Bachelor thesis

Artificial Intelligence

Radboud University 

Music Genre Classification Using Gaussian Process Models and Gibbs samplers

Author:
Ella aan de Stegge
s4808819

First supervisor:
dr. M. Hinne
Artificial Intelligence
m.hinne@donders.ru.nl

Second supervisor:
dr P.L. Lanillos Pradas
Artificial Intelligence
p.lanillos@donders.ru.nl

April 21, 2021

Abstract

Music genres are groups with certain musical characteristics. They can be used to create structure within the large amount of music that exists, for example via genre classification. Genre classification has been automated using Machine Learning. In previous research, Gaussian Processes have successfully been used to classify genres with a reasonable accuracy (60,9% over 6 genres), using relatively few samples (20 per genre).

In this paper, several genre classification approaches are compared: a Gaussian Process, a Gibbs sampler and as a baseline a Support Vector Machine. First, the optimal kernel for both the Gaussian Process and Support Vector Machine is determined. Then, the models are compared. The models are trained with a range of different sample sizes and accuracy is determined using the same testing set. The accuracies that the models achieve are then compared.

Using the Support Vector Machine with the polynomial kernel yielded the highest accuracies with the polynomial kernel and the Gaussian Process achieved the highest accuracies with the Matèrn kernel. When trained on the same data, the Gaussian Process outperforms the Support Vector Machine. The Gaussian Process reaches an accuracy of 73,1%, while the Support Vector Machine reaches an accuracy of 61,7%. The Gibbs sampler proves to perform relatively poor on this classification problem, as it yields an accuracy of 48,2%.

Contents

Chapter 1

Introduction

We are quick and relatively reliable in guessing the genre of a song. How quick and reliable exactly? After three seconds of a song, we can make a guess that is accurate 70% of the time [Gjerdingen and Perrott, 2008]. Music genres are used to organize the massive amount of music that exists into groups that bear a certain similarity with each other. That similarity can be strictly musical, but also cultural or historical. The advantage of clustering by genre is that genres are closely related to music preferences. Chances are that if you like a song from a certain genre, you will like other songs from that genre.

Music Information Retrieval (MIR) is the science of retrieving features of music. Examples of such features are: genre, emotion, artist, instruments, melody and chord progression. In turn, these features can be used for music recommendation, creation and manipulation. Machine learning is one of the disciplines used in MIR, and it is easily applied as enough labelled music training data exists. Still, determining the genre of an audio sample is a difficult and only partially objective task. Genres change over time, different countries have different genres, genres have overlap and subgenres exist. All the more reason to automate the process of categorizing music into genres, especially for large-scale, international music distributors.

Automatic genre classification can be done by several machine learning methods. The two methods I am interested in are Gaussian Processes and Gibbs samplers. Both are methods based on Bayesian statistics. Bayesian statistics is a field of statistics where observed data is used to update beliefs about the parameters of a statistical model. It is based on Bayes' theorem, where likelihood, prior and marginalization are used to compute the posterior.

Gaussian Processes assign probabilities to all possible functions that may fit the data. The observed training data affects the probability for these functions. If the observed data can be explained by a certain function, that function becomes more probable. If not, that function becomes less

probable. The second method, Gibbs sampling, is a Markov Chain Monte Carlo (MCMC) algorithm. It generates a Markov Chain of all samples, where each sample is correlated with nearby samples. Gibbs sampling can be used to find unknown parameters.

Lansdown compared music genre classification via a Support Vector Machine (SVM), an Artificial Neural Network, a Random Forest and a Gradient Boosting Machine with 5 genres: Electronic, Folk, Hiphop, Pop and Rock [Lansdown, 2019]. The SVM outperformed the other approaches with an accuracy of 68%. Furthermore, the SVM performed more consistently over all genres, whereas the other models yielded a lower accuracy recognising the Pop-genre. Lansdown also evaluates multiple feature extraction methods and the audio feature's value in this task, which will be further discussed later in this thesis.

When an SVM and a Gaussian Process model were compared on their classification accuracy with different sample sizes, the latter outperformed the SVM in all cases [Markov and Matsui, 2013]. This experiment shows the efficiency of a Gaussian Process: it needs less samples to reach a higher accuracy. The samples were classified in six genres: Classical, Electronic, Jazz-Blues, Metal-Punk, Rock-Pop and World. Additionally, multiple kernels were compared: linear, exponential, rational and Matèrn. The performance gap between the GP and SVM was largest with the exponential GP kernel.

Raczynski & Vincent used Gibbs sampling for music genre classification [Raczynski and Vincent, 2014]. They used a topic model, Latent Dirichlet Allocation (LDA), to model the data. In this study, the observed data consisted of musical chords. They used Gibbs sampling to identify parameters that explain the data. They used 9 genres: Pop, Blues, Celtic, Bop, Bossanova, Prebop, Romanticism, Classical, Baroque. In their experiment, LDA yielded an accuracy of 60,4%.

The approach in this paper is different: LDA is not used to model the data. Instead, the data is modelled via a Dirichlet mixture model. This is a relatively straight-forward model that will be explained in Chapter 2.

In this paper, the efficiency of Gaussian Processes and Gibbs sampling are compared with a Support Vector Machine acting as a baseline classifier. Training set size and kernel type will be varied between experiments. Training set sizes range from 20 samples per music genre to 100 samples per music genre. The testing set holds 120 samples per music genre. Seven music genres are used: Rock, Electronic, Hiphop, Folk, Historic, Pop and Classical. Prior to the experiments, optimal kernels are first identified for the GP and SVM. Ultimately, all models are compared.

I expect to find a difference between the GP and SVM: the GP will yield a higher accuracy than the SVM. I also expect a difference between the Gibbs sampler and SVM: the Gibbs sampler will yield a higher accuracy than the SVM.

The results of this research will be of interest for automatic genre classification. The advantage of Bayesian techniques for automatic genre classification is that the beliefs can be updated once more data becomes available. This is beneficial for music distributors, as new genres can emerge and grow over time. New data would then become available and could be used to update the model.

