Mathematics Expression Calculator Based on Speech Recognition

Wengie Wang
jwang862@wisc.edu

Yuhan Xie
xie75@wisc.edu

Hengjiali Xu
hxu259@wisc.edu

Abstract

We use machine learning algorithms to create a mathematics expression audio calculator. By concentrating our research on speech recognition, we hope to create an audio calculator that can do some basic math calculations including addition, subtraction, multiplication and division among integers 0 to 9. In this project, we examine several classification algorithms to predict the spoken digits from the Free Spoken Digit Dataset (FSDD) [6] and the spoken operators from the dataset created by ourselves. Before training machine learning models, we extract and concatenate Mel Frequency Cepstral Coefficients (MFCCs) and Log Filterbank Energies from the raw audio data. The models we use include Logistic Regression, Random Forest, and Radial Basis Function (RBF) Kernel Support Vector Machine (SVM). We also apply Adaptive Boosting (AdaBoost) to Logistic Regression model in order to increase the prediction accuracy. Since all of our models have their own strengths and weaknesses, we decide to use plurality vote by assigning weights of 2 to Random Forest and RBF Kernel SVM and a weight of 1 to Logistic Regression with AdaBoost. Although most of our models achieve a test accuracy higher than 90%, there are problems of overfitting since the training accuracy of all the models are 100%.

1. Introduction

In recent years, speech recognition technology has become an increasingly popular concept. From individuals to organizations, speech recognition has been widely implemented, making people's life more convenient and efficient. Specifically, for individuals, speech recognition enables people to multitask by speaking directly to intelligent assistants such as Google Home, Amazon Alexa and Siri. According to Bakanay [2], a global technology company helping brands with conversational AI and Analytics solutions, speech recognition benefits businesses by providing intelligent customer services to improve self-service in a way that enhances customer experience while reducing organizational costs.

Although speech recognition has been extensively used by both individuals and businesses, it has been rarely applied to scientific fields. Inspired by the speech recognition system, we are curious about how to turn the computer into a math assistant that could do some basic mathematical calculations based on the computation problems people say. Instead of typing the algebra expression into a calculator, people could simply say the problems they want to solve to the computer and get the answers immediately.

The idea of the audio calculator can be beneficial for various group of people. To be detailed, this calculator could help educate younger children who may not be familiar with calculators. Moreover, it could turn math study into a more interesting process because interacting with computer verbally is more similar to interacting with humans. As a result, children might get more excited about studying math than they regularly do. On the side of teachers, they could possibly benefit from this technique in the way of finding math education easier and more effective. Most importantly, this audio calculator would benefit the physically disabled people who are unable to use keyboards or calculators and help them do math calculations more conveniently. While most smart assistants in the world help us solve daily life problems, we try to discover how computers could become intelligent assistants in the mathematical field.

In our project, we constrict the speech recognition to four most fundamental math calculations, including additions, subtractions, multiplications and divisions among integers 0 to 9. Specifically, we use machine learning algorithms to take math expressions in the form of user speeches, produce the equations in digital mathematical expressions and compute the results.

2. Related Work

For speech recognition problems, one major task is to determine what kind of features should be selected. While spectrogram, a visual representation of the frequency spectrum of a signal [13], has been widely applied to represent audio signals, recently a lot of works on speech recognition have been using Mel-frequency Cepstral Coefficients (MFCCs). For example, Suksri et.al [10] stated that MFCCs are the most common parameter used in speech recognition problems. Besides, another feature named Filter Banks has also been used in speech recognition problems. Fayek [4] compared Filter Banks and MFCCs and concluded that both of them are useful. According to him, on one hand, though the procedures of calculating filter banks and MFCCs are similar, MFCCs can be beneficial when the model is susceptible to high correlated inputs since they decorrelate the filter bank coefficients. On the other hand, filter banks are popular since the computations are motivated by the nature of the speech signal and the human perception of such signals.

