minimize the Mahalanobis distance between the data points and the Gaussian component means, thus fitting the distributions as closely as possible to the data.

The GMM provides a way to model complex data distributions by combining simpler Gaussian distributions. When adding a new data point, we want to calculate how likely it is to belong to each data cluster.

3.6 Expectation-Maximization algorithm (EM)

The Expectation-Maximization (EM) algorithm provides a systematic way to estimate the parameters of the Gaussian mixture when labels or component assignments are not directly observable. EM iteratively alternates between two steps until convergence.

In the E-step, the posterior probabilities, also known as responsibilities, representing the probability that each data point x_i belongs to component k:

$$\gamma_{ik} \;=\; p(z_i = k \mid x_i, \theta) \;=\; \frac{c_k\,\mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{\ell=1}^{K} c_\ell\,\mathcal{N}(x_i \mid \mu_\ell, \Sigma_\ell)} \;=\; \frac{c_k\,|\Sigma_k|^{-\frac{1}{2}}\,\exp\!\Big(-\frac{1}{2}(x_i-\mu_k)^\top \Sigma_k^{-1}(x_i-\mu_k)\Big)}{\sum_{\ell=1}^{K} c_\ell\,|\Sigma_\ell|^{-\frac{1}{2}}\,\exp\!\Big(-\frac{1}{2}(x_i-\mu_\ell)^\top \Sigma_\ell^{-1}(x_i-\mu_\ell)\Big)} \tag{6}$$

where z_i is the latent variable indicating which type of component.

In the M-step, we update the parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}$ to maximize the log-likelihood given by Equation 5, using the responsibilities computed in the E-step. The parameter updates are derived by taking derivatives of the expected complete-data log-likelihood and setting them equal to zero, yielding the following update rules:

Update Means:

$$\hat{\mu_k} = \frac{1}{N_k}\sum_{i=1}^{N} x_{ik}. \tag{7}$$

Update Covariances (Johnsen 2017, p. 15):

$$\hat{\Sigma}_i = \frac{1}{N_i}\sum_{i=1}^{N_i} x_{ik}\,(x_i-\mu_k)(x_i-\mu_k)^\top. \tag{8}$$

Effective component counts (the total responsibility assigned to each component):

$$N_k = \sum_{i=1}^{N} x_{ik}. \tag{9}$$

Update Mixing Weights (proportion of data assigned to each component):

$$c_{ik} = \frac{N_k}{N}. \tag{10}$$

By iteratively performing these two steps, the EM algorithm converges toward parameter values that locally maximize the data log-likelihood, fitting the GMM to the observed data.

3.7 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a specialized type of artificial neural network that is particularly designed to handle data with a spatial structure, such as images, spectrograms, or videos. CNNs have proven to be extremely effective for automatically learning patterns like edges, shapes, textures, and more complex structures within such data.

At its core, a CNN is composed of several layers. Firstly is the convolutional layer. Here, small filters, also called kernels, slide across the input data to detect local patterns. These filters are learned automatically during the training process.

Classified : → True : ↓	Pop	Metal	Disco	Blues	Reggae	Classical	Rock	Hiphop	Country	Jazz
Pop	11	1	3	1	0	0	0	2	1	1
Metal	0	11	3	0	0	0	3	3	0	0
Disco	3	1	5	1	1	1	4	2	1	1
Blues	1	1	1	6	0	1	2	3	3	2
Reggae	1	1	0	0	7	1	2	1	3	4
Classical	1	0	0	0	0	13	0	0	2	3
Rock	2	2	6	4	0	0	2	1	2	1
Hiphop	1	3	3	3	2	0	1	5	1	1
Country	1	2	2	2	1	0	1	2	5	3
Jazz	2	0	2	2	0	3	1	1	0	9

Table 1: Confusion matrix of the 5-NN classifier using the Euclidean metric. Diagonal entries (in green) are correct predictions; off-diagonals (in red) are misclassifications.

After the convolution operation, an activation function is typically applied. The most common activation function is the ReLU (Rectified Linear Unit), which introduces non-linearity into the model. Without non-linearity, no matter how many layers we stack, the entire network would behave like a single linear transformation. The activation function allows the network to model more complex behaviors and relationships in the data.

Another essential component of CNNs is pooling layers. A pooling layer reduces the spatial size of the feature maps by summarizing small regions, usually by choosing a spesific value (e.g mean, max) within a region. This step not only decreases computational load but also helps control overfitting and ensures that the model is more robust to slight translations or distortions in the input data.

After several rounds of convolution, activation, and pooling, CNNs often flatten the data into a one-dimensional vector and pass it through fully connected (dense) layers. These dense layers combine all the features learned so far and perform tasks such as classification based on the extracted information (GeeksforGeeks 2020).

