Lab Note 8: SVD/PCA

Introduction

In the last lab, we have successfully implemented a controller that allows our car to drive straight. Our end goal for S1XT33N is to allow it to follow 4 different voice commands and execute the corresponding action. In order for S1XT33N to be able to identify the voice commands we give it, we need to implement a classifier that is able to distinguish between 4 sufficiently different commands. However, algorithms like voice recognition and classifiers that can identify the spoken words are beyond the scope of this course, and aren't easily implementable on microcontrollers like the Arduino that have limited resources available. We will instead be implementing more of a "speech pattern" classifier, where we use the overall shape of the waveform of the word recordings, rather than the actual words themselves, for classification. In this lab, we will be implementing such a classifier using singular value decomposition (SVD) and principal component analysis (PCA).

Recorded audio signal always has some amount of noise, so it is highly unlikely that you can generate two identical-looking waveforms, even if you're recording the same word. Therefore, we need to somehow measure the similarities between the waveforms. A waveform with 100 time steps can be represented as a 100 dimensional vector, with its entries being the amplitude of the waveform at that time step. Ideally, we would be able to make multiple recordings of each word and compute each word's "mean recording" or "mean waveform". Then, during live classification, when a voice command is given, we can measure its Euclidean distance to the different means, and classify it as the word whose mean is closest to it.

However, if we want to do this computation on the Arduino, we will run out of memory and storage. The Arduino, like other microcontrollers, has a limited amount of resources that we can work with. As a result, storing each word's mean vector and operating on such large vectors is infeasible. To get around this, we need a way to compress our vectors down to a reasonable size; PCA, which we implement using SVD, provides this. Of course, this, as with any compression, comes with information loss. If we are careful about how we use it, however, we can minimize that loss.

This lab note will not cover the math behind SVD or PCA. If you would like a review, please take a look at the class notes (SVD and PCA).

Data Collection and Preprocessing

Before we can implement our classifier, we first need to provide data for it to work with. There are two steps for this: recording your words, and preprocessing the recordings to clean them up before using them for our classifier.

Recording Your Words

First, we need to collect the data. Because we want to make our classifier more robust, we need to collect multiple recordings for each different word. That way, we can take into account the variations of the different recordings with the classifier. For this lab, we need at least 40 recordings of each word.

You will want to keep our classification methods in mind when choosing the words to record. Because we're taking the shape of the recorded waveform, and not the actual word that's being said, you will want to choose words with different numbers of syllables, and enunciate each syllable. For example, a word like "pear" would result in a single peak in the recorded waveform, while a word like "watermelon" would result in four, making it easier for our classifier to distinguish between the two commands. Words with a hard versus soft ending can also work well; think about the differences between a word like "cat" versus a word like "shoe".

When recording each word, try to say them the same way each recording (stress the same syllables, maintain the same speed, etc). Doing so will allow you to align your words more easily in the next step, and result in cleaner classification results. When you're giving the voice commands live in the future labs, you will have to say the words in the same way as your recordings, so it may help to record yourself with a phone saying the words so you remember how to say them later.

Data Preprocessing

Now that we have our data, we need to clean it up before we can use it for our classifier. Since your word probably didn't take up the entire recording's duration, there will be empty samples of noise or silence. We want to remove this as much as possible, since it's not feasible to expect that you start saying the word at the exact same sample number in each recording. To that end, there will be three parameters that you can tune for our data preprocessing:

- Length - The total number of samples that your word takes to say in each recording.
- Threshold - The fraction of the maximum value that our preprocessor uses to decide that the word has been started in the recording. For example, if the threshold is 0.5 and the maximum value in the recording is 500, the start of the word corresponds to when the recording first reaches a value of 250.
- Pre-length - The number of samples that will be included as part of the word from before the threshold was reached.

The data preprocessing can be summarized as follows: The get_snippets function searches through the recording for the sample number where the value of that sample is equal to the *threshold* times the maximum value in the recording. It then goes back by *pre-length* number of samples, considers that to be the start of the word, and counts the next *length* number of samples to be your entire word. Note that in order to run SVD/PCA, we need all of our preprocessed data samples to have the same length, so these three parameters are used for all of your words.