In order to solve speech recognition problems, various machine learning algorithms have been implemented in previous research. Bazzi and Katabi [3] have used SVM as a classifier to recognize spoken digits in English. They extracted MFCCs and performed different SVM classifiers. The best performance they obtained was 94.9% accuracy using a 1-versus-9 SVM classifier. Kalamani et.al [7] have conducted a study of speech recognition using Extreme Learning Machine (ELM) and K-Nearest Neighbor (kNN) and concluded that ELM classifier has achieved a higher accuracy than kNN.

Gavaert et al. use a different truncation method in their article to ours. They calculate the entropy of samples and decide the endpoints according to a threshold of 0.5 [5]. This has an effect of truncate out the less significant half of the sample. However, this may not be useful in our dataset as some of the samples in it contains only or mostly useful information. Truncate out the less significant half will result in discarding useful information. Thus we implement another truncation method which decide endpoints according to the normalized amplitude.

3. Proposed Method

3.1. Data Preprocessing

Our raw data consists of samples that are noisy and start from arbitrary points. Therefore, Preprocessing is required to reduce noises and align features. To reduce noises and amplify informative data, we conduct pre-emphasis on all raw samples. We align the features of samples by truncating out uninformative data according to the normalized absolute amplitude. The setup of pre-emphasis and truncation will be discussed in Section 4.2.

3.2. Feature Extraction & Performance Metric

As discussed in related work, MFCCs are the most popular feature used in speech recognition problem. To be detailed, it has the potential advantage of reducing the complexity in implementing feature extraction [10]. The frequency mapping also allows the spectral frequency features of signal to imitate the human auditory perception. In our project, MFCCs are obtained using the `mfcc()` function in the `python_speech_features` package. To guarantee that we include as many important features as possible, we also select Filter Banks features. That is, we calculate Log Filter Bank Energies using the `logfbank()` function in the `python_speech_features` package and concatenate both MFCCs and Log Filter Banks to build the design matrix.

After extracting the features from the processed audio data, in order to get good performances of the models, we decide to train the spoken digits and operators individually. Since we assume an operation always goes in the order of digit, operator, and digit, it is reasonable to train the digits and operators separately with the purpose of attaining promising model performances.

Accuracy is the quantitative performance metric we use on our models. Specifically, the accuracy is defined as:

$$Accuracy = \frac{\#Correct\ Predictions}{\#Total\ Predictions}$$

Our main way of visualizing the performance of our model is through the confusion matrix of accuracy, with predicted and true labels (digits or operators) on each axis. The confusion matrix of the prediction accuracy gives us a clear visualization about how the models perform.

3.3. Logistic Regression

We select Logistic Regression as the baseline model, setting a benchmark for more complicated models. The multiclass Logistic Regression model calculates the probability of each digit or operator as the output, given the input feature vector. After we split the dataset into 80% of training and 20% of test set, we train our multiclass Logistic Regression model using the Logistic Regression classifier from Scikit-Learn library [12]. In order to improve the prediction accuracy, we apply Adaptive Boosting (AdaBoost) to the Logistic Regression model, combining multiple weak learners to a strong one. Specifically, an AdaBoost classifier is calculated as [9]:

$$h_k(x) = argmax_j \sum_r \alpha_r 1[h_r(x) = j],$$

where each h_r is a weak learner which takes and $1[h_r(x) = j]$ refers to the $0/1$ loss.

3.4. Random Forest

As a simple but powerful machine learning algorithm composed of bagging and decision trees using random feature subsets, Random Forest is considered one of the most popular machine learning algorithms [9]. Since Random Forest has already chosen a random subset of features in each decision tree, using PCA for feature reduction is not necessary or effective. We use the Random Forest classifier from Scikit-Learn library to fit our Random Forest model.

3.5. RBF Kernel Support Vector Machine

To better predict the digits and operators, we decide to use a more complicated model–Radial Basis Function (RBF) Kernel Support Vector Machine (SVM), which is effective when the number of features is larger than that of training examples. A popular Kernel method in SVM, Radial Basis Function is a function whose values are dependent on the distance from the origin or some point [14]. The formula for calculating the RBF Kernel is:

$$K(X_1, X_2) = exponent(-\gamma\|X_1 - X_2\|^2).$$

We use Grid Search to tune the hyper-parameters C, which is the inverse of strength of regularization and γ. Specifically, according to Yadav [14], as C and γ decrease, the model tends to underfit and as they increase, the model is likely to overfit. After tuning the hyper-parameters through Grid Search, we build the SVM using the optimal C and γ.

