Alzheimer's Disease Detection using Disfluencies Implemented Fine-Tuning BERT Model Ensemble

Sub-Project Report, for Research Project: Using speech and language to identify patients at risk for hospitalizations and emergency department visits in homecare

Yunhang Lin yl4860, Kaan Yarali ky2446
ZKLab, Columbia University

Abstract

Alzheimer's disease (AD) and related Dementia (RD) represent a looming public health crisis, affecting one-in-five older adults aged over 60. Therefore, recognizing symptoms at an early stage can be effective for others to provide assistants. In the ADReSS Challenge, it has provided several short dialogues between different experimenters on the basis of gender & age. Many challengers put forward a number of methodologies to do AD classification only through Spontaneous Speech.
In the works of previous semesters, acoustic features were highly emphasized and implemented a lot. While the pre-train model seems to perform better.
In our works, two preprocesses named Segmentation merging and Pause representation were introduced. With fine-tined BERT and model ensemble, we got a maximum test accuracy of 91.67% and maintained the stability of results as much as possible (reduced variance).

Keywords: Alzheimer's Dementia Detection, ADReSS 2021 dataset, fine-tined BERT, Model Ensemble

1. Introduction

Alzheimer's Disease (AD) is a progressive, neurodegenerative disease that affects the lives of more than 5 million Americans every year. The number of Americans living with AD is expected to be more than double that number by 2050 [1].

There is presently no cure for Alzheimer's disease, and early detection is critical for effective treatments. Previous research has shown that speech can be used to distinguish between healthy and AD patients [2]. Current identification of AD is an expensive and invasive method which are the biggest obstacles of this method to becoming widely used in the health industry. There has been extensive research on using machine learning and deep neural networks to identify AD because of their huge success in representation learning in various domains including vision, language modeling, speech, etc.

In this work, we used AWS Speech to Text API to extract transcriptions from the raw audio files. From our previous studies, we know that by using only audio input, models cannot learn discriminative representations for the downstream classification task. In this paper, we are focused on learning optimal feature representations by leveraging the representation power of the pre-trained BERT on transcriptions. In Spring 2022, we developed two strategies using BERT;

1. Method 1: No-Chunk
2. Method 2: Chunk + Time-Gap

In all of the approaches, we filtered out the transcriptions of the nurses and removed them to eliminate the noise coming from their transcription and size constraints of BERT (subword tokens needs to be less than or equal to 512).

Method 1 models the architecture by simply putting a dense classification network over the embedding corresponding to [CLS] when the total number of subword tokens is less than or equal to 512, such that one forward pass of BERT can see all the tokens [3]. Method 1 only uses the speech by the patient and removes the speech by the nurse in the ADReSS 21 dataset in order to limit the number of tokens.

In method 2, we used the BERT embeddings along with the timing and speaker information for each token and feed features corresponding to each token into LSTM as a timestep. In this work, we primarily studied the effect of augmentation on the performance of the model in method 3. This method is studied in detail in the work [4].

In this paper, we also wanted to use the pause information because of the improved performance of the test set. However, we added special tokens for the calculated pauses rather than adding pauses numerically to the input in Method 2.

2. Background

2.1 Data

The ADReSS 2021 dataset consists of audio recordings of spontaneous speech describing a Cookie Theft Picture (CTP) sampled at 44.1kHz, transcripts, and the metadata (age, gender, and Mini-Mental State Examination (MMSE) score) of non-AD and AD patients. The dataset is balanced by gender, age, and label where there were 78 AD patients and 87 non-AD patients audio recordings in the training set. In the test set, there are 71 samples from distinct patients out of which 35 are AD patients.

2.2 Prior Work

A combination of acoustic and linguistic features extracted directly from audio recordings, without human intervention, yielded a baseline accuracy of 78.87 % for the AD classification task using leave-one-out cross-validation in the ADReSS 2021 dataset [1]. [2] achieved an accuracy of 85.4 % on the ADReSS 2020 test set using both audio

features (i-vectors and x-vectors) and text features (word vectors, BERT embeddings, LIWC features, and clan features). [5] calculated the Mel Frequency Cepstral Coefficient (MFCC) feature maps from audio recordings and duplicated those feature maps twice to use pre-trained Mobile-Net [6] and got 58.80% accuracy.

