Dear Roy, Alex, Peter and Rishabh,

You picked the most challenging project among the four ones for which I provided templates. As you correctly found out, there is a big tension between statistical optimization of models vs. a good human listening experience. You spent a very recommendable effort in trying out different types of models, and you had nice ideas about preprocessing - so, all is fine with the effort that you spent and your dedication to this challenge! The main thing where there is still leeway for improvement is the technical writing - you rely too much on words, and leave out important technical detail. I could not exactly understand in several places what you actually and concretely did, and I am afraid that in some places you suffer from some remaining misconceptions (see my inline comments - best viewed with Acrobat Reader, other pdf readers may not display all the sorts of comments). I hope that altogether you could learn a lot from this hefty exercise, despite all the difficulties that you met, saw, and struggled with.

Best wishes for your future exploits in machine learning!

You find the grading on the next page.

Cheers, Herbert Jaeger (March 3, 2021)

Evaluation form for semester project report, Machine Learning

Name of students:	Roy David, Alex Hill, Peter Varga, Rishabh Sawhney
Name of grader:	Herbert Jaeger
Date:	March 3, 2021

Fill in violet fields with grades on scale 0-10. If not applicable, put weight to zero. For bonus, fill in green field; this number will be added to the final grade (capped at 10.0).The max for bonus is 2.0 -- that would be earned by a publication-ready report with relevant results.
For Latex use, enter a 1 in the red field when Latex was used, else 0.

	grade	weight %	total	criteria	comments
Presentation (30%)					
Figures, tables, pseudocode	8.00	5	0.4	good quality figs? Legibly-sized texts in graphics? Captions? Symbols explained in caption? Figs etc. referenced in text? Sources of imported graphics given? Figs informative (not redundant)? Pseudocode transparent?	
Technical writing	6.00	25	1.5	General appearance (general visual appearance; not over-formatted (too many fonts); meaningful, standard sectioning; nice title page; table of contents)? clear formulations? Concise formulation? Mastership of technical formulations / math formulations? Mathematical formalism used whenever appropriate? Good text flow? Appropriate language (not sloppy in technical sections, not too dry in motivation sections?) Clear flow of argumentation? Contents well structured? Correct English, typos? are experiments/implementations clearly, completely, succintly described? Could experiments be checked/reproduced? results clearly,	
Latex	1.00		0	was Latex used? If yes leave the 1.00, else flip to 0.00 in the red field	
Technical quality (70%)					
Penetration of subject	6.50	20	1.3	Good task statement? Challenges inherent in dataset perceived and adequately addressed? choice of preprocessing, feature extraction, learning algorithm motivated? the used algorithms are understood? Insightful examples? Could student assess relevance within field? Pertinent future research sketched?	
Technical work	7.00	30	2.1	Maths correct? Conceptualizations and reasoning make sense? ML basics done correctly? (e.g. reporting test errors not training errors, hyperparameter optimization). Appropriate, even clever design ideas? Results connected back to starting question? Statistical basics done properly where applicable (error bars!)? Implementations / experiments reasonably thought out? Planned professionally (modular, efficient, transparent, documented)? Insightful discussion? well-chosen references? Number appropriate?	
Exhaustiveness	8.00	20	1.6	Is the amount of work done adequate for the given project time? Variations of methods explored?	
Bonus	0.20			extraordinary achievements, e.g. much extra work, very independent work, very difficult topic, interdisciplinary connections, original thinking. Up to 2 bonus points possible	
Raw grade			***7.10***		

Comparison of Regression Models for Completing Bach's Famous Unfinished Fugue

Roy David **Alex Hill** **Péter Varga** **Rishabh Sawhney**
s2764989 s3417549 s4291093 s4133366
University of Groningen
{r.a.david, a.f.hill.1, p.varga.1, r.s.sawhney}@student.rug.nl

Abstract

In this project, we perform the synthesis of 25 seconds of music for Bach's unfinished Fugue through the use of Machine Learning models. This is done using three distinct models such as Linear Regression, Ridge Regression and Gradient Boosting Machine (GBM). We then compare the performance of the different models over the same set of training data provided to us in the Musical Instrument Digital Interface (MIDI) format. The pitfalls of the models are then analysed along with the data processing methods and underlying loss functions.

1 Introduction

The art of music has been around for centuries. Ever since the advent of Machine Learning and AI, a question that has frequently come across data scientists and music connoisseurs' minds is whether a computer can generate music at the same level (or better) than humans. This was the main factor in driving research in this side of the spectrum. However, generating music in practice given some sample set of ordered notes has turned out to be more challenging than previously imagined.

One of the significant events that made scientists realize the difficulty involved in producing music using sampled data was the Time Series Analysis and Prediction competition by the Santa Fe Institute in 1993. The challenge was to add about half a minute or 25 seconds to the six scores of music, one of them being an unfinished fugue by J.S Bach. This score was left incomplete due to Bach's untimely death and forms the core data set behind this project. The completion of this fugue is considered as an unsolvable problem which remains unresolved to this day.