4. Experiments

4.1. Dataset

We base our research on the data collected from Free Spoken Digit Dataset (FSDD), which is a simple speech dataset consisting of recordings of spoken digits from 0 to 9 in wav files at 8kHz [6]. To be detailed, the dataset includes 4 speakers and 2,000 recordings in English (50 of each digit per speaker). In addition to FSDD, we also create another dataset consisting of spoken operation signs, including addition, subtraction, multiplication, and division. Specifically, each of our group members record 50 audio clips of “plus”, “minus”, “times” and “over”. In total, we have 600 recordings of operators (3 speakers, 4 operation signs, 50 recordings for each sign).

4.2. Data Pre-processing

4.2.1 Pre-emphasis

The first step we do is to apply a pre-emphasis filter to our audio clips. Pre-emphasis is effective in emphasizing useful information such as human voice, which usually has high frequencies, and de-emphasizing useless information, such as noice, which usually has low frequencies (Figure 1, 2). The pre-emphasis filter accomplishes this by balancing the frequency spectrum using the following equation:

$$y(t) = x(t) - \alpha x(t-1) \text{ [4]},$$

where α is the filter coefficient, which usually takes the value of 0.95 or 0.97 [4]. In our project, we use α = 0.97.

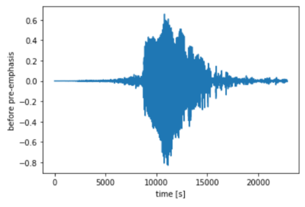


Figure 1. Before Pre-emphasis

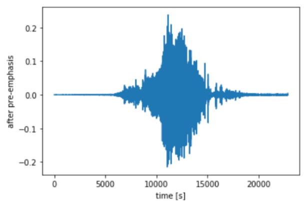


Figure 2. After Pre-emphasis

4.2.2 Truncation

Since our data from the FSDD [6] and the self-created operator dataset are manually recorded audio clips, the speaker may not start speaking at the beginning of the clip, and may not end speaking exactly at the end of the clip. That is, the endpoints (start point and end point) vary for each audio clip. To extract features, each audio clip is divided into several time intervals of same length $(10^{-3}s)$ and features are extracted from each time interval. Without proper truncation, blank chunk at the beginning and the end of audio clips may result in useless features (Figure 3). Therefore, we develop a truncation method according to the normalized absolute value of the amplitude of the original data (Figure 4). Specifically, we set start point and end point as the first and last interval that has absolute amplitude value larger than a threshold, and discard the intervals outside the two endpoints. We set the threshold of 0.1 after trying multiple values. (Figure 5).

4.2.3 Resampling

Another issue of the dataset is that the useful audio chunks are not in the same length after truncation, which may result in inconsistency in number of features. We resolve this issue by resampling all truncated samples to the same length (10,000 frames), so that the characteristic features of all samples of a word could align with each other (Figure 6). We conduct truncation and resampling on all samples of digits and operators.

4.2.4 Input Compatibility

To better fit our application goal, we enable input in the form of a complete sentence (e.g. “five plus one”). To extract the corresponding digits and operator, we perform a truncation similar to the one discussed previously (Figure 7). We first normalize the absolute value of amplitude, truncate out heading and tailing blank chunks with an amplitude threshold of 0.1. After we tune the threshold of waiting time between two words, and set $0.1s$ as the best parameter, we scan through the normalized amplitude, find the first blank chunk that is longer than the waiting time threshold. The first blank chunk serves as the separator of the first digit and the latter part. We then conduct the same truncation on