Chapter 2

Methods

2.1 Data

I will use the Free Music Archive (FMA) [Defferrard et al., 2017] as a database for this research. The FMA is a dataset created for Music Information Retrieval. It holds raw mp3-files, as well as metadata and preprocessed features. The metadata holds the information about the genre, which is used to create output array Y. Y is the desired output of the models, a list of genres. The preprocessed audio features that are used as input data X are provided by Echonest. Echonest is now owned by Spotify, and the preprocessing of audio features is included in Spotify for Developers [Spotify,]. The used audio features are summarized in Table 2.1.

Feature	**Value**
Acousticness	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Danceability	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Energy	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Instrumentalness	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Liveness	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Speechiness	$x \in \mathcal{R}$ and $0 \leq x \leq 1$
Tempo	Measured in beats per minute (BPM) ranging between 12 - 252
Valence	$x \in \mathcal{R}$ and $0 \leq x \leq 1$

Table 2.1: The features used for music genre classification

Almost all of those features are already scaled from 0 to 1, and to ensure easy Automatic Relevance Detection (ARD), the tempo feature is also scaled from 0 to 1. This way the length scales in the Gaussian Process kernel will give clear insight into the relevance of the individual features. ARD is a

method that can be implemented in models to determine the relevance of every feature. The implementation of ARD is further explained in Section 2.2.1.

The Echonest features were only calculated for a subset of the samples in the FMA. In this subset the genres are distributed as seen in Table 2.2.

Genre	**Number of samples**	**Y value**
Rock	3892	0
Electronic	2170	1
Hiphop	910	2
Folk	874	3
Historic	357	4
Pop	346	5
Classical	265	6
Jazz	*241*	-
International	*133*	-
Instrumental	*84*	-
Experimental	*17*	-

Table 2.2: The genres of the samples in the dataset

The experiments are performed using a range of different training set sizes and one testing set size. The training set sizes are 20, 40, 60, 80 and 100 samples per genre. The testing set size is 120 samples per genre. There should be no overlap between the training sets and testing set, so there should be at least 220 samples per genre. Therefore the genres International, Instrumental and Experimental are excluded. Jazz is also excluded because it barely exceeds the minimum amount of samples. That leaves seven genres: Rock, Electronic, Hiphop, Folk, Historic, Pop and Classical. For every experimental trial, a testing set with 120 samples per genre is constructed. Then, the five different training sets are constructed: train-20, train-40, train-60, train-80 and train-100. The training sets are constructed in such a way that $train\text{–}20 \subset train\text{–}40 \subset train\text{–}60 \subset train\text{–}80 \subset train\text{–}100$.

2.2 Models

The problem I will tackle is a classification problem. The models are trained with training samples with their corresponding classes, and should predict which class the sample is in, based on an input vector of 8 features per sample. A Gaussian Process, Gibbs sampling and a Support Vector Machine are compared on their prediction accuracy on the testing set.

2.2.1 Gaussian Process

The goal of a Gaussian process is to learn the parameters that best describe the supervised training data. In the case of classification, the Gaussian Process predicts the probability π_c that sample $\mathbf{x}$ is in class c. The π_c is based on the latent functions $\mathbf{f} = [f_1, ...f_c]$, that predict the degree of membership to class C. The latent functions are described by an independent Gaussian Process prior with a mean vector μ and a kernel function $K = k(\mathbf{x}, \mathbf{x}')$:

$$f \sim GP(\mu, K) \tag{2.1}$$

Multiple kernel functions can be used. One of the possible kernel functions is the squared exponential function: $k(x, x') = exp(-\frac{(|x - x'|)^2}{2l^2})$. Here, x and x' are two datapoints, $|x - x'|$ is the Euclidian distance between the two datapoints and l is the lengthscale. Because the output of the kernel function is divided by the lenghtscale, a large lengthscale results in a smaller output. This means that features with a large lengthscale become almost irrelevant because they do not affect the covariance much. Therefore, the lengthscales can be used for Automatic Relevance Determination.

In the research, four kernels are evaluated: Linear, Squared Exponential, Rational Quadratic and Matérn 3/2. The Linear kernel function is $k(x, x') = \sigma^2 + (x - c)(x' - c)$, where σ^2 is the variance. The Matérn 3/2 kernel function is $k(x, x') = \sigma^2 + \sqrt{3 * (\frac{|x-x'|}{l}} \exp(-\sqrt{3 * (\frac{|x-x'|}{l}})$. The Rational Quatratic kernel function is $k(x, x') = \sigma^2(\frac{1 + |x - x'|^2}{2\alpha l^2})(- \alpha)$. Here α determines the relative weighting of fluctuations. These kernel functions affect the shape the latent functions can take on. To find the optimal Gaussian Process for this music genre classification problem, the first experiment compares the four kernels.

The results of the latent functions are not probabilities of class membership yet. The results are turned into a discrete probability distribution over all classes by using a Robust Maximum function:

$$\pi_c(\mathbf{f}) = \begin{cases} 1 - \epsilon, & \text{if } c = arg\,max_c f_c \\ \epsilon/(C - 1), & \text{otherwise} \end{cases} \tag{2.2}$$

The goal of the Gaussian Process is to find the function that best explains and predicts the data. Finding the best function is a matter of calculating $p(\mathbf{f} \mid y)$, where y are the observed variables. If the likelihood was Gaussian, calculating $p(\mathbf{f} \mid y)$ could be done under closed form with matrix multiplication. But the likelihood is Categorical, not Gaussian. Therefore the posterior needs to be approximated with a Variational Gaussian Process.