4 Design of a K-NN classifier for all genres

For the first task, it is of interest to design a 5-NN classifier for all musical genres using the four musical features: *spectral rolloff mean, mfcc 1 mean, spectral centroid mean* and *tempo*. For a nearest neighbor classifier, there are multiple ways to decide what "nearest" actually implies, and is therefore an interesting problem to look into. An Euclidean distance was chosen at first to see if we were able to spot general patterns. The Euclidean 5-NN classifier had a 37.4% accuracy of detection. Table 1 shows the confusion matrix of the classifier.

To include a better weighing of features and scaling of distances, we attempted to switch the euclidean distance for the Mahalanobis distance for the 5-NN classifier. The distance is defined in Equation 2, which includes the inverse of the covariance between two datapoints of different distributions. This was implemented in our classifier by calculating classwise covariance matrices of the features. Then when calculating the distances, the measurement is assumed to be in the same distribution as the measured point, and weighted with the corresponding covariance matrix.

When implemented in Python it gave in total 41.9% correct classification. To use the Mahalanobis distance effectively there are certain assumptions one must make. These are further explored in the next task. Table 2 shows the confusion matrix from the Mahalanobis estimator.

Classified : → True : ↓	Pop	Metal	Disco	Blues	Reggae	Classical	Rock	Hiphop	Country	Jazz
Pop	10	2	3	1	1	0	1	2	0	0
Metal	2	12	3	0	0	0	3	0	0	0
Disco	2	0	9	1	1	1	3	0	1	2
Blues	0	1	0	5	1	1	5	1	4	2
Reggae	0	1	2	2	8	1	0	2	1	3
Classical	0	0	0	0	0	16	1	0	0	2
Rock	1	4	1	4	0	0	3	4	2	1
Hiphop	2	4	2	1	1	0	2	8	0	0
Country	1	1	2	3	1	1	0	4	4	2
Jazz	1	0	0	2	4	1	0	1	3	8

Table 2: Confusion matrix of the 5-NN classifier using the Mahalanobis metric. Diagonal entries (in green) are correct predictions; off-diagonals (in red) are misclassifications.

5 Comparison of features in pop, disco, metal, and classical

This task will look at the distribution of the features of pop, disco, metal, and classical from the previous task, to understand the performance of our classifier better. Figure 2 shows the probability distribution of the different genres for the 4 chosen features. As one can see from the graphs, there's a lot of overlap on all of the features; the means are similar and the variances are large enough to cause confusion. This leads to a reduced performance for linear classifiers such as the K-NN, where a decision rule is applied. This is especially problematic when differentiating between disco and metal. As seen in Figure 2, the means of these distributions are quite similar. This is also reflected in the confusion matrix in Table 2, where disco has been wrongly labeled as metal 3 times. For our classifier, classical is the most distinct and most correctly guessed label. This is reflected in the distributions. The means of the features of the classical class, barring tempo, are quite distinct from the other classes. This results in, as seen from the confusion matrices in Table 1 and Table 2 that we have a success rate of respectively 65% and 80% for classifying classical music correctly.

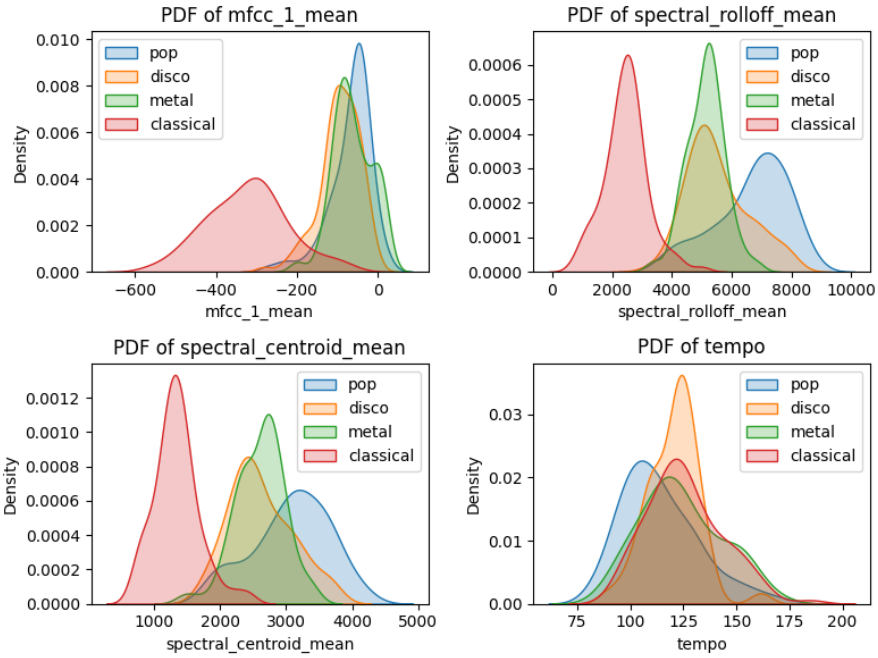

Figure 2: Distribution of music genres wrt. the features: *spectral rolloff mean*, *mfcc 1 mean*, *spectral centroid mean*, and *tempo*.