In Zklab, we achieved around 73% accuracy using the acoustic features in Summer 2021. GeMAPS[7] and feature sets from OpenSmiles were used as a feature extractor and those features were fed into various traditional machine learning algorithms such as Random Forest. In Fall 2021, we used YamNet [8] as a pre-trained feature extractor and fed it to a custom LSTM network and got 64% accuracy on the test set. In Spring 2021, method 1 and method 2 got 70%, and 76% accuracy respectively.

3. Methodology

There are several methodologies put forward based on some previous papers and works. On the basis of traditional linguistic pre-train NLP model training, we try to combine some acoustic representations so as to improve the performance of the final ensemble model. So that it can facilitate the generalization of the model to reduce the unavoidable overfitting owing to the small dataset.

Respectively from the perspective of the preprocessing part and model training part, detailed approaches will be illustrated.

Preprocessing:

Although there are dozens of acoustic representations that can be used to predict Alzheimer's disease, it is hard to implement the feature engineering. In other words, different acoustic representations are difficult to find out their contributions to the result leading to time-consuming exploration and trials. According to the results of the previous reports, the final evaluation is unstable and barely satisfactory.

However, relying on linguistic messages alone may also lose some important representations during translation. Therefore, among all acoustic characteristics, disfluency and pause are considered the most valuable information which is also easy to convert to text expressed as punctuations.

The pauses were encoded in the input word sequence. They are grouped into three bins: short (under 0.5 second); medium (0.5-2 seconds); and long (over 2 seconds). The three kinds of pauses are respectively represented as punctuations ",", ".", and "…". The original punctuations in the transcripts would be removed in advance.

Two examples that show the difference before and after add pause representations are below.

1. *adrso039 (original): Table seems found The thing is running over chief for cooking drying dishes sauces long and then this Well yeah Heres some outside wonder guarding against*
2. *adrso039 (after preprocessing): Table , seems found , The thing is , running over ... chief for , cooking , drying , dishes , sauces ... long , and , then ... this , Well yeah Heres some ... outside ... wonder . guarding against*

Why choose such punctuations to express pauses is because, intuitively, comma, period and ellipsis already stand for pauses of different lengths, from short to long. For most pre-train models like BERT, the model has been exposed to a large number of texts with these symbols. In the process of training, it is familiar with the meaning represented by the word vectors corresponding to these punctuation marks. So, during fine-tuning, if abnormal pauses appear in weird positions or come up too many times, the model should relate it with Alzheimer's disease.

The statistical results found that subjects with AD have more pauses in every group, but the difference between subjects with AD and non-AD is particularly noticeable for longer pauses.

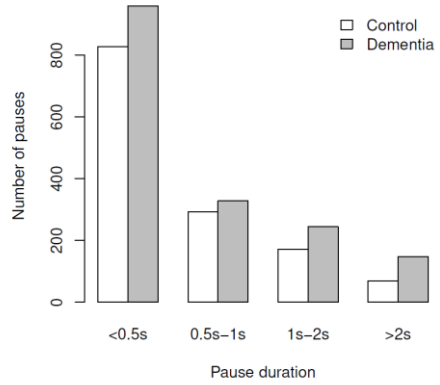


Fig. 1: The number of different type of pauses

Cross validation:

The previous training and validation sets were fixed when doing training iterations, even though we have shuffled the files in the very beginning, which will cause biased dataset in the aspect of classification class.

In the previous work, the first 133 files are fixed as training sets. The last 33 files are fixed as validation sets. So that bias always exists (AD % = 57.14% in training dataset, AD % = 33.33% in validation dataset). And the last 33 files are always not involved in training.

Therefore, cross validation is very suitable to balance such problems. Depending on the classification task, the proportion of sample categories should be considered when selecting the test set and training set, i.e., stratified sampling. Hence, StratifiedShuffleSplit was selected to take charge of it.

Compared with traditional K fold split, it will provide train/test indices to split data in train/test sets. This cross-validation object is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds.

The folds are made by preserving the percentage of samples for each class.

