Alzheimer's Detection using Speech Analysis

BIO / CS F266 - Study Project under Prof. Veeky Baths

Aditi Umashankar (2018AAPS0329G)

Piyush Maheshwari (2018A7PS0153G)

Shreyansh Joshi (2018A7PS0097G)

Yash Kumar (2018A7PS0126G)

Index

Introduction

- Alzheimer's disease (AD) is a progressive neurodegenerative disease that affects nearly 50 million individuals across the globe and is one of the leading causes of deaths worldwide.
- The average age of onset is around 65 years. In the early stages, it can be hard to distinguish Alzheimer's symptoms from the common effects of ageing.
- Symptoms include memory loss, language impairment, behavioural changes, decline in physical and social abilities, and eventually the inability of the patient to function independently.
- There is no cure for the disease, and the complications in the later stages due to loss of brain function may even result in death.

Motivation

- Since the damage to the brain is irreversible, and there is no cure, the only existing treatment is slowing the progression of the disease and managing the symptoms for some time.

- Hence, early detection is imperative. Current diagnosis tests include MRI, CT, as well as invasive tests that analyse the CSF (Cerebral Spinal Fluid) for certain indicators, some of which can be expensive or inaccessible to public.

- Speech analysis can help in detection of the disease in its early stages, since a prominent sign of AD is language dysfunction.

- Analysing speech does not require any sophisticated equipment, is non-invasive, can be done quickly and is inexpensive, making it highly scalable.

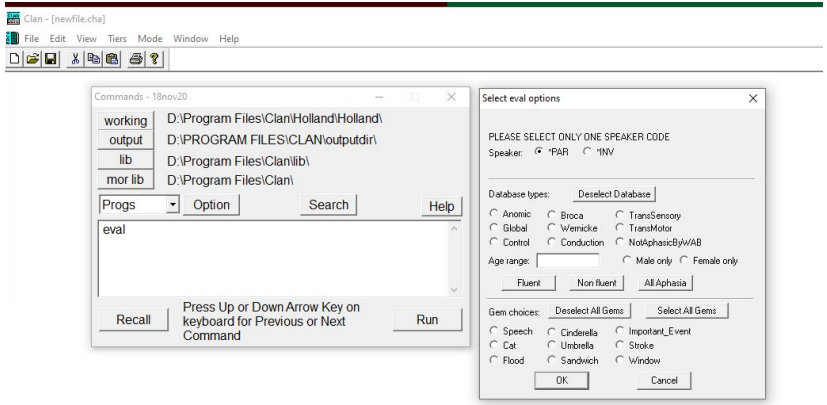
In this project, we worked on detection of Alzheimer's disease using speech analysis with machine learning and deep learning based models.

Data

We acquired transcripts of conversations with people, many of whom had different forms of dementia, in CHAT Format (Codes for the Human Analysis of Transcripts) from DementiaBank's Pitt Corpus. The transcripts were parsed using the CLAN software to extract features specific to our needs.

We used the following commands:

- IPSYN for Syntactic Complexity
- EVAL for Word-type ratios, Grammatical Analysis and Count of Utterances
- FLUCALC for Fluency and Pauses



CLAN software's interface

Data Extraction

The commands FLUCALC and EVAL were run on the transcripts folder-wise. This resulted in 10 CSV files, 5 for each command.

Unlike these commands, IPSYN command had to be run on each file for certain number of utterances separately. We therefore automated the process by writing a an AutoGUI Python Script to input custom values for each file in a folder. The code is shown in the below image:

```
import pyautogui , time, pandas as pd
time.sleep(3)
x=pd.read_csv("./sentence.csv")

for i in range(240):
    fname=(x['File'][i])
    uttsize=(x['Num_Uttr'][i])
    pyautogui.typewrite("ipsyn {} +t*PAR -leng +c{} +f{}sentence".format(fname, uttsize, fname))
    pyautogui.press("enter")
```

Running the commands resulted in around 1300 files from which we extracted the required data.

Data Extraction

We wrote a script for parsing and extracting information from the output files created by IPSYN command in CLAN. The extracted information was then stored in a CSV file with 4 values corresponding to each file.