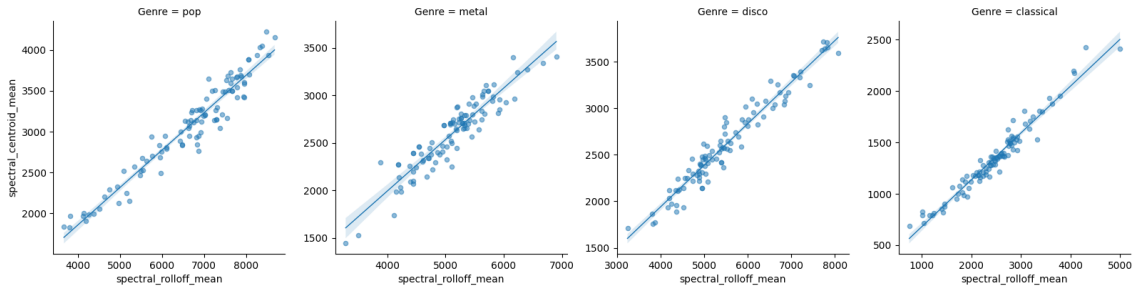

Figure 3: Scatter plot of features including a linear regression

As one can see from Figure 2, the tempo feature has significant overlap of all classes in comparison to the other features. Where all other features have one or two quite distinct class distributions, the tempo feature lacks this and would seemingly be a worse candidate for classification compared to the others. Moreover, looking at Figure 4 reveals that tempo is strongly anticorrelated with these other features—when tempo increases, many spectral and MFCC values decrease, and vice versa. A K-NN classifier using plain Euclidean distance treats each axis independently, so a simultaneous positive deviation in tempo and negative deviation with another feature appears as a large distance, even though those changes lie along the data's principal anticorrelated direction. In practice, this means that similar tracks get pushed apart in Euclidean space, potentially making the K-NN select the wrong neighbors, negatively affecting classifier accuracy. Using the Mahalanobis distance prevents exaggerating those neighbor distances, but the problem of overlapping distributions remain.

During task 1, we used the Mahalanobis distance as a metric as well as the Euclidean distance, to increase the performance of our classifier. For the Mahalanobis distance to be a suitable metric, the data must follow a linear distribution, as well that the calculated covariance matrices of the classes are a good representative of the true covariance. Figure 3 shows a scatter plot of *spectral rolloff mean* and *spectral centroid mean* for the 4 classes. Here, the distribution of the class features seems to follow a linear relationship. This strengthens our assumption of a linear distribution, highlighting why the Mahalanobis distance was better at classifying than using the Euclidean.

6 New 5-NN classifier

In this task, we will attempt to use other features with our 5-NN classifier to increase performance further. So far, only *spectral rolloff mean*, *mfcc 1 mean*, *spectral centroid mean*, and *tempo* have been used for the classification of genres. As seen in task 2, *tempo* seems to be a poor choice for classification, as it has a large overlap over all analyzed genres. Therefore, it was decided to swap this feature out for something else.

To decide the last feature that provided the highest accuracy of classification, an iterative search was performed, testing the accuracies of all features paired with the boundary of using *spectral rolloff mean*, *mfcc 1 mean* and *spectral centroid mean*. The feature proving the highest accuracy for the new 5-NN classifier was found to be *mfcc 4 std*. The new accuracy with these features ended up being 38.38% with Euclidean and 50.51% using the Mahalanobis distance. Table 3 shows the confusion matrix of this classifier.

7 Alternative classifiers

This task will attempt to create other classifiers with a higher classification accuracy using the training data given.