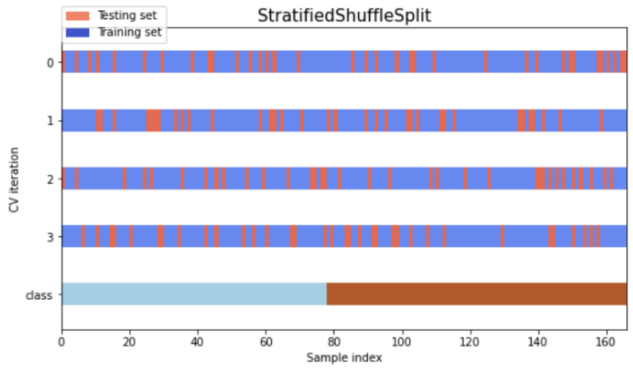


Fig. 2: Visualization of StratifiedShuffleSplit

For instance, in Fig. 2, we split the dataset into 4 groups, each group contains 20% validation data and 80% training data. Training data in blue color and validation data in orange color. There are two classes in the dataset: AD and CN. In the picture, the light blue color represents CN, and the brown color represents AD. The data sets were selected uniformly in different classification layers according to the proportion.

Model architectures

Google's BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based sentence representation in natural language processing. It is also capable of producing individual token embedding in the context of a given sentence[]. BERT is basically an Encoder stack of transformer architecture. A transformer architecture is an encoder-decoder network that uses self-attention on the encoder side and attention on the decoder side. BERT base has 12 layers in the encoder stack. It produces feature embeddings having 768 hidden units.

BERT is used for more than 11 language tasks in the field such as Sentiment analysis, etc. A massive dataset of 3.3 Billion words has contributed to BERT's continued success(Wikipedia (~2.5B words) Google's BooksCorpus (~800M words)). It was trained using 64 TPUs over the course of 4 days.

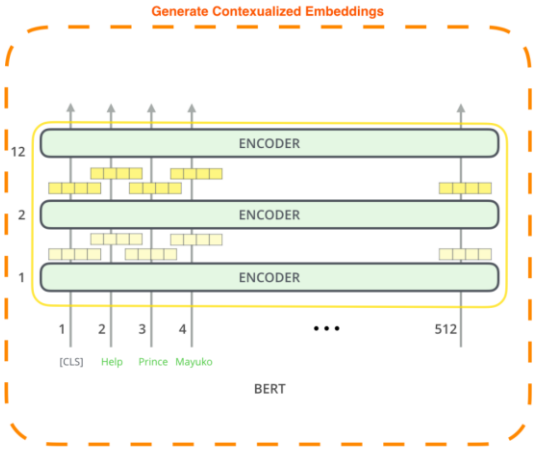


Fig. 3: Architecture of BERT

For the initial training, it was trained for two different tasks;

1) Masked Language Modeling (MLM)
2) Next Sentence Prediction (NSP)

In MLM, some words are randomly masked in the sentence and the model is trying to predict those words.

NSP (Next Sentence Prediction) is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not.

BERT is trained on both MLM (50%) and NSP (50%) at the same time. Unlike other large learning models like GPT-3, BERT's source code is publicly accessible (view BERT's code on Github) allowing BERT to be more widely used all around the world.

BERT uses special tokens [CLS] and [SEP] to understand input properly.[SEP] token has to be inserted at the end of a single input.[CLS] is a special classification token and the last hidden state of BERT corresponding to this token (h[CLS]) is used for classification tasks.

Ensemble

Previous studies have found that when fine tuning BERT for downstream tasks with a small data set, the model has a high variance in performance. Even with the same hyperparameter values, distinct random seeds can lead to substantially different results. Our experiment further confirms this conclusion.

Therefore, a model ensemble to some degree can reduce the standard deviation and increase the mean overestimates based on a single run.

Simple voting strategy was introduced as the ensemble method. There are two methodologies that extend from this:

1. The voting model based on the same random seed

2. The voting model based on different random seeds

The overall logistic is, by means of training models separately, we get dozens of trained models with fine tuning. Then, all the models do prediction for the test sets. So that it will get a series of prediction results. The category with the largest proportion of predicted results is the final answer.

4. Implementation

4.1 Data Preprocessing

Segmentation merging

In previous works, transcripts were archived as JSON files in forms of a verbatim display of its start time and end time. There are also time divisions for different speakers stored by CSV files. Only the segmentation parts of PATIENTS should be included in model training.

In the original segmentation files, it marks the beginning time and end time of different speakers, as shown in the left subfigure of Fig. 3. Since we only need the content of patients, we only select the rows with PAR speakers.

However, the original segmentation is on the basis of clauses. Each clause will have a corresponding time segmentation in JSON files which contains the time points of words. While the beginning time and end time of the two files are not exactly matching.