A Variational Gaussian Process is used to find the multivariate Gaussian distribution that is closest to the posterior. The optimal Gaussian

approximation for the posterior is denoted by $\hat{q}(\mathbf{f})$. The multivariate Gaussian distribution consists of mean value μ and covariance matrix Σ. Finding the optimal values for these hyperparameters is done by minimizing the Kullback-Leibler divergence. The Kullback-Leibler divergence measures how different the approximation is from the posterior:

$$\mathrm{KL}(\mathrm{q}(\mathbf{f}) \parallel \mathrm{p}(\mathbf{f})) = \int \mathrm{q}(\mathbf{f}) \ln \frac{\mathrm{q}(\mathbf{f})}{\mathrm{p}(\mathbf{f} \mid \mathbf{y}, \theta)} d\mathbf{f} \tag{2.3}$$

This formula is not tractable, because the posterior $p(\mathbf{f} \mid y)$ is unknown. Fortunately, Opper and Archambeau found that the Gaussian distribution with the smallest Kullback-Leibler divergence can be approximated as follows [Opper and Archambeau, 2009] :

$$\hat{q}(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}, [\mathbf{K}^{-1} + \mathrm{diag}(\boldsymbol{\lambda})]^{-1}) \tag{2.4}$$

Where μ is the mean, K is the kernel and $\mathrm{diag}(\boldsymbol{\lambda})$ is a diagonal matrix with $\boldsymbol{\lambda}$ on its diagonal.

For the experiments in this paper, the package GPflow [Matthews et al., 2017] is used to build Gaussian Processes in Python. GPflow uses Opper and Archambeau's findings to find the optimal approximation.

2.2.2 Gibbs sampler

A Gibbs sampler is a special case of the Metropolis-Hastings algorithm. It is used to obtain random samples from complex distributions. The algorithm is based on the premise that sampling from conditional distributions $P(A \mid B)$ and $P(B \mid A)$ is easier than sampling from the joint distribution $P(A, B)$. The joint distribution can be approximated from the samples drawn from these conditional distributions. Figure 2.1 shows pseudo-code for the Gibbs sampling algorithm.

Initialization: Initialize $\mathbf{x}^{(0)} \in \mathcal{R}^D$ and number of samples N

- **for** $i = 0$ to $N - 1$ **do**
- $x_1^{(i+1)} \sim p(x_1|x_2^{(i)}, x_3^{(i)}, ..., x_D^{(i)})$
- $x_2^{(i+1)} \sim p(x_2|x_1^{(i+1)}, x_3^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_j^{(i+1)} \sim p(x_j|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, ..., x_D^{(i)})$
- $\vdots$
- $x_D^{(i+1)} \sim p(x_D|x_1^{(i+1)}, x_2^{(i+1)}, ..., x_{D-1}^{(i+1)})$

return $(\{\mathbf{x}^{(i)}\}_{i=0}^{N-1})$

Figure 2.1: Pseudo-code for Gibbs sampler [Mikhailiuk, 2019]

A Gibbs sampler can find find the hyperparameters that best explain the observed data, which consist of X_{train} and Y_{train}, and to predict the class values for testing data X_{test}. The first step of initializing a Gibbs sampler is determining the relations between different variables and finding the underlying structure. The genre classification problem can be described with a Dirichlet prior and Categorical likelihood. Figure 2.2 is the generative model for this genre classification problem and the graphical model is drawn in Figure 2.3.

$$
\begin{aligned}
\alpha 1 &= (1,1,1,1,1,1,1,1) && \\
p &\sim \text{Dirichlet}\,(\alpha 1) && \\
g_m \mid p &\sim \text{Categorical}\,(p) && m = 1...M \\
\mu_{ij} &\sim \text{Beta}\,(1,1) && i = 1...C, j = 1...F \\
\sigma_{ij} &\sim \text{Gamma}\,(0.01, 0.01) && i = 1...C, j = 1...F \\
\alpha_{ij} \mid \mu_{ij}, \sigma_{ij} &= \mu_{ij} * \sigma_{ij}^2 && i = 1...C, j = 1...F \\
\beta_{ij} \mid \mu_{ij}, \sigma_{ij} &= (1 - \mu_{ij}) * \sigma_{ij}^2 && i = 1...C, j = 1...F \\
Xtrain_{nj} \mid \alpha_{aj}, \beta_{aj} &\sim \text{Beta}\,(\alpha_{aj}, \beta_{aj}) && a = Ytrain_n, n = 1...N, j = 1...F \\
Xtest_{mj} \mid \alpha_{bj}, \beta_{bj} &\sim \text{Beta}\,(\alpha_{bj}, \beta_{bj}) && b = g_m, m = 1...M, j = 1...F
\end{aligned}
$$

Figure 2.2: Generative model for Genre Classification problem

Here M is the number of testing samples, N is the number of training samples, C is the number of classes, and F is the number of features. The

generative model consists of a mean μ, variance σ^2 and shape parameters α and β based on the values of μ and σ^2 for every feature and every class. The priors for both μ and σ are uniform. Data points in X_{train} are drawn from a Beta distribution with parameters α and β depending on the class that that data point falls into. The classes are specified in Y_{train}.

The predictions for the genres of the testing data X_{test} are stored in g. The list g is drawn from a Categorical distribution, which takes the list of event probabilities, p, as a parameter. p is drawn from a Dirichlet distribution, which takes the list $\alpha 1$, the concentration parameters. The data points in X_{test} are then drawn from a Beta distribution, where the class indices are taken from the computed list of genres g.

It is important to note that shape parameters α and β cannot be zero. This is implemented in the program with an if-clause. If $\mu_{ij} * sigma_{ij} = 0$, $\alpha_{ij} = 0.000001$ and if $(1 - \mu_{ij}) * \sigma_{ij} = 0$, then $\beta_{ij} = 0.000001$.

The Gibbs sampler will find optimal values for parameters μ and σ, so that predictions become as accurate as possible. The predictions are stored in vector g. g is compared with Y_{test} to compute the accuracy.

