MASTER'S THESIS | LUND UNIVERSITY 2018

Classifying energy levels in modern music using machine learning techniques

Otto Nordander, Daniel Pettersson

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2018-28

Classifying energy levels in modern music using machine learning techniques

Otto Nordander
avo10ono@student.lu.se

Daniel Pettersson
dat13dpe@student.lu.se

June 19, 2018

Master's thesis work carried out at Soundtrack Your Brand.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Johan Brodin, johan@soundtrackyourbrand.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Music is a big part of many people's lives, certain characteristics of music may be more appropriate in certain situations, depending on mood, setting, or time of day. One interesting characteristic is energy level, by categorizing music into different energy levels a user is given the means to finding more fitting music.
We present a solution based on artificial neural networks and transfer learning that is able to classify 10 different energy levels on a linear scale from 1 to 10 independent of genre. The proposed solution reaches a mean distance of less than 1 from the annotated class on a broad range of music genres. We believe this is within expectation given that the problem is difficult even for a human.

Keywords: Music information retrieval, music classification, signal analysis, artificial neural networks, transfer learning, ordinal regression, macro mean absolute error, imbalanced data

Acknowledgements

We would like to thank our supervisor Pierre Nugues for guiding us and contributing with valuable input throughout this work. We would also like to thank our supervisor at *Soundtrack Your Brand*, Johan Brodin for tutoring and encouragement. Finally we want to thank *Soundtrack your Brand* for giving us the opportunity to perform this thesis and providing data, equipment and guidance.

Contents

Chapter 1
Introduction

1.1 Problem formulation

An element that has not been as studied is the perceived energy in a music track, where perceived energy is a subjective measurement of how energetic a song track is. For this problem the energy lies on a scale from 1 to 10, where 1 is a very low energy, and 10 is a very high energy. To give the reader an idea of different energy levels we present five songs, *An Ending (Ascent)* by *Brian Eno* with a very low energy, *Bridge over troubled water* by *Simon & Garfunkel* in the low energy spectrum, *All along the watchtower* by *Bob Dylan* belonging to the medium energy ranges, *Walking on sunshine* by *Katrina & The Waves* with a high energy, and finally *Ace of Spades* by *Motörhead* representing a very high energy.

This forces the question, is it possible to use different measurable traits extracted from signal analysis and meta data in order to train a machine learning model that classifies an energy level similarly to how humans do it?

Difficulties with the proposed problem is how accurately machine learning can adapt to very subjective and widespread data. Furthermore a problem of such subjective nature might prove to be difficult; there are many factors that can bias the classification set by a human. For instance a certain genre might ignite considerable more energy to a certain listener while leaving another listener cold. This has to be taken into consideration.

The problem can be formulated in a few points listed below;

- Can machine learning techniques generalize well enough with such a subjective problem?
- What are suitable methods of evaluation for such a problem?
- What is a useful scale for robust energy labeling?

1.2 Background

Music information retrieval (MIR) is a subject that has been gaining popularity over the last years. MIR is used to categorize, edit, and analyze music tracks. A few applications are genre classification, danceability estimation, and instrument recognition. Signal analysis and machine learning are helpful and popular tools to aid the process. Numerous studies have focused on classifying which moods a song might induce to the listener (Cernian et al., 2017; Dang and Shirai, 2009; Nuzzolo, 2015). Common approaches are to use various signal analysis components combined with lyrical analysis to train a machine learning model in order to classify the mood.

1.3 Structure

The report is structured in the following manner:

- **Chapter 2** contains theoretical concepts that are used throughout the work.
- **Chapter 3** consist of the approach that was used to create a basis for solving the problem, based on the theoretical concepts described in Chapter 2.
- **Chapter 4** covers results that have been obtained.
- **Chapter 5** provides a discussion of the previously mentioned results.
- **Chapter 6** discusses various interesting techniques for future work and conclusions of the thesis.

1.4 Previous work

Music information retrieval is a popular subject for research, however the available research on different energy levels is sparse. The most related research has been done with mood classification which arguably might be related to energy. Hopefully different moods might indicate different energy levels in a song, for instance a sad and calm song could have a lower energy level than a happy song (Nuzzolo, 2015).

Choi et al. (2017) showed in their paper that transfer learning could be used as a general purpose method for various music tagging problems. Their paper presents results comparable with *state of the art* (SOTA) methods for several different music tagging problems including tagging mood in music. With their method, a *convolutional neural network* (CNN) model is trained to tag a broad set of different labels and then using components from the network, to extract features from a 30 second music clip. The extracted features can then be used to train a different model in other music classification tasks.

Lee et al. (2017) performed experiments on music classification using raw waveforms as input to different *convolutional neural networks* in order to classify tracks with various tags. The authors compare different network architectures with 1-dimensional convolutional layers. An idea is that the network learns to distinguish different characteristics of the audio signal which hopefully can generalize to energy classification.

1.5 Work division

Throughout the course of this thesis we have been working together to the highest extent possible, mostly because we wanted both to have insight in to all parts of the project. However in certain situations we have divided initial research between the two of us, e.g. when researching different oversampling techniques. The writing of the report was done in the same manner, by together deciding the layout and structure, and writing individually but reading and revising as a group. We estimate that the writing of the report was equally divided.

Chapter 2

Theory

Throughout this section, we will describe the necessary background theory to understand the remainder of the thesis. First and foremost digital signal processing theories are presented, as it is a vital part of the thesis. Secondly various tools and theories taken from machine learning are explained; machine learning was used heavily during this thesis and plays a major role in its process. The reader is expected to have a basic understanding of algebra and statistics. Knowledge about machine learning in general is an advantage, however the concept is described briefly in Section 2.2.

2.1 Digital Signal Processing

Digital signal processing can be explained as information retrieval and altering of digital signals. A music track can be seen as a digital signal and hence inherits various methods for processing it. For instance one can transform the signal between different domains such as frequency and time or altering the digital size of the signal in several ways. A few key operations for this thesis are explained in this section (Stein, 2000).

2.1.1 Sampling

A signal sample is a value selected at a specific time t from a signal s. A digital signal is a collection of subsequent samples. The time frame between each sample is called *sampling frequency* or *sample rate*. The resulting quality of a digital signal highly depends on the sample rate. The sampling theorem (Equation 2.1) states that in order to fully reconstruct an analog signal, the sample rate is constrained to a minimum of twice the maximum frequency of the signal. A 30 second signal that is sampled with a sample rate of 22,050 Hz will result in 22,000 $\times$ 30 = 661,500 sampled values (Stein, 2000). However a lower sample rate has the advantage of the digital size being reduced which is less cumbersome to store and process in a machine learning setting.

$$
\begin{array}{ll}
frame_0: & \boxed{x_0 \quad x_1 \quad x_2 \quad x_3} \quad x_4 \quad x_5 \quad x_6 \quad x_7 \\
frame_1: & x_0 \quad x_1 \quad \boxed{x_2 \quad x_3 \quad x_4 \quad x_5} \quad x_6 \quad x_7 \\
frame_2: & x_0 \quad x_1 \quad x_2 \quad x_3 \quad \boxed{x_4 \quad x_5 \quad x_6 \quad x_7}
\end{array}
$$

Figure 2.1: Example of window size $N_f = 4$ and overlap percentage $p = 50\%$.

$$\frac{f_s}{2} > f_{max} \tag{2.1}$$

2.1.2 Mel Frequency Cepstrum

In speech recognition, it has proven efficient to use an audio representation called *mel frequency cepstrum* (MFC), which has shown to greatly affect the area.

The representation is obtained by transforming the raw audio samples (Section 2.1.1) into the frequency domain by applying the *discrete Fourier transform* (DFT), after which the mel scale is used to mimic the nonlinear scale of frequencies as perceived by humans (Equation 2.2). The MFC uses the aforementioned non linear correlation to build a representation of the power of the amplitude in different frequencies over time (Fang et al., 2001).

$$Mel(f) = 2595log_{10}(1 + \frac{f}{700}). \tag{2.2}$$

To go from the raw signal to MFC the signal needs to be divided into frames. These frames are small windows in the signal where it is assumed that the signal is sufficiently stationary.

The window size N_f is the number of samples contained in one frame. For each frame, an overlap percentage p is chosen. The overlap percentage is chosen as the number of samples that each of the frames overlaps. From the example (Figure 2.1), a sample set x of size 8 is divided into 3 frames.

A *Hann window* is used to calculate the coefficients for each sample in each frame. The equation for the *Hann window* is as follows:

$$w(n) = sin^2(\frac{\pi n}{N_f - 1}). \tag{2.3}$$

Then for each frame i of size N_f, the set is windowed using the *Hann window* transformed into the frequency domain by using the DFT. Each frame is transfered into the frequency domain as $\{X_1^{(i)}, X_2^{(i)}, \dots, X_{N-1}^{(i)}, X_N^{(i)}\}$, where $N_{DFT} = 1 + \lfloor \frac{N_f}{2} \rfloor$:

$$X_k^{(i)} = \sum_{0}^{N_f - 1} w(n) x_n e^{-\frac{2\pi i}{N_f} kn}. \tag{2.4}$$

To compute the power-spectral density, each frame is concatenated in a matrix Y as follows;

$$Y_{i,j} = \frac{1}{M}|X_i^{(j)}|^2, \tag{2.5}$$

where M is the number of frames.

The last step is to generate a *mel filterbank* which is the step that transforms the power-spectral density of the frames to into a representation which follows the perceived correlation between frequencies. The *mel filterbank* is composed of triangular filters where the position and width of each filter is calculated using the *mel scale* (Equation 2.2). The number of filters can be chosen arbitrarily when constructing the filterbank, but typically 25 to 45 filters are used. In the report Fang et al. (2001), 35 filters were found to have the best performance on automatic speech recognition. The equation from *mel* to *Hz* can be derived from (Equation 2.2) as:

$$Hz(m) = 700(10^{m/2595} - 1). \tag{2.6}$$

For M filters, $M + 2$ equally spaced points $f^{(m)}$ on the *mel scale* are chosen with the upper limit of $Mel(f_{max})$, where f_{max} is half the sample rate of the signal sample rate (Equation 2.1). These points are converted to frequency as $f_i^{(Hz)} = Hz(f_i^{(m)})$, where i is a vector index. To match these points to the index of Y each point is converted as:

$$f_i = \lfloor \frac{N_{DFT} + 1}{f_{max}} f_i^{(Hz)} \rfloor. \tag{2.7}$$

The filter bank uses the function below to calculate the triangular filters;

$$M_m(k, f) = \begin{cases} 0, & \text{if } k \leq f_{m-1} \\ \frac{k - f_{m-1}}{f_m - f_{m-1}}, & \text{if } f_{m-1} \leq k < f_m \\ 1, & \text{if } f_m = k \\ \frac{f_{m+1} - k}{f_{m+1} - f_m}, & \text{if } f_m < k \leq f(m+1) \\ 0, & \text{if } k > f_{m+1}, \end{cases} \tag{2.8}$$

where k is the index in Y and m is the filter index. The filterbank can then be put into matrix form:

$$F_{a,b} = M_{a+1}(f_{b-1}), \tag{2.9}$$

where $a \in [1, 2, \ldots, M - 1, M], b \in [1, 2, \ldots, N_{DFT} - 1, N_{DFT}]$.

$$\hat{Y} = Y * F, \tag{2.10}$$

where $\hat{Y}$ is the MFC for the chosen number of filters M (Lin et al., 2015).