In order to compose such kind of music, there has been extensive research using different kinds of models ranging from state space models such as Markov Chains, time series models involving the different flavours of Linear Regression as well as utilizing the immense power of Convolution Neural Networks as well as Recurrent Neural Networks.

In our project, we focus on utilizing Linear Regression as a start for the target of generating additional 25 seconds to the fugue. This is followed by involving slightly more complex models such as Ridge Regression and ultimately the Gradient Boosting Machine. To evaluate the quality of the generated notes, the models each utilize an appropriate loss function (cross entropy or MSE). Using this loss function we were then able to train the model and fine-tune the hyper-parameters via cross validation.

2 Data

The dataset we used for our project is identical to the one which was presented by the Santa Fe Institute. Table 1 shows an example sequence of the dataset. The total number of rows is 3824, which amounts to a length of approximately 3 minutes and 11 seconds, with a sampling rate of 10000Hz and 1/20s duration per row. Each column corresponds to a voice from the standard vocal range, also mentioned in [1]. Each number indicates an individual pitch, encoded in the standard Musical Instrument Digital Interface (MIDI) format, formalized by the MIDI Association. According to this format, the possible values are within the range of $\{0, 1, \ldots, 127\}$, going from lowest (8.18Hz) to highest (13289.75Hz) frequency, as detailed in [2]. The related frequencies from pitches can be retrieved by the following formula:

$$f = 440 * 2^{k-69}/12, \quad (1)$$

where k stands for the value of the given encoded pitch (e.g. 69 corresponds to $A4 = 440$Hz).

Soprano(1)	Alto(2)	Tenor(3)	Bass(4)
60	57	42	38
60	56	42	38
60	56	42	38
60	54	42	38
60	54	42	38
61	52	49	37

Table 1: Excerpt from the dataset with the different voicing as columns.

Hereinafter we are going to refer to each voice/singer by its respective column number (e.g. first voice/singer meaning Soprano), and to each pitch value as a single note. An example voice as a function of discrete time n can be seen in Figure 1.

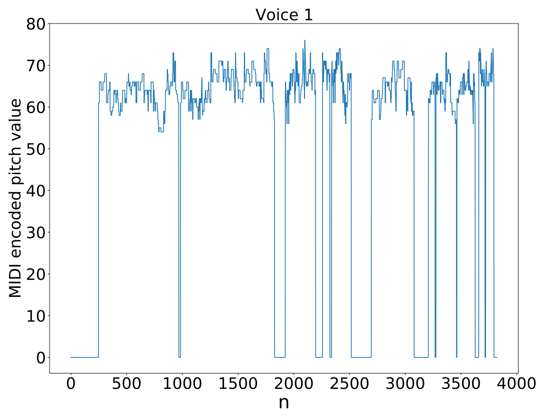


Figure 1: Original notes as a function of time (Voice 1).

3 Methodology

In our approach we decided to first create a simple model to get acquainted with the problem and the difficulties of this challenge. Afterwards, we introduce two more refined alternatives whose purpose is to exploit the hurdles we discovered during the implementation of the first simple model, and improve upon them. Our first model is a simple Linear Regression model using little pre-processing and post-processing, and some trial-and-error based fine-tuning to help with music generation. The second model is a Ridge Regression model, which is aimed to be a more sophisticated model than the modest Linear Regression. Besides the regularization Ridge Regression provides, we are also using more pre-processing, post-processing, and increased stochasticity with less reliance on fine-tuning and more focus on performance on the provided dataset. For our third model we decided to venture on a completely different direction and try a Gradient Boosting Machine. This was solely aimed at getting more variance/randomness within the predicted notes, since we had noticed from the first model a strong tendency to give higher probabilities to either the most frequent note present in a voice or the current note (depending on the pre-processing). For music generation, all our models generated roughly 25 seconds of music for each voice separately that were combined afterwards.

For all of our implementations we used Python3 as our main programming language, with MATLAB being a supplement to generate the audio files from text files.

All the source code detailing the model architectures, for this project can be found on `https://github.com/RUG-ML/Bach_Unfinished_Fugue`.

3.1 Linear Regression

The mathematical foundation of Linear Regression that finds the weight vector/matrix which fits the best to the dataset is as follows:

$$W'_{opt} = (XX')^{-1}Xy. \quad (2)$$

Linear Regression solves for the parameters by minimising the squared sum of residuals. This allows for a single closed form solution based on the data, and thus no iterative optimisation has to take place.

Window-based Linear Regression can often be used as a first approach for time-series prediction tasks. It is fast and relatively easy to implement and with large windows and a careful regularization can perform well on many time-series prediction tasks. If not used as the final model, it will at least give a baseline for more sophisticated methods [3] (chapter 3.2).

3.1.1 Pre-processing and Post-processing

The pre-processing consisted of two steps used for both Linear Regression models, and an optional step, only used for one of the models. Firstly, convert the data into n-dimensional one hot encoded vectors. The n was determined by the amount of unique notes within the voice (set of notes). This is needed for Linear Regression, in order for it to output a probability vector. Secondly, convert the data into a window size of one, where X is in time t (the now/current note), and y is in time $t + 1$ (next note). Here an instance of $x \in X$, would be the current note, where X would be all the input notes, an instance y would be the next note, where y would be the target note and all y's would therefore be all the target notes. Lastly, the optional step is to reduce the ranges of the notes to one. We will discuss this further in subsubsection 3.1.3.

