# Machine Learning Project

## Finishing a genius' work

**Group 10**
Boris Luttikhuizen (s2718669)
Dirk Jelle Schaap (s2745135)
Joppe Boekestijn (s2754215)
Lex Koelewijn (s2757036)

February 2, 2020

## Abstract

In this machine learning project we set out to tackle the well-known problem of completing Bach's unfinished fugue. We will use linear regression in order to predict a possible continuation of the unfinished piece of music. More advanced algorithms, with more flexibility, are also possible. Due to the strenuous assertion of the quality of artificially created music, and the limited data, applying linear regression is a logical approach, given the low complexity and good results on relatively small data sets. The data set consists of the beginning of the unfinished fugue provided by the Santa Fé Institute for their 1993 competition. This still leaves us with the challenge of figuring out how to encode the notes. Another issue is that ascertaining the quality of artificially generated music is difficult, seeing as there is no objective numerical measure of what constitutes "good" music. Our goal is to demonstrate the performance of linear regression in music generation and explore ways to measure the quality of the final product. We observe that linear regression shows promising results for the task at hand. The generated music sounds rich. We discuss the quality of the music and the effect of overfitting.

# 1 Introduction

Johann Sebastian Bach has generally been regarded as one of the greatest composers of all time since the 19th-century Bach Revival [1]. In his lifetime he has created an enormous oeuvre consisting of over 1000 compositions [8]. "Bach is revered for his technical command, artistic beauty, and intellectual depth" [4]. The Art of Fugue is an incomplete musical work of unspecified instrumentation by Bach [6].

A subdomain of machine learning has focused on time series prediction over the past few decades . In time series prediction, the goal is to make predictions on data which is temporally dependent on each other. This has been applied to numerous fields, such as stock prediction, weather forecasting, and music generation.

In this project we wanted to use machine learning techniques to try and create a continuation of Bach's famous unfinished fugue. We understand the infeasibility of this endeavour. Not only would it be difficult to mimic the technicality of one of the greatest composers of all time, it is also challenging to assert the quality of the final product. Normally one would want to create an algorithm that learns the underlying rule in the data set and generalizes well to unseen data. However, we do not have a numerical measure to compare the performance of generated music with unseen music. A low generalization error would mean that it mimics seen music, and that is not necessarily our aim.

We want to measure the success of the eventual product subjectively, by listening to the recording. Furthermore we want to assure that overfitting is limited in our model.

# 2 Methods

## 2.1 Data set

In the 1990s a time-series prediction competition was held by the Santa Fé Institute. One of the data sets that was made available for this challenge, was an unfinished fugue written by Bach. In this project, we used that same data set. It is split up into four voices, with each voice represented by a column vector. Each time step is represented by an extra row. Each of the 3824 rows represents 1/16th of a bar. Since this is hard to describe in just words, please refer to Table 1 for extra clarification. The values represent the notes and the repetitions of a value represent the duration of a note. The value of 0 means no sound, and the value of 60 represents the note of a middle C.

## 2.2 Theory

### 2.2.1 Feature extraction

The data set that was provided to us in a *.txt* file, containing a numerical representation of the four voices of the fugue (Soprano, Alto, Tenor and Bass). As

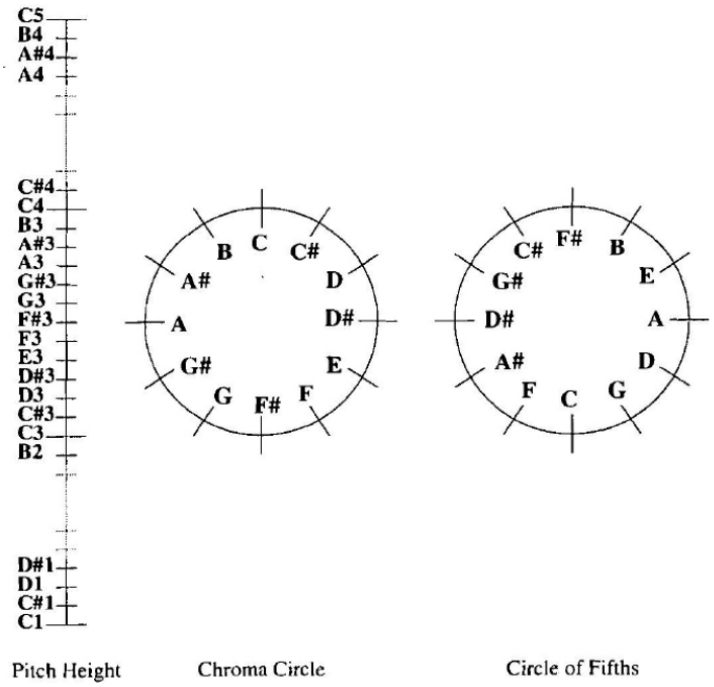


Figure 1: Circle of Fifths and Chroma Circle [3].

explained above, each voice is represented by a sequence of integers representing pitch in the way of piano key indices. The 0's represent a break in that particular voice. This data could be used for linear regression and it will be later, but there are several ways to extract more information from these numbers.