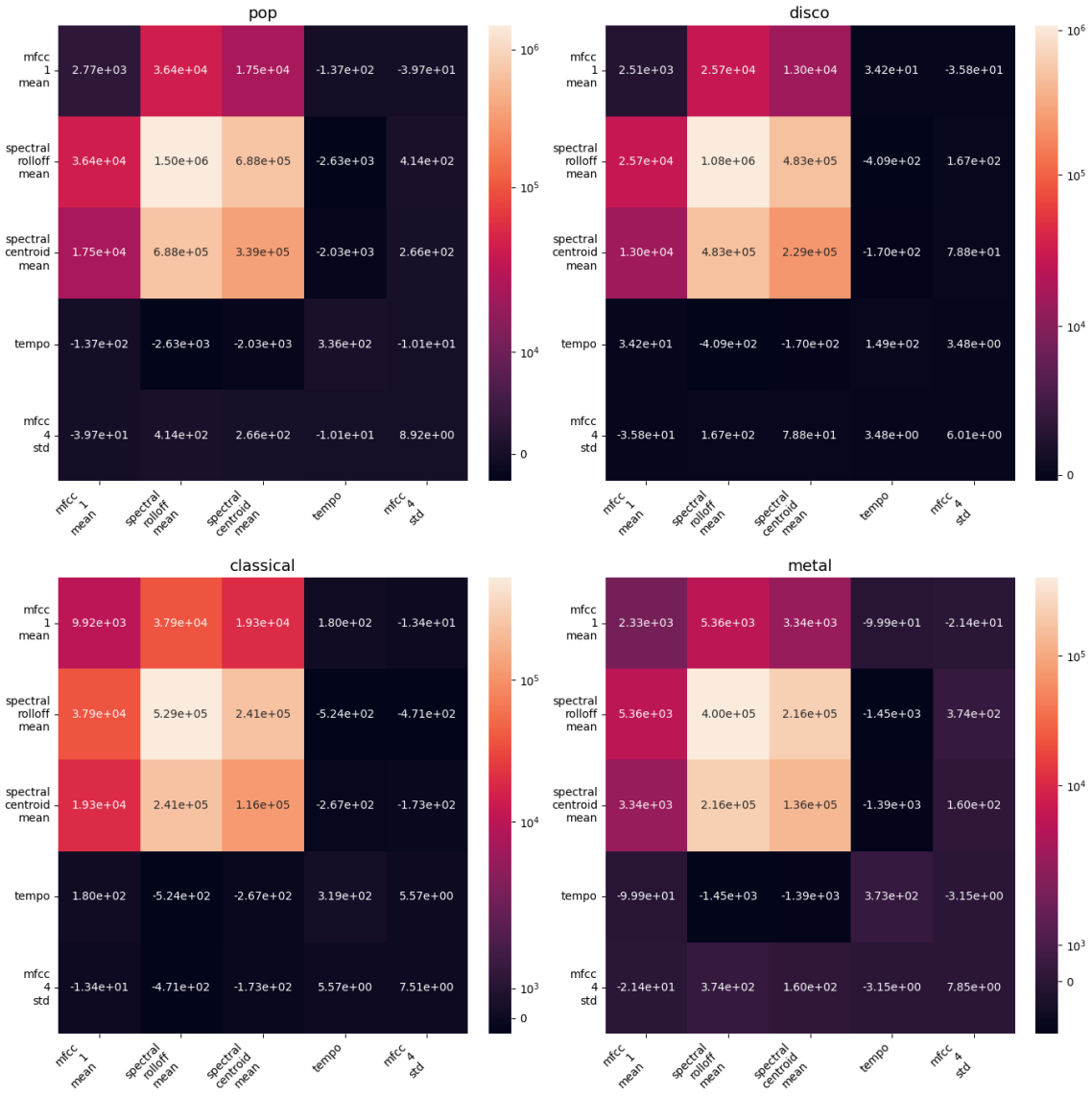

Figure 4: Covariance matrix heatmap. The diagonal shows the variance of the feature, while the off-diagonal shows the covariance between two features. Note negative values, indicating anticorrelation.

Classified : → True : ↓	Pop	Metal	Disco	Blues	Reggae	Classical	Rock	Hiphop	Country	Jazz
Pop	11	1	1	0	2	0	2	3	0	0
Metal	1	16	0	0	0	0	2	1	0	0
Disco	3	1	3	0	3	0	4	4	1	1
Blues	0	2	1	8	1	0	1	4	1	2
Reggae	0	3	0	0	8	0	1	4	1	3
Classical	0	0	0	0	0	15	1	0	0	3
Rock	2	2	4	3	0	0	3	4	0	2
Hiphop	3	1	1	0	2	0	0	13	0	0
Country	0	2	0	4	3	0	0	1	6	3
Jazz	0	0	0	1	0	1	0	1	0	17

Table 3: Confusion matrix of the 5-NN classifier using the *spectral rolloff mean, MFCC 1 mean, spectral centroid mean*, and *MFCC 4 standard deviation*

7.1 7-NN classifiers

An initial attempt to create a better classifier was to increase the amount of neighbors in our K-NN classifier from the previous task. This gave a marginal improvement for k=7, with an accuracy of 53.2%. Increasing it further to k=8 and k=9 resulted in poorer results, probably due to oversmoothing. Instead of just using 4 features, one can instead include all features of the 30 second spectrogram in the 7-NN classifier. This gave a performance of 49.4% with the Mahalanobis distance, which is a poorer performance. This is probably due to training too many features with too little data, which results in a worse calculated covariance matrix.