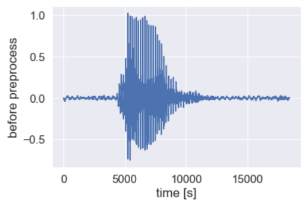


Figure 3. Before Processing

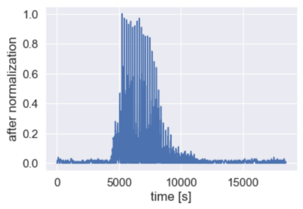


Figure 4. Normalization

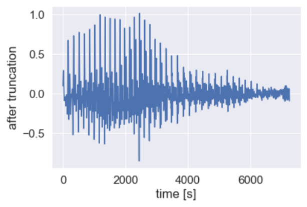


Figure 5. After Truncation

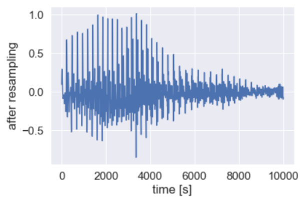


Figure 6. After Resampling

the latter part, separating out the operator and the second digit. Below is an example of sentence “5 plus 4” (Figure $7-10$).

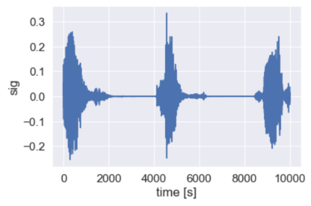


Figure 7. Original Input

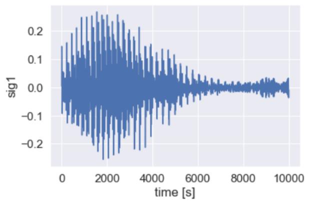


Figure 8. First Digit

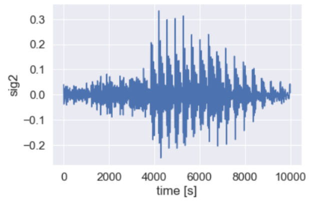


Figure 9. Operator

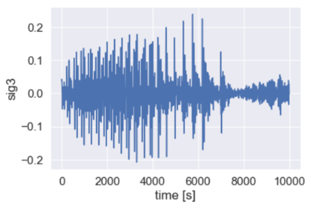


Figure 10. Second Digit

4.3. Logistic Regression

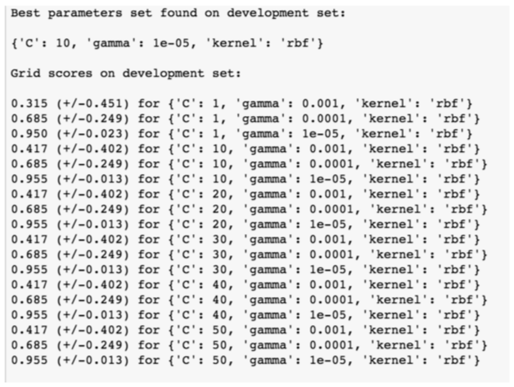
As we build the Logistic Regression model, we set the `multi_class` to ‘multinomial’, `solver` to ‘lbfgs’ and `max_iter` to 10,000. We then apply L2 regularization to our models since we presume that all input features impact the output. To apply the AdaBoost algorithm, we import the `AdaBoostClassifier` from `sklearn.ensemble` and set the number of estimator `n_estimators` to 100 and `random_state` to 0.

4.4. Random Forest

We fit Random Forest models on digits and operators separately using Random Forest Classifier from Scikit-Learn library [12]. Specifically, we initialize the number of trees in the model by setting `n_estimator` to 150. We try Adaptive Boosting to boost model performance. However, since Random Forest is an ensemble method, AdaBoost does not change the performance of it much.