2.2 Machine learning

Machine learning is the concept of *learning* a function that maps from an input to an output. A *learning algorithm* is fed data from which it learns to draw conclusions by use of statistics; the data consists of samples of numerical or categorical features that have been

extracted beforehand. The concept has proven very useful in tasks that may be too difficult or time consuming for a human to accomplish, additionally it can often be performed in a much larger scale than when doing it manually.
A simple application of machine learning: given a temperature and a location as input; map the given input to a climatological season. It is rather obvious that the learned function can not be consistently correct, therefore the goal is to minimize a defined error function that describes the problem well. A few widely used error functions are described in Section 2.2.2.

There are a number of learning algorithms available, a few of them being *logistic regression*, *gradient boosting*, and *neural networks*. The mentioned algorithms are all part of a subset of machine learning called *supervised learning*. A supervised learning algorithm in comparison to an unsupervised algorithm has information regarding the desired output for a given sample during learning. In the previous example with the season prediction the learning algorithm would be fed a sample with features and the algorithm can compare the predicted season with the actual season as annotated in the data, also referred to as *ground truth*. One downside to supervised learning is the need of annotated outputs or labels. Many times the labels need to be annotated manually, which can be time consuming depending on the problem (Goodfellow et al., 2016).

2.2.1 Loss function

In machine learning a loss function is almost always used, also called *cost function*, or *error function*. The machine learning algorithms task is to minimize the loss function which should indicate how well the algorithm is performing. An example of a loss function is *mean absolute error* (Equation 2.12); it calculates the mean absolute distance from predicted samples to the *ground truth*. The loss function used often differs depending on the type of problem at hand, as can be seen in Section 2.2.2 (Goodfellow et al., 2016).

2.2.2 Classification vs. Regression

A classification problem in a machine learning setting is to predict categories to which an input belongs. For example which type of animal a picture resembles. The function that the algorithm produces is of the form: $f : \mathbb{R}^n \to \{1 \dots k\}$ where k is the predicted class for a given input. A common loss function used for classification is *cross entropy* (Equation 2.11) where p is the probability distribution of a single sample and q is the predicted probability distribution (Goodfellow et al., 2016). In a classifier with *cross entropy*, the error of a mis-classification is independent of the distance between classes.

$$H(y, \hat{y}) = -\sum_{i=1}^{n} y_i \log \hat{y}_i \tag{2.11}$$

Regression is used to predict continuous numerical values given a set input. The resulting algorithm looks as such: $f : \mathbb{R}^n \to \mathbb{R}$. A mis-classification in a regression algorithm is different to a classifier; the error for a single input is a continuous value, for example the

Euclidean distance between the *ground truth* value and the predicted value. An example of a regression problem is to predict the price of a train ticket, the algorithm would try to estimate a continuous price instead of a set price. Common conventional loss functions for regression are *mean squared error* (MSE) and *mean absolute error* (MAE) (Equation 2.12) (Goodfellow et al., 2016).

$$\begin{aligned} MAE &= \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \\ MSE &= \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \end{aligned} \tag{2.12}$$

A regression algorithm can be used in conjunction with some after processing to create a classifier. For example a numerical output value can be rounded to the nearest integer which represents a certain class, this showcases a method of *ordinal regression*, briefly described in Section 2.3.2.

An advantage with such an approach is that the algorithm knows dependence between classes; i.e. the mis-classification error is proportional to the distance between the classes. An example demonstrating the difference is shown in Equation 2.13 and 2.14. The calculations show that the cross entropy error is independent of between class distance.

$$H = \begin{cases} -2.659 & \text{given } y = 9, \quad \hat{y} = 5 \\ -2.659 & \text{given } y = 10, \quad \hat{y} = 1 \end{cases} \tag{2.13}$$

$$MAE = \begin{cases} 4, & \text{given } y = 9, \quad \hat{y} = 5 \\ 9, & \text{given } y = 10, \quad \hat{y} = 1 \end{cases} \tag{2.14}$$

Equation 2.13 shows the calculated error produced by the common loss function *categorical crossentropy*, it clearly indicates that the distance between classes does not affect the resulting error. Conversely, Equation 2.12 implies that the regressional loss accounts for the class dependence based on the equal distance between classes.

2.2.3 Feed forward networks

Artificial neural networks is a technique that resembles how the human brain functions. The technique can be used to solve problems that may be too advanced or complicated for humans to solve. Neural networks consists of combinations of neurons, where a neuron is a simple calculating unit. The neurons in turn create layers in the network; a network can have infinitely many neurons in a layer, and infinitely many layers in the whole network. However the more neurons, the more computationally expensive it becomes to train. Figure 2.2 shows a simple feed forward network. In a feed forward network, each node will perform a calculation and feed the result forward to the next layer. Between each layer are so called weights, weights are what enables the possibility of altering the underlying function that the network produces. The output of each node except the input nodes are calculated as shown in Equation 2.15, where z is the output, ϕ is the activation function,

ω is the weight vector, and x is the input vector (Goodfellow et al., 2016).

$$z = \phi(\sum_{i=1}^{n} \omega_i \times x_i) \tag{2.15}$$

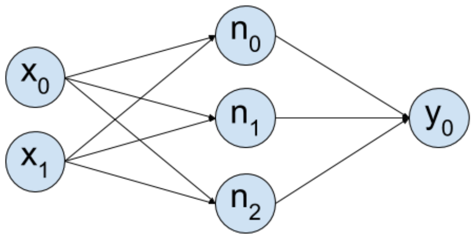


Figure 2.2: A small feed forward network with 2 input nodes, 3 hidden layer nodes and a single output node.

The weights between each layer are normally updated through *back-propagation*. In order to understand back-propagation one has to understand how the network *learns* a problem. Given a set of inputs **X**, a target vector **y**, the network tries to predict the correct values corresponding to an input. For each prediction an error is calculated, and the weights in the network are updated according to how much each node contributed to the error. The goal for the network is to minimize the resulting error during training; a *loss function* is defined that calculates an error given a sample input, a predicted output, and a ground truth output. Common loss functions are described in Section 2.2.2. The gradient of the error is calculated with respect to each weight in the network. The weights are then updated accordingly. For a network with n layers, the output $\hat{y}$ is given by Equation 2.16, this is called *forward propagation* or *forward pass*; the output of each layer is passed forward to the next layer.

$$\hat{y} = \phi_n(\mathbf{W}_{n-1} \times \phi_{n-1}(\mathbf{W}_{n-2} \times \phi_{n-2}(\ldots \phi_1(\mathbf{W}_1 \times \phi_0(\mathbf{W}_0 \times \mathbf{x})))) \tag{2.16}$$

2.2.4 Dropout layer

Dropout is a technique used to create a neural network that can be seen as a combination of several networks, also called *ensemble networks*. This is done in a single network and can be achieved by randomly disconnecting non-output neurons in the network with a probability p. In practice this will train a subset of the original network at each iteration. Figure 2.3 shows a network and each network that can be constructed by applying dropout to layer n.

2.2.5 Convolutional neural networks

A *convolutional neural network* (CNN) is a special feed forward network making use of convolution, the name of the network stems from the use of convolutional layers. Convolutional layers apply a special operation called convolution to the input; given a filter F,

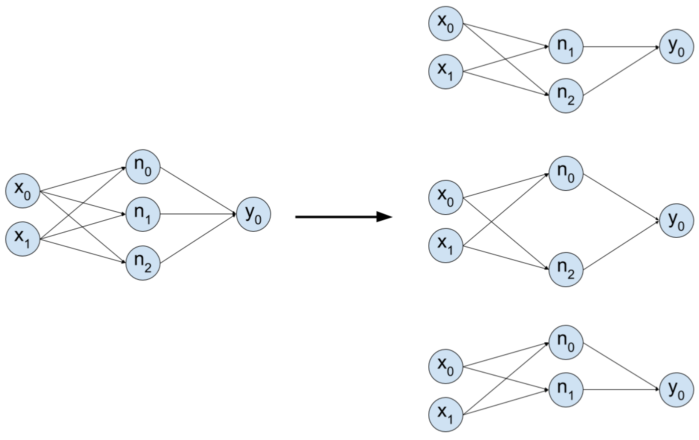


Figure 2.3: Combinations of a neural network that can be constructed by applying dropout to layer n.

and an image I, the convolution is defined as follows:

$$C = \sum_{i=1}^{m} \sum_{j=1}^{n} I(m,n)F(i-m, j-n). \tag{2.17}$$

CNNs have shown great progress in specific situations, one of the most common application is image recognition.

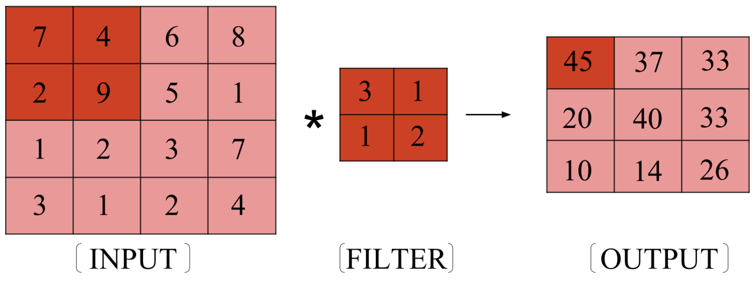


Figure 2.4: 2d convolution of a 4×4 image and with filter size 2×2.

The filter consists of updatable weights, and as seen in Equation 2.17 the output varies depending on the input and the filter. The output is sometimes referred to as a feature map. According to Goodfellow et al. (2016) filters with a smaller size than the input can discover tiny features such as edges. Moreover the number of parameters that need to be stored can be significantly reduced in comparison to a regular feed forward network layer, for a filter of size $m \times n$, only $m \times n$ weights need to be stored independent of input size.

2.2.6 Pooling layers

In CNNs various methods are used to reduce the size of images and hence the number of parameters that needs optimizing. In this section, the theories behind *max pooling* and *global average pooling* are described. Pooling layers operate on the feature maps produced by the convolutional layers in the network. The intent of pooling layers is to reduce the resolution and to prevent overfitting of noise (Stutz, 2014).

Max pooling can be thought of as iteratively sliding a window of size $n \times m$ over a feature map and taking the maximum value in each iteration. The number of steps between each iteration is called *stride*. Consequently the dimensions of the feature maps are reduced. A formula for calculating the resulting dimensions is shown in Equation 2.18, where W is the width, H is the height, S is the stride, n is the width of the sliding window, and m is the height of the sliding window.

$$W_{new} = \frac{(W_{old} - n)}{S + 1} \tag{2.18}$$

$$H_{new} = \frac{(H_{old} - m)}{S + 1} \tag{2.19}$$

For instance if a feature map is of size 4×4 and a max pooling of size 2×2 with stride 2 is used, the resulting dimensions would be 2×2, this can be seen in Figure 2.5.

