Spoken Character Recognition

Yuki Inoue (yinoue93), Allan Jiang (jiangts), and Jason Liu (liujas00)

***Abstract*—We investigated the problem of spoken character recognition on the alphabets, numbers, and special characters. We used the Mel-Frequency Cepstral Coefficients (MFCC) as the feature points to characterize the spoken characters, and then reduced the dimensionality of the feature vector by applying the Principal Component Analysis. Four different machine learning algorithms were trained using the dimension-reduced feature vectors, and we compared their performance. For the alphabet set, we realized that many of the letters with similar sound structures were confused for each other, so we instead took a two-layer approach: first determine which "character set" an input is in, and then classify the sample within the set. We used the K-Means algorithm to determine the character set. For the best performance, we achieved 58.02% accuracy for the alphabets, 92.05% accuracy for the numbers, and 82.85% accuracy for the special letters.**

I. Introduction

Online security is more important than ever. With the explosion of the amount of private information online, it has become a commonplace for websites to require single and sometimes even double security features to protect the users' private information. This includes requiring the user to type in a one time code, which is usually a randomly-generated string of characters containing the alphabets, numbers, and special characters. Typing in these characters can be an annoyance to many, especially in mobile settings, where data entry is far more constrained than in a laptop or desktop setting due to lack of dual screen capabilities for most smartphones.

Also, consider the situation in which one has to type in unfamiliar words such as foreign names. Since the character order is seemingly random to the user, he/she has the same problem of having to switch constantly between the screens.

Both of these situations can be remedied if we have a software that can interpret the voice input of the user and type out the result. In this project, we investigate the spoken character recognition for the alphabets, the numbers, and the special characters, to aid those people who have to type in sequences of characters. To solidify the problem, we will define the input to our algorithm as a one-second WAV file, in which a single character (a, 5, *, etc.) is spoken, and using a supervised learning algorithm (one of Logistic Regression, Naive Bayes, Support Vector Machine, or Neural Network) to output the prediction on which character was spoken.

II. Related Work

Almost all prior work in using machine learning to tackle spoken character recognition leveraged a cepstrum-based pre-processing pipeline to generate a set of features to be fed into a back-propogation neural network (BPNN). The main variation in prior works were in how they performed their pre-processing steps and how they created their neural network architecture.

A. Hand-chosen features

Existing classifiers that achieved the highest performance are neural networks that were fed hand-picked features. Roger Cole and Mark Fanty's English Alphabet Recognition (EAR) system achieves about 90% accuracy by using hand-picked signal processing features that were specific to improving performance on the English alphabet [3], [4]. They did not perform classification on other characters (such as numbers). On a high level, Cole and Fanty's method was a four step process: first they filtered and digitized their audio samples, second they used a neural network to track when a pitch began and ended, third they measured features over time and split them into different linguistic segments, and fourth they used another neural network to classify the segments they identified. Although they report the highest accuracy, they were only able to do so due to their explicit phonetic segmentation and the use of speech knowledge to select the best features to capture the linguistic information. As such the features chosen in their work are good for English, but are not generalizable to other languages.

In an earlier work by Burr, specially chosen filters and delays were used to extract features from letter data [2]. In particular, the processing pipeline for letters was hand-tuned to be different from the data processing pipeline used for spoken digit recognition in this paper. The pre-processing in this paper used fewer linguistic features specific to English letters and reported about 85% accuracy on letters and about 99% accuracy on digits.

B. Neural Network Architecture

Other papers used different network architectures with different transfer functions. For example, in a paper by Adam and Salam in 2012, the audio pre-processing step used was MFCC (which is highly standard), and the neural network architecture used was 400 input layer nodes, 150 hidden layer nodes, and 26 output layer nodes for the alphabet [1]. They used the hyperbolic tangent transfer function in the hidden layer and a linear transfer function in the output layer. They report 65% accuracy with this architecture. Finally, in a paper by Reynolds and Tarassenko in 1992, a Radial Bases Function neural network classifier was used on general characters in multiple languages. They did not report accuracy on the English alphabet specifically, but their overall accuracy was about 70%.