7.2 Gaussian Mixture Modelling (GMM)

As seen in Figure 5, plotting the empirical PDFs of our selected audio features for each genre shows that most of these PDFs are roughly bell shaped. This in turn suggests that each genre's feature vector can be well modeled by a Gaussian distribution. A GMM can then be used to approximate the class-conditional density using the formulas provided in the theory section. Furthermore, the earlier usage of the Euclidean distance metric proved to be significantly less effective than the Mahalanobis distance, which implies that the data clusters are highly nonspherical. This would in theory make hard classifiers that use the Euclidean distance metric such as K-Means less favorable than GMM, which is better for handling elliptical data types.

We have chosen to implement a hybrid supervised EM-algorithm, as the data provided has true labels, the covariances are initialized from labeled data. The model therefore starts in a good region of parameter space rather than completely random, which is more typical for EM. This ensures that the EM algorithm converges faster and more accurately using the same number of iterations.We then run the E- and M-steps as normal to fine-tune the component means and mixture weights based on soft assignments, as this avoids poor local maxima using the equations 7 and 10.

When picking features for the GMM, we include as many of the distributions as possible while disregarding features with highly non-Gaussian distributions such as *spectral flatness* and *spectral contrast var*, as seen in 5. After finally implementing the Gaussian Mixture Model, the accuracy came out to be 66.67%.

When we inspect genres like rock, the confusion matrix 4 reveals that the guesses are highly inaccurate and blend with especially with disco. This is due to rock having highly overlapping *spectral* and *MFCC* profiles as seen in Figure 5, which confuses the GMM more than the K-NN. While rock gets confused with the other genres, the other genres don't get confused with rock as their data are better separated, illustrated when comparing the features in pop, disco, metal and classical in task 2.