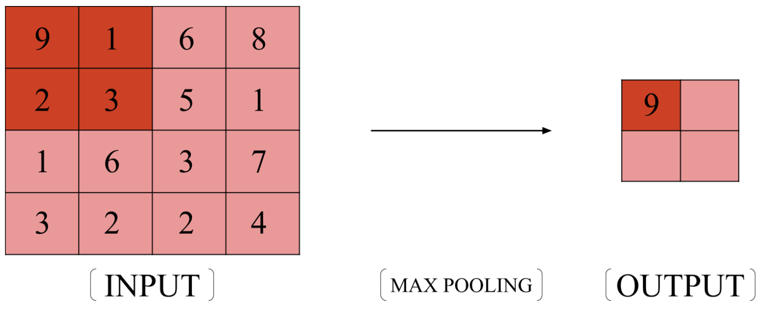


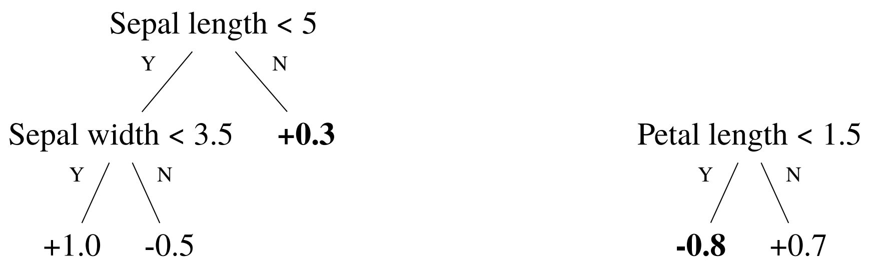
Figure 2.5: Max pooling with size 2×2 and stride = 2.

Lin et al. (2013) proposes a method called *global average pooling* (GAP) to prevent overfitting during training of a *convolutional neural network*. A scalar value is extracted from each feature map in a convolutional layer by averaging over it (Equation 2.20).

$$GAP = \frac{1}{W \times H} \sum_{i=1}^{W} \sum_{j=1}^{H} I_{i,j} \tag{2.20}$$

2.2.7 XGBoost

XGBoost is an implementation of the Gradient Boosting algorithm proposed by Friedman (2000) with changes to reduce overfitting and improved speed. The algorithm uses an ensemble of trees to produce a prediction $\hat{y}_i$ for the sample x_i, this is done with the results of the trained trees as $\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$ where $f_k(x_i)$ is the output for tree k. Figure 2.6 shows a simple example of how the output of $f_k(x_i)$ is decided for a problem of categorizing flowers with features sepal length, sepal width, and petal length. For a sample x_i with sepal length 6, sepal width 3, and petal length 0.5 with the pair of pre-trained example trees 2.6 the resulting prediction equals $\hat{y}_i = 0.3 - 0.8 = -0.5$.



(**a**) Tree 1

(**b**) Tree 2

Figure 2.6: Example trees

The algorithm is a supervised model so during training the algorithm requires an annotated set $(\mathbf{X}, \mathbf{Y})$. Where the values in $\mathbf{Y}$, y_i is distinguished from the predicted values $\hat{y}_i$ by being annotated before the training phase. During training the algorithm uses an additive strategy, by adding one tree f_t at a time that minimizes the objective function obj.

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t), \tag{2.21}$$

where t is the iteration, l is the loss function and Ω is the regularization function which penalizes the complexity of the tree. A second-order *Taylor expansion* is used to approximate the objective function and the constant terms are removed. The approximated objective function thus takes the form of the following expression:

$$\tilde{obj}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t), \tag{2.22}$$

where $g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ and $h_i = (\frac{\partial}{\partial \hat{y}_i^{(t-1)}})^2 l(y_i, \hat{y}_i^{(t-1)})$ are the first and second partial derivatives of the loss function w.r.t. the predicted label. Because of the infinite ways to structure trees, each split of the tree is chosen iteratively with the use of the gradient statistics to evaluate the tree f_t's influence on the obj function. This is solved with a greedy approximation with the calculation of the score obj_{split} for each split. In a split, the samples $\mathbf{X}$ are bagged in two different sets based on the condition of the split $I = I_L \cup I_R$. Where I_R consists of the sample set for the right and I_L for the left binary split. The scoring is calculated as follows;

$$obj_{split} = \frac{1}{2}\Big[\frac{(\sum_{I_L} g_i)^2}{\sum_{I_L} h_i + \lambda} + \frac{(\sum_{I_R} g_i)^2}{\sum_{I_R} h_i + \lambda} - \frac{(\sum_{I} g_i)^2}{\sum_{I} h_i + \lambda}\Big] - \gamma,$$

where λ and γ are parameters set for the regularization function Ω. The first component can be viewed as the score for the left split, the second as the right split, and the third as the score for not splitting the node. Then for each possible condition, the highest scoring split condition is chosen. If no positive score is found, the tree is pruned at that node. This is based on the regularization parameters and if the score is higher for a split than a non split (Chen and Guestrin, 2016).

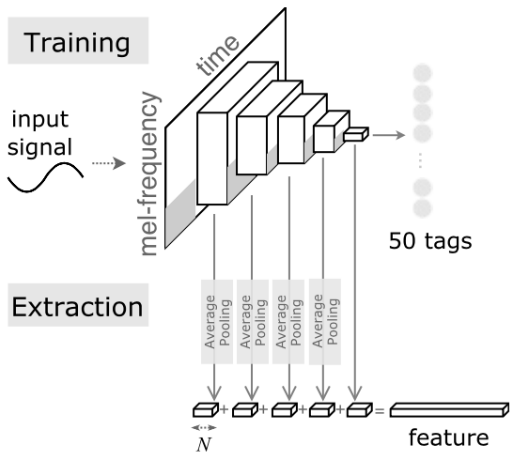


Figure 2.7: Block diagram of the transfer learning features extraction and the *cnn* model from the paper (Choi et al., 2017).

2.2.8 Transfer learning

Transfer learning is a concept that has proven useful in several machine learning studies. It is characterized by re-using previously learned information on a new related target task. The core idea of transfer learning is to transfer knowledge from one source domain to a target domain, for example if a network has been used to recognize dogs, it might prove efficient to reuse the network parameters as a starting point to recognize cats. Transfer learning is especially useful in cases where sufficient reliability or amount of data is not accessible to reach convergence during training. Furthermore it can greatly reduce the amount of training needed (Choi et al., 2017).

A convolutional neural network often learns general patterns of images in the early layers of the network, such as edge detection. Additionally the deeper layers are more specific to the problem that the network was optimized for. Naturally it makes sense to re-use the parameters from the shallow layers in transfer learning. A common approach is to extract values to be used as features, this can be seen as a feature extraction. Another approach is to simply replace the output layer with a layer corresponding to the new target (Yosinski et al., 2014).
A feature extraction can be done in many different ways; for example one can extract the raw values from the layers in a network, either before or after activation. Another approach suggested by Choi et al. (2017), is to apply global average pooling on each layer after its activation. A feature vector is constructed by concatenating each averaged feature map. An illustration is shown in Figure 2.7.

2.2.9 SMOTE

Synthetic Minority Over-sampling Technique (SMOTE) is an algorithm, where the minority classes are over-sampled. The motivation behind this approach as stated in the paper by Bowyer et al. (2011) is to increase the performance of a classifier on the minority classes of an unbalanced set. A class is considered a minority class when the population of said class is less than the majority class which has the highest frequency of all of the other classes in the dataset. The oversampling process in *SMOTE* generates synthetic samples rather than oversampling with replacement (Bowyer et al., 2011).

Oversampling with replacement works as follows. For each population of the minority classes, randomly sample n_i samples with equal probability from this population.

SMOTE synthesizes these samples by choosing a set $N^{(i)}$ of the k nearest neighbors for a sample x_i in a minority class. Each sample generates a certain number of synthetic examples. The algorithm synthesizes the samples by taking n_i neighbors from the set $N^{(i)}$ at uniformly random, with equal probability. For each of the randomly chosen neighbors $N_j^{(i)}$ a new synthesized sample $\hat{x}_{i,j}$ is created with the same label as x_i where $\hat{x}_{i,j}$ is calculated as a random point between x_i and $N_j^{(i)}$. Then n_i, the number of synthesized examples that should be generated from each sample x_i, is calculated from the percentage increase P in the minority class. Each sample x_i contributes an equal number of synthesized samples, which leads to $n_i = \frac{P_i}{100}$. P is often chosen as the percentage increase of a minority class to generate a sufficiently balanced dataset, i.e. where each class has the same population size.

$$\hat{x}_{i,j} = x_i + r * (N_j^{(i)} - x_i), r \sim U([0, 1]) \tag{2.23}$$

The synthesized samples $\hat{x}$ are then appended to the dataset (Bowyer et al., 2011). *SMOTE* is sometimes combined with under sampling techniques after the resampled set has been created.

One popular technique is to remove all samples contained inside *Tomek links* in the sample set. A pair of samples $(x_i, x_j), i \neq j$ is a Tomek link if the classifications for the samples differ and x_i is closer to x_j then each sample of the same class as x_i and vice versa (More, 2016).

2.3 Metrics

In the context of this thesis metrics are used for evaluating the performance of the classifiers on the *ground truth* of the samples, but can be used for other type of comparisons of data.

2.3.1 Accuracy metrics

Three different accuracy metrics are used in this thesis. These metrics are shown below, where $E = |y - \hat{y}|$, i.e. the absolute distance between *ground truth* and predicted class.

$$acc = \frac{n_{E=0}}{n_{total}} \tag{2.24}$$

$$acc_{one_{off}} = \frac{n_{E\leq 1}}{n_{total}} \tag{2.25}$$

$$acc_{two_{off}} = \frac{n_{E\leq 2}}{n_{total}} \tag{2.26}$$

2.3.2 Macro regression errors

Ordinal regression is defined as a classification problem where the classes y_i relate to each class as $y_1 < y_2 < \ldots < y_{n-1} < y_n$. The distance between each class can be known or unknown. For the subset of ordinal regression problems where the scale is known and of equal distance for each class, Baccianella et al. (2009) evaluated various metrics. In their paper they introduced *macro mean absolute error* (MAE^M) as a metric for ordinal regression problems with unbalanced datasets. They showed that MAE^M was a suitable replacement for MAE. MAE^M is derived from the more common error metric *mean absolute error* (MAE) (Equation 2.12). MAE^M is similar to various macro scores used in classification tasks.

$$MAE^M = \frac{1}{N} \sum_{j=0}^{N} \frac{1}{|C_j|} \sum_{x_i \in C_j} |\phi(x_i) - \hat{\phi}(x_i)|, \tag{2.27}$$

where N is the number of classes, ϕ and $\hat{\phi}$ are both classifiers and in the context of this report ϕ is the classifiers that outputs the *ground truth* from each sample. C_k is the set of samples that belongs to each class k which is derived from the *ground truth* classifier ϕ (Baccianella et al., 2009).

2.3.3 Confusion matrix

The confusion matrix is a valuable tool for measuring the performance of two classifiers for both binary classification and multi-class classification problems. The matrix shows the distribution for two classifiers with a shared sample set. Typically one of the classifiers is the *ground truth* classification and the other classifier is an algorithm predicting classes.

Table 2.1: Example confusion matrix for classification of three different species of the Iris flower; Iris setosa, Iris virginica, and Iris versicolor.

		Predictions		
		Iris setosa	Iris versicolor	Iris virginica
Ground truth	Iris setosa	35	10	0
	Iris versicolor	15	30	10
	Iris virginica	0	10	40

Table 2.1 shows an example of a confusion matrix for classification of different Iris flower species. The diagonal of the matrix shows the number of correct classification for each class and all other elements in the matrix shows the number of predictions for each class and what the *ground truth* classification is. This metric gives an overview on how the classifier performed for each of the classes.

The metric can also be normalized for easier digestion by using percentages of the distribution of the *ground truth*. This is done by dividing each element with the row or column sum for that element, dependent on which classifier is defined as the *ground truth*.

