Classification of Neuronal Responses to Auditory Stimuli

Kevin T. Farr
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01002
ktfarr@umass.edu

1 Introduction

In this project, I tested several methods of classification on neural data in response to sensory stimuli, in this case, auditory stimuli, attempting to find a model or method that provided reasonable accuracy in the general case. Discerning patterns in neural data, particularly if they can be used to predict responses to stimuli, provides insight into the underlying structure of what causes specific neurons to fire, and what different firing patterns could mean. This project was interesting because of the complexity of the unknown systems and variables that produced the data I worked with. While the similarity metric I implemented proved problematic when applied to classifiers, it was a kind of measure that I had not previously considered could be applied to real world data. Based on the experiments performed, Support Vector Machines approached 70% classification accuracy on a withheld test set, and with further tuning, I believe could be a viable model for the data. By contrast, Support Vector Machines within an AdaBoost ensemble performed poorly, achieving a maximum accuracy of 41%.

2 Related Work

In their paper, Caras et al[1] describe using a pattern classifier to correlate neuronal activity with the intensity of auditory stimuli. In their experiment, they expose songbirds to varying decibel levels of the same sound, going from 10dB to 90sB, and repeating each exposure 10 times. However, rather than learning a subset of this labeled and data, and then attempting to classify the remaining portion, their pattern classifier instead selects a single spike train at random (with replacement) from each trial, and then uses that as a template to classify the remaining 81 spike trains, and repeats this process 1000 times. Then, the correct classifications are averaged over the 1000 estimates, and this is used to indicate whether or not the cell discriminates between different sound intensities. Each spike train is preprocessed with a gaussian filter to smooth the data, and then the similarity is calculated using the Rcorr function. The template that provides the highest Rcorr score is the label a given spike train is assigned to for that iteration.

In their review of different spike similarity metrics, Victor[4] provides an overview of the most commonly practiced and most promising methods for measuring similarity between spike trains. The first, and seemingly most widely used, involves dividing the spike train into uniform time bins, and counting the number of spikes within a given bin. The trains then have a smoothing kernel applied to them, and their scalar products provide the distance metric. Next, Victor discusses cost-based metrics, which attempt to find the minimum number of steps required to transform one spike train into another, thus a lower score indicates that the trains are similar, while a higher score indicates that they are sufficiently different.

Vargas-Irwin[3] et al expands upon the notion of a cost-based metric for spike train similarity, putting forth a process called Spike Train SIMilarity Space (SSIMS). Their pairs the cost-based metric with dimensionality reduction using t-Distributed Stochastic Neighbor Embedding (t-SNE) in order to project the high-dimensional spike trains into a two or three dimensional representation. Then, using this representation, they attempt to learn the structure indicated by correlations between these firing patterns.

The t-SNE technique, proposed by van der Maaten et al[2], is a modified form of Stochastic Neighbor Embedding, which transforms high dimensional Euclidean distances into conditional probabilities pairwise between all the points, based on a gaussian function with centers at each point. However, SNE is described as being very difficult to optimize, and can fall into poor local minima. By contrast t-SNE, uses a different version of the cost function that uses simpler gradients, and employs a different distribution in low dimensions to compute the similarity between points, which helps to solve the optimization issues of SNE.

3 Methodology

For this project, I used one major pipeline for the initial data processing and classification, and a secondary one that was focused on testing my similarity function against different collections of data cases.

3.1 Support Vector Machine and AdaBoost Pipeline

The first step of the pipeline was to load and process the data from the CSV files containing the data labels, and the text files containing the spike times, and turn each single unit into a matrix with 80 rows, and a variable number of columns. To this end, I wrote several helper functions in the DataProcessing file, which take paths generated by glob, labels stripped from the path names provided (though this is not used in the final pipeline), and an argument called bin length. The most common way of analyzing spike trains that I found was to divide the stimulus interval into uniform bins of a length in milliseconds, and the value in the array corresponding to that bin is the total number of spikes that took place during that time. Without prior knowledge of which binning would produce the best results, I used a series of increases bin sizes (1,2,4,8,16,40,80,120,. Generating these arrays is performed by the array_from_raw_data function, which then returns the unit arrays to stack_arrays, which conglomerate the individual units, shuffle the rows, and save the result to the Data directory as Numpy arrays.