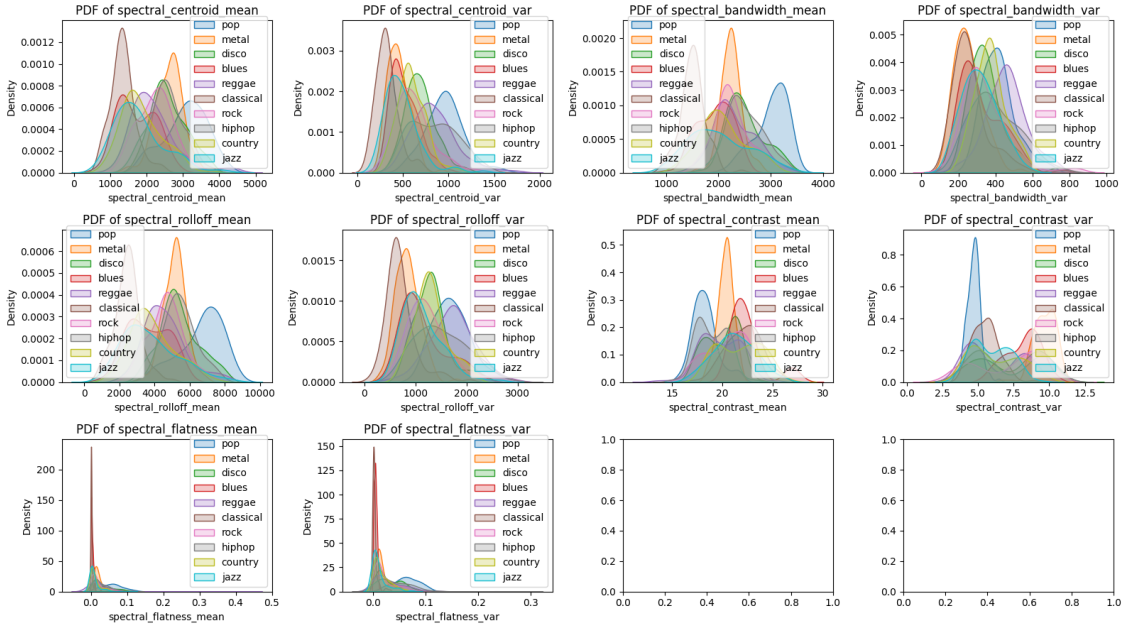

Figure 5: PDF of spectral features

Classified : → True : ↓	Pop	Metal	Disco	Blues	Reggae	Classical	Rock	Hiphop	Country	Jazz
Pop	13	0	3	0	0	0	0	2	1	1
Metal	0	17	1	0	0	0	1	1	0	0
Disco	0	0	6	0	4	0	4	2	3	1
Blues	0	0	1	18	0	1	0	0	0	0
Reggae	2	0	1	2	12	0	0	1	2	0
Classical	0	0	0	0	0	17	0	0	0	2
Rock	2	1	2	1	6	1	5	0	2	0
Hiphop	0	0	0	0	5	0	0	15	0	0
Country	1	0	0	3	0	0	2	0	13	0
Jazz	0	0	0	1	1	1	0	1	0	16

Table 4: Confusion matrix for the GMM classifier.

7.3 CNN classifier

Additionally, an attempt was made to create a classifier using a convolutional neural network (CNN) to classify music genres, and using all available data (5s, 10s and 30s spectrogram) grouped together for each song as a data vector. The network was trained on a shuffled data set with all data features. This was to ensure no bias towards any classes from their order. With a selection of 20% test data, and no validation data to maximize the amount of training data. Validation data was used during the development of the model to look for overfitting. When this was seen as "good enough", these validation data were turned into training data. The network was implemented in Python with TensorFlow using a sequential network with 8 layers. The implementation after training 100 epochs gave a success rate of a whopping 79.3%!. The training data had an accuracy of 79.7%, revealing that overfitting played little part. The corresponding confusion matrix can be seen in Table 5. As a CNN is not of particular relevance to the subject curriculum or task, the technical details of the model will be cut due to length concerns. The code and model structure can be found in the appendix.

Classified : → True : ↓	Pop	Metal	Disco	Blues	Reggae	Classical	Rock	Hiphop	Country	Jazz
Pop	16	0	0	0	1	0	1	1	0	1
Metal	0	19	0	1	0	0	0	0	0	0
Disco	1	0	8	1	1	0	2	3	3	1
Blues	0	1	0	16	0	0	1	1	1	0
Reggae	0	0	0	0	15	0	1	4	0	0
Classical	0	0	0	0	0	19	0	0	0	0
Rock	0	1	3	1	4	0	9	0	1	1
Hiphop	0	1	1	0	2	0	0	16	0	0
Country	1	0	0	1	0	0	4	0	12	1
Jazz	0	0	0	0	1	1	0	0	1	16

Table 5: Confusion matrix of the CNN classifier

8 Conclusion

In this study, we compared two fundamentally different supervised classification methods - K-Nearest Neighbors (K-NN), and a Gaussian Mixture Model (GMM) trained using a hybrid supervised Expectation-Maximization (EM) - for the task of music genre classification on the GTZAN dataset.

We began by reviewing the mathematical foundations of distance-based and probabilistic classifiers. The K-NN relies on the Euclidean or Mahalanobis distance to count and estimate the majority class surrounding the new data, and makes no distributional assumptions beyond local neighborhoods. We implemented a 5-NN classifier using both Euclidean and Mahalanobis distance metric, using class-specific covariance estimators for the latter. The Mahalanobis distance metric consistently performed better than the Euclidean distance due to taking into account data structure and correlation such as anticorrelation between genre features. Comparing the genre features revealed that tempo for the four different genres had similar distributions which makes the task of distinguishing them harder for the classifier. This created the motivation for replacing this feature. This provided a classifier which yielded significantly better results.

As for the alternative classifier, several different classifiers was tested. Increasing the number of K-Nearest Neighbors resulted in a bit more accurate classifier. GMMs was also tested, which assumes that each genre's feature vectors attains from a Gaussian distribution, parameterized by a mean vector and a covariance matrix, and combine these via mixing weights. The EM algorithm then iteratively tunes these parameters to maximize the log-likelihood of the data. For the GMM, we wrote an EM algorithm that initializes the Gaussian means and covariances of each component's parameters from labeled data (supervised initialization), then performs soft-assignment E-steps and M-steps to tune means and mixture weights, skipping covariances in the M-step as these are already initialized during calculations of the Mahalanobis distance.

As for the CNN classifier, it was a nice learning exercise in neural networks and artificial intelligence. Although the performance was good compared to the other classifiers in this paper, a more careful selection of features, network structure, and parameters could maybe result in an even better classifier.

All in all, the K-NN classifier attempts showed that the data could be efficiently classified with great effect just by using simple tools. There was also a lot to gain from analyzing the data through tools such as histograms and covariance matrices, which further aided in the creation of better classifiers. This exercise helped us better understand these tools of data classification and analysis, and the experience gained will aid us in the future when tackling similar projects.