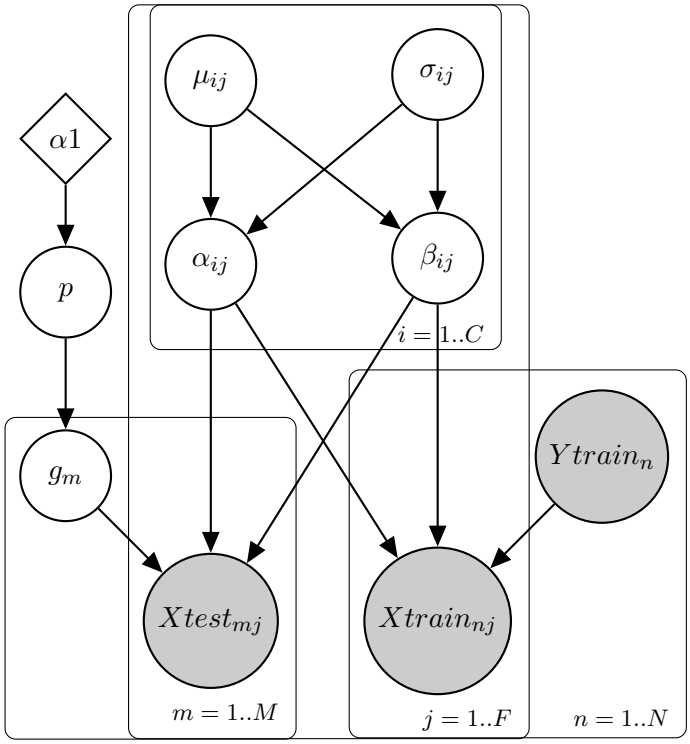

Figure 2.3: Graphical model of Gibbs sampler

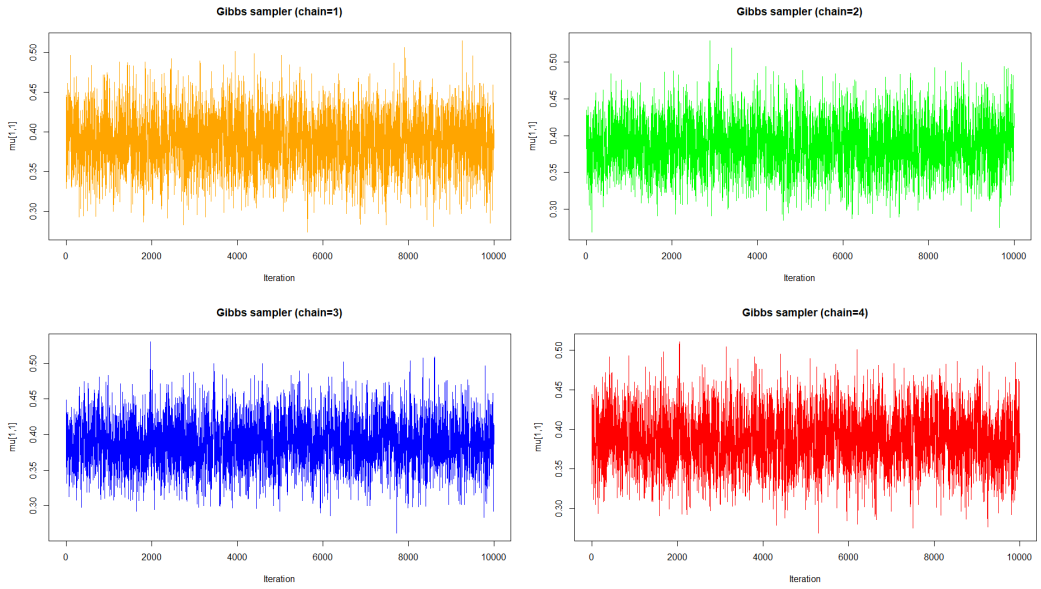

Figure 2.4: The value of mu[1,1] over 10.000 iterations and 4 chains

It is common to throw away an amount of samples at the beginning of a chain. It can take a while before a stationary distribution is reached and then the samples may not be an accurate reflection of the distribution. To evaluate whether a burn-in period is necessary, Figure 2.4 shows the value of $\mu[1,1]$ over a large amount of iterations. This demonstrates that there is no visible significant convergence and thus no significant burn-in period. For the experiments, a burn-in period of 200 iterations is used. This was chosen as a starting point and yielded positive results. Therefore, this burn-in period was retained in all experiments using the Gibbs Sampler. In addition, 10.000 iterations did not provide more specific results than 1.000 iterations and they prove to be very time-consuming. Therefore I decided to work with 1.000 iterations per chain.

2.2.3 Support Vector Machine

The goal of a Support Vector Machine is to find a hyperplane that classifies the data points correctly. A Support Vector Machine finds the optimal hyperplane: one that maximizes the margin. A large margin means a bigger distance between data points of different classes. To maximize the margin, you need to minimize the loss function.

The SVM works with different kernels. In the first experiment, the linear kernel, Radial-Basis Kernel and polynomial Kernel are compared. The optimal kernel is then used for the comparison with the other techniques.

SVM's are implemented in the Scikit-learn library [Pedregosa et al., 2011].

It is a relatively quick and easy algorithm and therefore functions as a good baseline performance measure.

Chapter 3

Results

In the first two sections, the two best kernels are determined for the SVM and the GP. Then the performance of the SVM and GP with the optimal kernels are compared with the Gibbs sampler.

3.1 SVM Kernels

The first experiment consists of finding the most applicable kernel for the Support Vector Machine. There are 50 trials performed for each kernel and each condition. The results of this experiment are shown in Figure 3.1.

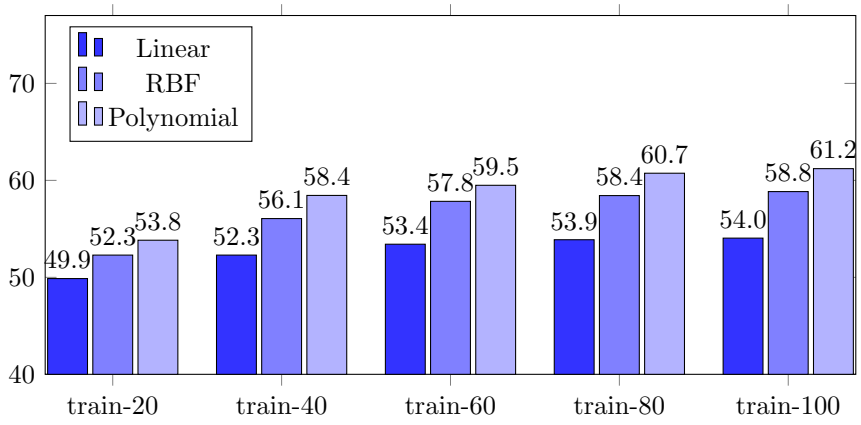

Figure 3.1: Average accuracy over 50 trials of Support Vector Machines with different kernels

The Support Vector Machine with the polynomial kernel outperforms the other kernels in each condition. It yields an accuracy of 61.21% with 100 training samples. The RBF kernel yields the second highest accuracies and the linear kernel the lowest. The polynomial kernel is used in the final comparison between the SVM, Gibbs sampler and Gaussian Process.