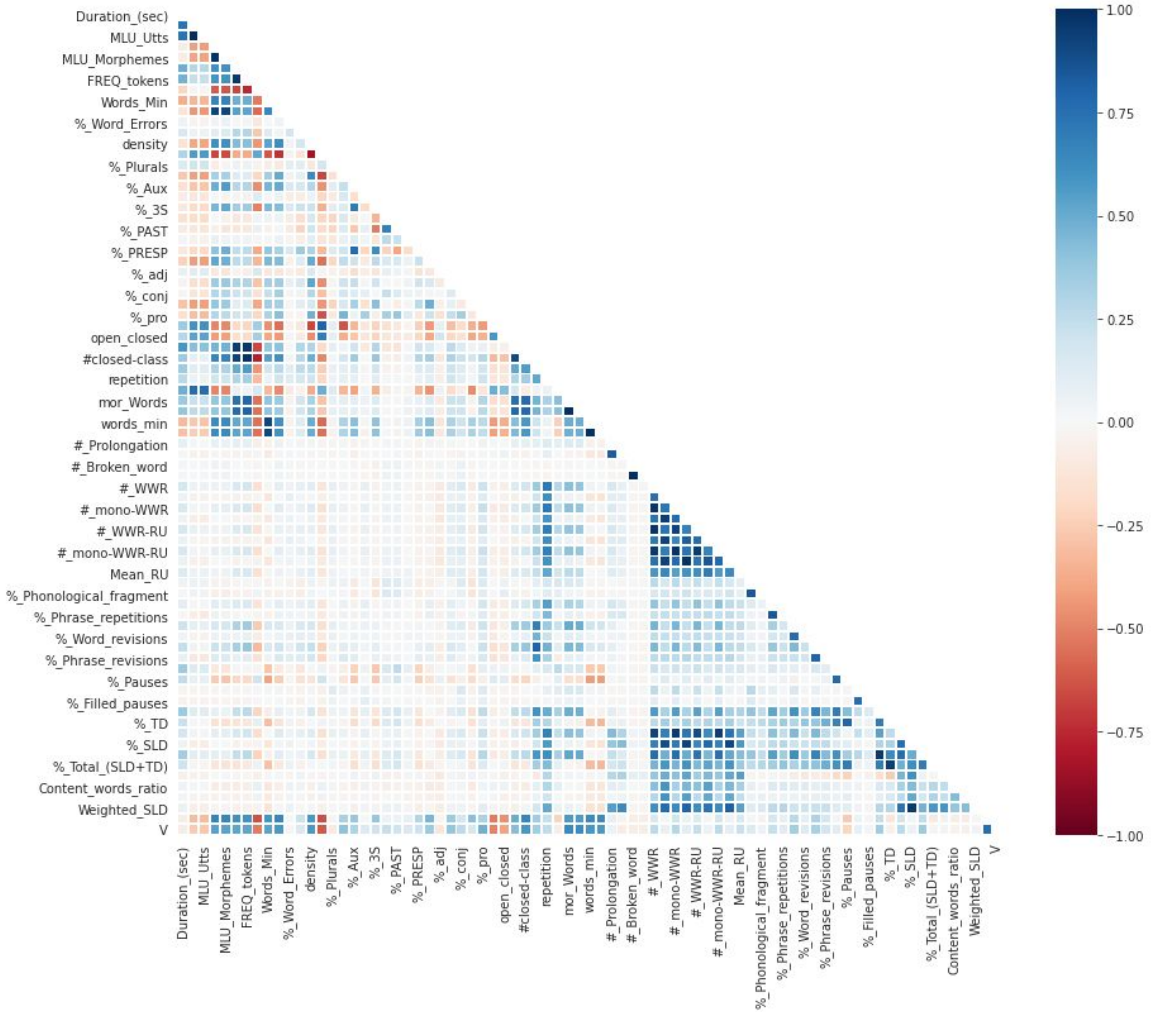
```
file1 = open("files.txt","r")
csvfile=open("sentence.csv",'a')
csvfile.write('File,N,V,Q,S\n')
for i in file1:
    final = open('sentence/'+i[:-1],"r")
    a=i[:5]+'.cha,'
    for x in final.readlines()[-6:-2]:
        a+=x[4:-1] + ','
    a=a[:-1]+'\n'
    csvfile.write(a)
```

IPSYN Script

This CSV file was merged with the CSV files resulting from EVAL and FLUCALC commands using the file name fields. The final CSV file had around 100 features. We then proceeded to clean the data.

Data Cleaning

There were some redundant features, in the CSV file we'd generated . We plotted a correlation matrix (as seen in the figure on the right) to find highly correlated features and dropped them. Apart from this, we manually dropped quite a few columns (features), that were found to be overlapping with some other features.