Bibliography

Cinematerial (2023). *Oppenheimer Poster*. Accessed: 2024-07-21. URL: https://www.cinematerial.com/movies/oppenheimer-i15398776/p/2afqhdxx.

Duda, Richard O., Peter E. Hart and David G. Stork (2001). *Pattern Classification*. 2nd. New York, NY: Wiley-Interscience.

GeeksforGeeks (2020). *Introduction to Convolution Neural Network*. Accessed on April 27, 2025. GeeksforGeeks. URL: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ (visited on 27th Apr. 2025).

Johnsen, Magne H. (2017). *Compendium: Classification*. Course Compendium, TTT4275. Norway: Norwegian University of Science and Technology (NTNU).

McFee, Brian et al. (2015). *Librosa: Audio and Music Signal Feature Extraction*. https://librosa.org/doc/main/feature.html. Last accessed: April 27, 2025.

Tzanetakis, George (2002). *Marsyas Datasets: Audio collections for research and development*. http://marsyas.info/downloads/datasets.html.

9 Appendix

9.1 K-Nearest Neighbor Classifier

```
# python class with methods for a kNN classifier

# Create covariance matrix for each of the 10 genres
for i in range(self.num_classes):
    self.covariance_matrix.append(np.cov(self.X_train[self.Y_train==i], rowvar=False))
self.covariance_matrix = np.array(self.covariance_matrix)

def euclidean_distance(self, X1, X2):
    """
    Compute the Euclidean distance between two points.
    Parameters:
        X1 : array-like, shape (n_features,)
        First point.
        X2 : array-like, shape (n_features,)
        Second point.
    Returns:
        distance : float
            The Euclidean distance between X1 and X2.
    """
    d = np.dot(X1-X2,X1-X2)
    return np.sqrt(d)

def mahalanobis_distance(self, X1, X2):
    """
    Compute the Mahalanobis distance between two points.
    Parameters:
        X1 : array-like, shape (n_features,)
        First point.
        X2 : array-like, shape (n_features,)
        Second point.
    Returns:
        distance : float
            The Mahalanobis distance between X1 and X2.
    """
    X1genre = self.Y_train[np.where(self.X_train == X1)[0][0]]
```

```
    d = np.dot(np.dot(X1-X2,np.linalg.inv(self.covariance_matrix[X1genre])),X1-X2)
    return np.sqrt(d)


def predict(self, X_test,metric='euclidean'):
    """
    Predict the class labels for the provided test data.
    Parameters:
        X_test : array-like, shape (n_samples, n_features)
            Test data.
        metric : str, optional (default='euclidean')
            The distance metric to use ('euclidean' or 'mahalanobis').
    Returns:
        predictions : array-like, shape (n_samples,)
            Predicted class labels for the test data.
    """
    # Selects distance function
    if metric == 'euclidean':
        distance_func = self.euclidean_distance
    elif metric == 'mahalanobis':
        distance_func = self.mahalanobis_distance
    else:
        return 0
    final_output = []
    for i in range(len(X_test)):
        distances = []
        for j in range(len(self.X_train)):
            distance = distance_func(self.X_train[j] , X_test[i])
            distances.append((distance, j))
        # Sort distances and select KNN
        distances.sort(key=lambda x: x[0])
        k_nearest = distances[:self.k]

        # Get labels of neighbors and majority voting
        votes = [self.Y_train[j] for (_, j) in k_nearest]
        ans = Counter(votes).most_common(1)[0][0]
        final_output.append(ans)

    return final_output][predictions[i]] += 1
    return cm
```

9.2 Gaussian Mixture Model

```
# Expectation-Maximization (EM) algorithm for Gaussian Mixture Model (GMM)
def _e_step(self, X):
"""
E-step: compute responsibilities for each component given data.

Parameters:
    X : array-like, shape (n_samples, n_features)
        Data matrix.

Returns:
    responsibilities : ndarray, shape (n_samples, n_components)
        Posterior probabilities of each component per sample.
"""
    n_samples, n_feat = X.shape
```

```
    K = self.n_components
    resp = np.zeros((n_samples, K))

    for k in range(K):
        mu_k     = self.means[k]
        sig_k    = self.covariance_matrix[k]
        invsig_k  = np.linalg.inv(sig_k)
        detsig_k  = np.linalg.det(sig_k)
        norm_k  = self.weights[k] / (
            (2*np.pi)**(n_feat/2) * np.sqrt(detsig_k)
        )

        for n in range(n_samples):
            diff    = X[n] - mu_k
            d2      = diff.dot(invsig_k).dot(diff)
            resp[n, k] = norm_k * np.exp(-0.5 * d2) # numerator of eq. 6

    # Normalize so each row sums to 1
    resp /= resp.sum(axis=1, keepdims=True)
    return resp

def _m_step(self, X, responsibilities): # Update means, covariances, and weights based on respons
    """
    M-step: update means and weights based on current responsibilities.

    Parameters:
        X : array-like, shape (n_samples, n_features)
            Data matrix.
        responsibilities : ndarray, shape (n_samples, n_components)
            Current posterior probabilities.
    """
    # eq. 9
    N = responsibilities.sum(axis=0)
    n_samples, n_feat = X.shape

    for k in range(self.n_components):
        # eq. 7
        self.means[k]   = (responsibilities[:, k] @ X) / N[k]
        # eq. 10
        self.weights[k] = N[k] / n_samples
```