3.2 GP Kernels

The second experiment consists of finding the most applicable kernel for the Gaussian Process. There are 20 trials performed for each kernel and condition. The results of this experiment are shown in Figure 3.2.

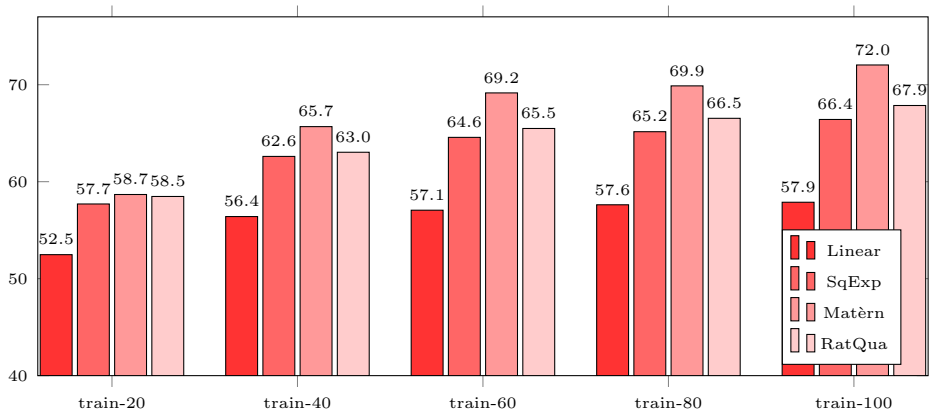

Figure 3.2: Average accuracy over 20 trials of Gaussian Processes with different kernels

The GP with the Matèrn kernel outperforms the other kernels in each condition. It yields an accuracy of 72%. The difference with the other kernels becomes bigger when more training samples are used. The Rational Quadratic kernel and Squared Exponential kernel yield similar accuracies. The Linear kernel achieves the lowest accuracy in all cases. The Matèrn kernel is used in the final comparison between the SVM, Gibbs sampler and Gaussian Process.

3.3 GP vs. SVM vs. Gibbs sampler

The last experiment is a comparison of all three approaches. The accuracies for the SVM and GP are computed over 50 trials. The accuracies for the Gibbs sampler are computed over one trial due to time constraints. One trial consists of 4 chains and 1000 iterations. The accuracies are computed over all those iterations minus the burn-in iterations. The results of this experiment are shown in Figure 3.3.

