CS221 Final Project

Tyler Boyd-Meredith and Jordan Marcy

December 6, 2012

1 Abstract

The aim of this project was to build a classifier that could be trained to discriminate between musical pieces based on their composer. We hoped to generalize this to an arbitrary number of composers. In order to accomplish the first goal, we implemented logistic regression, optimized by stochastic gradient descent (SGD). In order to accomplish the second goal, we implemented: (1) a baseline algorithm; (2) a nearest neighbors algorithm; (3) softmax regression. We gathered our data using the humdrum toolkit and the existing library of music scores represented as **kern files.

2 Data Collection and Feature Extraction

In order to successfully implement our classification algorithm, we needed: (1) a large enough amount of data, and (2) for the data to be in a form that could be fed to the algorithm as an input. We found a data type called **kern, which allowed us to accomplish both of these goals (see Fig 1 for diagram of data processing pipeline).

**kern, prange, and rcheck

**kern files encode all the symbols of music scores into a text file that can be read easily by a computer. The label comes first, telling us who wrote the piece (along with other info like when it was written). The music is represented as a set of matrices (one matrix corresponds to each staff). Each chord or single note is represented as a row in the matrix. Bundled into each row is information about which measure the chord/note is in, whether it is played on the downbeat or not, etc).

**kern files are part of a toolkit called Humdrum. Humdrum is widely used, and there are many tools to turn these **kern files, into text files, which cherry pick data according to the interests of the programmer. We used two of these functions: prange and rcheck.

Prange returns a text file the number of occurrences of each note in the piece. We used this tool to extract the following features: (1) *highest pitch*, (2) *lowest pitch*, (3) *total note count*, (4) *most frequent pitch*, (5) *pitch class frequency* (i.e. how often does A appear, how often does A# appear, etc), (6) *octave class frequency* (i.e. $\frac{\#notes\ that\ appear\ in\ each\ octave}{\#total\ notes}$).

Rcheck returns a text file containing a matrix in which each note in the piece is represented in order as a row in the matrix. Each row in the matrix tells us the absolute beat within the piece that the note is played on, the duration of the note, and the midi number of the note. We used rcheck to extract the feature: *most frequent note duration*.

Binary Classification

To start with, we created a binary classifier that took in the **kern data of two composers, and used a training set to determine the composers of the testing set. We compared the use of two algorithms, Nearest Neighbors and Stochastic Gradient Descent using logistic regression. These two algorithms use different strategies in order to correctly classify unknown data sets. Both algorithms used the feature vectors from the **kern data, as described in the Data Collection and Feature Extraction section. We also included a baseline algorithm that determined the composer of each testing example based on the composer the showed up most frequently in the training examples. For each binary classifier, this baseline algorithm return .5, because we balanced the number of each composer's music pieces in the testing set.

The Nearest Neighbors algorithm took in each feature vector of the validation set and compared the distance from that vector to every feature vector in the training set. The distance for each feature was defined as the absolute value of the difference between the feature values. The prediction was based on the composer of the training data sample that was closest in distance. The Nearest Neighbor algorithm performed with a mean validation error rate of 0.2254 over the ten pairings of the five composers. The variance of the error rates was 0.0173 with the worst rating coming from the Beethoven/Chopin binary classifier (0.5) and the best coming from Beethoven/Scarlatti classifier (0.0667).

In contrast, the Stochastic Gradient Descent algorithm created a weight vector that was used to obtain the dot products of the weight and feature vector in order to determine which composer the musical piece most closely corresponded with. To create this weight vector, we used logistic regression as a loss function and minimized the total loss in respect to the weights and the training examples. The Stochastic Gradient Descent algorithm performed with a mean validation error rate of 0.2106 over the ten pairings of the five composers. The variance of the error rates was 0.0143 with the worst rating coming from the Mozart/Chopin and Mozart.Scarlatti binary classifiers (0.45) and the best coming from Beethoven/Scarlatti classifier (0.0). See figure 2 for more validation error values.

The benefits gained from using the Stochastic Gradient Descent algorithm were due to its ability to generalize the data. Nearest Neighbors, although it was able to work well, is not very different from rote memorization. If the two composers in the binary classifier had written pieces where there was over lap in style, Stochastic Gradient Descent could generalize each composer by minimizing loss, while Nearest Neighbors would depend only on the closet

data point to each validation example. In terms of time complexity, Nearest Neighbors runs in O(#testing * #training) time while SGD runs in O(#Rounds * #Features * #Training). Nearest Neighbors also takes up less memory, because it does not have to keep track of the weight vector, although most of the memory for both algorithms is used to store the training and testing feature vectors.