Let's take an easy example of a certain file (adrso039.csv). The original transcript is like this,

Table seems found. The thing is running over chief for cooking, drying dishes, sauces long and then this. You see anything else? Well, yeah. Here's some outside wonder guarding against. Okay. Okay. Good.

If we delete non-patient fields according to the original time period, after removing all the original punctuations, the result would be like,

Table seems The thing is running over chief for drying dishes sauces long and then this Well yeah Heres some outside guarding against

If segmentation merging applied, it would be like

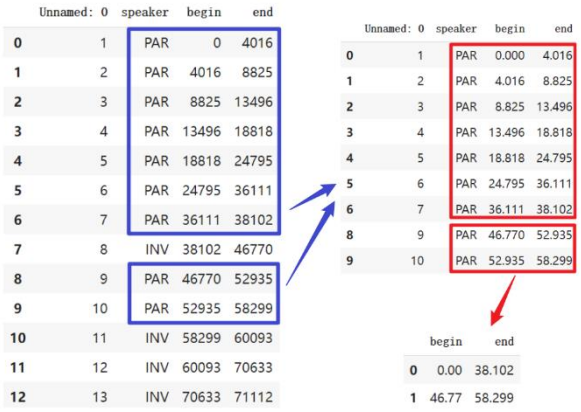
*Table seems **found** The thing is running over chief for **cooking** drying dishes sauces long and then this Well yeah Heres some outside **wonder** guarding against*

It is not difficult to find that the bold words are missed in the last text, especially when the word is in the boundary of two clauses. Because the time intervals in two files are not exactly matched, if the deviation causes the interval of word to span two intervals of clause in original segmentations, it is easy to ignore the word during preprocessing.

Since we have so few texts to begin with, the loss of words can have unforeseen consequences. There is a clear improvement in terms of inclusion of previously missed words.

Here is how it works. As shown in Fig. 3, For adjacent PAR time periods, if the start time of the next period is equal to or smaller than the end time of the previous period, they would be regarded as one interval and then merge together.

Finally, many small intervals depending on clauses would be connected to a large one.



	Unnamed: 0	speaker	begin	end
0	1	PAR	0	4016
1	2	PAR	4016	8825
2	3	PAR	8825	13496
3	4	PAR	13496	18818
4	5	PAR	18818	24795
5	6	PAR	24795	36111
6	7	PAR	36111	38102
7	8	INV	38102	46770
8	9	PAR	46770	52935
9	10	PAR	52935	58299
10	11	INV	58299	60093
11	12	INV	60093	70633
12	13	INV	70633	71112

	Unnamed: 0	speaker	begin	end
0	1	PAR	0.000	4.016
1	2	PAR	4.016	8.825
2	3	PAR	8.825	13.496
3	4	PAR	13.496	18.818
4	5	PAR	18.818	24.795
5	6	PAR	24.795	36.111
6	7	PAR	36.111	38.102
8	9	PAR	46.770	52.935
9	10	PAR	52.935	58.299

	begin	end
0	0.00	38.102
1	46.77	58.299

Fig. 4: Visualization for Segmentation Merging

Pause representations

Since we have had the start time and end time of every word, we can judge whether there is an interval between two words, then we can regard it as a pause. We simply assume that there are three kinds of pause that are respectively represented by different symbols.

```
Pauses = {"short":",","medium":".","long":"..."}
```

We wrap each small preprocessing step as a separate function. They are respectively getting intervals from files, deleting other speakers, removing the original punctuations, and inserting pause representations.

By means of Regular matching and Numpy manipulation, it is easy to accomplish all steps and output the content after preprocessing, which will be convenient for the following model training.

	Id	Label	Content
0	adrso039	AD	Table , seems found , The thing is , running o...
1	adrso028	AD	How she will , find her , mothers wish , Mhm ,...
2	adrso031	AD	, what ... From what I can see hes going to up...
3	adrso036	AD	, All the , Whats going on in this picture Was...
4	adrso033	AD	... Hey ... I , really , dont know , because t...
...	...	...	...
161	adrso302	CN	Egypt , Well start with the girl . Shes , goin...
162	adrso274	CN	Yeah , All right . boy is , taking a , cookie ...
163	adrso280	CN	... Uh huh , Yeah ... its full of ... boy , co...
164	adrso307	CN	... the boys , the girls making fun of the boy...
165	adrso276	CN	its the same , picture , It should give me a d...