The only post-processing step for this model, was to map the chosen index of the probability output vector to the corresponding index of the set of notes from that voice. In this way the note corresponding to the index was retrieved and used for the music generation part.

3.1.2 Model

Our first model is the most simple and straightforward one. It has the least amount of preprocessing and post-processing of the data, smallest window size (of one) and uses simple Linear Regression to get a probability output vector. Such an output vector would resemble the probabilities for each note within the set of notes present in a voice. These probability vectors where compared to the target/teacher output to determine performance on the dataset. We used cross-validation to examine the performance of the model on the dataset and as a possible indication for the quality of music generation.

For the music generation, a random note was picked from the top 10 notes with the highest probabilities. Depending on whether the sequences of the notes were reduced to one or not, this would avoid converging to the same note(s). If the sequences of the notes were reduced to one representative note and removing repetitions (e.g. $[0, 0, 0, 1, 2, 2]$ to $[0, 1, 2]$), this would increase the amount of unique notes used in the generated voice. If the sequences of the notes were not reduced, this would avoid converging to the current note. After that, the length/range or sequence of the note (e.g. $[1, 1, 1]$ is a sequence of 3), was determined via a weighted choice from all the ranges/sequences present within the voice of that note. This was carried out to maintain the style of the voice. We added a $*2$ multiplier to this chosen range to improve (subjective) performance of the music to extend the range of the generated/chosen note.

In the end we opted for two models: one that would perform better on the dataset, the other with more (subjective) ”music logic” and supposedly better musical performance. However since the music generation part was the same for both models, disregarding the prediction of ranges/sequences and the addition of randomness with regards to picking the next note the difference in terms of (subjective) music performance could be negligible.

The model choices consisted of: the default Linear Regression implementation from scikit-learn [1]; a k value of 5 for cross-validation; and picking from the top 10 notes with the highest probabilities.

3.1.3 Optimisation

The only optimisation step for this model was done by trying to improve the pre-processing of the data. Firstly, as described in subsubsection 3.1.1, this consisted of converting all instances to one-hot encoded vectors with a window size of one. Secondly, similarly to the first one but with the restriction of the range of sequences of notes set to one, so the next note would always differ from the current one. This to increase the probabilities of the next note being different ~~than~~ the current one, to increase the variance of the next predicted note (i.e. training a model to predict the next note where the probability that the next note differs from the current note is low (if the ranges of the notes are not reduced to one) *vs.* training a model to predict the next note where the probability that the next note differs from the current one is high (if the ranges of the notes are reduced to one)).

A more general optimisation for the music generation part included to involve a certain degree of randomness to avoid converging to the note(s) and a range determiner based on a random weighted choice (as discussed in subsubsection 3.1.2).

3.2 Ridge Regression

The mathematical notations and terminology that we use within this subsection strongly follows

[1] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

[3] (chapters 3 and 6), with occasional modifications for consistency with preceding and subsequent sections.

Ridge Regression [4] is a type of Linear Regression that manipulates the original formula in (2) to find the optimized weight matrix for our model h, by the addition of a multiple of the identity matrix I before inverting XX', which results in the following:

$$W'_{opt} = (XX' + \alpha^2 I_{nxn})^{-1} Xy. \quad (3)$$

The effect of adding this regularization term with a correctly chosen α is not only an increased numerical stability, but it also decreases the empirical risk and helps in minimizing the risk. The empirical risk for our Ridge Regression is defined as

$$R^{emp}(h) = \frac{1}{N} \sum_{i=1}^{N} L(h(x_i), y_i), \quad (4)$$

where $h(x_i)$ is the model's output upon input x_i (a single datapoint), and y_i (a single label) is the teacher output. As temporal learning tasks are often able to capture more information of the data by subsequent past data points being used for predicting y_i, we have decided to execute the learning algorithm with an input window $x_{i-w+1}, x_{i-w+2}, ..., x_i$, rather than a single input x_i, where w denotes the size of the window.

3.2.1 Pre-processing and Post-processing

There are two types of models that we have built with Ridge Regression, called *note model* and *octave model*. These two models' outputs are combined together to form a single output. Thus, to achieve the possibility of combination, we had to implement two kinds of pre-processing before the models could begin learning from the dataset. A detailed description of these models is included in subsubsection 3.2.2.

Firstly, in hope of a less complex distribution and a better generalization ability, we reduced every note to a simplified representation. The method of the reduction is based on an implicit property of the given note, namely its octave. By extracting the octave from each note $x_{i_{i=1,...,N-w}} \in \{0, 1, ..., 127\}$, they can be securely reduced into a significantly lower range resulting in $x_{i_{i=1,...,N-w}} \in \{0, 1, ..., \cancel{12}\}$ without any loss of information. An example plot (of the first voice) on the progression of notes as is illustrated in Figure 2. A break among the notes in Figure 1 is associated with a value of 0, whereas in Figure 2 it is equal to -1 due to the 0 value being reserved for the C note in the new format.