Chapter 3
Approach

In this chapter, we will present approaches based on the theories described in the previous chapter. Machine learning is a broad subject and many different techniques and approaches are available. In order to set the scope of the project, different theories will be applied to make it more manageable.

Our process is heavily inspired by the methodology *cross-industry standard process for data mining* (CRISP-DM); it consists of six key stages: *business understanding*, *data understanding*, *data preparation*, *modeling*, *evaluation*, and *deployment*. Influenced by this we chose to incorporate *business understanding* (Section 3.1) and under the Section 3.3 we show our results from the experiments in *data understanding*, *data preparation*, and *modeling*. As this thesis is primarily a theoretical study, the *deployment* stage is delegated to the company.

In Section 3.2.1 below, we present the data and its structure. This is to inform the reader of the basics of the data, the possibilities and, limitations for this problem.

This data is then used in Section 3.3, where the data is explored for the purpose of trying to construct a machine learning model through a series of experiments.

Finally, we will describe our exact implementation steps for producing the final model in Section 3.5 and how to evaluate it in Section 4.1.

3.1 Business understanding

In order to produce results that are usable and relevant from a business perspective, certain key points have to be taken into consideration. During this thesis we had an iterative discussion with *Soundtrack Your Brand* that have influenced our work.

Firstly, we came to the understanding that it was preferred to use technology that was easy to computationally distribute on the *Google Cloud API*; a natural choice was to use

Table 3.1: Provided data structure with relevant columns preserved.

energy	track_uri	track_preview_url	annotator_id	genres
int	str	str	str	str

neural networks with Keras and Tensorflow as our final model.
Secondly, discussions with the annotator team at *Soundtrack Your Brand* led to a guideline to interpret the magnitude of errors; a mis-classification of distance 1 is considered acceptable, while a distance of 2 is a small error, and finally errors larger or equal than 3 are considered large errors. The reasoning was that it is difficult for an annotator to reliably distinguish between two neighboring classes. It's unknown to which degree the different classes relate to each other, which led to the decision to keep the linear mapping between each class.
Finally, through discussion with *Soundtrack Your Brand*, we derived that from a user perspective it's desirable to prioritize a classifier that is often close or correct than a classifier that is often correct but with a higher degree of large errors.

3.2 Data

In this section, the initial data that was used throughout the project is presented. This includes a small data analysis, all fields that are contained in the dataset as well as in what format the data is provided. Furthermore we will explain how the data has been annotated and extracted, and our process for obtaining features.

3.2.1 Provided data

The dataset used in this study is provided by *Soundtrack Your Brand*; it has been manually annotated by a handful of individuals hired by the company. Each entry in the data corresponds to a music track and is annotated with an energy level as perceived by the annotator which is the target classification. The energy level ranges from 1 to 10 and should indicate how much energy the song brings to the listener. For example an ambient very slow song has a lower energy level, while a fast heavy metal song has a higher energy level. In order to obtain robust annotations, a few guidelines were established as follows:

- A track can have varying energy levels throughout the different segments, but it's judged by the segment with the highest perceived level;
- The full song is included in the analysis;
- If the annotator is uncertain about two energy levels, a higher energy level is preferred.

Data is provided as a comma separated value file with values as shown in Table 3.1. *En-*

ergy is the perceived energy as annotated by the annotator, *track_uri* is a unique *Spotify* identifier for the specific track, *annotator_id* is the unique identifier for the person that annotated the track, *track_preview_url* is a hyperlink to a 30 second sample of the music track available for download. Finally, *genres* is a comma separated string with genres that match the track; as they are annotated by the annotator team we cannot feed it as a feature to our machine learning algorithm, it is merely used for analysis and statistics.

The dataset included approximately 52×10^3 annotated tracks, but only around 42×10^3 samples had a valid *track_preview_url* column; in this case it is valid if the *track_preview_url* is a reachable link to a product sample of a track. The product sample is a section of 30 seconds extracted from a given track. It enables us to extract additional features complementing what is given by *Spotify Web API*; thus we decided to focus on the sub-set that contain valid *track_preview_urls*. We made the assumption that the *track_preview_url* was a product sample of the actual song used in the annotation phase.

The distribution of energy annotations with a valid *track_preview_url* is found in Figure 3.1. The figure clearly shows a very unbalanced distribution of energy ratings which will be covered more in Section 3.3.2.

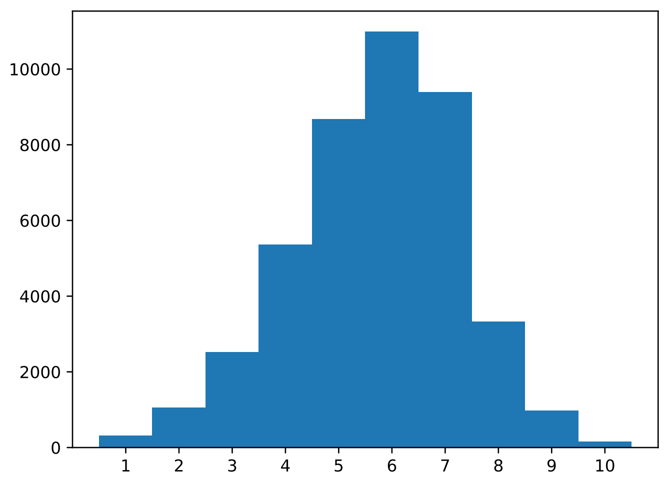


Figure 3.1: Energy label distribution

3.2.2 Features

In the experimental phase of the thesis three different feature spaces were taken into consideration, namely *Spotify features*, *transfer learning features* and, *signal analysis features*. The different feature sets will be described below.

3.2.2.1 Spotify features

Spotify provides an API that can be used to extract various audio features from songs (Spotify, 2018). Table 3.2 shows the features that we evaluated. Unfortunately, the approaches used by Spotify to obtain the features are undocumented. However the documentation indicates use of probabilistic models and signal analysis. These features will continually be referred to as $F_{Spotify}$.

Table 3.2: Features provided through the Spotify API.

Feature	Values
danceability	[0, 1]
energy	[0, 1]
key	[0, 11]
loudness	[-60dB, 0dB]
speechiness	[0, 1]
acousticness	[0, 1]
instrumentalness	[0, 1]
liveness	[0, 1]
valence	[0, 1]
tempo	[0, 1]
time_signature	[1, 200]

3.2.2.2 Transfer learning features

Choi et al. (2017) presented a convolutional neural network trained to classify various music features, according to the authors the network can be used to extract general features from music. The transfer learning network takes the raw signal as input and transforms it to an *MFC* representation as described in Section 2.1.2. We extracted the result from the activation function in each of the convolutional layers and performed an *global average pooling* on each layer, resulting in a feature set $F_{transfer}$ with 160 features. Figure 2.7 shows an overview of the transfer learning process.

3.2.2.3 Signal features

Finally we chose to explore digital signal analysis to build a feature set F_{signal}. Essentia is a library that offers several pre-built algorithms to process signals. Some features used in music genre classifications are tempo, root mean square energy and different tonal features (Baniya et al., 2013), which are all available in Essentia (Bogdanov et al., 2013). Moreover, Essentia offers preprogrammed music feature extractors containing aforementioned features among others.

Table 3.3: Total feature set.

Subset	# features
$F_{Spotify}$	12
F_{signal}	300
$F_{transfer}$	160

3.3 Experiments

In order to derive the optimal system to classify music we conducted experiments to investigate which techniques and settings that were best performing. We divided the experimental phase into the following subsections; *data understanding*, *data balancing*, *class dependence*, *feature selection*, *annotator models*, and *hyper parameters search*. Our workflow, each setup, and the thought process through the different experiments will be explained in this section. The experimental setups are listed chronologically and conclusions drawn from each experiment are used in consecutive experiments.

3.3.1 Experimental setup

Firstly the set where split into a **train** and **test** set. The **test** set was formed by uniformly random sampling 20% of the whole dataset, consequently the **train** set was chosen as the remaining samples. In total the **train** set consists of 34,203 samples and the **test** set of 8,551 samples.

During the initial part of our experimental phase we elected to use *XGBoost* for exploring data balancing, feature selection, and the effect of preserving the class dependence when training a model. The major advantage of using *XGBoost* over the already decided architecture of neural network is its significantly lower training time due to parallelization and small hyper parameter space. Additionally, *XGBoost* have recently shown great results in machine learning competitions (Chen and Guestrin, 2016).

To validate these experiments, we used *cross validation* with four folds on the **train** set. The *cross validation* procedure works by splitting the dataset into four different sets, training on three parts and measuring the performance on the remaining. This is done for all the different permutations of sets.

For the experiments with neural networks, we used a validation split of the the **train** set instead of using *cross validation*.

3.3.2 Data understanding

In order to fully understand and model the proposed problem, we have to understand the underlying data. The tools used in this phase was Python in conjunction with the Pandas library for managing large *comma separated value* files and visualization, as well as Numpy for numeric computation.

Our first step in order to understand the data was to extract the class distribution (Figure 3.1). As seen in the figure it was very skewed towards the mid-range energy labels; this is what is considered an unbalanced class distribution. A problem that may arise from an imbalanced dataset is that the algorithm fully ignores certain minority classes. The minority classes' impact in the algorithm will be suppressed by the majority given that the difference is sufficiently large. This can be seen as the algorithm being *lazy*, we need to prevent the outcome of the inherent *laziness* of the algorithms by combating the imbalance. A more fine grained distribution is shown in Table 3.4. Intuitively, energy 1 and

Table 3.4: Distribution of labeled energy of 42,754 songs.

Energy	**n**	**n / total**
1	310	0.007
2	1,056	0.024
3	2,521	0.059
4	5,359	0.125
5	8,671	0.203
6	10,983	0.257
7	9,393	0.220
8	3,329	0.078
9	976	0.023
10	156	0.004
Total	42,754	1

10 will be difficult to predict in a ML model judging by the very low number of samples present.

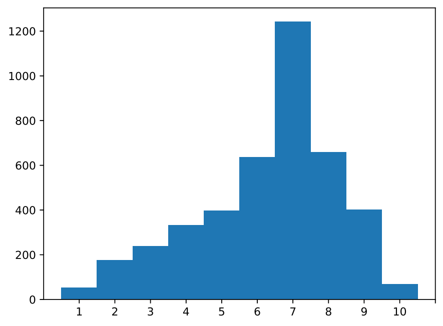
MAE^M was chosen as the evaluation metric (Section 2.3.2), since Baccianella et al. (2009) showed the metric's usefulness for ordinal regression problems with unbalanced data.

During data analysis we discovered that different annotators tended to disagree to a certain extent; this was discovered when some of the tracks had been labeled more than once. Obviously it is not feasible that all songs are labeled identically by different annotators, considering for instance personal taste in music; however the discrepancy was larger than we initially thought. Naturally we examined the label distributions per annotator. Label distributions from the four annotators with the highest number of songs labeled are shown in Figures 3.2a- 3.2d.