4.5. RBF Kernel Support Vector Machine

We conduct Grid Search on the development sets to tune the hyper-parameters for RBF Kernel Support Vector Machine and get the best parameters with the best grid score. We then train the SVM with the optimal hyper-parameters C of 10 and γ of 0.00005 (Figure 11).



```
Best parameters set found on development set:

{'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}

Grid scores on development set:

0.315 (+/-0.451) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.950 (+/-0.023) for {'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}
0.417 (+/-0.402) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}
0.417 (+/-0.402) for {'C': 20, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 20, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 20, 'gamma': 1e-05, 'kernel': 'rbf'}
0.417 (+/-0.402) for {'C': 30, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 30, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 30, 'gamma': 1e-05, 'kernel': 'rbf'}
0.417 (+/-0.402) for {'C': 40, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 40, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 40, 'gamma': 1e-05, 'kernel': 'rbf'}
0.417 (+/-0.402) for {'C': 50, 'gamma': 0.001, 'kernel': 'rbf'}
0.685 (+/-0.249) for {'C': 50, 'gamma': 0.0001, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 50, 'gamma': 1e-05, 'kernel': 'rbf'}
```

Figure 11. Tuning hyperparameters C and γ

4.6. Software

- Python
  - ML: python_speech_features, Scipy, Scikit-Learn, glob, sys, nltk, Librosa
  - Visualization: Matplotlib, Seaborn
  - Others: Pandas, NumPy
  - Environment: Jupyter Notebook

All models were implemented in scikit-learn 0.22 [12] and trained on Jupyter Notebook. The source code is available at https://github.com/ScorpionXiezi/Audio-Calculator

4.7. Hardware

- Audacity: this is a free and open-source digital audio editor and recording application software, available for Windows, macOS, Linux, and other Unix-like operating systems. [11]

5. Results and Discussion

5.1. Logistic Regression

In the Logistic regression model, we achieved 82.17% test accuracy for spoken digits and 92.86% for operators. As a relatively simple model, it has several problems. Specifically, logistic regression model assumes that each feature value is independent of each other, which does not take the sequence into consideration. This is a strong assumption and might not be appropriate to our dataset, given that audio data have sequential patterns. What's more, since logistic regression might not perform well when there are more than 5 output classes, the test accuracy for spoken digits using logistic regression is relatively low as there are 10 output classes for digits compared to that for operators, which only have 4 output classes.

To improve the logistic regression model, we use AdaBoost by combining the weak learners to make a stronger one, improving the test accuracies slightly. Specifically, the test accuracies of Logistic Regression with AdaBoost are 83.3% for the spoken digits and 93.6% for operators respectively.

5.2. Random Forest

In the Random Forest model, the test accuracies for digits and operators are 90.5% and 94.87% respectively. As shown in the confusion matrix (Figure 14), Random Forest does a good job in predicting both digits and operators. We think that this is because Random Forests works well for high dimensional data since it chooses only a subset of the features [8]. Our data has over 4,800 features, which is extremely high dimensional.

5.3. RBF Kernel Support Vector Machine

In the RBF Kernel SVM model, the test accuracies for the spoken digits and operators are 93.6% and 93.9% respectively. The RBF Kernel SVM model performs the best among all the models, since it has the highest test accuracy. Specifically, as the number of features is 4,836 and the number of training examples is 1,600, SVM model works well when the number of features is higher than that of the training examples.

5.4. Model Comparison

Model	Digit	Operator
Logistic Regression	82.17%	92.86%
Random Forest	90.5%	94.87%
Logistic Regression-Adaboost	83.3%	93.6%
Random Forest-Adaboost	89.5%	94.9%
RBF Kernel SVM	93.6%	93.6%

Table 1. Model Test Set accuracy

According to Table 1, Random Forest and RBF Kernel SVM have higher accuracies in predicting spoken digits [6]. All five models are reasonably accurate on operator dataset. It is worth noticing that Random Forest does not benefit from AdaBoost algorithm as it uses bagging in its own implementation.