Once the data have been processed, they are fed into a loop that iterates over the predefined bin sizes, and uses SciKit Learn's Train-Test-Split to produce the indices that correspond to a random 80-20 split of the data. Next, the 80% are again divided into training and testing portions using the same method and making another 80-20 split. Next, using Numpy's logspace, I generated the search space for the Support Vector Machine's hyperparameters. For the C parameter, which controls how heavily the SVM penalizes incorrect classification, I kept the log base at 10, and generated 25 samples in the range of 10^{-3} and 10^{3}. For Gamma, the term of the SVM using the radial basis function kernel that controls the influence an individual data point has on the model (a smaller Gamma indicates a larger influence of a data point), I used the number of columns in the data matrix as the base, and generated samples in the range of 10^{-5} to 10^{2}. These distributions are used by SciKit Learn's RandomizedSearchCV, which cross-validates the supplied model on a random sampling from the distributions for a user specified number of iterations.

Once the optimal parameters were collected, I used them to initialize an untrained SVM, and used that as the base classifier for the AdaBoost Classifier, which trains additional classifiers of the same type as the base in order to correct for classification errors. For time concerns, I tested the AdaBoost classifier first on 5, 15, 25, and finally 50 estimators.

3.2 Similarity Function Pipeline

The second pipeline involves testing custom similarity function, found in the SpikeSimilarity class. The function, based on the algorithm provided in Victor[4] and detailed further in Vargas-Irwin[3], takes two spike trains using 1 millisecond binning as input, and returns a score computed by calculating the minimum number of atomic moves it takes to transform spike A into spike B. For this process, atomic moves are adding a new spike, deleting a spike, and moving a spike from one position to another. Adding and deleting confer a weight of 1 to the score, while shifting is proportional to the distance moved. For the move weight, I settled on 1/100, meaning a spike could be shifted up to 100 milliseconds left or right before incurring the same penalty as an addition or deletion. Additionally, a spike cannot cross the path of another spike, so this provides another restriction I had to account for.

Initially, I tried to call the function directly in models that allowed it, but found that it was too slow to accurately gauge how well it was working. Instead, I chose to limit the size of the data transformed by the similarity function to around 15% of the total data. First, using 5 random slices of the full data set (retrieved with the train-test-split function) I precomputed the X training distance matrix by passing the X matrix and the function into SciPy's pairwise distance function, pdist. Then, to compute the X test matrix, I used SciPy's cdist function, supplying the X train and X test and the similarity function. Once computed, I trained and tested support vector machines using the precomputed kernel option. Next, the pipeline retrieves twelve single units, and shuffles them into training and testing sets, again computing the similarity matrices. Finally, five single units are processed without splitting, and the results are fed into the t-distributed Stochastic Neighbor Embedding module, which reduces the dimensionality of the data to two or three, while attempting to preserve the distances in higher dimensions. The primary use is visualization of high dimensional data, and I used the results to plot 2d and 3d scatter plots, colored with the true class labels.

4 Data Set

The data set was obtained from a series of electrophysiology recordings performed on 54 zebra finches, 27 males and 27 females. The subjects were anesthetized, and a single electrode was inserted in the caudomedial nidopallium (NCM) in the left hemisphere of the brain. A specific site was selected when it displayed a response to auditory playback stimuli. Once the site was selected, four different sound files were played, with twenty pseudo random repetitions each. The sound files were three examples of male bird song and one of white noise, each two seconds in duration. During playback, the neural activity was recorded by Spike 2 software.

After the experiment, single cells were isolated from the multi-cell recording by Spike 2, using principle components analysis. Each recording produced an average of two to three distinct cells, for a total of 119 across all subjects. Each unit had an interspike interval of more than one millisecond, indicating that the single units were indeed individual cells. Finally, the single units were extracted as text files, with the time signatures of each spike event in the memory buffer, and a keys file, which is a CSV containing the time signatures of the stimuli, and the label of the stimulus.