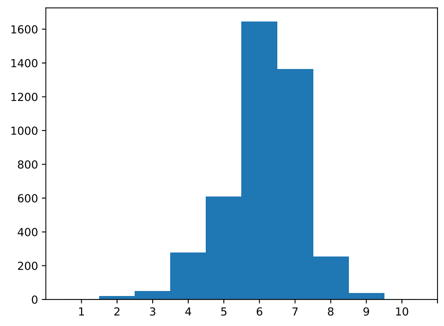
As part of our analysis, we constructed an intermediate model by use of *XGBoost* with the goal to investigate outliers in the data. The samples that produced the largest errors during prediction by our temporary model were manually reviewed by the annotator team at *Soundtrack Your Brand*; our conclusions were that the dataset had a significant number of mis-classifications. The majority of the mis-classified songs that we found were re-classified iteratively, consequently the quality of the labels improved over time.

3.3.3 Data balancing

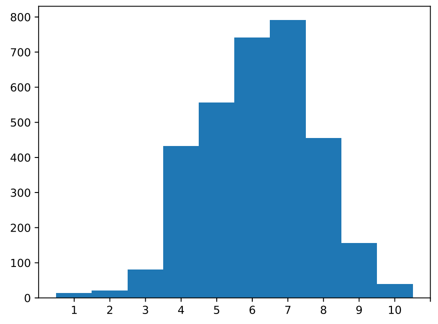
During the experiments from Section 3.3.2 we noticed huge imbalance in the class population of the dataset. We theorized that the performance of the machine learning algorithms was strictly bounded to the class imbalance. Several ways of dealing with imbalanced data were compared against each other versus the performance of leaving the class population as is. The methods that were compared are; *random over sampling with replacements*, *SMOTE*, *SMOTE* with *Tomek links* under sampling, and *random under sampling*. The first three methods (*random over sampling with replacements*, *SMOTE* and, *SMOTE* with *Tomek links* under sampling) is covered under Section 2.2.9 and *random under sampling*



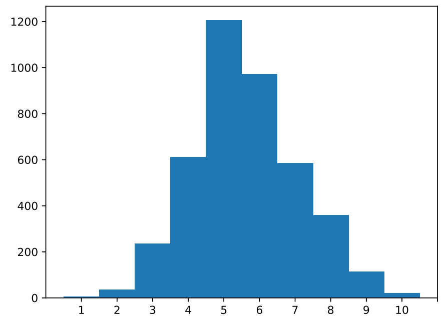
(a) Distribution of energy classes by annotator alias *th*.



(b) Distribution of energy classes by annotator with alias *RZ*.



(c) Distribution of energy classes by annotator with alias *9j*.



(d) Distribution of energy classes by annotator with alias *8l*.

Figure 3.2: Class distributions for four annotators.

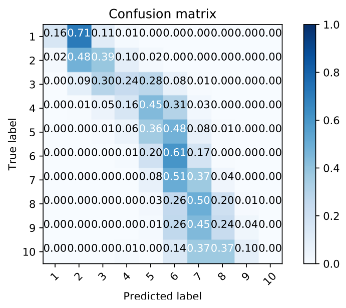
is an approach in which samples are removed uniformly random from each class except the one with the smallest population until the class distribution is even.

The experiments were carried out with the feature space as $F_{Spotify} \cup F_{transfer} \cup F_{signal}$, which will be examined in later experiments. We chose *XGBoost* as the machine learning algorithm for these tests, with MSE as the loss function. The predictions were made using *cross validation* on the **train** set.

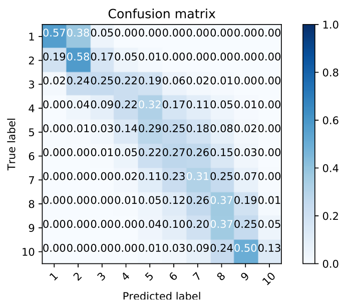
Table 3.5: Oversampling comparison with XGBoost for the metric MAE^M.

Technique	**MAE$^{\mathrm{M}}$**
No resampling	1.16
Random oversampling	**0.97**
Random undersampling	1.02
SMOTE	0.98
SMOTE and Tomek	0.98

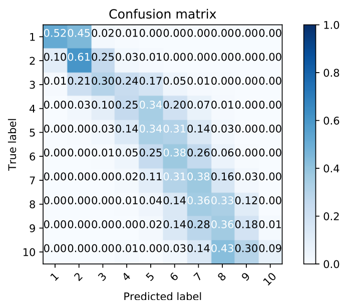
The results from the tests are shown in the Table 3.5 and Figures 3.3. Both the *confusion matrices* and the MAE^M metric show a performance increase compared to using no resampling method. All the methods examined here are based on numbers drawn from uniformly random distributions. As such the results are prone to fluctuate. This effect is minimized to some degree as the results are produced using cross validation.



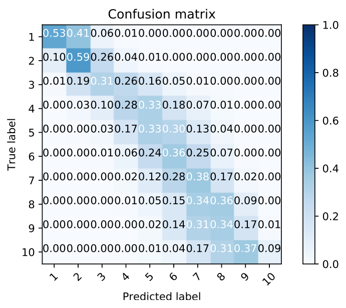
(a) Resulting confusion matrix for no resampling.



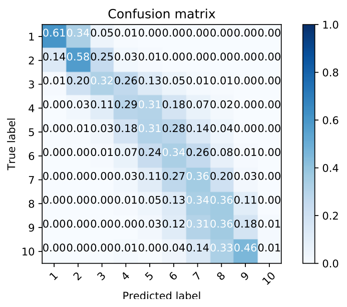
(b) Resulting confusion matrix for random under sampling.



(c) Resulting confusion matrix for random oversampling with replacements.



(d) Resulting confusion matrix for SMOTE oversampling.



(e) Resulting confusion matrix for SMOTE oversampling with Tomek links under sampling.

Figure 3.3: Confusion matrices for the oversampling comparison with *XGBoost*.

From the Figures 3.3, all of the resampling techniques (Figures 3.3b-3.3e) show an increase in performance on the low and high spectrum of labels in comparison to no resampling (Figure 3.3a). All the oversampling approaches (Figures 3.3c, 3.3d, 3.3e) show similar behavior in the mid spectrum, while *random under sampling* (Figure 3.3b) gives a larger spread in the same section. The probable cause for this is that the majority of the samples are in the middle spectrum as Table 3.4 shows, therefore the music diversity in this range probably differ a great deal and with *random under sampling*, the classifier trains on a small subset of this diversity and therefore can't reliably classify music in that spectrum.

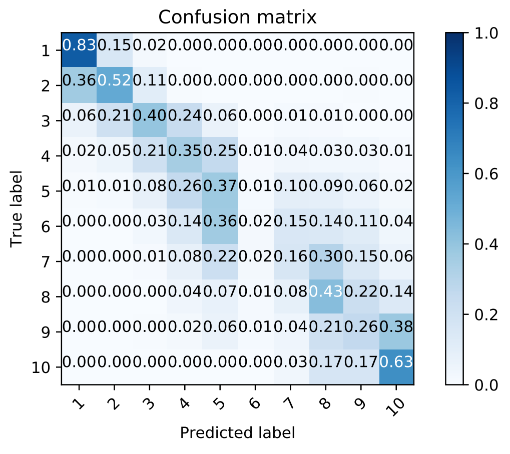
3.3.4 Class dependence

The problem posed can be evaluated in several different ways. At its core, it is a classification problem, i.e. it is binary if the classification is correct or not. However it also poses as a regression problem given that the energy levels lie on a linear scale from 1 to 10.

The confusion matrices from Figure 3.4a were derived using the $F_{Spotify}$ feature set and the resampling technique random over sampling.

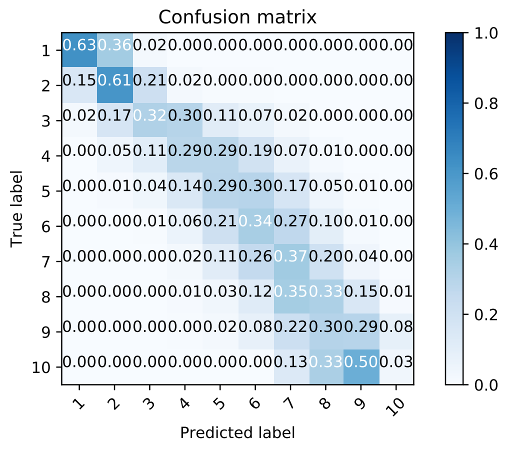
When using a classifier architecture, the range of faulty classifications was rather broad; as can be seen in Figure 3.4a. This likely stems from the fact *categorical crossentropy* has no information on class dependence as discussed in Section 2.2.2.

On the contrary, a regression architecture resulted in narrower ranges of mis-classifications as seen in Figure 3.4b. For the specific problem at hand, it's advantageous to obtain a model that produces predictions with minimal deviation from the ground truth (see Section 3.1); our experiments indicate that a model without knowledge about class dependence performs worse in this regard, see Figures 3.4a and 3.4b. Consequently, we chose to work with the problem as an ordinal regression problem, with the reasoning that a misclassification of one level should not be penalized to the same degree as a misclassification of five levels. When optimizing the various machine learning algorithms, we used a regression architecture. Finally, the resulting values were clipped between 1 and 10 and rounded to the nearest class.



True label \ Predicted label	1	2	3	4	5	6	7	8	9	10
1	0.83	0.15	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.36	0.52	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.06	0.21	0.40	0.24	0.06	0.00	0.01	0.01	0.00	0.00
4	0.02	0.05	0.21	0.35	0.25	0.01	0.04	0.03	0.03	0.01
5	0.01	0.01	0.08	0.26	0.37	0.01	0.10	0.09	0.06	0.02
6	0.00	0.00	0.03	0.14	0.36	0.02	0.15	0.14	0.11	0.04
7	0.00	0.00	0.01	0.08	0.22	0.02	0.16	0.30	0.15	0.06
8	0.00	0.00	0.00	0.04	0.07	0.01	0.08	0.43	0.22	0.14
9	0.00	0.00	0.00	0.02	0.06	0.01	0.04	0.21	0.26	0.38
10	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.17	0.17	0.63

(a) Resulting confusion matrix of a classifier architecture using XGBoost with categorical crossentropy loss.



True label \ Predicted label	1	2	3	4	5	6	7	8	9	10
1	0.63	0.36	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.15	0.61	0.21	0.02	0.00	0.00	0.00	0.00	0.00	0.00
3	0.02	0.17	0.32	0.30	0.11	0.07	0.02	0.00	0.00	0.00
4	0.00	0.05	0.11	0.29	0.29	0.19	0.07	0.01	0.00	0.00
5	0.00	0.01	0.04	0.14	0.29	0.30	0.17	0.05	0.01	0.00
6	0.00	0.00	0.01	0.06	0.21	0.34	0.27	0.10	0.01	0.00
7	0.00	0.00	0.00	0.02	0.11	0.26	0.37	0.20	0.04	0.00
8	0.00	0.00	0.00	0.01	0.03	0.12	0.35	0.33	0.15	0.01
9	0.00	0.00	0.00	0.00	0.02	0.08	0.22	0.30	0.29	0.08
10	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.33	0.50	0.03

(b) Resulting confusion matrix of regression architecture using XGBoost with *MSE* loss.

Figure 3.4: Comparison of resulting confusion matrices from a classifier architecture and a regression architecture.

3.3.5 Feature selection

Considering that the feature space is very large, especially the number of features that can be extracted by signal analysis, we felt the need to discard subsets of the features. This was done by an experimental setup as shown below. In each iteration, a combination of the feature subsets are chosen.

1. Iteratively, choose a subset F of feature categories;
2. Re-balance **train** data with random over sampling;
3. Use *cross validation* on the **train** set to generate predictions;
4. Calculate MAE^M, and a *confusion matrix* and store for comparison;
5. Iterate step 1-4 until all feature combinations have been explored.