From Figure 12 - 14, for digits, SVM and Random Forest are better at predicting "six" and "seven", compared with

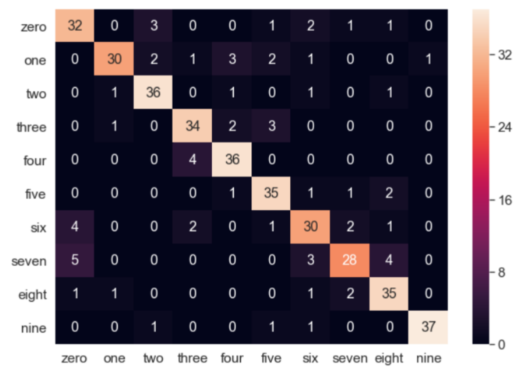


Figure 12. Confusion Matrix of Logistic Regression (AdaBoost) for Digits

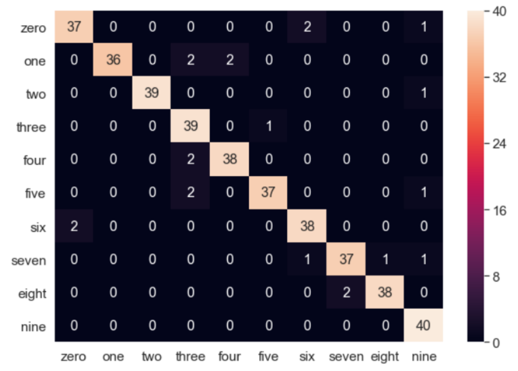


Figure 13. Confusion matrix of RBF Kernel SVM for digits

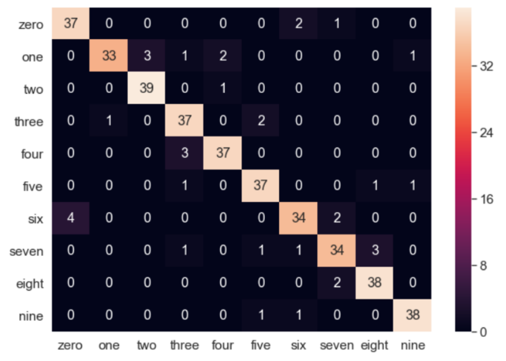


Figure 14. Confusion Matrix of Random Forest for Digits

logistic regression model. While logistic regression with Adaboost, Random Forest and SVM have different advantages at predicting some digits, SVM and Random Forest are more accurate in general. For operators, all models perform reasonably well on the operator dataset because of the small class space (Table 1).

5.5. Plurality Vote & Final Model

Given the performance of all five models, we decide to combine three of them: Random Forest, Logistic regression with AdaBoost, and RBF Kernel SVM. We use the prediction of weighted plurality vote with weight 1, 2, 2 respectively as our final prediction result. In addition, as SVM performs better on prediction digit “six” than the other models, we directly use the result of SVM for digit prediction when its prediction is “six”.

6. Conclusions

The models we build include Logistic regression with and without Adaboost, Random Forest with and without Adaboost, and RBF Kernel SVM. We then apply plurality vote on each kind of the models in order to take advantage of their relative strength, and assign higher weights on the models that perform better in predicting specific class labels. The Confusion matrix of the final model is shown in Figure 15 and Figure 16. The test set accuracies with plurality vote for digits and operators are 96.25% and 96.55%.