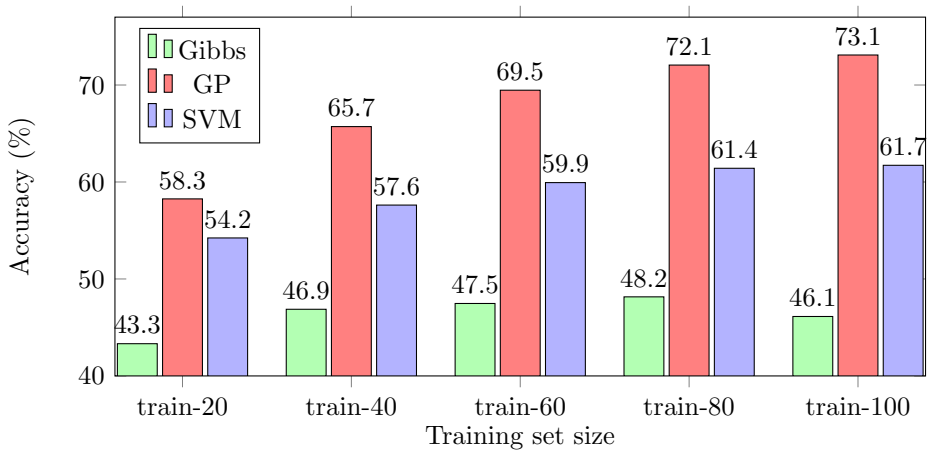

Figure 3.3: Results of all three techniques

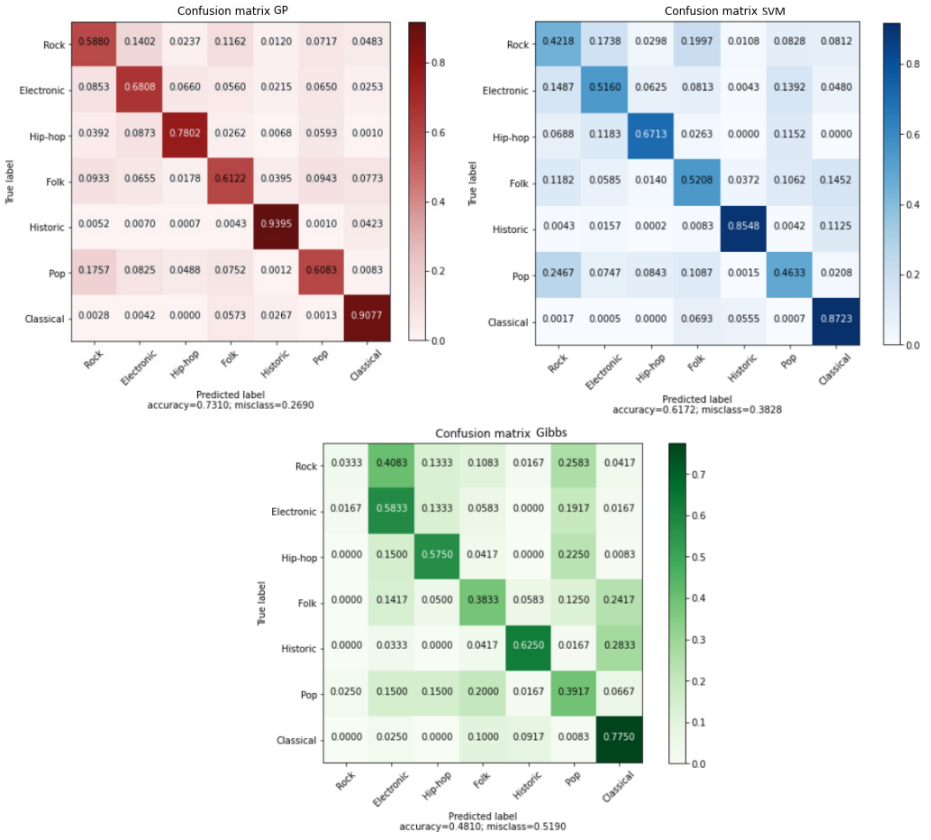

Figure 3.4: Confusion matrices for all three approaches, trained with train-100

The Gaussian Process outperforms the Support Vector Machine and Gibbs sampler in all conditions. With 100 training samples per genre, the GP yields an accuracy of 73,1%, the SVM yields an accuracy of 61,7 % and the Gibbs sampler yields an accuracy of 46,1%. Interestingly, the baseline classifier SVM outperforms the Gibbs sampler. The differences between the accuracies of the GP and the SVM become bigger with more training samples. The accuracy of the Gibbs sampler increases when more training samples are used, until 80 training samples are used. However, when 100 training samples are used, the accuracy decreases.

Figure 3.4 shows the confusion matrices for each method. The confusion matrices of the GP and SVM are rather similar, but the confusion matrix of the Gibbs sampler deviates. The Rock genre has an puzzling low sensitivity. Only 3,3% of Rock samples are correctly categorized.

3.4 Automatic Relevance Detection

The lengthscales of Gaussian Processes inherently correspond with the relevance of features. Table 3.1 shows the lengthscales of a GP using the Matèrn 3/2 kernel.

Feature	**Lengthscale** (l)	$1/l$
Acousticness	0.617	1.62
Danceability	0.583	1.715
Energy	0.525	1.905
Instrumentalness	1.112	0.899
Liveness	0.445	2.249
Speechiness	0.469	2.132
Tempo	2.034	0.492
Valence	0.76	1.316

Table 3.1: Lengthscales of all features of a GP using a Matèrn 3/2 kernel

Liveness has the smallest lengthscale and therefore differences in liveness are very relevant. Tempo has the smallest lengthscale and therefore differences in tempo are less relevant.

Chapter 4

Conclusions

In conclusion, the polynomial kernel of the SVM proved to be best for this music classification problem. For the Gaussian Process, the best kernel turned out to be the Matèrn kernel. The Gaussian Process outperforms both the Support Vector Machine and the Gibbs sampler on this particular music classification problem. It achieved a high accuracy of 73,1%.

The relevance of the Echonest features were found with ARD. Liveness proved to be the most relevant feature. The order of features from most relevant to least relevant is: liveness, speechiness, energy, danceability, acousticness, valence, instrumentalness, tempo.

The Gibbs sampler's performance was unexpectedly poor and did not exceed the SVM's baseline performance. It could be interesting to look into the reason why Rock samples were classified so poorly. The low accuracy of Rock samples causes a great decrease in the average accuracy. Interestingly, Rock samples were often misclassified as Electronic samples. It could be an improvement to combine both classes into one large Rock/Electronic class, classify them with a variation of the current model and use a binary classification model to classify Rock/Electronic samples into their respective classes. However, this would increase the running time, which is already a matter of concern.

Another improvement that could be made, is using a collapsed Gibbs sampler instead of the normal Gibbs sampler. Collapsed Gibbs samplers are known for their quick convergence. In the experiments there was no visible convergence, maybe the collapsed Gibbs sampler would reach a more significant stationary distribution.

The data and how it was handled is not a true reflection of real-world music. The training sets and testing set consisted of equally distributed genres, even though some genres are more present in the real-world than others. In future research, this could be reflected in the data and the models. For example, the models could be trained with balanced data, but tested with unbalanced data.

Another issue I ran into, was the time-consuming nature of the models. While the SVM was quick, the other models were not. Especially the Gibbs sampler was time-consuming. The Gibbs sampler was only tested with one random series of training sets and one test set. The SVM and GP were tested with 50 trials, where the training sets and testing set were different. Perhaps the Gibbs sampler needs more trials to accurately portray the accuracy of the approach.

Lastly, the Gaussian Process and Gibbs sampler are significantly slower than the SVM. Such practical factors were not taken into account in the evaluation of the methods for this research. However, for real world applications, such considerations are very important next to accuracy.

Overall, the results of the Gaussian Process and Support Vector Machine are impressive. Perhaps Gibbs samplers are better used for feature extraction than for classification.

Bibliography

[Defferrard et al., 2017] Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. (2017). FMA: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference (ISMIR)*.

[Gjerdingen and Perrott, 2008] Gjerdingen, R. O. and Perrott, D. (2008). Scanning the dial: The rapid recognition of music genres. *Journal of New Music Research*, 37(2):93–100.

[Lansdown, 2019] Lansdown, B. (2019). *Machine Learning for Music Genre Classification*. PhD thesis.

[Markov and Matsui, 2013] Markov, K. and Matsui, T. (2013). Music genre classification using gaussian process models. *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*.

[Matthews et al., 2017] Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6.

[Mikhailiuk, 2019] Mikhailiuk, A. (2019). Sampling distributions with an emphasis on gibbs sampling, practicals and code.

[Opper and Archambeau, 2009] Opper, M. and Archambeau, C. (2009). The variational gaussian approximation revisited. *Neural Computation*, 21(3):786–792.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Raczynski and Vincent, 2014] Raczynski, S. A. and Vincent, E. (2014). Genre-based music language modeling with latent hierarchical pitman-yor process allocation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 22(3):672–681.

[Spotify,] Spotify. Spotify for developers. https://developer.spotify.com/.