III. Dataset and Features

The sample collection was done by ourselves. We recorded three samples from each person, each corresponding to the three character groups (i.e. the alphabets, numbers, and special characters). For each of the recordings, we asked the speaker to pronounce one character every time we tapped their shoulders. We made sure that the shoulder taps are separated by at least 1 second, so that each samples are spaced out enough and thus do not interfere with each other. After we collected the samples, we subdivided them into one second sound bites of the sound samples using a MATLAB script. For example, an alphabet recording was subdivided into 26 one second sound bites, each capturing an alphabet letter. The script first takes the square of the signal (i.e. calculates the power of the signal), then applies a low pass filter by taking a moving average. The resulting signal waveform looks as the bottom right. Finally, the script subdivides the signal by looking at the peaks of the filtered signal. Because the peak of the filtered power signal occurs at the same part for all of 1 syllable characters and for most of the 2 syllables characters, the script is able to line up the samples well.

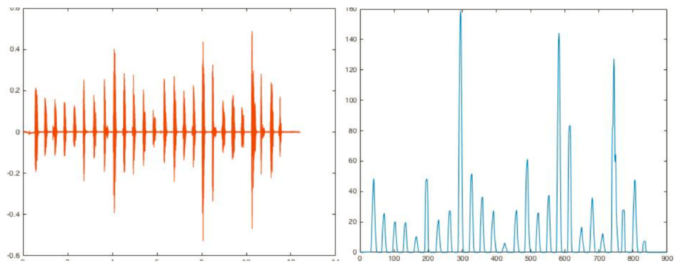

Fig. 1: Raw sound file (left), sound wave squared (right)

Overall, we collected 1,600, 670, and 602 sound samples for the alphabets, numbers, and special character sets, respectively. With one-second sound samples, we still felt that the data was not in a digestible format. We applied the Mel-Frequency Cepstral Analysis to each sample and performed Principle Component Analysis on the resulting feature vectors to find the most significant feature points. Both MFCC and PCA are explained in detail in the next section.

After the data collection, we tested our learning algorithm as follows:

1) Randomly choose 90
2) Test on the remaining 10
3) Repeat steps 1 and 2 one hundred times

Where step 3 is added to reduce the variation from the choice of the training set.

IV. Methodology

A. Voice Synthesis Basics

Voice synthesis can be modeled with two steps. The first step is the carrier tone synthesis using vocal cords. Since the carrier tone is just a "raw" tone created by simply vibrating ones vocal cord, it cannot be deciphered as a recognizable word. The carrier tone depends on the physical composition of the speaker such as the length of the vocal cord, making this step unique to each individual and also invariant on the word being spoken. The second step involves shaping this carrier tone using the contour of the mouth. This can roughly be modeled as putting a filter to the carrier tone, and it determines which word is being spoken. Since the objective of this project is voice recognition and not speaker recognition, the feature points should be chosen as such to characterize the filter applied to the carrier tone, not the carrier tone itself. However, this is easier said than done, for we can only collect post-filtered soundbites. One of the ways to extract the filtering characteristic is to use the Mel-Frequency Cepstrum Coefficient Transform.

B. Mel-Frequency Cepstral Coefficient Transform (MFCC)

The MFCC allows for the extraction of the filtering characteristic of the audio files. Though we will avoid the in-depth explanation of how MFCC can achieve such a result as it is not the main focus of the project, MFCC essentially takes the Fourier Cosine Transform of the log magnitude of the Fourier Transformed audio sample, allowing it to analyze the frequency response of the filter applied onto the carrier tone. Since the main focus of the project is to apply machine learning algorithms onto the extracted feature points, and cared less about how we extract those feature points, we did not implement the MFCC. Instead, we used `mfcc.m` in the Auditory Toolbox by Malcolm Slaney [6] to apply the MFCC transform to the sampled audio. An example MFCC output is as follows.