Retained Features

Following were the 50 columns we retained for training:

'Age', 'Sex', 'Group', 'Duration_(sec)', 'MLU_Utts', 'MLU_Morphemes', 'FREQ_TTR', 'Words_Min', 'Verbs_Utt', '%_Word_Errors', 'Utt_Errors', 'density', '%_Nouns', '%_Plurals', '%_Verbs', '%_Aux', '%_Mod', '%_3S', '%_13S', '%_PAST', '%_PASTP', '%_PRESP', '%_prep', '%_adj', '%_adv', '%_conj', '%_det', '%_pro', 'noun_verb', 'retracing', 'repetition', 'mor_Utts', 'mor_syllables', 'syllables_min', '%_Prolongation', 'Mean_RU', '%_Phonological_fragment', '%_Phrase_repetitions', '%_Word_revisions', '%_Phrase_revisions', '%_Pauses', '%_Filled_pauses', '%_TD', 'SLD_Ratio', 'Content_words_ratio', 'Function_words_ratio', 'N', 'V', 'Q', 'S' .

Data Preprocessing

There were errors in data as well, including NaN entries, misspelled entries - random decimal points or bad spelling/capitalisation. Hence we needed to process the data further.

For instance, we dropped the rows with invalid values of the feature ‘S’ with the following code.

```
rows= []
df1['S'].value_counts()
for i in range(len(df1)):
  if str(df1.loc[i]['S'])[0] == ':':
    rows.append(i)
df1.drop(rows, axis = 0, inplace = True)
```

The entries for the column ‘Group’ in some rows were misspelt. We replaced them with the correct values. Furthermore, we dropped the rows with ‘Group’ as ‘Other’, ‘Dementia’ and ‘.’ since there were very few samples for the same.

```
df1.loc[df1["Group"] == "possibleAD", "Group"] = 'PossibleAD'
df1.loc[df1["Group"] == "Probable", "Group"] = 'ProbableAD'
df1.drop(df1[df1["Group"] == "Other"].index, inplace=True)
df1.drop(df1[df1["Group"] == "Dementia"].index, inplace=True)
df1.drop(df1[df1["Group"] == "."].index, inplace=True)
```

Data Preprocessing

We dealt with NaN values in all columns, by replacing them with the mean corresponding to the group (class of alzheimer) that row belongs to. This was done using the following code snippet.

```
# Need to replace the NaNs in columns with the mean corresponding to their group

columns = ['Words_Min','noun_verb','syllables_min','SLD_Ratio','Function_words_ratio','N','V','S','Q']    # these columns have NaNs
for col in columns:
  for grp in ['Control', 'MCI', 'Memory', 'PossibleAD', 'ProbableAD', 'Vascular']:
    df1.loc[df1['Group'] == grp, col] = df1.loc[df1['Group'] == grp, col].replace(np.nan,df1.groupby('Group')[col].mean()[grp])
```

We also had to encode columns having strings in them to integers, before feeding the data to our model. For instance, we encoded 'Male' as 1 and 'Female' as 0 in the column 'Sex'. Finally, before training, we also standardized the inputs to the model using the function **StandardScaler()** from the ML library **scikit-learn.**

We arrive at the following number of samples:

```
Group
ProbableAD    762
Control       243
MCI           162
PossibleAD     68
Vascular       20
Memory         12
Name: Group, dtype: int64
```

Methodology

We decided to train multiple classifiers to compare their accuracies of prediction using speech factors. We trained multi-class classifiers as well as binary classifiers since some classes had too little samples for the models to learn anything useful.

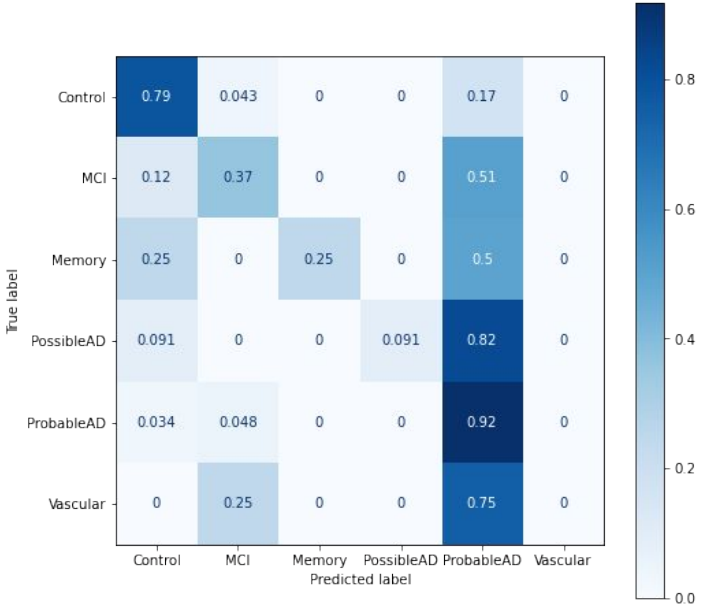
The following models were trained and tested:

- XGB Classifier
- Support Vector Classifier
- Decision Tree
- K-Nearest Neighbours
- Random Forest
- Logistic Regression
- Neural Networks

For all the classifiers, we have plotted a normalized confusion matrix to analyse their performance and compare them.