The reader is referred to Table 3.3 as a reminder of the feature sets examined.

Results from the feature experiments are shown in Table 3.6, and Figures 3.5a-3.5g. Our experiments show no improvement by adding F_{signal} to the feature set; considering that the signal analysis is the most time consuming feature extraction process we chose to leave it out of the final feature set.

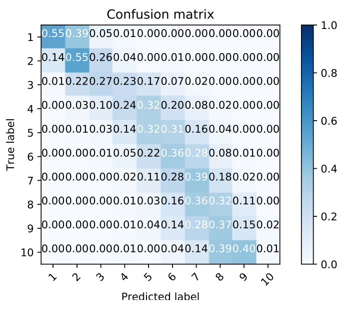
Table 3.6: MAE^M for the various combined feature categories.

F	MAE^{M}
F_{signal}	1.03
$F_{Spotify}$	1.04
$F_{transfer}$	1.03
$F_{signal} \cup F_{Spotify}$	0.99
$F_{signal} \cup F_{transfer}$	**0.98**
$F_{Spotify} \cup F_{transfer}$	***0.98***
$F_{signal} \cup F_{Spotify} \cup F_{transfer}$	**0.98**

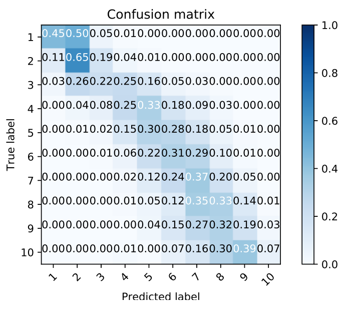
3.3.6 Annotator models

Because of the inherent subjectivity of the problem and lack of data to acquire robust measurement of inter annotator agreement; we experimented with models trained on single annotator data, with the feature set $F_{Spotify} \cup F_{transfer}$. We theorized that a model trained on a dataset extracted from the same annotator could eliminate contradicting cases where annotator preferences may affect the energy label. Table 3.7 shows the result of four classifiers trained on different annotators. The annotators used in this experiment were the four annotators which had labeled the largest quantity of tracks, the distributions for these annotators are shown in the Figures 3.2a- 3.2d.

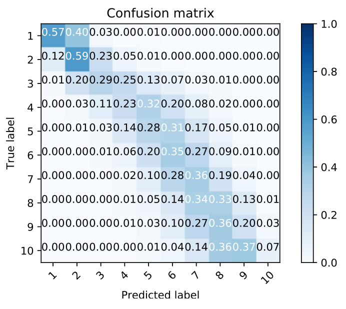
As seen in the Table 3.7 all the annotators outperformed the previous experiment from Table 3.6, however no single annotator model nor ensemble resulted in higher MAE^M when evaluating on the remainder of the dataset. In conclusion we decide not to use single annotators given the poor performance on the rest of the dataset.



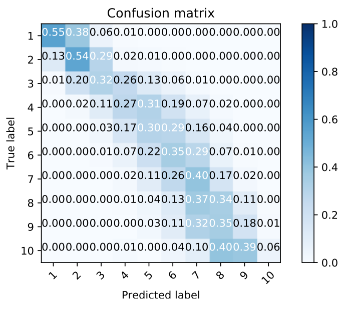
(a) Resulting confusion matrix where $F = F_{signal}$.



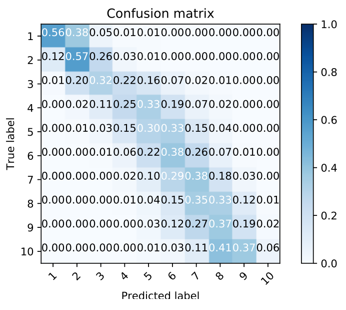
(b) Resulting confusion matrix where $F = F_{Spotify}$.



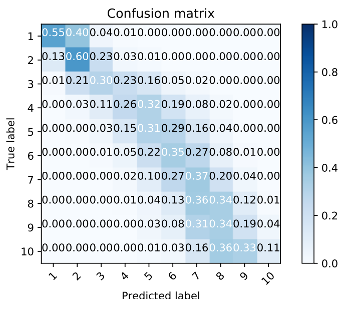
(c) Resulting confusion matrix where $F = F_{transfer}$.



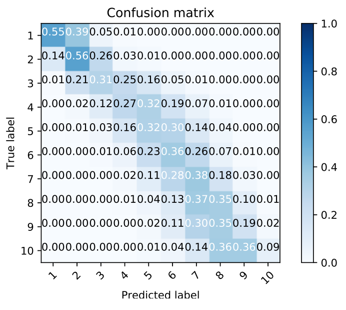
(d) Resulting confusion matrix where $F = F_{signal} \cup F_{Spotify}$.



(e) Resulting confusion matrix where $F = F_{signal} \cup F_{transfer}$.



(f) Resulting confusion matrix where $F = F_{Spotify} \cup F_{transfer}$.



(g) Resulting confusion matrix where $F = F_{signal} \cup F_{Spotify} \cup F_{transfer}$.

Figure 3.5: Confusion matrices for different feature sets.

Table 3.7: Results from construction models on single annotator data. Where MAE^M annotator column is the result from measuring the model on the annotators own data with *cross validation* and remainder column measures the models performance on the rest of the dataset. The final row consists of the performance of the mean vote of each annotator.

Annotator alias	MAE^M annotator	MAE^M remainder
th	**0.72**	1.16
RZ	0.88	1.28
9j	0.99	**1.12**
8l	0.90	1.15
Ensemble	-	1.26

3.3.7 Hyper parameter search

In the space of neural networks, there are plenty of adjustable parameters which alter how the algorithm learns the problem. These are called *hyper parameters*. The choice of different hyper parameters can greatly affect how the algorithm performs, and as such it is vital to explore the parameter space in order to obtain the optimal model. Moreover, the preprocessing step also contains adjustable parameters that we will investigate in Section 3.4.

To find the optimal parameters for our neural network, we constructed a pipeline that enables the possibility to evaluate different combinations of parameters; this was used in conjunction with *Bayesian Optimization* methods to estimate optimal values for parameters. The results from the parameter search are shown in Table 3.9.

Table 3.8: Variable parameters used in Bayesian optimization to find optimal hyper parameters.

Variable	Type	Values
n layers	Discrete	1|2|3
n neurons	Discrete	32|64|128|256
batch size	Discrete	10|50|100|150
learning rate	Continuous	$[5 \times 10^{-6}, 5 \times 10^{-4}]$

3.4 Neural network experiments

After we have concluded the hyper parameter space for our final neural network, we want to reiterate the previously conducted experiments for rebalancing data. We trained a neural network using the best hyper parameters found in Section 3.3.7 to once again explore the various sampling techniques.

Continuing from the first experiment (Section 3.5), we chose to run experiments with different parameters for *SMOTE* and *SMOTE and Tomek*; specifically we altered the number of neighbors. The results are shown in Table 3.10. As the result differ from the results

Table 3.9: Bayesian optimization search with respective losses, ordered as conducted by the optimization algorithm.

# layers	# neurons	Learning rate	Batch size	MAE^M
2	256	1.82×10^{-4}	150	0.845
3	64	1.55×10^{-4}	50	0.837
2	64	1.55×10^{-4}	50	0.827
1	64	1.54×10^{-4}	50	0.896
3	32	1.45×10^{-4}	150	0.849
1	128	2.29×10^{-4}	150	0.900
2	128	4.57×10^{-4}	150	0.822
3	64	2.40×10^{-4}	10	0.865
1	32	4.04×10^{-4}	10	0.888
1	32	5.00×10^{-4}	150	0.885
3	128	1.97×10^{-4}	50	0.830
3	32	2.35×10^{-5}	100	0.843
3	256	8.80×10^{-6}	10	0.911
1	32	1.01×10^{-5}	50	0.853
2	32	1.55×10^{-4}	50	0.835
1	32	1.96×10^{-4}	100	0.859
3	64	2.97×10^{-4}	150	0.849
1	256	2.04×10^{-4}	150	0.876
2	32	2.99×10^{-4}	100	0.836
3	128	1.10×10^{-4}	100	0.830
2	128	1.35×10^{-4}	50	0.838
2	256	4.78×10^{-4}	10	0.873
1	128	4.17×10^{-4}	10	0.932
2	32	3.59×10^{-4}	150	0.827
3	128	4.99×10^{-4}	150	0.851
2	128	2.60×10^{-5}	10	0.884
3	128	1.06×10^{-4}	100	0.837
3	128	1.16×10^{-4}	100	**0.816**
3	128	1.28×10^{-4}	100	0.824
2	128	3.35×10^{-4}	50	0.819
2	256	4.99×10^{-4}	50	0.837

gathered from the hyper parameter search when using the same *hyper parameters*, we concluded that the weight initialization of the network introduces some randomness in the results.

3.5 Implementation

This section describes our final implementation in the detail which is needed for the reader to reproduce all the steps in order build the final model. The model is chosen as the highest performing model in the context of the methods of evaluation described in the evaluation

Table 3.10: Macro mean absolute error for different resampling approaches.

Technique	**MAE^M**
Random oversampling	0.843
SMOTE (k = 3)	**0.830**
SMOTE (k = 4)	0.832
SMOTE (k = 5)	0.858
SMOTE (k = 6)	0.836
SMOTE and Tomek (k = 3)	0.831
SMOTE and Tomek (k = 4)	0.849
SMOTE and Tomek (k = 5)	0.830
SMOTE and Tomek (k = 6)	0.855

Section 4.1 in this report. The first Section 3.5.1 consists of the instructions on how to build the feature set that the model trains and evaluates on. The second Section 3.5.2 states what hyper parameters are chosen and how to train the model.

3.5.1 Building the feature set

The final feature vector used for each sample consists of a concatenation of the feature sets $F_{Spotify}$ and $F_{transfer}$. As seen in Figure 2.7, the input data to the *transfer learning* model is a raw sound signal. In our case, this raw waveform is sampled at the sample rate of 16000. This is achieved with the *python* library *librosa* (McFee et al., 2017). The process of creating the feature vector is shown as a flow diagram in Figure 3.6, where n = 160 and m = 12.

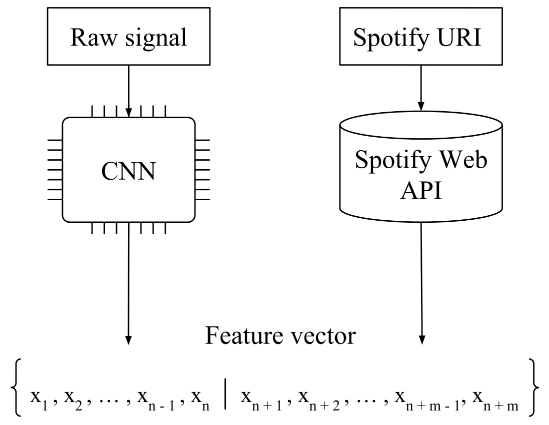

Raw signal
Spotify URI
CNN
Spotify Web API
Feature vector
$\{x_1, x_2, \dots, x_{n-1}, x_n \mid x_{n+1}, x_{n+2}, \dots, x_{n+m-1}, x_{n+m}\}$

Figure 3.6: Diagram of the creation of a feature vector, where the object marked as *CNN* is more detailed in 2.7; n is the number of features from *transfer learning* and m is the number of features from the *Spotify* Web API.

