Machine Learning applied to Speech Recognition

Adam WILD

May 1, 2018

Abstract

With more powerful GPUs comes more powerful computation allowing Machine Learning and Deep Learning algorithms to perform well at analyzing and making sense of a large amount of data. Our goal here is to apply these algorithms to Speech Processing and more precisely to Speech Recognition. ECE 6255 is the occasion to test and further our proficiency at Signal Processing and Machine Learning by applying it to Speech Recognition.

Introduction

This project will associate principles and algorithms coming from courses such as Digital Speech Processing and Machine Learning to solve the problem of speech recognition. In this paper, we will introduce the dataset used by our scripts, the framework that we developed, the algorithms used to classify our speech extracts and finally the speech processing algorithms used to extract meaningful features from our speech data. The results and the effect of the parameters used will be discussed when we will talk about speech processing.

1 Dataset

The dataset is made of basic speech commands, we got the dataset from [**?**]. In this dataset, we have the 10 digits, which we will use primarily in this paper. Every speech extract of the dataset is exactly 1 second long and is sampled with a frequency of 16 kHz, which means that we have 16000 samples for each speech extract. Therefore, we will not have to resize them as part of a pre-processing step. We do not need to use Time Warping here, even if this technique could be useful in other cases, it will not be used in this paper. The distribution is the following:

word	zero	one	two	three	four	five	six	seven	eight	nine
# of elements	2,376	2,370	2,373	2,356	2,372	2,357	2,369	2,377	2,352	2,364

The people recording this dataset had to do so in a room, with no specification of equipment but with little to no surrounding noise. It means that the data is clean. After listening and visualizing some of the samples, it is clear that most of the data is zeros and the actual word being pronounced actually takes 6000 out of the 16000 samples for a speech extract. We will see later on how this affects or not the algorithm being used.

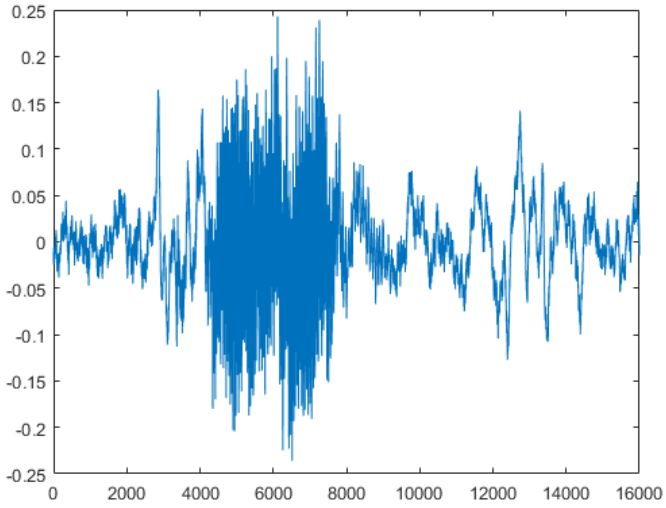

Figure 1: Example of a 'three'

2 Framework

We will refer to the readme file associated with the code for instruction on how to run the same experiments that we did. However, we will discuss the overall structure here. The structure is built on two scripts and three files. The three files are files, vectors and output_classifiers. files contains all of the speech extracts in their raw form, that is wave files. By using data_to_vect.m, we extract features from the files specified using an algorithm specified to build different datasets that are to be stored in the vectors file. We then use run_classifiers.py to test different machine learning algorithms to analyze how they perform at distinguishing different words.

The framework has been developed so that lots of tests can be performed to analyze the effects of the numerous hyperparameters involved at each step. Few lines need to be changed to each time and the code should be able to run on any machine as long as the structure of the file and folders is respected.

3 Machine Learning

The problem can be seen as a multiclass classification problem. The data is stored as (X,y) pairs, the goals being to predict y given a X value. For every classifier that we apply, we use 70% of the dataset as training samples and 30% as testing sample, every example shown below will be the performance of classifiers on the testing set. We randomly shuffle the samples used for training or testing when we train the classifiers, so you may (will) get different results from what we got. However, the results should not vary enough for you to come to different conclusions. We also used 10-fold cross validation and keep the best performing algorithm each time.

