Zebra Finch Song Classification

Kyle Liberti
Boston University
Boston, MA 02215
kliberti@bu.edu

Matthew Kennedy
Boston University
Boston, MA 02215
mkennedm@bu.edu

Abstract

In order to understand how memories are stored, many scientists study biological brains. Due to the complexity and ethics of experimenting on humans, many scientists have turned to smaller animals, such as the zebra finch. The brain anatomy of the zebra finch is simpler than that of a human but it is not much different. One perk of studying zebra finches is that they can only sing one song, a song stored in memory which activates a specific neural pattern when sung. By observing the neural firing patterns of these birds, we can learn a lot about how memories are stored and maintained over time; in turn, we can better understand human memory. The current approach of collecting data is to record images of the brains of zebra finches whenever the birds make a sound; however, we are only interested in the images taken when the birds are singing. Even though we have a general idea of what the desired neural patterns look like, neural imaging is labor intensive and computationally expensive. Therefore, we used audio data taken along side the images to detect whether a bird was singing and not. Using cosine similarity scores and Pearson correlation coefficients, we were able to distinguish between songs and noise, but not between songs sung by different birds. Fortunately, we were able to solve this problem using k-means algorithm. It is still too soon to make any real claims, we still need more data from more birds and more tests to justify that we can identify zebra finches by their songs.

1 Background

We are working with the Laboratory of Neural Circuit Formation at Boston University to help with the mechanics of detecting zebra finch songs. When zebra finches sing, there is a specific neural firing pattern that takes place in their brains. By observing these patterns over time, the lab gains greater insight on how memories are retained in biological brains. Moreover, the visual data of the firing neurons is very desirable but computationally expensive to extract. Therefore, we need to find a way to isolate these neural events on camera by detecting the songs that are associated with them.Given long audio streams, we need to be able to seperate zebra finch songs from noise. We also need to able to differentiate between songs from different birds. Since the image data is paired with the audio data for every bird, it is important that the song we detect belongs to the bird of which is being recorded. Otherwise, our detection algorithms will suggest extracting undesired visual data.

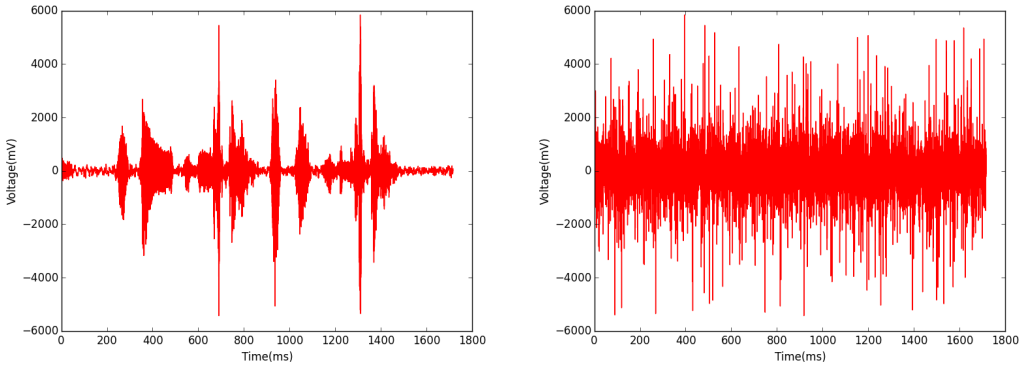
2 Data

We were provided with 146 zebra finch songs, sung by two different birds. Each song consists of a list of voltage values where the index of the list represents the time value. Each bird can only sing one song and that one song can vary in voltage signals and duration.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2.1 Noise generation

In order to start classifying the songs, we needed to create a benchmark to test against our song data. We could have generated a list of random values in a specified range but that kind of generated noise would not help us as our benchmark because it would be too different from our voltage values. The voltage values do have somewhat of a pattern and range specific to every song. Therefore, the best decision for now is to create noise from or data. So we using numpy to randomly pick songs from or data and create random permutations of the voltage values. In this way, we have created noise with a more realistic range to the numbers and forces to come up with better methods to seperate our songs and noise.



(a) Zebra finch song

(b) Random permutation of zebra finch song

As one can infer from the graphs above, the song has a voltage pattern which is destroyed by our noise permutaion; however, now the voltage values have the same range as that of the song.

2.2 Initial results

For our initial tests, we decided to average the voltage values across our songs of one of the birds and compare that average to individual songs of that bird and random noise permutations using their cosine similarity. We then binned the cosine similarity scores using Scott's rule in the histogram below.