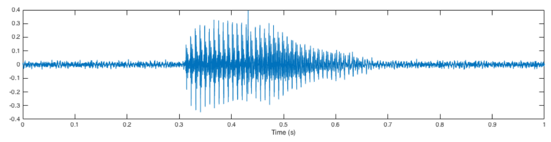

Fig. 2: Raw audio snippet of spoken letter

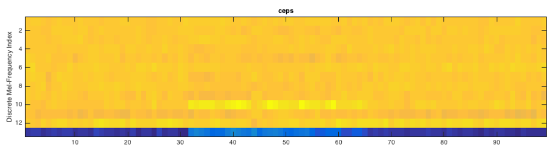

Fig. 3: Example MFCC feature output heatmap

As can be seen from the figure above, the MFCC outputs 13 feature points per sampled time period, and as we ultimately chose to subdivide the 1 second sound bite into 100 subsections, the MFCC roughly outputs 1300 feature points per sound bite.

We tried to add more features other than the direct MFCC output to enhance the performance, such as adding the 1st/2nd derivatives of the MFCC and duration of the signal input, but nothing seemed to improve the result significantly. Therefore, we decided to just use the raw outputs of MFCC as the feature vector.

C. PCA

With the parameter settings of the MFCC feature extraction that produced the best performance, the size of the feature vector became roughly 1300. Though it is hard to exactly say how large the dimension of a feature vector should be to a given problem, 1300 dimensional space was just too large, as most of the ML algorithms run very slow for such a large feature vector. We therefore decided to use the Principal Component Analysis (PCA) to reduce the feature dimension. PCA reduces the dimension of a dataset to dimension N by taking the N right singular vectors that correspond to the largest singular values as the basis. With this change in basis, the dimension of the feature points is lower, but the resulting covariant matrix should still approximately be the same. After PCA, the 1700 dimensional feature vector was turned into a 30 dimensional feature vector.

D. K-Means Algorithm

The K-Means Algorithm is an algorithm used in unsupervised learning problems. The main idea of the algorithm involves minimizing the following objective function over the set of clusters S_i:

$$\arg\min_{s} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{1}$$

The algorithm estimates the solution to the objective function by calculating the cluster centroids, which are the estimates of the μ. The algorithm has two steps. First, each data in the sample set is put into a cluster that is closest to a cluster centroid. This is the assignment step. Then, the location of each cluster centroid is recalculated by taking the mean of the sample points that it contains. This is the update step. In our project, the K-Means Algorithm was used to group letters that have a similar MFCC feature points. We expected that since the MFCC extracts features that are strongly correlated with the spoken characters, by we can automate the grouping of similar sounding characters by applying the K-Means Algorithm to the samples we collected.

E. Naive Bayes

The first algorithm we used to test our data was the Naive Bayes model. We started with this, because of its ease of implementation and general robustness. The assumptions behind Naive Bayes are the each sample is independent from one another. Although we had speakers repeat through the letters three times, we still posit that the independence holds true. The rationale behind this is the fact that we specified subjects to put pauses in between each character spoken. Thus, when a subject says “a” followed by “b”, the way “b” is spoken is independent of the pronunciation for “a”. We used the Gaussian Naive Bayes model, since it seemed like a safe assumption that data would be somewhat normally distributed.

F. Support Vector Machine

Next, we moved on to using a Support Vector Machine, also known for its reliability. Much of the magic behind the Support Vector Machine lies in its kernel method, which is a mathematical trick used to perform calculations on feature vectors of (potentially) infinite dimension. We experimented with three types of kernels: Gaussian, Poly, and Linear. During testing, the Gaussian kernel consistently performed much worse than both the Polynomial and Linear kernels.