We tested several classifiers, some had overall bad performance for any given dataset (like linear regression) or took too long to learn to be efficient and thus we discarded them (like support vector machines). Since we have to run a lot of experiments, such algorithms were discarded. We tuned the hyperparameters on small datasets to determine which one we should keep for further experiments. We kept the parameters that helped generalize the best and discarded weak learners and learners that overfitted. Here are the classifiers that we kept.

3.1 Perceptron or Artificial Neural Network (ANN)

The perceptron or artificial neural network is the algorithm that we discussed in class. The idea of this algorithm is to use a weighted sum of the inputs and determine the weights of the synapses through the backpropagation algorithm. Thus, creating a non-linear function giving a vector as the output. The highest value in this vector gets the value of 1, every other gets the value of 0, we can then map this vector to the corresponding word and get a result. [**?**] [**?**].

3.2 Random Forest Classifier (RFC)

The RFC is a simple, fast to run and efficient algorithm to run for classification problems. Using the training set, several Decision Trees will be created to help classify new data. The algorithm is based on information gain, each split of a given tree will maximize the information gain, thus separating the remaining dataset efficiently. The random forest will then let the different trees give a classification for an element of the testing set, the value that is the most common among the predictions of the decision trees is the output of the RFC.

4 Linear Prediction Coefficients (LPC)

4.1 The method

The goal is to reduce the 16000 samples to get something that is computably reachable. For this we will the use LPC. The idea relies on reproducing the speech signal thanks to few coefficients. The estimated signal is computed using the following formula:

$$\hat{s}[n] = -\sum_{k=1}^{p} a_k.s[n-k]$$

These p coefficients are computed while minimizing the error in least squares sense.

$$E = \sum_{n=-\infty}^{\infty} [s[n] + \sum_{k=1}^{p} a_k.s[n-k]]^2$$

Following this formula, given that we have a lot of 0 values in our signal, they will not impact the a_k coefficients since they will be zeroed out in the error calculation. The coefficients will be computed to fit the non-zero values, where the information actually is and thus being features representative of the speech extract.

We tried different values and computed their error. If the error is too large, the signal that is approximated is not similar to the original one and therefore it would be impossible for a classifier to perform properly. We tested with several values of p and got error ranging from 10^{-4} for $p = 2$ to 10^{-5} for $p = 20$. The accuracy is good for these small speech extracts and thus the order used for the LPC had little impact, however, we still tuned that hyperparameter to improve the overall performance.

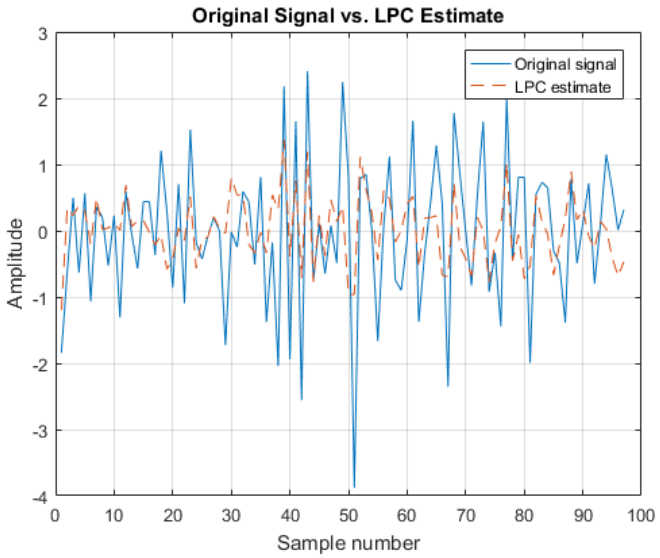

Figure 2: Approximation with p=3

For instance, for $p = 3$, the curve already follows the pattern of the real curve, therefore, prediction and classification is possible even at low values of p.