Aalon Kuqi discusses the chroma circles in his bachelor project on music generation [3]. The current representation of our data does not capture the fact that that a larger difference in pitch is sometimes less disturbing than a smaller difference. We will therefore use a different representation of pitch described in the paper by Kuqi. We transform the 1-dimensional pitch vectors of each voice to a 5-dimensional vector. The first dimension is a quantity proportional to the natural logarithm of the original pitch representation. The second and third dimension are derived from the coordinates on the chroma circle. The final two dimensions are derived from the coordinates on the circle of fifths. This 5-dimensional representation mentioned by Kuqi is based on Shepard's theory of generalization. The best way to explain their use of this theory is by means of an illustration. On the left side of Figure 1 we see the logarithmic representation of the pitch and the two aforementioned circles are on the right side of the figure. The logarithmic scaling ensures that that tonal half-steps are at an equal distance from each other. A tonal half step corresponds to the smallest distance commonly used between pitches in Western music [10]. The structure

of the circles is based on common structures in music and the psychological similarity between different pitches. The ordering of the chroma circle is, like the logarithmic transform, based on tonal half-steps, i.e. all neighbors are a tonal half-step apart from each other. The pitches in the circle of fifths are a perfect fifth apart. Perfect fifths belong to the group of perfect intervals in music, which own their name to high degree of consonance [9]. Consonance is a term used to measure how pleasant or acceptable two successive tones in music sound [7]. The influence of these three components in the calculation of a distance between to pitches can be tweaked by changing the diameter of the two circles. In our experimentation we will use the same transformations as Kuqi and fix the diameter of both circles to the length of an octave in the pitch axis. The exact transformation of our integer vectors to these new 5-dimensional representation (we will, similar to Kuqi, also include duration as a 6th dimension), can be found in both the bachelor thesis of Kuqi [3] and in our Jupyter Notebook.

### 2.2.2 Linear Regression

Linear regression is the main technique that we applied in our attempts of creating a continuation of the famous unfinished fugue by Bach. Linear regression is a simple, but in many practical cases useful technique to transform input signal windows to output data [2]. We will first give a quick recap of the basic ideas behind linear regression and ridge regression based on the lecture notes by prof. dr. H. Jaeger [2].

Consider a data set $(\mathbf{x}_i, y_i)_{i=1,\dots,N}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y \in \mathbb{R}$. We want to find the weight-vector $\mathbf{w} \in \mathbb{R}^n$ that minimizes the quadratic loss function:

$$\mathbf{w} = \arg\min_{\mathbf{w}^*} \sum_{i=1}^{N} (\mathbf{w}^* \mathbf{x}_i - y_i)^2. \tag{1}$$

It can be shown that the solution to this problem is given by:

$$\mathbf{w}' = (XX')^{-1} Xy, \tag{2}$$

where $X$ has the input vectors $\mathbf{x}_i$ as its columns. The inversion step in the solution above can cause numerical instability when $XX'$ is close to singular. This instability can be resolved by adding a positive scalar $\alpha^2$ to the diagonal of $XX'$ before inversion:

$$\mathbf{w}' = (XX' + \alpha^2 I_{n \times n})^{-1} Xy. \tag{3}$$

The addition of the positive diagonal matrix in the optimal solution is not only motivated by the instability of the inversion problem, it is also used for regularization. To restrict the flexibility of linear regression one can penalize solutions of large magnitude. To do this, we minimize the following loss function instead:

$$\mathbf{w} = \arg\min_{\mathbf{w}^*} \sum_{i=1}^{N} (\mathbf{w}^* \mathbf{x}_i - y_i)^2 + \alpha^2 ||\mathbf{w}||^2. \tag{4}$$

The minimization of this loss function will once more lead to the optimal solution in Equation 3. The magnitude of $\alpha$ now determines how strongly we regularize our linear regression.

When we apply linear regression to our problem, the desired output values will be 6-dimensional vectors and the sample size will depend on the window size that we use for our prediction. We will therefore now present the more general solution that allows for output vectors of the form $\mathbf{y} \in \mathbb{R}^k$. Consider a data set $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,N}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^k$. The optimal solution $W_{opt} \in \mathbb{R}^{k \times n}$ to the loss function analogous to the scalar one in Equation 4 is given by:

$$W'_{opt} = (XX' + \alpha^2 I_{n \times n})^{-1} XY, \tag{5}$$

where $Y$ contains the output vector $\mathbf{y}'$ on its rows and $X$ is as before.