TABLE I: Labelling errors of different SVM kernels on classification of special characters

Kernel	Linear	Poly	Gaussian
Error	0.139	0.183	0.822

The rationale behind this is that the equation for the Gaussian Kernel is:

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{2}$$

As an exponential equation, this effectively maps our feature vectors to infinite dimensions. This creates a model that is more complex than our dataset actually is, thus yielding high error rates. On the other hand, the linear kernel is merely represented as a dot product, with the polynomial kernel being a finite linear combination of these dot products. With these less complex models, we had much better results that consistently outperformed our Naive Bayes model. Perhaps, with more data, we would be able to use more feature points from PCA and possibly yield better results with the Gaussian kernel. In addition, when selecting a penalty, the L2 penalty yielded better results than the L1 penalty, suggesting that outliers should be treated with extreme caution.

G. Logistic Regression

Logistic regression performs surprisingly well as compared to other classification algorithms. During our trials, we saw that it performed almost as well as the SVM with a linear kernel.

For Support Vector Machines and Logistic Regression, we used the one-versus-rest scheme. The strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n_classes classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. (http://scikit-learn.org/stable/modules/multiclass.html#one-vs-the-rest)

H. Back Propagation Neural Network

The last classifier we used to fit our data was a back propagation neural network. The backpropagation neural network estimates the gradients of a cost function over a network architecture and updates the weights of the transfer functions from layer to layer accordingly. The strength of neural networks is that network architectures with many hidden nodes allow the classifier to create increasingly complex hypotheses.

We trained a network with a single hidden layer with 52 nodes for alphabets, 20 nodes for digits, and 24 nodes for special characters. We used the Maxout transfer function for the hidden layer and the Softmax transfer function in the output layer.

V. Experiments/Results/Discussion

A. Alphabets

The classification of alphabets was the hardest out of the three character sets considered. Obviously with alphabets having the most number of categories to label (26 compared to 12 for special characters or 10 for numbers), it is expected to be the hardest. However, that was not the only complexity related to the alphabet classification. Unlike the numbers or the special characters, there are some alphabets that sound very similar. This has been pointed out by different papers [2]–[4], and are typically called "sets." More specifically, the sets noted by other researchers are the E-set, containing the family of alphabets that have an e ending (i.e. B/C/D/E/G/P/T/V/Z), and the M-set, containing M and N.

We first ran the ML algorithms without taking a special note of the sets. Figure 4 shows the result for the Naive Bayes, and we see that certain letters vastly underperform others. For exapmle, letters such as "h" and "w" perform well, but other letters, especially the letters in E-set, are grossly mislabeled. At this point, we suspected that this is due to the E/M-sets. And surely enough, the confusion matrix in Figure 5 confirms that the letters in the E/M-sets are being mislabeled within the sets. But we did not stop there. Looking at the confusion matrix more closely, we saw more than the M/E-sets. For example, Q/U are mislabeled as each other, and so are A/J/K. This makes sense, as those letters sound similar to human ears. Therefore it is logical that when MFCC extracted the features for those letters, the resulting feature points are similar. This result motivated for some way to make an educated guess on the sets for the rest of the 15 letters.

Since we do not have the prior knowledge of how different letters are clustered with each other, set determination is an unsupervised problem. To tackle the problem, we ran the k-means square algorithm on the samples.

Each plot represents a cluster, and the bars represent how many letters are part of that cluster. As expected, letters that sound similar such as Q/U and A/J/K mentioned before, are in the same cluster. One thing to note is that L/O, which do not necessarily sound similar, are in the same cluster. K-means algorithm allowed us to discover sets that we would have otherwise not have been able to guess. After fine tuning, we ended up with 8 different sets.

Empowered by the knowledge of sets, we once again ran the test. The learning algorithm now has two layers – the first step is to determine which of the 8 sets the sample belongs in, and the second step actually determines which letter it is. The results are summarized in Table 2.