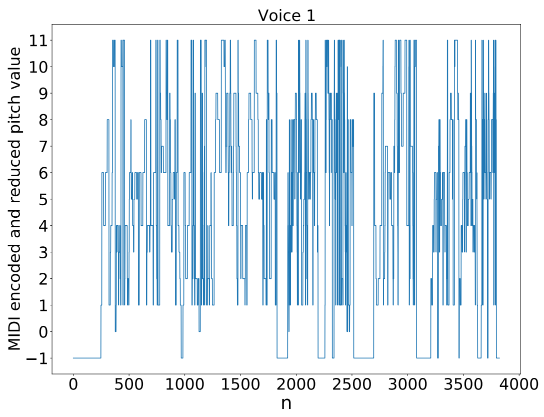


Figure 2: Reduced notes as a function of time (Voice 1).

As the second step of the first type of pre-processing, similarly to the discrete-to-continuous transformation in subsubsection 3.1.1, we used a one-hot encoding on the dataset. The resulting input vectors are in the form $x_i \in (\{0, 1\}^{13})^w$. Additionally, the original output probability vector $y_{i_{i=1,...,N-w}} \in \{0, 1\}^{128}$ becomes a new probability vector $y_{i_{i=1,...,N-w}} \in \{0, 1\}^{13}$, where each entry in y_i is associated with a certain type of standard note from music theory, e.g. [C, C#, D, E, .., B]. The window-based regression problem for binary-encoded the notes is defined as $f : (\{0, 1\}^{13})^w \rightarrow \{0, 1\}^{13}$.

Afterwards, as the second type of pre-processing (implemented for the octave model), we plotted the extracted octaves in hope of observing an inherent rule or a repeating pattern. Figure 3 depicts the octaves of the first voice with the same progression of time as in Figure 1, for example. From Figure 3 it is clear that the octave values with respect to the entire first voice remain in the set $S_1 \cancel{\in} \{-2, 3, 4, 5\}$ of $[S_1, S_2, S_3, S_4]$. The value -2 stands for a break - not initially included in the MIDI Note Number Chart (see Appendix for an overview). The same behaviour of the notes, remaining in a set of 4 octaves throughout the entire song, was observed in every other voice as well, with the exception of different elements being present in the respective sets. As a result, we restricted the possible number of entries of the output vector to include only four values, with the same one-hot encoding method being performed on the inputs and outputs that we used for the notes. Thereby, the window-based regression problem for the binary-encoded octaves is defined as $f : (\{0,1\}^4)^w \rightarrow \{0,1\}^4$. A complete set of plots of all voices and their octave progression can be seen in the Appendix.

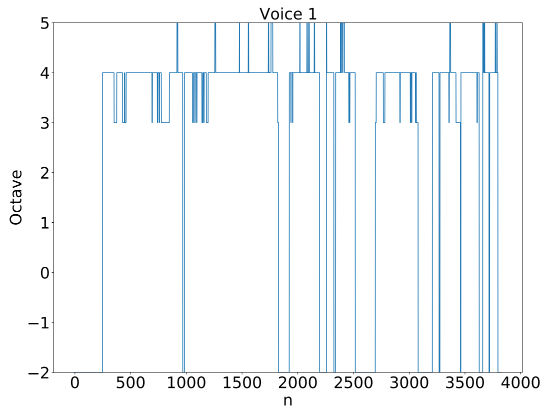


Figure 3: Octaves as a function of time (Voice 1). Refer Figure 6b,Figure 6c and Figure 6d for the other voicings in the Appendix.

3.2.2 Model

The main reason behind our choice of training multiple types of models is the hypothesis of a Ridge Regression being able to perform better given a smaller range of values, and thus we expect a lower empirical risk and expected risk in the separated predictions. As a consequence of the extraction of octaves, it is not feasible to solve the task with only two models in total for all four voices. Therefore, we created eight (1-1 from each type per voice) models in total.

The first type of model is the *note model*, which was trained on the reduced 3824 datapoints (of the corresponding voice). The second type of model is the *octave model*, which was trained on the extracted octave of each note, amounting to an equal, 3824 data points per voice. The main parameters of the models that determine their performance consist of: w (size of the input window), α (multiplier of identity matrix), m (an element-wise power operation's value on the output vector).

Following the optimisation, detailed in subsubsection 3.2.3 and subsection 4.2, the two models' outputs are combined together at each prediction. The combination results in a single note $\hat{y}_{opt}$ per step that bears the original MIDI encoding format. The basic formula to combine the predictions is

$$\hat{y} = \hat{y}_{note} + 12(\hat{y}_{octave} + 1). \qquad (5)$$

However, (5) alone does not satisfy the corner case of only one of the model types producing a break, thus the following adjustment had to be made. We considered a note prediction to be more dominant than an octave prediction, thereby when an octave prediction is a break, but the note is not, then a random octave is picked from the four possible octaves of the given voice. This leads to a scenario where the only occurrence of a break in (5) is when $\hat{y} = -13$, thus resulting in

$$\hat{y}_{opt} = \begin{cases} \hat{y}, & if \;\; \hat{y} \neq -13 \\ 0, & otherwise \end{cases} . \qquad (6)$$

All the models with Ridge Regression have been implemented with scikit-learn's built in Ridge Regression model [2].