4.2 Influence of p for all digit values

After tuning the hyperparameters of the machine learning tools on small dataset, we fixed them for the rest of the tests. For this part, the variable is the value of p and all digits are to be predicted. Here is the accuracy that we got:

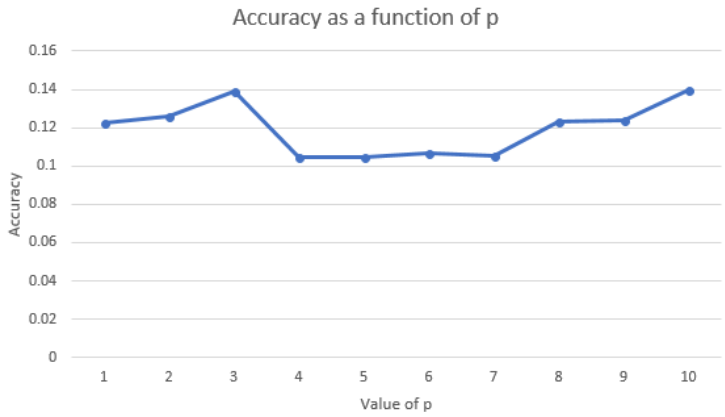

Figure 3: Accuracy as a function of the order of the Linear Predictor

As said before, this is the accuracy given by the best performing classifier for each point. There are two information given by this graph. First the value of p has little impact on the accuracy. Since the error is already extremely low for small values of p, increasing p has little effect. We tested for larger values (such as $p = 20$) but found no improvement. The second deduction that we can make is that having to predict 10 digits at once is too hard for this method. The accuracy is just above 10% which is the performance that a random classifier would have.

To improve the performance, we decided to diminish the number of classes to be predicted.

4.3 Prediction values for 5 digits

The are different ways to tell words apart. The vowel triangle and the spectrogram are two of these methods. We can put different vowels on different part of a triangle determined by the different formant values of a vowel. The bigger the differences in these values, the easier it is to tell these values apart. Since the LPC method is not efficient with all 10 digits, we decided to use 5 of them that are significantly different to improve performance, we used: 3, 4, 7, 9 and 0. They have significantly different vowels and recognizable spectrograms. Therefore, a classifier would have less of a hassle to tell them apart.

Word	Sounds	ARPABET
Zero	/z I r o/	Z–IH–R–OW
One	/w ʌ n/	W–AH–N
Two	/t u/	T–UW
Three	/θ r i/	TH–R–IY
Four	/f o r/	F–OW–R
Five	/f a^y v/	F–AY–V
Six	/s I k s/	S–IH–K–S
Seven	/s ɛ v ə n/	S–EH–V–AX–N
Eight	/e^y t/	EY–T
Nine	/n a^y n/	N–AY–N
Oh	/o/	OW

Figure 4: Sound lexicon of digits

The output of the classifier can be seen under output_classifiers/015.csv. The accuracy reaches 44% which is still under the performance that we would like to achieve but still 24% more efficient than a random classifier.

Other features were evaluated for this method such as the number of speech examples in each dataset. For these examples, we had 1,000 instances for each word. Increasing the number of instances does not increase the performance the classifiers, we tested with 2,000 instances and found no improvement. However, for small dataset such as these, it is difficult to draw a definitive conclusion. As seen in class, methods using Deep Neural Networks only managed to be efficient when provided with 100 of hours worth of speech. Going from 1,000 to 2,000 instances may not improve performance but we cannot predict what would be the result of this method for 1,000,000 instances.