## 2.3 Implementation

The final system used to create the predictions is a ridge regression using chroma features where some pre- and post-processing was done on the original data. We implemented this using a Jupyter Notebook, the Python programming language and a well known package *SKLearn* [5]. The data consists of a a list of integers, which is not the most informative encoding. We start out by encoding the signal as notes with their respective durations where we chose the maximal duration to be 16 or 4 bars. When a note is repeated more than 16 times in will be split up in multiple tuples of (note, duration). The data set is then transformed using a sliding window of size 40 to create a data set consisting of windows of 40 notes with their duration and 1 parent signal output of a note plus duration which is the note at the next time step. At this stage a single window consists of 80 values of (note, duration) pairs. This window based data set is once more processed such that each note is represented by its chroma representation thus turning a single integer into a 5 feature representation. Now each data point is a 6-dimensional vector consisting of the chroma representation as well as its duration. A single window now consists of 40 time steps of 6-dimensional vectors which are all concatenated into a single 240-dimensional vector. A one hot vector encoding is now generated for each window as well as its parent output. The resulting vector representation a training data window using chroma representation and its duration in a one hot encoding is a 3200-dimensional vector. Thus each individual training window is now represented by a 3200-dimensional vector and accompanying 32-dimensional teacher output. Of this data a train and test split of 80/20 is generated and the resulting training data will be used in order to train the ridge regression.

To give a numerical estimation of how well the system performs the mean squared error (MSE) of the predicted notes in the test data compared to the actual notes is compared. Now that we have a trained regressor all that remains to do is to predict a new signal. This is done in an iterative fashion such that each newly predicted note will influence the next prediction. We start by looking at the final 40 notes of the signal and convert this to the chroma-duration one

hot encoding that the regressor requires and let it predict the new output. This newly predicted signal is appended to the original signal and we repeat the process with what are now the last 40 notes of the signal, in other words we have shifted the entire window one time step.

In order to retrieve the optimal window size we attractively trained a new system entirely, where we loop through a window size range of 1 to 100. The mean squared error of each system is recorded and the resulting plot thus shows the MSE versus window size. Based on this plot we estimated that the optimal window size lies in the range of 30-40 and thus the system was trained using a window size of 40. Another parameter of great influence is that of the alpha, which specifies the amount of regularization applied, in the ridge regression. Initially the model was trained using the default alpha of 1, however a grid search was performed in order to find the optimal alpha suitable for the problem at hand and this proved to be 1500.

## 2.4 MIDI

In order to be able to listen to the music created by the algorithm, we made use of the common Musical Instrument Digital Interface (MIDI) standard. It allows us to encode the output in such a way that it can be transformed into an audio file that is ready for playback. Lucky for us, the range of values in the original data set perfectly matches with the MIDI standard. The conversion from Table 1 to 2 is quite simple yet crucial. The data is now stored in the format of (note, duration). In this way, it is now possible for the MIDI software to know what note to play and for how long.

The second step is to encode the resulting converted data (as seen in Table 2) into a `MIDI` file. We did this using a package available for the Python programming language, called *MIDIUtil* [11]. This software is able to handle multiple tracks with each having separate tones and durations.

The third and final step is to convert this `MIDI`-file into a `WAV`-file, so that it can be easily played and listened to. We found another Python package capable of doing this, called *midi2audio* [12]. This combines *FluidSynth* with a so-called "soundfont" and renders an audio file in the `WAV` format.

| voice_0 | voice_1 | voice_2 | voice_3 |
|---|---|---|---|
| 0 | 54 | 50 | 47 |
| 0 | 54 | 50 | 47 |
| 61 | 54 | 50 | 47 |
| 61 | 54 | 49 | 47 |
| 61 | 54 | 49 | 47 |

Table 1: Sample of stored music data.

| voice_0 | voice_1 | voice_2 | voice_3 |
|---|---|---|---|
| (0, 2) | (54, 5) | (50, 3) | (47, 5) |
| (61, 3) | | (49, 2) | |

Table 2: Sample of converted data.

## 2.5 Evaluation

In our opinion, one of the most difficult things about this project is deciding on a numerical measure for the quality of the generated music. Common measures

such as the MSE or some other prediction error seem useful. However, we quickly found out that these measures do not necessarily indicate good music. In the end, we settled on fully judging it ourselves and relying on our own, subjective opinions. The music presented in the Results section is a compilation of our best and worst findings.

# 3 Results

Please listen to the audio files in the `ZIP`-file accompanied by this report. In the folder called `predictions/` there are three files of interest:

1. **`all_voices_alpha_1.wav`**: This file is our best effort and contains data for all four possible voices.
2. **`voice_1_alpha_1.wav`**: This file contains just one voice. Here we have used the regularisation value $\alpha = 1$.
3. **`voice_1_alpha_1500.wav`**: This file is identical to the previous file and contains just one voice. However, here we have used the "optimal" regularisation value $\alpha = 1500$.