3.2.3 Optimisation

In order to find the best-performing model on the validation sets during a k-fold cross-validation for each individual model for both model types, we conducted a grid-search for all the relevant parameters. The possible values for the size of input window ranged from 1-150, the values we searched from for α are 0.001, 0.01, 0.1 and 1. The last parameter, m, was tested for values 1 (no change), 2, and 3.

The main expectations we had were that an increasingly higher w should presumably result in less reliance on α - good generalization ability without forcing it, paired with increasingly higher performance (e.g. less ~~test~~ error) via a positive correlation with m. The final values of the parameters can be found in Table 3 within subsection 4.2. The abbreviations $N_{i=1,..4}$ and $O_{i=1,..4}$ stand for the note and octave models, respectively.

3.3 Gradient Boosting Machine

The final method we implemented is a Gradient Boosting Machine (GBM). This method is quite different to the previous two models, with the techniques of evaluation (loss function) and the preprocessing steps differing greatly. Gradient Boosting Machines work by iteratively adding decision trees where each new tree aims to correct the 'mistakes' of the trees previously added. The specifics of Gradient Boosting Machines is an interesting matter beyond the scope of this paper, for the details on such models please see [5, 6]. In this

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html`

ensemble-based model, the sub-models/decision trees are regressors as opposed to the more standard classification. This model will also aim to add a greater level of variance within the generated music, in the hopes of improving upon the previous two models.

3.3.1 Pre-processing

Similarly to the previous methods, the pre-processing of the data for this model began with extracting features from the data taken from standard music theory. Most notably, the octaves and notes of the four singers were extracted, and also a binary variable depicting whether each singer was taking a break or not. A striking difference between this model and the previous models is that each individual model (for each singer) is built upon features taken from all four singers simultaneously. In this way you could say the artificial singers 'listen to each other' whilst singing. Once the features from all four singers were extracted, a window size w was decided and then the same musical features (octaves, notes, etc.) were then taken for the previous w time-steps.

3.3.2 Model

To implement this in Python we used the package CatBoost which is very well suited to dealing with categorical features in the training data. As Gradient Boosting Machines are an ensemble of models, to alter the flexibility and accuracy of the ensemble, one looks at the diversity of sub-models and also their inherent accuracy. As the data is time-series, we couldn't use the technique of bagging to randomly sample data for the decision trees. However, we could certainly use random feature selection. Furthermore, each tree within the ensemble was limited to a maximum depth of 3. This aims to achieve the goal of minimising the risk of overfitting as each tree is relatively short. Additionally, the number of decision trees within the ensemble can be altered to boost model accuracy or combat overfitting. To deduce the optimal number of trees for this model we will use k-fold cross validation (with $k = 5$) to find the 'sweet spot' between the bias and variance, and consequently minimize the generalisation error.

To ensure that the model didn't select the same note over and over again, a sub-model was built to predict the lengths of each note. Once the length of the current note had been realised, the model was 'forced' to pick a new note to the one previous. Additionally, we added some small noise to the notes when the music was detected to be overly repetitive.

3.3.3 Optimisation

To optimise this model, we decided to use k-fold cross validation for multiple parameters in a grid search. In Figure 4 we have the cross validation for each of the four singers against the 'Number of Trees' parameter.

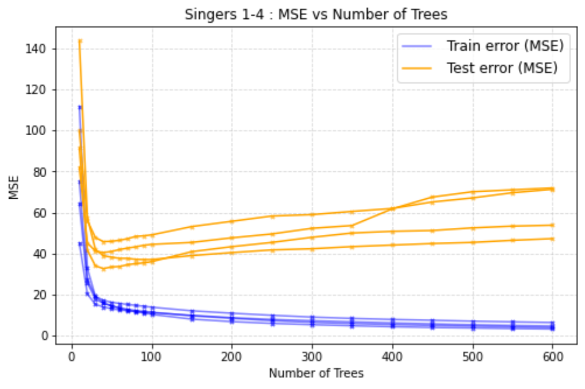


Figure 4: Mean-squared error (MSE) vs Number of Trees using K-fold Cross Validation

Four each of the four singers, there is a clear trend for both the training error (Empirical Risk) and the test error (Risk). As expected, the training error gradually decreases as the model flexibility (given by the number of trees) increases. Conversely, the test error first dramatically drops as the model flexibility increases, and then slowly increases again as the model starts to become overly complex. The gradual increase of the test error can be attributed to the model overfitting on the training data. For our final selection for the number of trees we want to solve for the following equation:

$$h_{opt} = \underset{\theta \in \mathbb{R}^{\geq 0}}{\text{argmin}} \frac{1}{N} \sum_{x_i \in \mathcal{x}_{test}} L(h(x_i), y_i) \ . \qquad (7)$$

Where θ can be any hyper-parameter we want to optimise for. In the case of the number of trees in Figure 4 we can see that the optimal model is given when the number of trees is approximately 35. This methodology can then be applied to all of the hyper-parameters simultaneously using a grid-search. The grid-search will test a range of values for the different parameters. The results of this grid search will be discussed in the results section.