9.3 CNN classifier

```
#structuring the data
def load_and_prepare_data():
    files = [
        'Classification music/GenreClassData_5s.txt',
        'Classification music/GenreClassData_10s.txt',
        'Classification music/GenreClassData_30s.txt'
    ]

    train_data_list = []
    test_data_list = []
    datasets = []

    # -- Read files and split train/test
    for file in files:
```

```
        df = pd.read_csv(file, delimiter='\t')
        df.columns = df.columns.str.strip()  # Remove whitespace from column names

        test_df = df[df["Type"] == "Test"]
        train_df = df[df["Type"] == "Train"]

        train_labels = train_df["GenreID"].values
        test_labels  = test_df["GenreID"].values

        # Drop unnecessary columns
        train_df = train_df.drop(columns=["File", "Genre", "GenreID", "Type","Track ID"])
        test_df  = test_df.drop(columns=["File", "Genre", "GenreID", "Type","Track ID"])

        # Store the arrays
        train_data_list.append(train_df.values)
        test_data_list.append(test_df.values)

    # -- For each of (train_data_list, test_data_list),
    #    we combine the 5s, 10s, and 30s sets with the right repetition.
    for data_list in [train_data_list, test_data_list]:
        data_5s, data_10s, data_30s = data_list

        # Repeat to match sequence length
        data_10s_repeated = np.repeat(data_10s, 2, axis=0)
        data_30s_repeated = np.repeat(data_30s, 6, axis=0)

        # Concatenate along the feature dimension
        combined_data = np.concatenate((data_5s, data_10s_repeated, data_30s_repeated), axis=1)


        # The first pass is for train data, second for test data
        if len(datasets) == 0:
            datasets.append([combined_data, train_labels])
        else:
            datasets.append([combined_data, test_labels])

    # datasets[0] = train, datasets[1] = test
    return datasets[0], datasets[1]

# Normalizing the training dataset
global_mean = tf.reduce_mean(train_ds, axis=(0, 1))
global_std = tf.math.reduce_std(train_ds, axis=(0, 1))
def normalize_data(x, y):
    x = tf.cast(x, tf.float64)  # Ensure the feature tensor remains float64
    x = (x - global_mean) / global_std
    y = tf.cast(y, tf.int64)  # Ensure the label tensor remains int64
    return x, y

#normalizing and shuffling training data
tf_train_ds = (
    tf_train_ds
    .map(normalize_data)
    .shuffle(buffer_size=700)
    .batch(batch_size)
    .prefetch(tf.data.AUTOTUNE)
)

#CNN model used for classification
```

```
model = Sequential([
    tf.keras.layers.Input(shape=(6, 186, 1)),  # Add channel dimension if missing
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # 10 genres
])

#compiling the model
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy']
)

#evaluating the model
best_model = tf.keras.models.load_model('models/MLP_modelv2.keras')
# Evaluate the best model on the test dataset
test_loss, test_accuracy = best_model.evaluate(tf_test_ds, verbose=1)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")

y_true = []
y_pred = []

# Iterate over the dataset
for x_batch, y_batch in tf_test_ds:
    # x_batch has shape (batch_size, 6, 186) { keep as is
    predictions = model.predict(x_batch, verbose=0)  # No need to expand dims
    y_true.extend(y_batch.numpy())                   # Append all true labels
    y_pred.extend(np.argmax(predictions, axis=1))    # Get predicted class

# Convert to numpy arrays
y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Compute and plot confusion matrix
cm = confusion_matrix(y_true, y_pred)
print(cm)
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 6, 186, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 3, 93, 32)	0
conv2d_3 (Conv2D)	(None, 3, 93, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 1, 46, 64)	0
flatten_1 (Flatten)	(None, 2944)	0
dense_2 (Dense)	(None, 128)	376,960
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 10)	1,290

Total params: 397,066 (1.51 MB)

Figure 6: Model parameters and structure of the CNN network