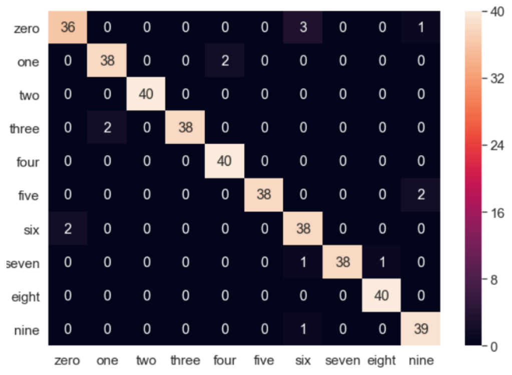


Figure 15. Confusion matrix of final model for digits

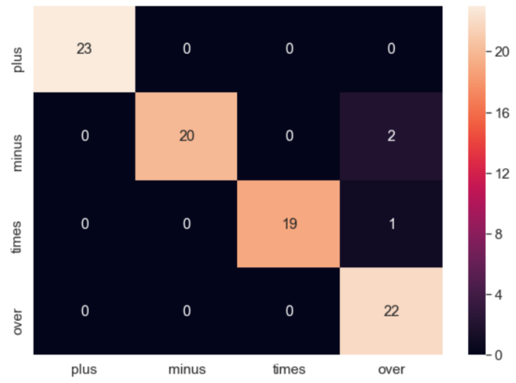


Figure 16. Confusion matrix of final model for digits

6.1. Limitations

All of our models achieve good performances on digit and operator predictions. However, when fitting the models on the training set, we get training accuracies of 100% for all of the models. This means that our models have the problem of overfitting. We believe overfitting arises for the following two reasons: (1) our sample size is relatively small; (2) the variation among samples is small. First of all, we only have 2,000 samples for digits and 600 samples for operators. Each class of digits has 200 samples and each class of operators has 150 samples. This is a pretty small sample size considering we have 10 ten classes of digits and 4 classes of operators. In addition, we are not able to perform calculations involving digits other than 0 to 9 due to the lack of data. For operators, so far our models could not recognize other wordings of operators such as “add”, “subtract”, “multiply” and “divided by”. Therefore, we need to enlarge our dataset to resolve the problem of overfitting and make our calculator more generalized.

Secondly, our dataset has a small variation due to the fact that we have a limited number of speakers. There are only four speakers for digits and three speakers for operators. This could potentially cause our models to hold a bias on those speakers. That is, our models might predict exclusively well on those speakers, while perform poorly on other speakers. We need to include more speakers to increase the variation among the samples.

In summary, the major limitation of our project is the quantity and variation of our dataset. So far, the FSDD is the only online free spoken digit dataset we could find.

6.2. Future Work

To resolve the problem of overfitting, we need to build a larger dataset. We could look for more data online or manually record the data from different speakers. We would also try neural networks such as Convolutional Neural Networks (CNN) [1]. For ensemble methods, we try HistGradientBoostingClassifier. However, due to hardware limitation, we could not successfully apply it to our models, and we would like to try using it if appropriate hardware is available.

7. Acknowledgements

We would like to express our great appreciation to our professor Dr. Sebastian Raschka for offering advice for our project.

8. Contributions

Wengie Wang: She wrote codes for Logistic Regression and SVM model and wrote sections of Abstract, Introduction, Logistic regression, RBF Kernel SVM, Software, Hardware in the report.

Yuhan Xie: He wrote codes for pre-processing, feature extraction, prediction and wrote sections of Data preprocessing, model comparison, and conclusion in the report.

Hengjiali Xu: She wrote codes for Random Forest, the presentation slides, and sections of random forest, related work, feature extraction, limitations, and future work in the report.

References

[1] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4277–4280, March 2012.

[2] H. Bakanay. The advantages of speech recognition. http://aiweb.techfak.uni-bielefeld.de/content/bworld-robot-control-software/.

[3] I. Bazzi and D. Katabi. Using support vector machines for spoken digit recognition. pages 433–436, 01 2000.

[4] H. Fayek. Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html.

[5] W. Gevaert, G. Tsenov, and V. Mladenov. Neural networks used for speech recognition. *Journal of Automatic Control*, 20, 01 2010.

[6] Jakobovski. Free spoken digit dataset. https://github.com/Jakobovski/free-spoken-digit-dataset.

[7] M. Kalamani, D. S. Valarmathy, and S. Anitha. Automatic speech recognition using elm and knn classifiers. 2015.

[8] J. Kho. Why random forest is my favorite machine learning model. https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706.

[9] S. Raschka. Stat 479: Machine learning lecture notes chapter 7 ensemble methods. https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/07_ensembles/07-ensembles__notes.pdf.

[10] S. Suksri. Speech recognition using mfcc. 09 2015.

[11] Wikipedia. Audacity (audio editor). https://en.wikipedia.org/wiki/Audacity_(audio_editor).

[12] Wikipedia. Scikit learn. https://en.wikipedia.org/wiki/Scikit-learn.

[13] Wikipedia. Spectrogram. https://en.wikipedia.org/wiki/Spectrogram.

[14] A. Yadav. Support vector machines(svm). https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589.