Each feature vector is stacked in a matrix $\mathbf{X}$, and each energy label in a vector $\mathbf{y}$.

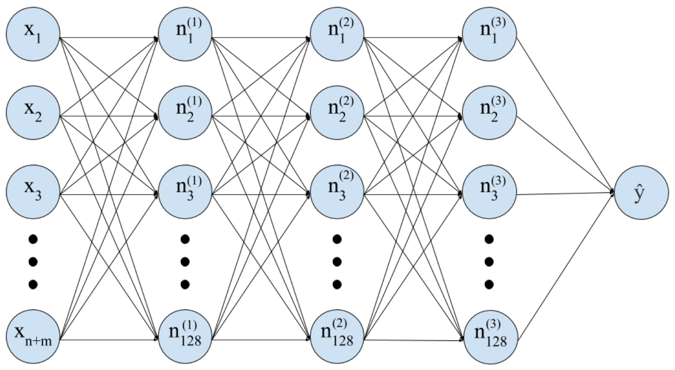


Figure 3.7: Neural network architecture.

3.5.2 Training neural network

Initially, the aforementioned $(\mathbf{X}, \mathbf{y})$ dataset is split into three datasets, namely **train**, **validation**, and **test**. The **train**-set is rebalanced using *SMOTE*. A neural network (see Figure 3.7) is trained using the parameters shown in Table 3.11 for 100 epochs with *Adam* optimizer and a learning rate of 1.16×10^{-4}; the epoch that produces the best performing model (measured by MAE^M on the **validation** set) is saved and used for evaluation. For initializing the weights of the network, randomly uniform initializers were used.

Table 3.11: Final implementation of the neural network.

Type	Setting
Dense	128
Dropout	0.5
Activation	*tanh*
Dense	128
Dropout	0.5
Activation	*tanh*
Dense	128
Activation	*tanh*
Dense	1
Activation	linear

Chapter 4
Result

In this chapter, we present results of our neural network architecture (Table 3.11) in comparison of the result of the baseline model and the *XGBoost* classifier. The training of the model is described in detail in Section 3.5.2. We evaluated and compared with a baseline model on the previously mentioned **test** dataset; the evaluation procedure is described in Section 4.1. Furthermore we present statistics of the classifications per energy label and genre.

4.1 Evaluation

In order to deploy a robust and reliable model, some form of final evaluation has to be conducted. The purpose of the evaluation is to ensure that the model works as expected and produces reasonable results. During each of the experiments we conducted, quantitative evaluations were made by comparing metrics for each model. However in the final stages of extracting a model qualitative evaluations are valuable. In this section, we will describe our process of evaluating the model from a machine learning perspective as well as a business perspective. For this purpose, we will construct two models using the parameters obtained in our experiments; this includes one neural network and one *XGBoost* model. Additionally, we will use a *baseline* model in order to compare a naive approach with the work conducted in this thesis.
We chose our baseline classifier based on the most naive approach without any prior knowledge about the problem at hand. This includes the steps for balancing the data, knowledge of best performing feature set, as well as understanding the relation between classes. With this reasoning, we chose not to rebalance the dataset consisting of the least complex feature set, namely $F_{Spotify}$. As our baseline classifying algorithm we selected *logistic regression* with one-versus-all classification; where the algorithm has no information about relations between classes.

As previously described in Section 2.3.2 *macro mean absolute error* defines a suitable

Table 4.1: Model comparison on MAE^M and MAE scoring on the test set.

Model	**MAE^{M}**	**MAE**
Baseline	1.30	0.89
XGBoost	0.88	0.89
Neural network	0.84	0.93

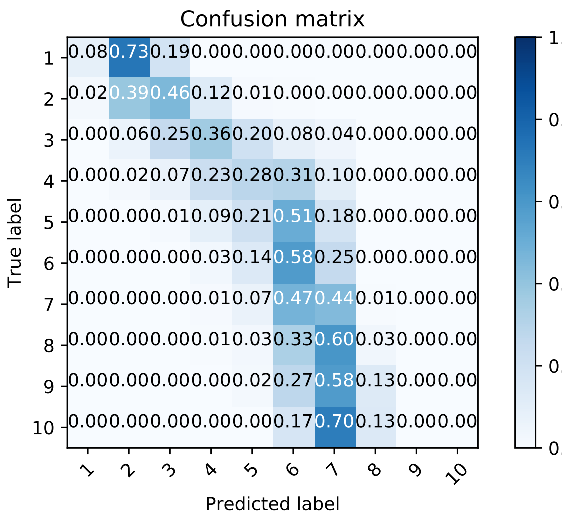
Table 4.2: Label statistics for the neural network.

Label	**MAE**	**acc**	**$\text{acc}_{\text{one\_off}}$**	**$\text{acc}_{\text{two\_off}}$**	**# Samples**
1	0.19	0.81	1.0	1.0	59
2	0.49	0.53	0.98	1.0	217
3	0.77	0.42	0.86	0.96	509
4	0.99	0.33	0.77	0.93	1057
5	1.03	0.3	0.76	0.94	1755
6	0.93	0.34	0.78	0.96	2225
7	0.91	0.34	0.8	0.95	1857
8	0.89	0.36	0.8	0.96	638
9	1.13	0.22	0.75	0.93	204
10	1.1	0.23	0.73	0.93	30

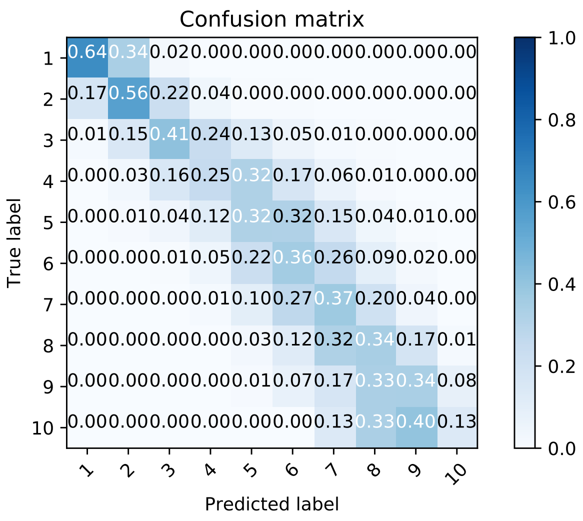
error for the problem. Our final evaluation will be based on a combination of MAE^M and manually assessing the resulting confusion matrix as it gives a good overview of the performance. The assessment has to take into consideration the business aspect of the problem as stated in Section 3.1. It is vital that the model performs equally well for each class which is why the aforementioned methods were chosen.

Table 4.3: Genre statistics for the neural network.

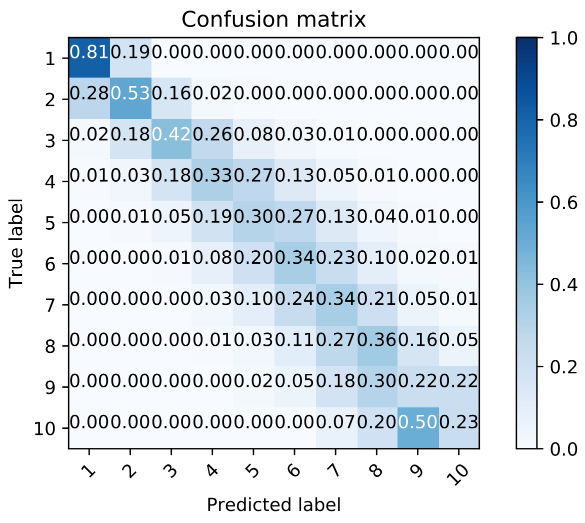
Genre	MAE	acc	$\text{acc}_{\text{one\_off}}$	$\text{acc}_{\text{two\_off}}$	# Samples
african	0.74	0.43	0.85	0.98	86
ambient	0.76	0.44	0.87	0.94	188
bass	1.02	0.28	0.78	0.94	126
blues	0.86	0.38	0.79	0.97	117
christian	1.23	0.3	0.63	0.89	199
classical	0.76	0.42	0.85	0.97	229
country	0.98	0.32	0.78	0.94	459
dance/edm	0.92	0.33	0.8	0.97	1001
dancehall/reggaeton	0.86	0.34	0.83	0.98	160
disco	0.75	0.4	0.87	0.98	206
experimental	1.05	0.32	0.76	0.91	176
folk	1.0	0.31	0.78	0.94	377
funk	0.71	0.44	0.87	0.99	278
hiphop	0.95	0.3	0.78	0.97	459
house	0.86	0.37	0.82	0.95	822
house/techno	0.67	0.33	1.0	1.0	3
indie	1.06	0.31	0.73	0.92	1577
jazz	0.69	0.44	0.88	0.98	507
lounge	0.92	0.34	0.8	0.95	661
mariachi	0.8	0.5	0.8	0.9	10
other	0.99	0.34	0.74	0.94	188
pop	0.96	0.33	0.78	0.95	2892
r&b	0.82	0.38	0.82	0.99	482
reggae/dub	0.91	0.26	0.83	1.0	54
rock	1.17	0.29	0.69	0.89	1453
salsa	0.64	0.45	0.91	1.0	33
samba/bossa nova	0.76	0.37	0.88	0.99	89
singer-songwriter	0.98	0.3	0.79	0.94	572
soul	0.71	0.43	0.87	0.99	607
techno	0.96	0.34	0.78	0.94	117
traditional	3.0	0.0	0.0	0.0	1



(a) Resulting confusion matrix of the baseline classifier on the test set.



(b) Resulting confusion matrix of the XGBoost classifier on the test set.



(c) Resulting confusion matrix of the neural network classifier on the test set.

Figure 4.1: Confusion matrices for the three different models.

Chapter 5
Discussion

In this chapter, we discuss the obtained results. This will be done in the context of the problem formulation (Section 1.1). Each question will be discussed and answered with respect to the results and possible drawbacks that we encountered. Moreover we will try to give reason to different choices we made during the process, e.g. design choices and assumptions about the data.

The results indicate that machine learning algorithms are able to generalize a solution to the problem to some extent. Generalization was improved in rate with the number of songs that were utilized; additionally the dataset was refined during the work as issues and wrong annotations were found. We drew from the ever increasing dataset, the first iteration of experiments was followed by a second one when the final dataset was acquired.

One very interesting factor in the generalization problem is the annotator field, by using a dataset annotated by a single annotator the algorithm's performance was heavily affected. This may indicate that different annotators have different ideas of what *energy* is in a song. However, it may also indicate that the quality of the annotations may differ, it's impossible to distinguish what the cause is without analyzing the dataset manually which would be very tedious and time consuming. Our assumption is that annotators' taste in music impacts the resulting energy labels, and as such removing samples from annotators that diverge from the overall norm in the dataset should improve performance. Interestingly and perhaps obviously the best model (measured by MAE^M) was achieved using a single annotator. The conclusion is that the model is able to capture the annotators personal thoughts about music in general; however there was no single annotator that had annotated sufficient amount of music types and genres for the idea to be practicable.
Furthermore, our aim was to generalize the model. One can argue that the subjectivity of the problem is removed if a single annotator is used. Observations that we found during experiments with single annotator classifiers were that certain often small subtypes of music were lost, for instance only a handful of annotators labeled Chinese traditional music.

Our concern was that valuable and necessary information would be lost by ignoring specific annotators, a trade off had to be made, either achieve a better predictive capacity or covering a larger scope of music.