The extent to the utility of such a hyper-parameter optimisation is questionable in our case.

One must keep in mind the true performance measure of models in this paper; the quality of the resulting music. Unfortunately, as there is currently no loss function for this particular measure we must rely only on our own ears and use a trial-and-error method to fine-tune the model. The hyper-parameter that made the biggest change to the quality of the music was undoubtedly the window size of the regression. For this reason, the final window size was hand picked by us to attempt to achieve the tremendous goal of producing 'good' music. The window size that produced the most promising results was $w = 50$. This makes intuitive sense as music is very dependent on the past, and thus we want our models to reflect this intuition.

3.4 Evaluation

For evaluation of our regression models (Linear and Ridge Regression) we used the log loss [7], also known as logistic loss or cross-entropy loss [3], in combination with cross-validation [4]. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. This differs from mean squared error (MSE), typically used for regression. Since the output of this model consists of vector probabilities, the cross-entropy loss fits best.

Cross-validation for time-series prediction tasks is a variation of k-fold cross-validation, which returns the first k folds as train set and the $(k+1)$th fold as ~~test~~ set. Different from standard cross-validation, time-series cross-validation needs to keep the structure of the data to preserve the time aspect of it. Therefore a random shuffle of all data instances, as is done in standard cross-validation, cannot be done since this structure would then be lost. For all our models the value of k was set to 5.

For our third model (Gradient Boosting Machine) we used the same time-series cross-validation method with the same k value as for our first two regression models. Instead of the cross-entropy loss we used MSE as a loss function. Since this model outputs notes rather than n dimensional probability vectors, MSE is preferred.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html`

[4] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html`

4 Results

This section discusses the performance as well as the results achieved by the different models in greater detail. Due to different models, pre-processing steps resulting in different datasets, the evaluation methods can differ depending on the model and its respective dataset.

However, there are two common characteristics amongst all the models. Firstly, all the optimisations have been carried out in a fully supervised manner. Secondly, the final process of generating notes that are ultimately added to the original song with each model has been performed by continuously appending the currently predicted note to the dataset, and thus making the models gradually rely more on their own notes.

With these differences and commonalities in mind, we proceed to the individual results.

4.1 Linear Regression

Here we will go over the results of the Linear Regression model. This consists of comparing the model with the extra pre-processing step of reducing the ranges of the notes to one (as described in subsubsection 3.1.1 and subsubsection 3.1.3) *vs.* the model without this extra step. We will refer to these models as Linear Regression with reduced ranges to one (LR + RR) and Linear Regression with full ranges (LR + FR) respectively.

Note that these models are not directly comparable to each other since ~~there~~ respective datasets differ from each other. Making it harder for the LR + RR model, due to the increased variety of x and y pairs. Due to the reduced ranges to one, the next note or target y would always differ from the current input note or x. Whereas for the LR + FR model this dataset would contain a lot of the same x and y pairs, x being the current input note, being the same as the y, being the target output note.

With this in mind we do try to compare them, to give us an indication to what extend the model stored information from the data.

From Table 2 we can see the log loss scores from both the LR + RR model and the LR + FR model on their respective dataset. We see that the LR + FR model outperforms the LR + RR model in terms of log loss scores achieved on their respective dataset. This shows that the LR + FR model is better suited to its dataset than the LR + RR model is to its dataset.

This does not necessarily mean the LR + FR

Voice \ Model	LR + RR		LR + FR	
Soprano	train	1.34	train	**0.51**
	test	7.59	test	**2.98**
Alto	train	1.32	train	**0.59**
	test	7.25	test	**4.31**
Tenor	train	1.46	train	**0.55**
	test	6.54	test	**1.71**
Bass	train	1.53	train	**0.65**
	test	7.52	test	**1.98**
Mean	train	1.41	train	**0.58**
	test	7.23	test	**2.75**

Table 2: Linear Regression with ranges reduced to one (LR + RR) *vs.* Linear Regression with full ranges (LR + FR). Comparison between train and test log loss scores (lower is better) per voice. Best scores per voice are in bold.

model will generate the better music. Since (i) the music generation part is also influenced by a randomizer and range determiner and (ii) performing well on the dataset does not in itself mean that the model has a better idea of the structure of the music. This was determined by listening to the produced music samples as a group, deciding which music sample sounds better. The music sample generated by the LR + RR model was deemed better than the sample generated by the LR + FR model, though the difference between them is small.

4.2 Ridge Regression

Within this subsection we are going to discuss the obtained train and test error values of the Ridge Regression models, in relation to the investigated parameters, formerly mentioned in subsection 3.2.3.