5 Mel-Frequency Cepstrum Coefficients (MFCC)

5.1 The method

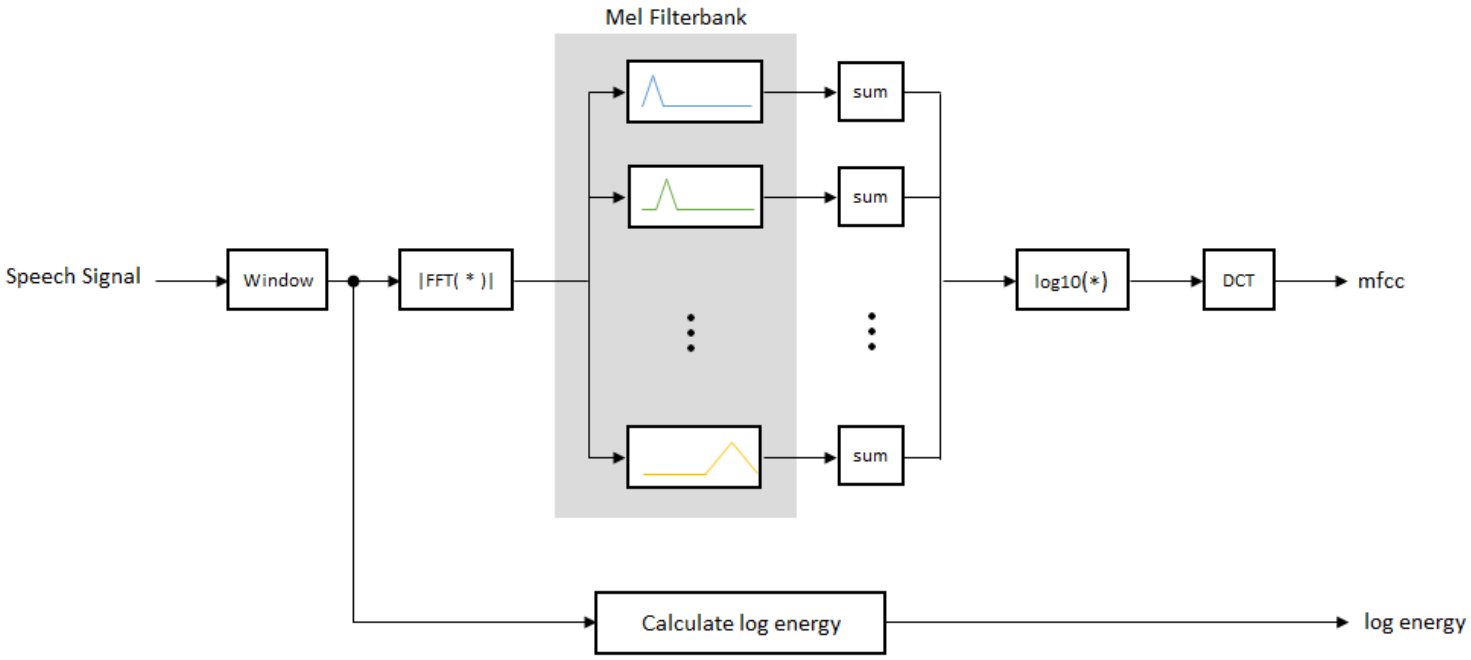

Figure 5: MFCC algorithm (source Matlab)

The method described in [2] also uses Dynamic Time Warping on each to vector to diminish the variations between words pronounced in different settings (different speakers, accent). The method was implemented using MATLAB.

To have an algorithm that performs better, we need to implement a technique that gives us more information from the wave file. The MFCC are coefficients that are derived from a cepstral representation of a speech file. The MFCC is a non-parametric method for modelling the human auditory perception system and if it is mostly used as an identification technique, its feature extraction can also be used to do speech recognition.

The maximum accuracy is reached at 64% for the testing set, see 'output_classifiers/017.csv'.

Conclusion

Even for a small dataset with little noise, doing speech recognition is no easy task. We reached the best prediction accuracy at 64% which is good but far from what is needed for an efficient algorithm. To improve the performance of our algorithms, we should use new techniques in relation to the ones exposed here, Deep Neural Networks and larger datasets. Other methods such as Hidden Markov Model or Gaussian Mixture Models to extract even more meaningful features for a classifier.

References

[1] Y. LeCun and M. Ranzato, "Deep learning tutorial," in *Tutorials in International Conference on Machine Learning (ICML13)*. Citeseer, 2013, pp. 1–29.

[2] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010.

[3] J. Lyons, "python-speech-features," *python-speech-features.readthedocs.io/en/latest*, 2013.

[4] J.-H. Lee, H.-Y. Jung, T.-W. Lee, and S.-Y. Lee, "Speech feature extraction using independent component analysis," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 3. IEEE, 2000, pp. 1631–1634.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[7] R. P. Lippmann, "Review of neural networks for speech recognition," *Neural computation*, vol. 1, no. 1, pp. 1–38, 1989.

[8] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support vector machines for speaker and language recognition," *Computer Speech & Language*, vol. 20, no. 2-3, pp. 210–229, 2006.

[9] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[10] P. Warden, "Speech commands: A public dataset for single-word speech recognition." 2017.

[11] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 5206–5210.