166 rows × 3 columns

Fig. 5: Output files after preprocessing

4.2 Deep Learning Network

We used the pre-trained BERT base as a backbone feature extractor to take advantage of transfer learning by inheriting knowledge from much larger datasets. In this problem, the dataset we have consists of only 166 training samples which makes the learning discriminative features applying deep learning methodologies very difficult. Therefore, we would like to exploit the representation power of pre-trained networks. However, this also brings some constraints which you cannot change while training. For example, the maximum number of tokens (512) and hidden state dimensionality (768). In the BERT paper [], authors mention that for fine-tuning the model does not need to train for many epochs. The process is typically very fast.

The network that we build on top of the CLS token has a high impact on the performance of the model. Therefore, we have studied a comprehensive hyperparameter tuning study using Weights and Biases [].

We defined the sweep configuration as;

1) **Batch Size:** 8,16
2) **Dropout:** (in between each dense layer): 0.3,0.4,0.5
3) **Hidden state dimensionality:** 64,128,256,512
4) **Number of dense layers:** 1,2,3,4

All of the combinations were tested using grid search API in Weights and Biases. In the original paper, authors do not recommend changing the optimizer type and learning rate so we did not change them.

4.3 Model Ensemble

After cross validation, the data is balanced. The percentage of AD and CN are approximately around 50%, as shown in Fig. 5. And we can fully use the whole dataset for training and validation.

```
new set:
Val set size  0.20481927710843373
Pos sample %:  0.5294117647058824
     TRAIN: [160 146  51 130   3] TEST: [105 136  20  36 121]
new set:
Val set size  0.20481927710843373
Pos sample %:  0.5294117647058824
     TRAIN: [ 92  11 119 155  50] TEST: [ 58  93  23 147   0]
```

Fig. 6: Proportion of classification after cross validation

As mentioned above, the performance of individual models would vary a lot due to the limited amount of data. So, a model ensemble is used to reduce such deviation. Also, statistical evaluation is needed to judge the overall performance of novel methodologies. Therefore, abundant training samples are required.

In order to improve the reusability of the trained model so as to save resources and time, the weight of the model is saved as long as training is finished. With different hyperparameters and cross validation, they are named after their attributes.

After long-time experiments, 13 groups of hyperparameters came out with excellent performance. Since the loss function and optimizer were determined, we combined the best hyperparameter candidates with different-seed cross validation, like permutations, so that we could get a bunch of individual trained models.

Randomly, 10 types of seed were elected, each seed would repeatedly train 5 times.

Since the dataset is reloaded and shuffled every time training, even though the random seed is the same, the results are different.

Hundreds of models can be used to implement the voting ensemble and obtain the statistical result. The detailed result and analysis will be displayed at Section 5.

4.4 Software Design

In the preprocessing part, every step was set to a function. The details can be seen in the code.

In the model ensemble part, for each batch of test sets, every time we input the linguistic messages, each model can output a prediction (1 and 0 represent True and False). Once we get all the predictions of models, we can select the majority voting. For example, 60% models choose 1: True, so the ensemble result is True.

5. Results

5.1 Project Results

Generally speaking, we have done a lot of experiments to improve the performance of prediction. Test accuracy and test loss are regarded as the final metrics to judge the evaluation. All results were repeated multiple times to get the statistical data. Among all of them, the maximum test accuracy is 91.67%. Maximum mean value of model

ensemble is 87.65%. Minimum standard deviation is 0.0134. Details are shown in the next subsection.

5.2 Comparison of Results

There are several pairs of results that are worth noticing.

5.2.1 Effectiveness of Preprocessing

In the preprocessing part, we have put forward two novel ideas manipulating the input linguistic text. To verify the effectiveness of the improvement, a simple experiment was implemented.

In the same experiment setting, by means of simplest training methodology, we respectively trained the model using the dataset:

- With segmentation merging
- With pause representation
- With segmentation merging + pause representation

Validation accuracy and test accuracy were regarded as the metrics to judge the performance of the model. The result was displayed as Table 1.

Table 1: Statistical results of test accuracy using different preprocessing strategies

	Validation Acc	Test Acc
Normal	0.83	0.62
Pause Representation	0.81	0.7
Segmentation + Pause	0.88	0.82

Normal means there is no preprocessing implemented. It is not difficult to find from the perspective of Test Accuracy, the preprocessing manipulations dramatically improve the performance of prediction.