Multi-class Classification

To discriminate between an arbitrary number of composers, we used Nearest Neighbors, as we did for binary classification and Stochastic Gradient Descent, but this time with a softmax loss function.

Nearest Neighbors generalizes nicely to the multiclass case, and the implementation did not change for multiple classes. When tested on 3-classes, the algorithm had a mean validation error rate of 0.2933 and variance of 0.0100. The best performance came from discriminating Chopin, Mozart, and Joplin (0.1300) and the worst came from discriminating Scarlatti, Joplin, and Beethoven (0.47). For 4-class classification, the algorithm had a mean validation error rate of 0.3300 and a variance of 0.0051. The best performance came when discriminating Chopin, Scarlatti, Joplin, and Mozart (0.2250) and the worst came when discriminating Mozart, Beethoven, Chopin, and Scarlatti (0.4250). For 5-class classification, the algorithm had a validation error rate of 0.4000.

In order to generalize Stochastic Gradient Descent to multi-class classification, the algorithm requires different methods for making predictions, calculating loss, and labeling examples. We used softmax regression to determine each of these changes. Instead of labeling composers 1 or -1, softmax requires them to be labeled 1 through k, where k is the number of composers to discriminate between. Rather than having a single set of weights, we keep track of k weight vectors. Predictions are made by determining which of the k weight vectors is most active when dotted with a testing example. Finally, loss for a specific training example x, whose class is j, is calculated according the following function (in which the notation $1\{a\}$ returns the truth value of a, so $1\{True\} = 1$, $1\{False\} = 0$):

$$log \frac{\exp^{w_j^T} x}{\sum_{i=1}^{k} \exp^{w_i^T} x}$$

Overall, our softmax algorithm performed passively, but not impressively. In particular, it did not perform as well as logistic regression when tested on 2-classes, although it did beat or tie with logistic regression on 5 comparisons (with a mean validation of 0.2954 and variance = 0.0233; results not presented), even though it is intended to be a generalization of logistic regression. In discriminating between 3-classes, mean validation error was 0.5000 (variance = 0.0012). The best performance in this category was on Chopin/Scarlatti/Beethoven, Chopin/Scarlatti/Joplin, Chopin/Mozart/Joplin, which had validation error of 0.4667. The worst performance was on Scarlatti/Joplin/Mozart (0.5667). In discriminating between 4-classes mean validation error was 0.5800 (variance = 0.0012). With the best performance on Beethoven/Chopin/Scarlatti/Joplin (0.5500). Finally, in discriminating between 4 classes,

softmax regression had a validation error of 0.7200. See figure 3 for more validation error values.

The biggest potential problem with our implementation of the softmax algorithm is the lack of independence between labels. Namely, if we have probability estimates for any of the k-1 labels, there is a unique value for the final probability estimate. Rather than computing k-1 probabilities, and computing the kth implicitly, we compute all k separately, which is likely to skew our results. In our research, we read that this problem could be mitigated effectively using the regularization term λ. However, in practice, we have not found this to be the case.

3 Discussion

As one can see, our binary and multi class classifiers had a wide-range of results, from perfect: SGD with logistic regression of Beethoven and Scarlatti, to no difference from the baseline: Nearest Neighbors of Chopin and Beethoven. Ultimately, the variability of our classifier must have come from two factors, our data and our features. The algorithms will do the best they are designed to based on our feature vectors, but require a lot of data, as well as features that generalize the composers well.

Once we made the decision to use **kern files, we were limited in the amount of data we could obtain by what was available on the website kern.humdrum.org. There were only around 60 **kern files for each composer, not allowing for a large amount of training data. Algorithms like Nearest Neighbors and SGD work best with larger data sets. This problem was compounded when attempting to create the multi class classifier. Although we were able to do better than the baseline, our multi class classifier often had an error rate above 0.5. Could this have been improved given an order of magnitude more data? Perhaps, but the amount of data was not the only factor in determining our results.

Another limiting factor in how well our algorithms performed was our features that we extracted from the data. We were limited by the type and amount of data that we could easily parse from the **kern files. The format of the **kern files would have made it incredibly difficult to build programs from scratch that could parse through them. Luckily, we had access to previously constructed programs like prange and rcheck. However, we were only able to look at the frequency of notes their durations. Music style tends to have more complexity than just these features, and extending the number of features could have decreased our error rate. A feature that we would have liked to have included, given enough time, would have been looking at the most common sequences of note pitches.