In Table 3 an overview of the resulting MSE for all four voices is given, for both the default and "optimal" values of $\alpha$.

| | $\alpha = 1$ | $\alpha = 1500$ |
|---|---|---|
| Voice 1 | 188,3 | 241,7 |
| Voice 2 | 218,0 | 132,1 |
| Voice 3 | 235,5 | 306,6 |
| Voice 4 | 236,0 | 216,4 |

Table 3: MSE scores per voice for the two values of alpha used.

# 4 Discussion

We were quite surprised by the final results. Especially the first audio file from the previous section within the time interval of 0:18 to 0:40. This section sounds very Bach-like and plausible. The music before and after that interval does not, however, and sounds a lot more chaotic.

The second and third files are examples of how "optimal" model parameters found by an automatic parameter sweep, result in terrible music. In the second audio file the value for $\alpha$ is kept at the default. This results in music that sounds chaotic in places, but decent overall. This value for $\alpha$ is also used for the first file. When we optimize $\alpha$ automatically to reduce the MSE measure, the result is the third file. It sounds terrible and reminded us more of a car alarm than Johann Sebastian Bach. In this scenario the value of $\alpha$ became so large, that it will essentially dominate the loss function during optimisation.

The latter thus implying that our original model with $\alpha = 1$ might be overfitting quite strongly on the original data and just reproducing learned elements from the original data. Due to the nature of music and the task at hand one might argue that this is what you want and overfitting should be accepted. The complexity, and artistic, nature of the task make it challenging to judge what level of overfitting is acceptable. In more general terms, the linear regressions seems quite capable of generating music that approximates Bach's style.

One essential step was the conversion from a list of integers representing notes to a (note, duration) representation. The earlier systems we tested without this data format did work, but they always predicted a repetition of a single note. This makes sense as for predicting the next note the current note has the most influence so the system gets stuck in this loop of predicting the same note. When switching to a format of note and duration separately this problem was solved, which agrees with the notion that now a repetition of a note is a single data point rather than a lot of the same notes followed after each other.

One important thing we that we have all learned during this project is that the pre-processing of data and the selection/construction of features from that data is *very* important. If you were to compare linear regression to deep neural networks, the conclusion would be that the first is relatively simple. You are inclined to believe that linear regression would likely not be able to solve complex problems confidently. However, with the right amount of attention and care, it can definitely still be powerful in complex tasks. Always throwing comprehensive algorithms at complex problems is not the answer. Experimenting with simpler algorithms forces you to be clever and truly learn to understand the problem at hand.

# References


[1] Blanning, T.C.W. *The Triumph of Music: The Rise of Composers, Musicians and Their Art*. Belknap Press of Harvard University Press, 2008.

[2] H. Jaeger. Machine learning: Lecture notes, 2019.

[3] A. Kuqi. Art in echo state networks: Music generation, 2017.

[4] Musica Batlica contributor. Bach, johann sebastian (1685 - 1750), 2019. [Online; accessed 1-February-2020]. URL: `https://www.musicabaltica.com/en/composers-and-authors/johann-sebastian-bach/`.

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

[6] Wikipedia contributor. The art of fugue — Wikipedia, the free encyclopedia, 2019. [Online; accessed 1-February-2020]. URL: https://en.wikipedia.org/wiki/The_Art_of_Fugue.

[7] Wikipedia contributors. Consonance and dissonance — Wikipedia, the free encyclopedia, 2019. [Online; accessed 1-February-2020]. URL: https://en.wikipedia.org/wiki/Consonance_and_dissonance.

[8] Wikipedia contributors. List of compositions by johann sebastian bach — Wikipedia, the free encyclopedia, 2019. [Online; accessed 1-February-2020]. URL: https://en.wikipedia.org/wiki/List_of_compositions_by_Johann_Sebastian_Bach.

[9] Wikipedia contributors. Perfect fifth — Wikipedia, the free encyclopedia, 2019. [Online; accessed 1-February-2020]. URL: https://en.wikipedia.org/wiki/Perfect_fifth.

[10] Wikipedia contributors. Semitone — Wikipedia, the free encyclopedia, 2019. [Online; accessed 1-February-2020]. URL: https://en.wikipedia.org/wiki/Semitone.

[11] M. C. Wirt. MIDIUtil 1.2.1, Mar 2018. URL: https://pypi.org/project/MIDIUtil/.

[12] B. Zamecnik. midi2audio 0.1.1, Nov 2016. URL: https://pypi.org/project/midi2audio/.