One of our major breakthroughs was the usage of oversampling to re-balance the dataset. We had tried various approaches to fight the imbalance, but with lack of success. When we used oversampling the performance was improved significantly, which illustrates the problematic behavior that stems from the class imbalance. First and foremost the method enhances the predictive capabilities of the model in regards to the extreme minority classes, 9s and 10s. Obviously the optimal solution to this problem would have been a perfectly balanced dataset without oversampling it; however the energy distribution of modern music is arguably not perfectly balanced from 1 to 10, which raises the question of how well the used energy scale depicts the actual energy? The scale that is used for energy labels ranges from 1 to 10; as previously stated, neighboring classes can be hard to reliably distinguish for a human annotator, as such an error of 1 is considered correct. Instinctively we thought to reduce the number of classes due to this fact; however, we chose not to do so, with the reasoning that a higher granularity could prove useful and it would be easier to lower the resolution than to enhance it.

When deciding upon our evaluation process we strove for a classifier that performed well on all classes. Examining the results in Table 4.1, the common metric *mean absolute error* (MAE), shows that the baseline classifier had a better predictive score than our neural network (which performed best according to our evaluation which serves the business perspective). However MAE is not a suitable metric when considering the major class imbalance in the dataset that it is evaluated on; the confusion matrix in Figure 4.1a clearly performs worse on the majority of classes than the matrix shown in Figure 4.1c. We argue that *macro mean absolute error* (MAE^M) is a much better indicator of how well the algorithm is performing with the given dataset. A possibility would be to program a MAE^M loss function to be used in our neural network, we believe that such a loss would eliminate the use for re-balancing the dataset. The reason behind the rather high MAE score of the baseline classifier stems from the great class imbalance; the classifier categorizes a majority of the tracks in energy levels 6 and 7, which are the classes with the highest number of samples in the dataset. We chose to present MAE as part of our results for comparison.

In our results, we decided to extract MAE per genre, note that a track can have one or more genres associated with it. According to *Soundtrack Your Brand*, the energy of a track is not biased by genre; our classifier performs equally well in each genre with the exception of *christian, rock, and traditional*, we believe those specific genres are very broad and contains songs that are not encapsulated by a single genre. Table 4.3 show that our model's predictive capability is not bound to specific genres.

Chapter 6
Conclusion

The first section serves to summarize the problem, our solution, and the result of our thesis work. It's followed by a section on further improvements and unexplored approaches to our thesis work.

6.1 Summary

The subject for our thesis work rose from the infeasibility to manually and consistently classify a large set of tracks of their perceived energy level. Our proposed solution is to automate the classification of tracks using machine learning. A lot of music classification has been tried on mood classification, which arguably relates to energy classification. Therefore, we set out to solve this problem with the use of machine learning.

This thesis work consisted of experimentation and exploration of machine learning approaches to classifying music, continuing we list our three major conclusions found. Firstly a problem of this nature can efficiently be treated as an *ordinal regression* problem. The energy levels have a dependency between them that contains valuable information which can be used to obtain a better machine learning model. Our experiments showed that a regression architecture is preferred over a classification architecture. Secondly, the *transfer learning* architecture proposed by Choi et al. (2017) produces characteristics for a track which correlates with the tracks energy level. Finally, class imbalance in the dataset requires action to be taken to derive a well-performing model. Rebalancing the dataset is one way of counteracting the side-effects of the imbalance during training, although other approaches may be applicable.

When choosing the model, the choice of a metric highly affects the outcome, and in our case a less common metric MAE^M is more in line with our problem than the more commonly used ones; see Chapter 4 where various metrics on the **test** set are presented.

6.2 Further improvements

One major point of inconsistency is in the way we extract the features from the transfer learning network. The transfer learning features are extracted from a 30 second sample of the track. Optimally this 30 second clip might be chosen from a measurable heuristic as to ensure consistency in the transfer learning feature set. As of now, the part of the song that is being used is chosen through an undocumented process from *Spotify*'s API. This results in that the majority of the features that are used are based on a process that is unfortunately not under our control. The worst case would be if the track segment covers the least energetic part of the track. It's improbable that the majority of the segments falls in the worst case category, but they might not represent the most energetic part of the song as the annotators where instructed to classify on. Possible solutions other than using a heuristic might be to split the song into as many non overlapping segments as possible and classify each of those segments of the track and choosing the highest classification of each of those segments. This leads to a model that relates more closely to the manual process in which the tracks are classified. This approach obviously leads to increased computational time per classification, but the trade-off might be worth the performance gain.

Other machine learning modeling approaches are of course interesting to explore. In the paper *Convolutional Recurrent Neural Networks for Music Classification*, the authors (Choi et al., 2016) showcased the recurrent neural network structure for music classification. The model outperformed the *SOTA* for multiple music classification tasks. The recurrent model makes a great deal of sense intuitively, because of the music signals time dimensionality. It would be of great interest to test the performance of a recurrent neural network on this problem and compare the results.

Furthermore we believe that by combining numerous other features of music, a dynamic model could be built. Theoretically, energy level is related to personal taste and music genre. Based on this fact a model could be derived for each genre or category of personal taste. If the dataset provided included several annotations per track, such an approach would be feasible. A possibility would be to cluster annotators by similar annotations and try to derive an understanding of music preference. We believe that such a strategy would be superior to our current one. However annotating all the tracks by each annotator is a time consuming task.

Bibliography

Baccianella, S., Esuli, A., and Sebastiani, F. (2009). Evaluation measures for ordinal regression. In *2009 Ninth International Conference on Intelligent Systems Design and Applications*, pages 283–287.

Baniya, B. K., Ghimire, D., and Lee, J. (2013). Evaluation of different audio features for musical genre classification. In *SiPS 2013 Proceedings*, pages 260–265.

Bogdanov, D., Wack, N., Gómez, E., Gulati, S., Herrera, P., Mayor, O., Roma, G., Salamon, J., Zapata, J. R., and Serra, X. (2013). Essentia: an audio analysis library for music information retrieval. In *International Society for Music Information Retrieval Conference (ISMIR'13)*, pages 493–498, Curitiba, Brazil.

Bowyer, K. W., Chawla, N. V., Hall, L. O., and Kegelmeyer, W. P. (2011). SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813.

Cernian, A., Olteanu, A., Carstoiu, D., and Mares, C. (2017). Mood detector - on using machine learning to identify moods and emotions. In *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, pages 213–216.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754.

Choi, K., Fazekas, G., Sandler, M. B., and Cho, K. (2016). Convolutional recurrent neural networks for music classification. *CoRR*, abs/1609.04243.

Choi, K., Fazekas, G., Sandler, M. B., and Cho, K. (2017). Transfer learning for music classification and regression tasks. *CoRR*, abs/1703.09179.

Dang, T. T. and Shirai, K. (2009). Machine learning approaches for mood classification of songs toward music search engine. In *2009 International Conference on Knowledge and Systems Engineering*, pages 144–149.

Fang, Z., Guoliang, Z., and Zhanjiang, S. (2001). Comparison of different implementations of mfcc. *Journal of Computer Science and Technology*, 16(6).

Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

Lee, J., Park, J., Kim, K., and Nam, J. (2017). Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *CoRR*, abs/1312.4400.

Lin, M., Xuemeng, Y., Mengran, Q., Chen, H., and Yuan, P. (2015). The programming of mel frequency cepstrum coefficient (mfcc) difference's characteristic extraction of parameters and emulation. In *2015 International Conference on Intelligent Transportation, Big Data and Smart City*, pages 895–897.

McFee, B., McVicar, M., Nieto, O., Balke, S., Thome, C., Liang, D., Battenberg, E., Moore, J., Bittner, R., Yamamoto, R., Ellis, D., Stoter, F.-R., Repetto, D., Waloschek, S., Carr, C., Kranzler, S., Choi, K., Viktorin, P., Santos, J. F., Holovaty, A., Pimenta, W., and Lee, H. (2017). librosa 0.5.0.

More, A. (2016). Survey of resampling techniques for improving classification performance in unbalanced datasets. *ArXiv e-prints*.

Nuzzolo, M. (2015). Music mood classification.

Spotify (2018). Spotify web api.

Stein, J. Y. (2000). *Digital Signal Processing*. John Wiley & Sons.

Stutz, D. (2014). Understanding convolutional neural networks.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc.

EXAMENSARBETE Classifying energy levels in modern music using machine learning techniques
STUDENTER Daniel Pettersson, Otto Nordander
HANDLEDARE Pierre Nugues (LTH), Johan Brodin (Soundtrack Your Brand)
EXAMINATOR Jacek Malec (LTH)

Estimera upplevd energinivå i musik, går det?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Daniel Pettersson, Otto Nordander**

Musik har en stor plats i det moderna samhället. Vi presenterar en lösning för att kategorisera musik baserat på upplevd energinivå. Detta öppnar upp fler möjligheter för att matcha den perfekta musiken till den perfekta situationen.

Energinivå i musik

Industrin för streamingtjänster är enorm, bara Spotify uppskattas ha cirka 159 miljoner användare med över 30 miljoner unika låtar, och allt kan nås via ett par knapptryck. För att göra det enklare att hitta i djungeln av musik så hjälper det att gruppera musik via dess drag. Ett drag som urskiljer sig är *upplevd energinivå*, det är ett begrepp som kan beskrivas som hur mycket energi som en musiklyssnare upplever när hen lyssnar på en låt. Energinivåer har valts att representeras som en numerisk skala från 1 till 10. För att få en praktisk förståelse för olika energinivåer har fem representativa låtar valts ut, de följer i ordning från väldigt låg till väldigt hög energi: *An Ending (Ascent)* av *Brian Eno*, *Bridge over troubled water*, *All along the watchtower* (*Bob Dylan*), *Walking on sunshine* av *Katrina & The Waves*, och *Ace of Spades* av *Motörhead.*

Maskininlärning & musik

Med hjälp av dagens teknologi är det möjligt att effektivt hitta mönster i stora mängder data. Det gör det möjligt för datorer att lära sig dessa mönster och associera data med en annoterad klassifikation, för att sedan förutse ny data med en klassifikation. Tekniken benämns som maskininlärning och är väldigt populär för att automatisera annoteringsproblem.

I musikens värld kan detta innebära att matematiskt omvandla musik till bilder och på så vis hitta mönster i bilderna. De låtar som ger upphov till ett visst mönster kan paras ihop med en specifik energinivå. Det föreslagna tillvägagångssättet kan visa sig vara dyrt och tidskrävande, då ett krav är att inneha stora mängder energi-annoterad musik med hög kvalitet. Systemet bygger på cirka 50.000 låtar som har blivit annoterade av ett 20-tal människor. Självklart är det en subjektiv fråga vad en låt ger för energi t.ex. påverkas lyssnaren av att hen har extra tycke till en viss genre eller artist. Det framtagna systemet grundas därav i flertal olika musiksmaker vilket leder till en mer generell klassificering av upplevd energi.

Automatiserad energinivå

Den absolut främsta fördelen med ett automatiserat system är hur mycket mer effektiva datorer är i jämförelse med människor på att kategorisera musik. Vid testningar uppskattas programmet kunna klassificera 10,000+ låtar per timme med ett medelavstånd på 0.84 ifrån den upplevda energinivån hos människor. Det framtagna systemet har visat goda resultat som förhoppningsvis kan ligga till grund för framtida studier av energinivå.