B. Numbers and Special Characters

Numbers and special characters are much easier to classify, as the characters sound very differently from each other. Therefore, we decided that there is no need to look for sets to improve on the accuracy. In general, there was no difference in dealing with numbers and special characters. The results are summarized in Table 2.

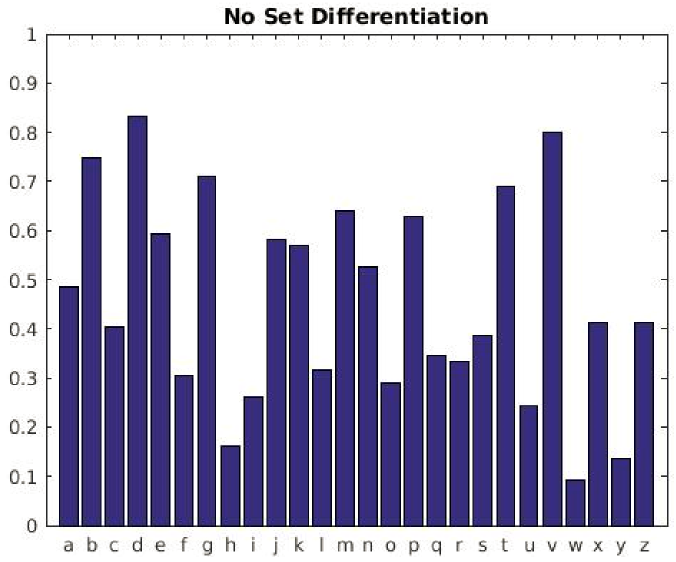

Fig. 4: Naive Bayes error without set differentiation

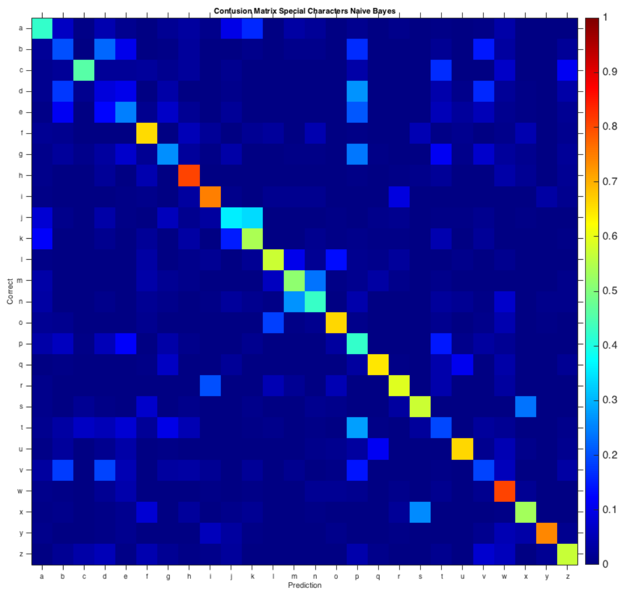

Fig. 5: Naive Bayes Confusion Matrix

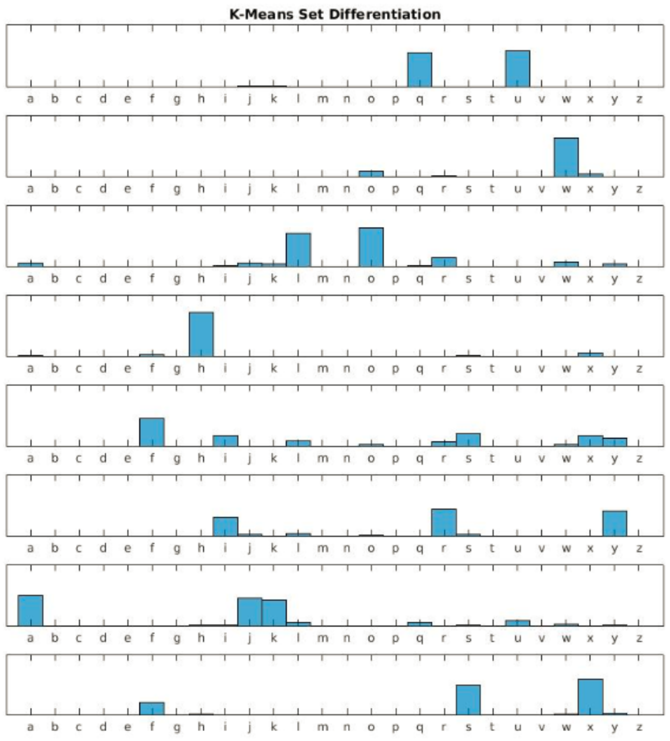

Fig. 6: K-Means clustering set differentiation

TABLE II: Results of different algorithms on spoken digit and character classification

Success Rate				
	NB	SVM	LogReg	NN
Alphabets	0.499	0.568	0.580	0.541
Numbers	0.828	0.893	0.920	0.838
Specials	0.776	0.861	0.829	0.657

The results are better as expected. One thing to note here is that unbeknownst to us when we were collecting, the “-” character was pronounced as “hyphen” and “dash”. Though this may not have worked if every single character labels were pronounced multiple ways, it is nice to know that the ML algorithms were robust to such a variation.

VI. Conclusion and Future Work

Unfortunately, just by looking at the numbers, our algorithm vastly underperformed when compared to works done by other researchers. However, there are few details that explain why this does not necessarily mean that our work is useless. First, our results indicate that we do not have enough sample size to fully utilize the ML algorithms. We observe an overfit for all of the algorithms except for the Naive Bayes, from the fact that all three produce excellent result (95%+ success rate) when we set the training and the testing samples to be the same. In short, the complexity of the ML algorithms are too high for our sample sizes. Therefore, with the inclusion of more data, our algorithm is predicted to do even better.

Also, many of the sound samples we collected were from those who do not speak English as their native language, unlike the “American English speakers” Cole and Fanty tested on. Therefore, our sound samples have larger noises added to them in the form of accents. This was intentional, as the main goal of the project is to create a software anyone can use.