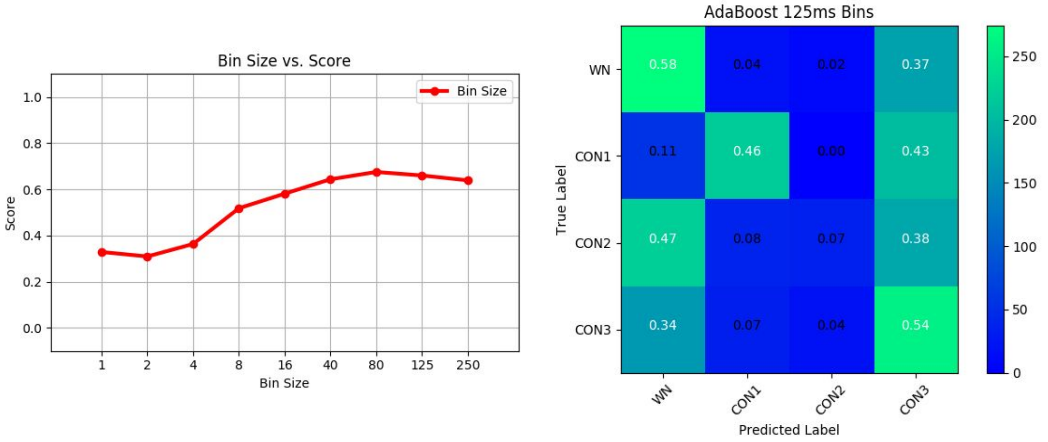
For each of the 119 units, there are 80 two second intervals of labeled data that can be extracted, providing 9520 total data cases, and given the interspike intervals, a maximum precision of one millisecond, or 2000 columns. At this maximum precision, each column would contain a 1 if there was a spike event at that time, and a 0 otherwise. The labels correspond to the different song stimulus played, with White Noise having a label of 0, the song CON1 being 1, CON2 being 2, and CON3 being 3.

5 Experiments and Results

To have a baseline for classification comparison, I decided to train a series of Support Vector Machines with a radial basis function kernel on the data set, each using data that had been counted

into bins different sizes. For the binning, I used 1, 2, 4, 8, 16, 40, 80, 125, and 250 millisecond, and the trained the models in a loop. Each iteration, the data were split into 80% training data, and 20% test data, stratifying it to keep equal proportions of classes, to test the models for general accuracy. I used RandomizedSearchCV to cross-validate the parameter selection using the second Train-Test split, finally returning the model that performed best, retraining it on the training data from the initial Train-Test split. After the model finished cross-validating, I tested it on the withheld data set, recording the score for each bin size. As expected, given the high dimensionality of the data, models trained with data that had been binned into 1, 2, and 4 milliseconds performed poorly in the general case. Accuracy improved dramatically beginning at the 8 millisecond bin size, reaching a peak at 80 milliseconds, as can be seen in Fig. 1.

In an attempt to improve the successful classification rate of the support vector machines, I chose the ensemble AdaBoost classifier, using a support vector machine as the base classifier. As in the original experiment, I began with an 80-20 split of the data into a training and testing set. Given their low accuracy and long run-time, I eliminated the 1-4 millisecond binnings, and the 250 millisecond binning, instead using only 8-125. Next, I further split the training set 80-20, and used the same randomized search to select hyperparameters for the base support vector machine. Once the cross-validation completed, I fit the AdaBoost classifier on the training set, and tested it, producing a series of confusion matrices. Surprisingly, the AdaBoost classifier performed poorly, achieving a maximum accuracy of 41.3 during the 125ms bin trial. Judging by the confusion



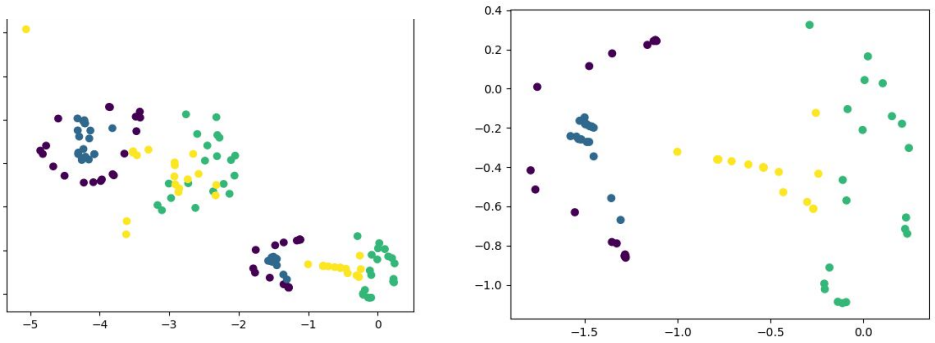
matrices, the classifier struggled with the data representing CON3, and overfit to compensate.