As it can be seen in Table 3, the window sizes tend to be a lot higher in the case of the octave models, which is an unexpected behaviour considering that the dimensions of the octave model are significantly lower.

Most surprisingly, with the best performing models the parameter α was always found optimal with a value of 1, whereas other window sizes often produced a high variance among them. A sign of better generalization potential on the dataset when the model initially overfits, but then gets regularized, instead of trying to overfit less without regularization. Additionally, the parameter m was

Model	d	α	m
N_1	31	1	2
N_2	17	1	2
N_3	52	1	2
N_4	17	1	2
O_1	89	1	2
O_2	62	1	2
O_3	120	1	2
O_4	73	1	2

Table 3: Optimal parameters of Ridge Regression models

found the most influential among all, a wrongly chosen m (e.g. 1 or 3 instead of 2) tends to render the window size completely meaningless and introduces drastic overfitting.

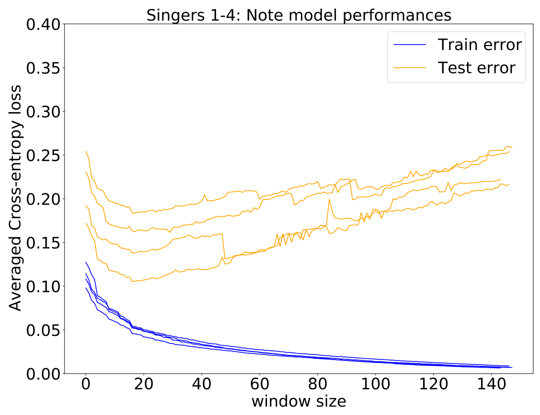


Figure 5: Performance of note models as a function of window size

As an example of the outcome of the window size search, the note models from all the four voices can be seen in Figure 5. All four test curves show a continuously present, sharp, rising-falling behaviour in some form, particularly intensified for window sizes greater than 60. This could be an indicator of inherent properties and patterns of the music that interfere with large window sizes.

Following the quantitative analysis of the chosen models, we listened to the generated music to see whether it was capable of adhering to the structure of the fugue or not, whether the repetition and length of notes managed to resemble the fugue's style. Unfortunately, the parameters found by optimization had a poor performance when it came to generating notes. This is especially due to the tendency of converging to a single note. Consequently, the window sizes were fine-tuned until we eventually agreed that a pseudo-Bach style has been attained.

4.3 Gradient Boosting Model

The grid-search for the GBM yielded the following final parameters:

Parameter	Grid Search Result
Number of Trees	60
Maximum Tree Depth	3
Learning Rate	0.01

Table 4: Grid search result for three key parameters for the Gradient Boosting Machine

With an average MSE of 43.914 on the validation data. These quantitative results are important for the optimisation of the model parameters, however, the qualitative analysis i.e. the quality of the music, is far more important to the overall project. Therefore, we dedicate the rest of this sub-section to the qualitative aspects of the generated music.

The first version of the GBM had the issue that the resulting music was too monotone. It would tend to predict the same values too often. In order to counteract this, the sub-model which predicted the lengths of each note was utilised alongside the addition of some small noise to the notes when the music was detected to be overly repetitive. In this way we were able to add a degree of stochasticity to the model and this significantly helped the output sound much more like regular music. However, the music was 'too sporadic' and lacked any long term structure or pattern. For this reason, the GBM was not the most successful model we managed to build.

5 Conclusions and Future Work

This project aimed at generating music to add roughly 25 seconds of music to Bach's famous unfinished fugue using three distinct machine learning models. Our first two models used some form of regression (Linear and Ridge respectively), our third model used a Gradient Boosting Machine. All our models did manage to create 25 seconds of music, which all sound different from each other. Subjectively we as a group ranked our generated 25 second music samples from best to worst: Ridge Regression, Linear Regression with ranges reduced to one, Gradient Boosting Machine. This was determined based on our subjective opinion of the generated music sample as an extension of Bach's fugue. All three models struggled to keep a strong sense of structure in the 25 seconds music generated. The music sample generated by our Ridge Regression model most successfully resembled musical structure.

All three models had the same overall goal, however, the individual goals were different between models. The goal of our first (simple) model, Linear Regression, was to get acquainted with the challenge and function as a baseline for our other (more sophisticated) models. Our second model, Ridge Regression, aimed at overcoming the challenges found in the process of creating the simple Linear Regression model, therefore eventually generating a better quality 25 second music sample. As described in subsubsection 3.2.2 the Ridge Regression model is different from the Linear Regression model in not only model architecture but also pre-processing, post-processing and music generation. The last model, GBM, aimed at overcoming the converging problem (after n generated notes the next generated note(s) would always be the same) our first two models suffered from. All three models achieved their individual goals.

The biggest challenge was to maintain some structure that is present in music. Writing music is a creative process without boundaries, however there are unwritten rules that make a set of notes sound pleasant. These unwritten rules are hard to retrieve using only simple machine learning methods, such as the ones we used, since music generation is more than just a time-series prediction task. A way to try and keep this structure is to have some model that has some memory of past notes since this does influence the next ones.