Finally, the feature extraction method we used is independent of the target language, unlike that of the other researchers, who used pronunciation patterns very specific to English, such as the notion of sonorant and fricative sounds. Therefore, there is a higher chance for our algorithm to perform better for different languages.

We have a couple of topics in our minds for the future. First, since the main objective of the research is to create an algorithm to classify characters regardless of the type (i.e. alphabets, numbers, special characters), instead of creating ML algorithms for each of the three types, we want to create an ML algorithm that can take any kind of inputs. Second, as mentioned in the discussion section, our algorithm was able to ignore the fact that the “-” was pronounced differently. Since this kind of variation in how the special characters are pronounced is to be expected, it would be nice if we can extend our project to account for that. More specifically, use the idea of sets (developed when we were studying alphabets) to account for such a variation. For example, both “hash” and “pound” will be in the same set of “#”. Such a system would be a very useful and practical method of data entry for mobile systems.

References

[1] T. Adam, M. Salam, et al. Spoken english alphabet recognition with mel frequency cepstral coefficients and back propagation neural networks. *International Journal of Computer Applications (0975-8887) Volume*, 2012.

[2] D. J. Burr. Experiments on neural net recognition of spoken and written text. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(7):1162-1168,1988.

[3] R. Cole and M. Fanty. Spoken letter recognition. In *Proc. Third DARPA Speech and Natural Language Workshop*, pages 385-390, 1990.

[4] R. Cole, M. Fanty, Y. Muthusamy, and M. Gopalakrishnan. Speaker-independent recognition of spoken english letters. *In Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 45-51. IEEE, 1990.

[5] J. Reynolds and L. Tarassenko. Spoken letter recognition with neural networks. *International Journal of Neural Systems*, 3(03):219-235, 1992.

[6] M. Slaney. Auditory toolbox. *Interval Research Corporation, Tech. Rep*, 10:1998, 1998.