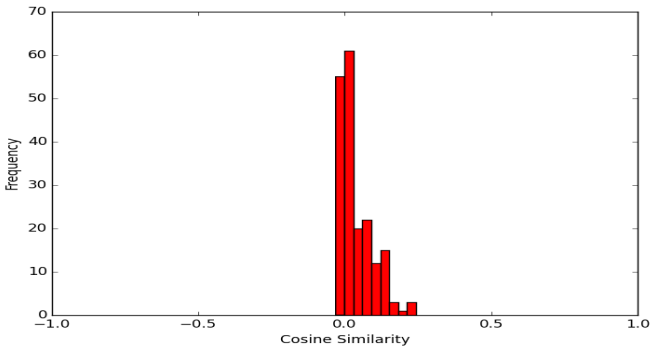


Figure 2: Cosine similarity of averaged songs and individual songs/generated noise

Our first results were unclear. We were hoping for a clear distinction between our generated noise and our song data but as one can see below, there was not much difference. Both the songs and the noise clumped around a cosine similarity score of 0.2. When we look at the graph for the generated noise and compare it to the averaged song data, the problem becomes more clear.

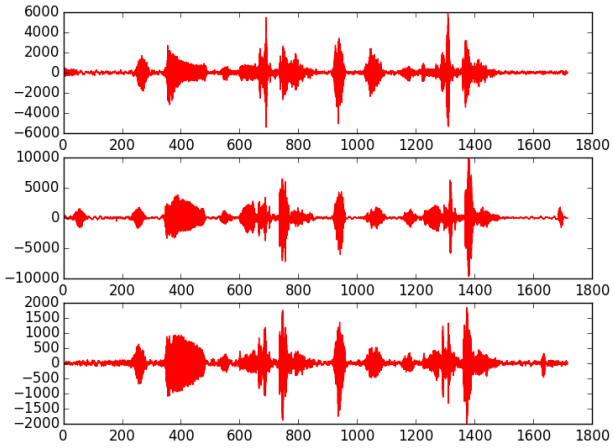


Figure 3: Comparision of different songs from same bird

When we averaged the song data over their voltage values we forgot 1) that there are millisecond differences in the recordings and 2) same amplitude peaks vary in voltage over the songs. Looking at the graph above, one can see how the songs differ from one another and how those songs look averaged together. Even though our zebra finch can only sing one song, he sings it differently every time. So when you average the raw data over the voltage values like we did, the result looks more like that of the generated noise than the actual song data.

2.3 Processing the data

To remedy our timing and voltage issues, we needed to clean our data. So we performed a Fourier Transform to all our songs to get the power and frequency over time and then we took the running mean over each song to 'smooth' the data.

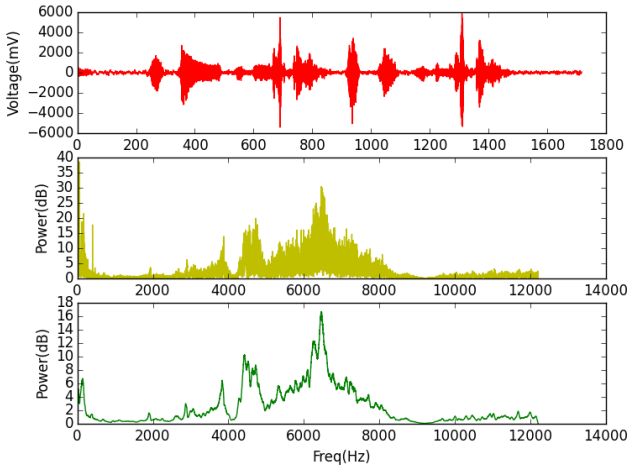


Figure 4: Transformation of data

After transforming our data, we were hoping to see not only the cosine similarity scores increase for our song data but also a clear distinction between our song data and our generated noise.

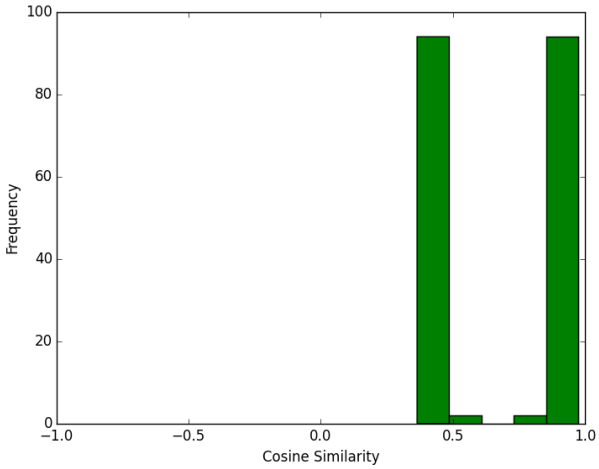


Figure 5: Cosine similarity of averaged songs and individual songs/generated noise

From the histogram, we can see that there are two distinct groups where our training data fell into, our generated noise and our song data. We also see that our cosine similarity scores increased across the board, even for the noise intervals which we generated. These results were a lot more promising but we still need better results!