Next, I tested my implementation of the spike similarity function described by Vargas-Irwin et. al., but found that my version wasn't optimized well enough to calculate the similarity of a large portion of the dataset, so I used the train-test split to pull out 15% of the data at random for training, and 5% for testing, and did this five times, selecting a different random seed for each iteration. Then, I precomputed the similarity function on the training set using SciPy's pdist function, producing a pairwise similarity matrix that I could pass in to the models I chose to train with it, along with a pairwise similarity matrix between the training set and the test set using cdist. Using the precomputed option, I trained five support vector machines on each random slice of the data, and each produced an accuracy score of 0.25, predicting the same label each time. Then, examining the probability class labels, I found that they had values of 24.86-25.34% for all data cases. Performance did not improve, regardless of hyperparameter selection. Next, I used the KNeighbors Classifier and KRadius, which produced similar results.

Given that neurons can have dramatically different firing patterns between cells in the same animal, I decided to test whether I could improve the performance of my similarity metric by choosing entire cells rather than selecting individual trials. To this end, I randomly selected 12

single units, repeating this process five times, and computed the similarity matrices for each. Training and testing support vector machines on these matrices produced the same results as the random slice, with hyperparameters having no visible effect. Additionally, I used both the standard scaler and normalizer to try and improve performance, but the accuracy remained at 0.25, and the models continued to output the same class label for all test cases.

Finally, I tested my similarity function on single units, because a neuron's firing pattern given a specific stimulus should be most similar to itself, and the distance function should reflect this. Rather than splitting the data for training and classification, I extracted five single units, computed their similarity matrices, as well as their 16-80 millisecond binnings. In order to better visualize the data, as described in van der Maaten et. al., I used SciKit Learn's t-distributed Stochastic Neighbor Embedding (t-SNE) to compare my similarity function with the other metrics available (Euclidean, Manhattan, and Cosine). The following figures show the mappings of cost-based similarities of EMEK048B and EMEK022A into two-dimensional space, colored with their true labels.



6 Discussion and Conclusions

As indicated by the experimental results, my implementation of the SSIMS algorithm requires significant optimization, as well as general improvements. First, it requires the data to be in the 1ms bin form, meaning it is operating over 2000 columns, and second, it will only function when the data are labeled 0 or 1. This meant that it was impossible to use SparsePCA or similar dimensionality reduction methods before computing the distance matrix. Given more time, I would redesign it.

The results from the single support vector machines exceeded my expectations, and I would be like to continue optimizing them. If I continue to use AdaBoost, I will use a weaker base classifier. However, it seems unclear whether or not such a classifier is producing accurate predictions, and whether those predictions would actually be helpful. The related research seems to indicate a desire for more specificity in their labeling, as well as being able to complete the classification task on comparable or smaller data sets.

References

[1] Caras, M. L., Sen, K., Rubel, E. W., & Brenowitz, E. A. (2015). Seasonal Plasticity of Precise Spike Timing in the Avian Auditory System. *The Journal of Neuroscience*, *35*(8), 3431–3445. http://doi.org/10.1523/JNEUROSCI.3407-14.2015

[2] van der Maaten, Laurens, Hinton, Geoffrey. (2008) *Visualizing Data using t-SNE*. *Journal of Machine Learning Research* **9**(7):2578-2605.

[3] Vargas-Irwin, Carlos E. et al. (2015) *Spike Train SIMilarity Space (SSIMS): A Frame-Work for Single Neuron and Ensemble Data Analysis*. *Neural Computation, 27(1), 1-31*. http://doi.org/10.1162/NECO_a_00684

[4] Victor, J. D. (2005). Spike train metrics. *Current Opinion in Neurobiology*, *15*(5), 585–592. http://doi.org/10.1016/j.conb.2005.08.002