Some limitations of this project was the limited amount of data. Because of this we could not resort to Hidden Markov Models or Deep Learning methods (that can have some memory of past notes) since these methods require large datasets to work effectively.

Future work could consist of experimenting with more advanced models that have some memory of past notes, to see the benefits (if any) for music generation and time-series prediction on musical data. Hidden Markov Models and Recurrent Neural Networks (e.g. with Long Short Term Memory) are examples. In order to work with such models, we would need to collect extra data. This data could be in the same style (e.g. writer or genre) of the music to be generated or with a different style, which would be a fun and interesting research topic in itself.

References

[1] A Weigend and M Dirst. Baroque forecasting: On completing ~~js bach's~~ last fugue. In *Time Series Prediction: Forecasting the Future and Understanding the Past: Proceedings of NATO advanced research workshop on comparative time series analysis, Santa Fe, New Mexico.*, pages 14–17, 1993.

[2] Inspired Acoustics. Midi note numbers and center frequencies. URL `https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies`.

[3] Herbert Jaeger. Machine learning lecture notes (kim.ml09). Bernoulli Institute, Rijksuniversiteit Groningen, January 2020.

[4] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970. URL `https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488635`.

[5] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in ~~n~~eurorobotics*, 7:21, 2013.

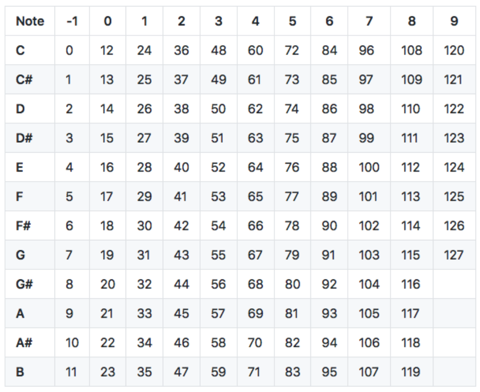
[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. ~~s~~pringer, 2006.

Appendix

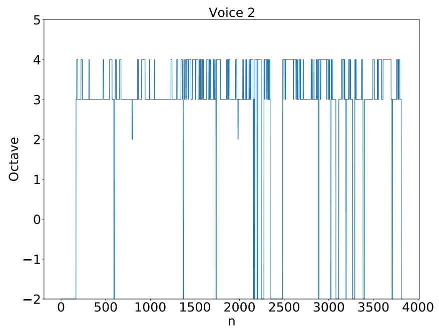
(i) The MIDI format used to obtain the notes given in the dataset. The ten musical notes from C all the way to B are mapped to numbers ranging from 0 all the way to 127. The notes are repeated as we go higher the octaves, the last octave being in the range of 120-127. This format was used in order to get the output as an audio file from the text file produced by the different models. A notation is provided in Figure 6a.

(ii) The results for the octave distribution with respect to time for the other voices from the Ridge Regression model can be viewed from Figure 6b, Figure 6c and Figure 6d.

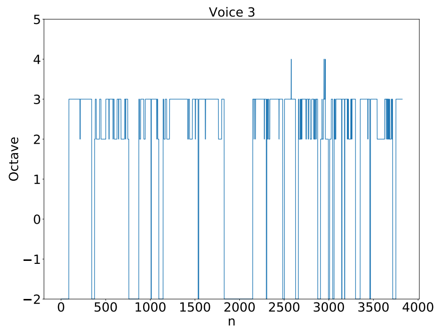


Note	-1	0	1	2	3	4	5	6	7	8	9
C	0	12	24	36	48	60	72	84	96	108	120
C#	1	13	25	37	49	61	73	85	97	109	121
D	2	14	26	38	50	62	74	86	98	110	122
D#	3	15	27	39	51	63	75	87	99	111	123
E	4	16	28	40	52	64	76	88	100	112	124
F	5	17	29	41	53	65	77	89	101	113	125
F#	6	18	30	42	54	66	78	90	102	114	126
G	7	19	31	43	55	67	79	91	103	115	127
G#	8	20	32	44	56	68	80	92	104	116	
A	9	21	33	45	57	69	81	93	105	117	
A#	10	22	34	46	58	70	82	94	106	118	
B	11	23	35	47	59	71	83	95	107	119	

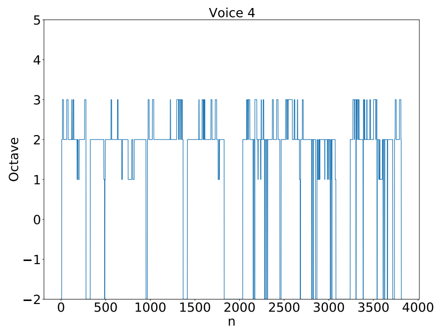
(a) MIDI Note Number Chart.



(b) Octaves as a function of time (Voice 2) for the Ridge Regression model.



(c) Octaves as a function of time (Voice 3) for the Ridge Regression model.



(d) Octaves as a function of time (Voice 4) for the Ridge Regression model.

Figure 6: MIDI chart for the notes and the octave distribution for the Ridge Regression model