3 Results

Since there was a small distinction between the cosine similarity scores of the noise and song data, we decided to compare the processed data using their Pearson Correlation Coefficients as well. By doing this, we hoped to separate these data heaps further away from one another.

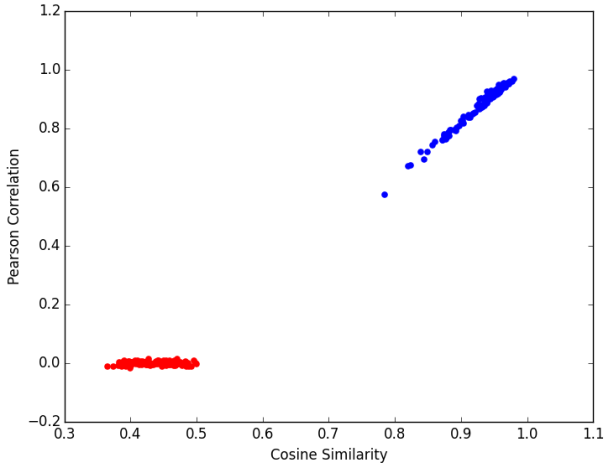


Figure 6: Noise and song clusters (one bird)

Here we show that we can group our generated noise and songs into distinct clusters, so we have a baseline to test against more of our data. Now we can move on to the more challenging problem of differentiating between songs created by different zebra finches. When we process the other birds songs and plot their scores with the other data points we get the graph below.

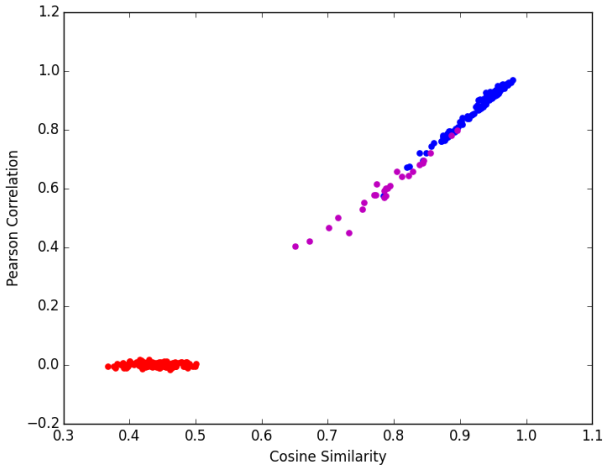


Figure 7: Noise and song clusters (two birds)

As one can see, we cannot clearly distinguish between the two birds (blue and magenta) using cosine similarity scores and Pearson correllation coefficients. We need to make sure that these clusters are distinct, matching each cluster of data points to a particular bird.

3.1 Clustering Method

We used k-means clustering because the format of the processed data lends itself to the strengths of the algorithm. The data that composed our means were frequency, power, and song length. Unlike other machine learning tasks that require humans to decide how much weight to give each facet of the data, the numbers used in our project came straight from the (processed) samples. The dataset also solved the usual challenge of deciding on k, the number of clusters. Since we know the recordings are coming from exactly two types of birds, the question of what our k should be has already been answered.

Each of the two clusters starts off with a random subset of the data. On a recent trial, the first cluster received an assortment of data from 80 recordings, with 66 going to the latter cluster. The last part of the initialization is to compute the centroid of each cluster.

For each recording, we had information about its length in milliseconds, its frequency at each point in time, and it power at those same points. The dimensions of the centroids were length, frequency, and power. In order to find the average frequency of a cluster, we consolidate the Python list of frequencies each member has down to the average frequency for that cluster member. The same operation was done for the lists of powers. Once all the members of a cluster have been read, the centroid is returned as a Python list containing average length, average frequency, and average power.

The rest of the work is done in a loop. Criteria for loop termination is contained in the function needReordering. This function checks how close each list of recording data ([length, average frequency, average power]) is to each of the centroids. If needReordering finds one member that needs to be in another cluster, it immediately returns True which means the subroutine will start. Another function, reorder, puts every list of recording information into the cluster it is closest to. Proximity is determined by the cosine similarity between a centroid and an individual recording's data. Eventually none of the cluster members need to be moved, signaling the end of the algorithm.

3.2 Evaluation

K-means clustering worked even better than we expected. When we run clusterize on the full dataset, the loop executes exactly once. This means that after the clusters are initialized randomly, the algorithm perfectly divides the recordings by comparing them to each centroid just one time for every recording. We've tested the algorithm dozens of times and the result is always the same despite being initialized randomly.