Investigate its reason. Segmentation merging prevents the risk of loss of some important words. Since we were very concerned about the accuracy of the description of the test subjects, the appropriate degree of words had a strong correlation with the result of predicting AD. As the number of datasets is so small, any word loss may lead to erroneous judgment for models.

Besides, the introduction of pause representation causes a huge facilitation for prediction. Punctuations call back the missing acoustic information, which demonstrates better performance than only using the linguistic text itself.

Although the results of the experiment are highly variable, which means that there may be some deviation for the values of conclusion, the increasing trend can be seen. So that such preprocessing is worthwhile.

5.2.2 Effectiveness of Model Ensemble

This is a simple experiment in addition to master training in order to find whether a model ensemble can prevent the high variance of results caused by a small dataset and improve the general performance.

By means of 7 different random seeds, each seed is used to run the model 4 times. Therefore, 28 runs were recorded and voted for the prediction, as shown in Fig. 7 black dots stand for individual experiment results. Orange line is the test accuracy of voting with 85.5%.

Some interesting things worth noting is that the maximum accuracy of the ensemble sample is 87% and the mean accuracy is around 82%. While the voting result is 85.5%. From a sample point of view, the trials whose accuracy are greater than 85.5% are only 5 (18% in the sample).

It proves that the value of an ensemble, to some extent, would be so much better than the mean value of a group of individual experiments. Even, it may be better than the maximum value (will be shown later).

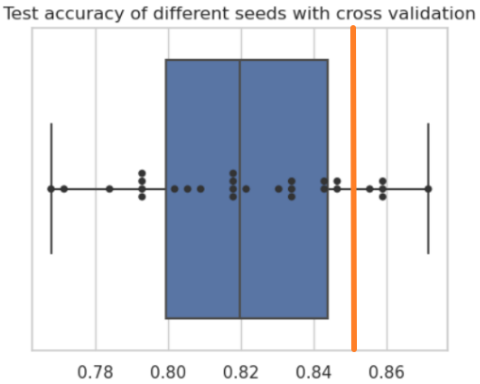


Fig. 7: Boxplot of Model Ensemble

5.2.3 Effectiveness of Hyperparameter Fine-tuning

Before fine-tuning the model, we store the statistical training data for fixed hyperparameters. Only with the strategy of cross validation, 20 types of seeds were selected. Each seed is used to run the model 10 times.

It has been proved that, in the scenario of the same seed, even though the input data is the same, the initialization of the model is the same, the result of the model will be different.

Because every time running a new training, the data loader for training and validation will be reloaded and shuffled, which means that every batch of data is different among different iterations of training. Therefore, even with the same seed, the training process is different. To some degree, it may compromise the deviation when shuffling the data.

With the saved weights, it is easy to combine all the models for voting after a long-time training process. The statistical result is shown in Table 2. When doing voting, every iteration would use 10 related models to do a model

ensemble. The result distribution of the single model is shown in Fig. 8.

Generally speaking, voting increases the mean value and decreases the standard deviation.

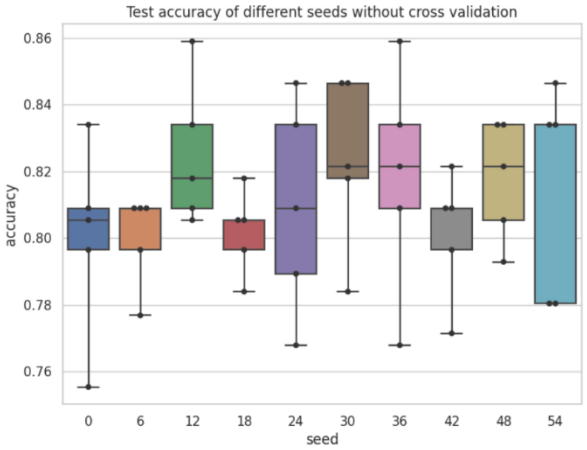


Fig. 8: Boxplot of individual models

Table 2: Statistical results of test accuracy using cross validation

	min	max	mean	std
Single	0.71825	0.90079	0.83083	0.03350
Same seed voting	0.81944	**0.91667**	**0.87654**	0.02812
Diff seed voting	**0.84722**	0.88889	0.86667	**0.01416**

Table 3: Statistical results of test accuracy using cross validation + fine-tuned hyperparameters

	min	max	mean	std
Single model	0.50796	**0.90081**	0.82042	0.03436
Same seed voting	0.81793	0.88397	0.85500	0.01967
Diff seed voting	**0.85536**	0.88393	**0.87440**	**0.01200**

Since we did 384 iterations of model training, 13 groups of best candidates for hyperparameters were selected, including model architectures, dropout, batch size and epoch. By means of such hyperparameters, cross validation was combined to do model ensemble. 390 individual models were trained. And the results are displayed in Table 3.