Appendix A

Appendix

A.1 Data preparation (Python)

```
1  import numpy as np
2  import pandas as pd
3
4  #Get echonestdata
5  Xdf = en[('echonest','audio_features')]
6
7  #Echonest was only applied to a subset, so I gather the
       track_id's to crossreference with the Y data.
8  xs = Xdf.index.values.tolist()
9  xs = np.asarray(xs)
10 print(xs.shape)
11
12 Ydf = tracks[('track', 'genre_top')]
13 Ydf = Ydf.replace({'Hip-Hop': 0, 'Rock':1, 'Pop':2, '
       International':3, 'Electronic':4, 'Experimental':5, '
       Instrumental':6, 'Folk':7, 'Classical':8, 'Jazz':9, '
       Blues':10, 'Old-Time / Historic':11})
14 all_genre_labels = ['Hip-Hop', 'Rock', 'Pop', '
       International', 'Electronic', 'Experimental', '
       Instrumental', 'Folk', 'Classical', 'Jazz', 'Blues', '
       Historic', 'No top genre']
15
16 #Only take the echonest subset of the track info
17 Ydf = Ydf.loc[xs]
18
19 #Convert to numpy array
20 Y = np.asarray(Ydf)
21 X = np.asarray(Xdf)
22
23 #Scale the features in X
24 for i in range(X.shape[1]):
25   min=np.min(X[:,i])
```

```
26      max=np.max(X[:,i])
27      X[:,i]=(X[:,i]-min)/(max-min)
28
29  hiphop_X = X[Y==0]
30  rock_X = X[Y==1]
31  pop_X = X[Y==2]
32  int_X = X[Y==3]
33  elec_X = X[Y==4]
34  exp_X = X[Y==5]
35  inst_X = X[Y==6]
36  folk_X = X[Y==7]
37  class_X = X[Y==8]
38  jazz_X = X[Y==9]
39  blues_X = X[Y==10]
40  hist_X = X[Y==11]
41  nan_X = X[pd.isna(Y)]
42
43  #Samples without top genre are useless in this experiment
44  #Use genres with at least 200 samples
45  genre_labels= ['Rock', 'Electronic', 'Hip-hop', 'Folk', '
        Historic', 'Pop', 'Classical']
46  datasets = [rock_X, elec_X, hiphop_X, folk_X, hist_X, pop_X
        , class_X]
47
48  c = len(datasets)
```

A.2 GP Code (Python)

```
1   import gpflow
2   import tensorflow as tf
3
4   test_size = 120
5   c = 7
6   f = 8
7
8   def gp_test(X_train, Y_train, X_test, Y_test, kernel):
9       """This method initializes an GP with a kernel, trains
            it with X_train and Y_train and tests it with X_test
             and Y_test.
10
11      Input Arguments: X_train (array: n by f), Y_train (
            array: n by 1), X_test (array: m by f), Y_test (
            array: m by 1), kernel
12      Returns: accuracy of the GP's prediction
13      """
14      # Robustmax Multiclass Likelihood
15      invlink = gpflow.likelihoods.RobustMax(c)  # Robustmax
            inverse link function
```

```
16      likelihood = gpflow.likelihoods.MultiClass(c, invlink=
            invlink) # Multiclass likelihood
17      data = (X_train, Y_train)
18
19      m = gpflow.models.VGP(
20          data,
21          kernel=kernel,
22          likelihood=likelihood,
23          num_latent_gps=c,
24      )
25
26      #Optimize model
27      optimizer = gpflow.optimizers.Scipy()
28      optimizer.minimize(m.training_loss,
            m.trainable_variables, options={"maxiter": 100})
29
30      pred, var = m.predict_y(X_test)
31      pred = np.argmax(pred, axis=1)
32      correct = np.sum(pred == Y_test[:,0])
33      accuracy = (correct/Y_test.shape[0])*100
34      return accuracy
35
36  def run_experiment3(datasets, trials):
37      """This method creates X_train, X_test, Y_train and
            Y_test for each trial, and finds the accuracies GP
            yields with each kernel
38
39      Input Arguments: datasets (list of arrays), trials (
            integer)
40      Returns: array of accuracies
41      """
42      train_sizes = np.array([20, 40, 60, 80, 100])
43      n_kernels = 4
44      max_train_size = train_sizes[4]
45      conditions = train_sizes.shape[0]
46      accuracies_gp = np.empty((n_kernels, conditions, trials
            ))
47      for i in range(trials):
48          #Initialize X_train and X_test
49          X_train = []
50          X_test = []
51          #Fill them with samples from each class
52          for d in datasets:
53              d2 = np.copy(d)
54              random.shuffle(d2)
55              test=d2[:test_size]
56              X_test.append(test)
57              train=d2[test_size:test_size+max_train_size]
58              X_train.append(train)
```

```
59
60          #Create Y_test
61          X_test = np.concatenate(X_test)
62          Y_test = np.zeros((test_size*c, 1))
63          for k in range(c):
64              Y_test[k*test_size:(k+1)*test_size, 0] = k
65
66          #Shuffle X_test and Y_test
67          idx = np.arange(c*test_size)
68          np.random.shuffle(idx)
69          Y_test = Y_test[idx]
70          X_test = X_test[idx]
71
72          X_train = np.asarray(X_train)
73
74          for j in range(conditions):
75              #Take the amount of samples for this condition
76              train_size = train_sizes[j]
77              X_train2 = np.concatenate(X_train[:,0:
                    train_size])
78              Y_train = np.zeros((train_size*c, 1))
79              #Create Y_train
80              for k in range(c):
81                  Y_train[k*train_size:(k+1)*train_size, 0] =
                        k
82              #Shuffle X_train and Y_train
83              idx = np.arange(c*train_size)
84              np.random.shuffle(idx)
85              X_train2 = X_train2[idx]
86              Y_train = Y_train[idx]
87
88              #Initialize kernels
89              k1 = gpflow.kernels.SquaredExponential(
                    lengthscales=[1]*f)
90              k2 = gpflow.kernels.Matern32(lengthscales=[1]*f
                    )
91              k3 = gpflow.kernels.Linear(variance=0.1)
92              k4 = gpflow.kernels.RationalQuadratic(variance
                    =0.1, lengthscales=[1]*f)
93              gp_kernels = [k1,k2,k3,k4]
94
95              #Find accuracies for each kernel
96              for n in range(n_kernels):
97                  kernel = gp_kernels[n]
98                  accuracies_gp[n, j, i]=gp_test(X_train2,
                        Y_train, X_test, Y_test, kernel)
99      return accuracies_gp
100
101 trials = 10
```

```
102 exp3_accuracies = run_experiment3(datasets, trials)
```