The algorithm remained successful even when it had little data to start with. We divided the 146 recordings into a training set and a testing set. The training set was further divided into one cluster for each bird. One by one, each recording from the testing set was compared to the clusters and added to the one it was closest to. Each cluster's centroid was recomputed after gaining a new member. We were still able to consistently divide the recordings perfectly.

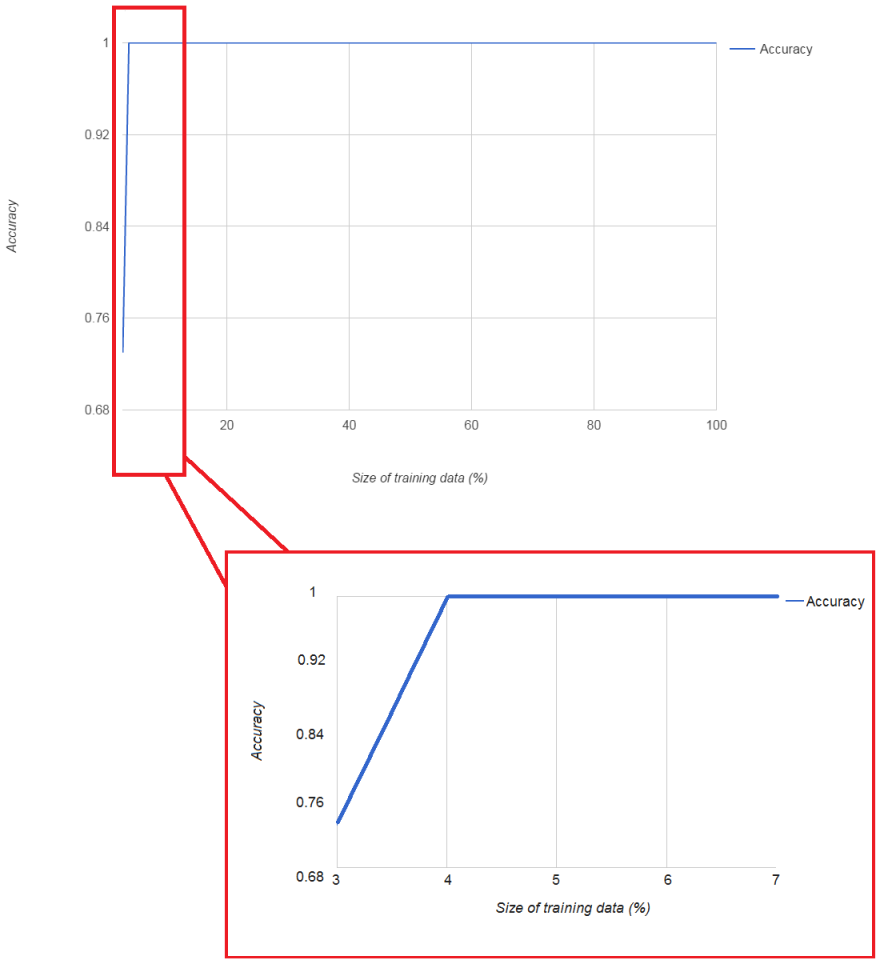


Figure 8: Size of training data vs accuracy

We can't safely reduce the percentage of the starting clusters below 4 percent. At 3 percent, the probability of one of the initial clusters having 0 elements is 25%[1]. If a cluster has no elements, then computing its centroid is impossible. Out of the 75% of the times, both clusters have at least one element, our accuracy falls from 100% to 70-92%.

4 Conclusion

It is still too soon to make any concrete claims that we can distinguish between the songs of zebra finches reliably. So far we have only tested our methods on songs from two different birds. Although we can differentiate between those two birds using k-means, there are also many factors specific to our data that are aiding our classification. One of these factors is that these songs vary in duration. It would be interesting to see our algorithms try to classify two songs that were of the same length. We

[1]We take the same percentage from each bird's recordings. 3% of 120 is 3.6 which isn't a problem for us since we can take 3 samples from that set. However, 3% of 26 = .78. Our training set won't have any recordings from the second bird.

still need to test on more song data from other birds. Once we can prove that we can classify between two birds reliably, we can start adding more birds into the data to see how many distinct clusters k-means can group effectively.

Acknowledgments

We would like to thank the Laboratory of Neural Circuit Formation, more specifically Tim Gardner and Will Liberti, who guided us throughout our project. Not only did they provide all the data but they gave us pointers on how to seperate the song data from the noise as well as how to process our data in order to get better results.