Compared with the results of Table 2, after adding fine-tuned hyperparameters, although the accuracy value does not improve any more, the standard deviation decreased a lot. That is what we want to see. Since the dataset is very small, low-variance results are what we expected. And the particular values are also delightfully high enough.

6. Future Work

Since we have explored and conducted a lot of potential improvement from the perspective of preprocessing and model training, the linguistic pre-train model has pretty much been studied. Future work may be like below:

- More accurate transcripts
- More subdivided pause representations
- More acoustic features that can be quantified as text messages
- Better hyperparameters fine-tuning
- More diverse model ensemble
    - Other pre-train models: ALBERT, ERNIE, etc.
    - Acoustic model: LSTM, etc.
    - Other voting strategies: bagging, etc.

7. Conclusion

In this paper, we have done a lot of work from different perspectives. In the part of preprocessing, two approaches were put forward. Segmentation Merging prevents the risk of losing important information. Pause Representation adds the necessary acoustic features to the linguistic messages as the input samples for model training. After two steps of preprocessing, the files were outputted in convenience with further training.

During the model fine-tuning. Hundreds of models were trained to sweep dozen groups of hyperparameters and finally got 13 groups' best hyperparameters. Combined with cross validation. 390 individual models were trained for model ensemble. With the strategies of Voting, the maximum test accuracy was reached 91.67%. And the maximum mean value and minimum standard deviation are respectively 87.65% and 0.012. Compared with previous works, it has dramatically improved.

8. Acknowledgement

Thanks for the cooperation of Kaan, Enze, Yifei and Manisha. Thanks for the support of Prof. Kostic.

9. References

[1] Luz, Saturnino, et al. "Detecting cognitive decline using speech only: The ADReSSo Challenge." arXiv preprint arXiv:2104.09356 (2021).

[2] Glass, James. "Classifying Alzheimer's disease using audio and text-based representations of speech." Frontiers in Psychology (2021): 3833.
[3] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
[4] F Ahmad: Fall 2022 - <Feroz's report - Work Spring 2022>
[5] Zhu, Youxiang, et al. "Exploring Deep Transfer Learning Techniques for Alzheimer's Dementia Detection." Frontiers in computer science 3 (2021).
[6] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).
[7] https://tfhub.dev/google/yamnet/1
[8] Balagopalan A, Eyre B, Robin J, et al. Comparing pre-trained and feature-based models for prediction of Alzheimer's disease based on speech[J]. Frontiers in aging neuroscience, 2021, 13: 635945.
[9] Yuan J, Bian Y, Cai X, et al. Disfluencies and Fine-Tuning Pre-Trained Language Models for Detection of Alzheimer's Disease[C]//INTERSPEECH. 2020, 2020: 2162-6.
[10] Zhang Z, Han X, Liu Z, et al. ERNIE: Enhanced language representation with informative entities[J]. arXiv preprint arXiv:1905.07129, 2019.
[11] Rohanian M, Hough J, Purver M. Multi-modal fusion with gating using audio, lexical and disfluency features for Alzheimer's dementia recognition from spontaneous speech[J]. arXiv preprint arXiv:2106.09668, 2021.
[12] Mohammadi E, Amini H, Kosseim L. Neural feature extraction for contextual emotion detection[C]//Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019). 2019: 785-794.
[13] Liu Y, Lapata M. Text summarization with pretrained encoders[J]. arXiv preprint arXiv:1908.08345, 2019.

8. Appendix

Appendix 8.1 Fine-tuned hyperparameters - table

	Archs	Dropout	Batch Size
1	[512,64]	0.4	16
2	[64,512]	0.5	16
3	[256,512,64]	0.3	8
4	[128]	0.3	8
5	[128,512,256]	0.4	16
6	[256,64]	0.3	16
7	[256,128]	0.4	16
8	[512,256]	0.5	16
9	[256,64,128]	0.3.	16
10	[512,256,64]	0.4	16
11	[128]	0.5	16
12	[64,512]	0.3	16
13	[256,512,128]	0.3	16