A final consideration is that our validation-test was performed on between about 20-50 pieces (with the same number of files for each composer). Though we used a leave m-out cross-validation scheme, we did not do multiple rounds of cross-validation. Thus, for each test, we do not know if our algorithm was performing unusually well, given the data, unusually poorly, or somewhere in between. Our results would be bolstered if we performed iterative leave m-out cross-validation.

Overall, we are pleased with our results, and in particular, we are surprised at how well

Nearest Neighbors did on this task, though slightly disappointed in the success of our softmax regression algorithm.

We'd like to conclude by acknowledging Craig Stuart Stapp, who created the **kern data format, without which we would have had a much more difficult time on this project.

4 Appendix

Figure 1 – Data Processing Pipeline

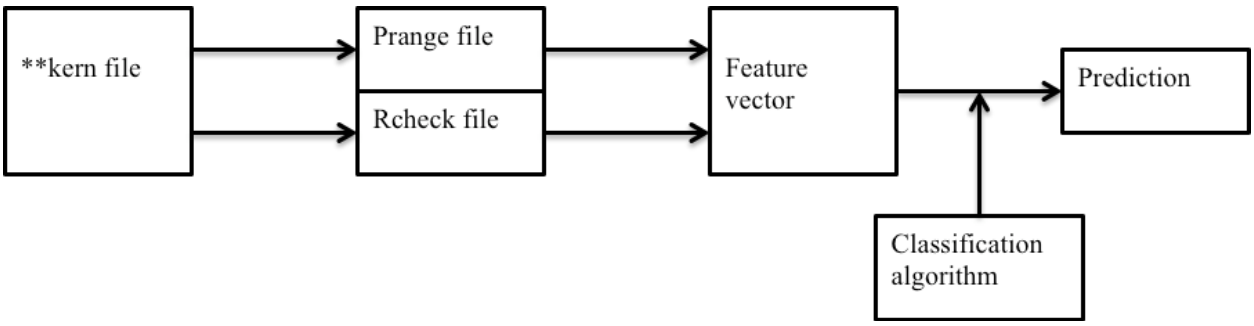

Figure 2 – Validation Error in Binary Classification

Testing Error Using SGD with Logistic Regression(Lr) and Nearest Neighbors(Nn)						
		Beethoven	Chopin	Joplin	Mozart	Scarlatti
Beethoven	Lr	-	0.25	0.26	0.30	0.00
	Nn	-	0.50	0.21	0.17	0.07
Chopin	Lr	-	-	0.14	0.45	0.05
	Nn	-	-	0.14	0.15	0.25
Joplin	Lr	-	-	-	0.37	0.21
	Nn	-	-	-	0.23	0.15
Mozart	Lr	-	-	-	-	0.45
	Nn	-	-	-	-	0.40
Scarlatti	Lr	-	-	-	-	-
	Nn	-	-	-	-	-

Figure 3 – Validation Error in Multiclass Classification

Testing Error Using SGD with softmax regression and Nearest Neighbors			
Composer Classes	Baseline	Nearest Neighbors	Softmax Regression
Chopin/Mozart/Joplin	0.6667	0.1333	0.4667
Chopin/Scarlatti/Mozart	0.6667	0.20	0.50
Chopin/Scarlatti/Joplin	0.6667	0.1667	0.4667
Chopin/Scarlatti/Beethoven	0.6667	0.30	0.4667
Chopin/Joplin/Beethoven	0.6667	0.3667	0.4667
Chopin/Mozart/Beethoven	0.6667	0.3667	0.50
Scarlatti/Mozart/Beethoven	0.6667	0.30	0.5333
Scarlatti/Joplin/Beethoven	0.6667	0.4667	0.5333
Scarlatti/Joplin/Mozart	0.6667	0.3333	0.5667
Joplin/Mozart/Beethoven	0.6667	0.30	0.50
Joplin/Mozart/Beethoven/Chopin	0.75	0.35	0.60
Mozart/Beethoven/Chopin/Scarlatti	0.75	0.425	0.575
Beethoven/Chopin/Scarlatti/Joplin	0.75	0.325	0.55
Chopin/Scarlatti/Joplin/Mozart	0.75	0.225	0.575
Scarlatti/Joplin/Mozart/Beethoven	0.75	0.325	0.60
All Five Composers	0.8	0.40	0.72