Implementing the Classifier

Now that we have our data, it is time to finally implement the classifier. We will split our data into a "training" and "test" set, where the former will be used to create the classifier, and the latter will be used to test it. There are two steps in creating our classifier: SVD/PCA, and identifying the centroids of the data clusters.

SVD/PCA

As mentioned before, we will be using SVD to implement PCA so that we can compress our data to an operable size for the Arduino while minimizing information loss. Recall that SVD takes in an input data matrix and returns 3 matrices:

- U, which contains the column vectors that are the PCA vectors for the columns of your input data matrix.
- S, which contains the singular values in order from greatest to least.
- V^T, which contains the row vectors that are the PCA vectors for the rows of your input data matrix.

Our data matrix is constructed such that its rows are our different recordings of the different words, and the columns are different samples/time steps. Because of this, the PCA basis we will use for compression will be formed by the vectors in V, not U. Due to memory limitations on the Arduino, we can only choose 2 or 3 of these vectors to use for our PCA basis. Use the singular values to evaluate how much variation in the original data our PCA basis, which we will choose to have 3 vectors, can represent.

Identifying the Cluster Centroids

Upon finishing PCA and identifying the new PCA basis, we will now project our training data into this basis and plot it. Upon doing so, you will see that the data forms different clusters in the PCA basis, one cluster for each word. To complete our classifier, we will now find the centroid of each cluster. There are many ways to do so, with the simplest one (and the one we will use) just being finding the average position of the data points belonging to that cluster. The centroids are what we will use for classifying our words; the centroid with the smallest Euclidean distance to our test data point represents the word that we will classify the data point as. With these centroids computed, we have now completed our classifier!

Arduino Implementation

When performing live classification on the Arduino, the conditions will not be as ideal as the conditions for classification in the Jupyter notebook. There will be background noise even when not giving voice commands, and sometimes the spoken words are very different from the recordings that were used to train the classifier (i.e. speaking too late). Therefore, we will use additional parameters, loudness threshold and Euclidean threshold, to reduce the misclassifications from the Arduino.

- Loudness threshold - The maximum value of the sample must be greater than this threshold to be considered a word. If the recorded data is too soft, we do not want to classify it as it is probably noise.
- Euclidean threshold - The distance of the sample to the nearest centroid must be less than this threshold to be considered a valid word. If the L2 norm (distance) is larger than the threshold, your classification algorithm should simply ignore it and wait for the next sample.

Wrap-up

To illustrate what we have done in this lab, let's say that our 4 words, after preprocessing the data, are all length-80 vectors. If we were to store the average vector of each word and compare each preprocessed live recording (another length-80 vector) to it, we would be computing distances from length-80 vectors 4 times, not to mention having to store all 4 vectors on the Arduino. By using SVD/PCA, we cut the number of length-100 vectors we need to store down to 2 or 3 PCA vectors. Not only that, but by projecting the live recording into our PCA basis, we have cut that vector length from 80 down to 2 or 3. When classifying the word, we now also only need to compute distances from length-3 vectors, as the centroids themselves are also in the new basis, cutting the required computation! As you can see, we have compressed our data and classifier significantly with PCA. In a limited-resource environment like the Arduino, this can mean the difference between whether an algorithm is implementable or not.

Design Considerations

Earlier we noted how S1XT33N's command classification will act as more of a "speech pattern" classifier than a proper voice classification system. The implementation details in this lab note describe how and why this is the case. And this has implications on the accuracy of detection for the words that we choose during our classification. If two words have very similar waveforms, then they are prone to get mixed up by our classifier. Hence, we want to choose 6 easily distinguishable words (or sounds) to ensure better classification accuracy.

When humans distinguish words, they listen for temporal and frequency differences to determine what is being said. However, S1XT33N will have to choose much simpler features for classification: syllables, intonation, and magnitude. Keep this in mind while choosing your words!

Note written by Steven Lu (2021)
Updated by Steven Lu, Megan Zeng (2022)
Updated by Junha Kim, Ryan Ma, Venkata Alapati (2023)