A.3 SVM Code (Python)

```
1  from sklearn import svm
2
3  test_size = 120
4  c = 7
5  f = 8
6
7  def svm_test(X_train, Y_train, X_test, Y_test, kernel):
8    """This method initializes an SVM with a kernel, trains
       it with X_train and Y_train and tests it with X_test
       and Y_test.
9
10     Input Arguments: X_train (array: n by f), Y_train (
         array: n by 1), X_test (array: m by f), Y_test (
         array: m by 1), kernel
11     Returns: accuracy of the SVM's prediction
12     """
13     # Initialize SVM
14     svm_model = svm.SVC(kernel=kernel)
15     Y_train = Y_train.reshape(Y_train.shape[0])
16     # Train SVM
17     svm_model.fit(X_train, Y_train)
18     # Test SVM
19     pred = svm_model.predict(X_test)
20     correct = np.sum(pred == Y_test[:,0])
21     accuracy = (correct/Y_test.shape[0])*100
22     return accuracy
23
24 def run_experiment2(datasets, svm_kernels, trials):
25     """This method creates X_train, X_test, Y_train and
         Y_test for each trial, and finds the accuracies SVM
         yields with each kernel
26
27     Input Arguments: datasets (list of arrays), svm_kernels
         (list of kernels compatible with SVM), trials (
         integer)
28     Returns: array of accuracies
29     """
30   train_sizes = np.array([20, 40, 60, 80, 100])
31   n_kernels = len(svm_kernels)
32   max_train_size = train_sizes[4]
33   conditions = train_sizes.shape[0]
34   accuracies_svm = np.empty((n_kernels, conditions, trials)
       )
35   for i in range(trials):
```

```
36      #Initialize X_train and X_test
37      X_train = []
38      X_test = []
39
40      #Fill them with samples from each class
41      for d in datasets:
42        d2 = np.copy(d)
43        random.shuffle(d2)
44        test=d2[:test_size]
45        X_test.append(test)
46        train=d2[test_size:test_size+max_train_size]
47        X_train.append(train)
48
49      #Create Y_test
50      X_test = np.concatenate(X_test)
51      Y_test = np.zeros((test_size*c, 1))
52      for k in range(c):
53        Y_test[k*test_size:(k+1)*test_size, 0] = k
54
55      #Shuffle X_test and Y_test
56      idx = np.arange(c*test_size)
57      np.random.shuffle(idx)
58      Y_test = Y_test[idx]
59      X_test = X_test[idx]
60
61      #Create different X_train
62      X_train = np.asarray(X_train)
63
64      for j in range(conditions):
65        train_size = train_sizes[j]
66        X_train2 = np.concatenate(X_train[:,0:train_size])
67        Y_train = np.zeros((train_size*c, 1))
68        for k in range(c):
69          Y_train[k*train_size:(k+1)*train_size, 0] = k
70        idx = np.arange(c*train_size)
71        np.random.shuffle(idx)
72        X_train2 = X_train2[idx]
73        Y_train = Y_train[idx]
74        for n in range(n_kernels):
75          kernel = svm_kernels[n]
76          accuracies_svm[n, j, i]=svm_test(X_train2, Y_train,
                X_test, Y_test, kernel)
77    return accuracies_svm
78
79  svm_kernels=['linear', 'rbf', 'poly']
80  trials = 50
81  exp2_accuracies = run_experiment2(datasets, svm_kernels,
      trials)
```

A.4 Gibbs Sampler Code (R)

```
require(rjags)
require(coda)

# THE DATA
X_train <- Training[,1:8]
Y_train <- Training[,9]
X_test <- Testing[,1:8]
Y_test <- Testing[,9]
f <- 8
c <- 7
n <- nrow(X_train)
m <- nrow(X_test)

# THE MODEL
model1.string = "
model {
    ## Prior
    alpha <- c(1, 1, 1, 1, 1, 1, 1)
    p ~ ddirch(alpha)

    for (i in 1:m){
        g[i] ~ dcat(p)
    }

    for (i in 1:c){
        for (j in 1:f){
            mu[i,j] ~ dbeta(1, 1)
            sigma[i,j] ~ dgamma(0.1, 0.1)
            a0[i,j] <- mu[i,j]*sigma[i,j]^2
            b0[i,j] <- (1-mu[i,j])*sigma[i,j]^2
            a[i,j] <- ifelse(a0[i,j]==0, 0.000001, a0[i,j])
            b[i,j] <- ifelse(b0[i,j]==0, 0.000001, b0[i,j])
        }
    }

    ## Likelihood
    for (i in 1:n){
        for (j in 1:f){
            X_train[i,j] ~ dbeta(a[Y_train[i], j], b[
                Y_train[i], j])
        }
    }

    for (i in 1:m){
        for (j in 1:f){
            X_test[i,j] ~ dbeta(a[g[i], j], b[g[i], j])
        }
```

```
47      }
48  }"
49
50  model1.spec = textConnection(model1.string)
51  mcmciterations = 1000
52  jagsmodel ← jags.model(model1.spec,
53                         data = list('X_train' = X_train,
54                                     'X_test' = X_test,
55                                     'Y_train' = Y_train,
56                                     'f' = f,
57                                     'c' = c,
58                                     'n' = n,
59                                     'm' = m),
60                         n.chains = 4
61  )
62
63  model1samples = coda.samples(jagsmodel,
64                               c('g'),
65                               n.iter = mcmciterations)
66
67  # Find accuracy
68  samplesMatrix = as.matrix(model1samples)
69  acc ← c()
70  for (i in 1:nrow(samplesMatrix)){
71    acc[i] ← sum(samplesMatrix[i,]==Y_test)/m
72  }
73
74  # Remove burn-in samples
75  acc ← acc[-c(1:200)]
76  acc ← acc[-c(1000:1200)]
77  acc ← acc[-c(2000:2200)]
78  acc ← acc[-c(3000:3200)]
79  mean(acc)
80  sd(acc)
```