Confusion Matrix - Interpretation

As an example, consider the normalised confusion matrix we obtained for neural networks of XGB Classifier.



We have plotted this matrix as a grid with coordinates (0,0) denoting (Control, Vascular). Each box contains the fraction of data from the actual class that the model predicted as the class on the x-axis.

Let's take the example of the Control group.

- At (0,5), we have true positive, i.e. we predicted correctly.
- From (1,5) to (5,5), we have false negatives, i.e. we predicted false, but it was actually a true value.
- From (0,0) to (0,4), we have false positives, i.e. we predicted true, but it wasn't a true value.
- The elements on the diagonal, show true negatives, which means they were predicted correctly for a non-control group.

A high value for the diagonal elements is desirable.

Confusion Matrix - Calculation

For each individual group, we have the following terms :

All True/False Positives/Negatives are the actual number of samples, not the fractions mentioned in the confusion matrix..
*Actual number of samples = fraction * total samples for 'true label' group*

- Recall : Out of all the positive classes, how much we predicted correctly. It should be as high as possible.
  (True Positives) / (True Positives + False Negatives)

- Precision : Out of all the positive classes we have predicted correctly, how many are actually positive.
  (True Positives) / (True Positives + False Positives)

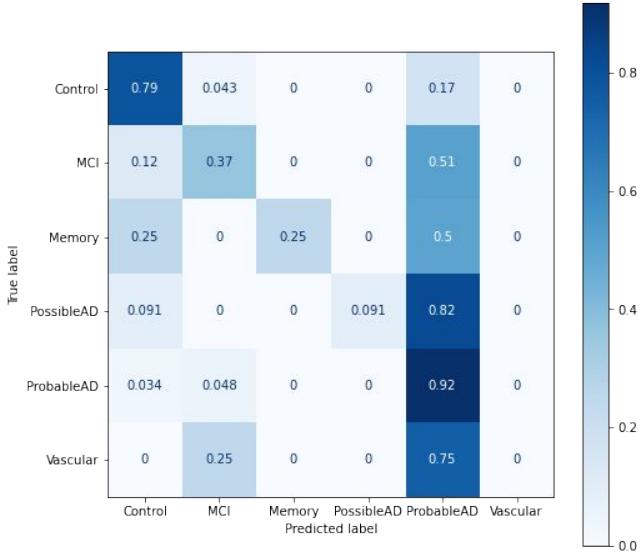
- Accuracy : Out of the entire data, the percentage of predictions made correctly.
  (True Positives + True Negatives) / (Total)

For our problem statement wherein early detection is essential, false negatives mean that a potential AD case was missed by the classifier, compared to a false positive. If the classifier predicts false positives, the person can simply take a professional's opinion to rule out the disease, but if the classifier does not recognise an actual positive case, the loss can be significantly larger in real life.

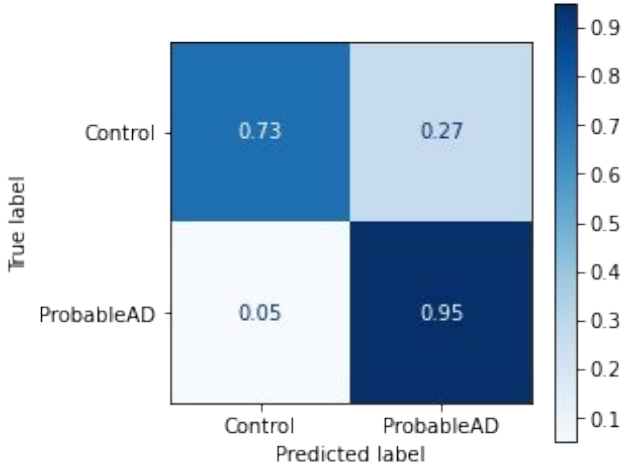
Classifiers - XGB Classifier

The tuned hyperparameters and confusion matrices are:

Multi-class:
colsample_bytree: 0.55,
max_depth: 2,
n_estimators: 90



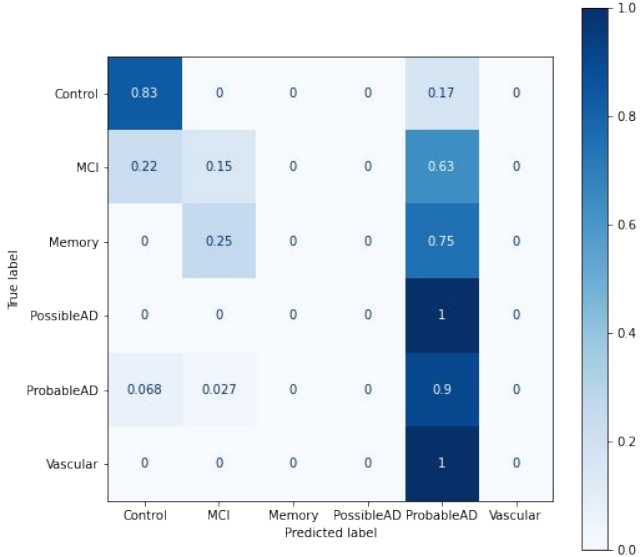
Binary:
colsample_bytree: 0.65,
max_depth: 2,
n_estimators: 90



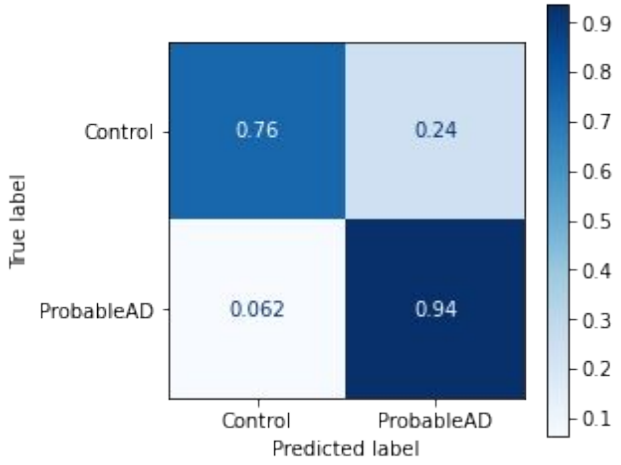
Classifiers - Support Vector Classifier

The tuned hyperparameters and confusion matrices are:

Multi-class:
C: 1.55,
kernel: 'rbf'



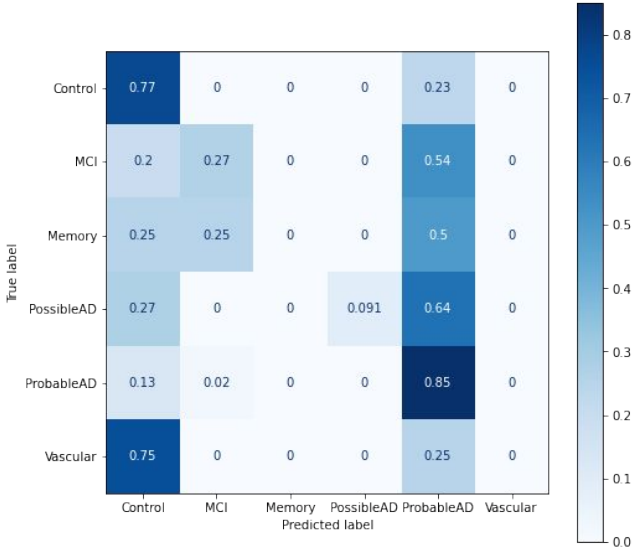
Binary:
C: 11.20,
kernel: 'rbf'



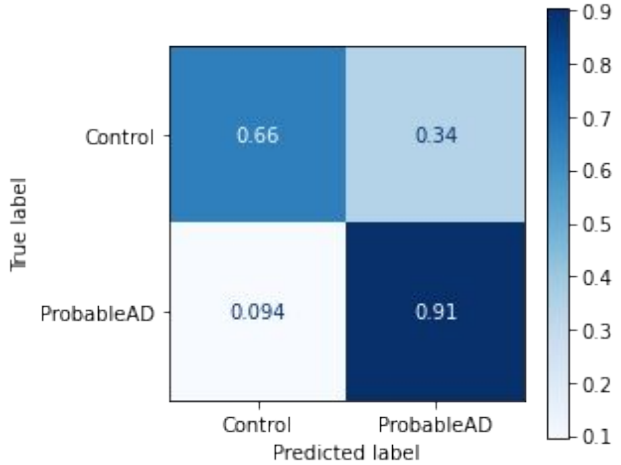
Classifiers - Decision Tree

The tuned hyperparameters and confusion matrices are:

Multi-class:
max_depth= 5,
min_samples_split= 0.039



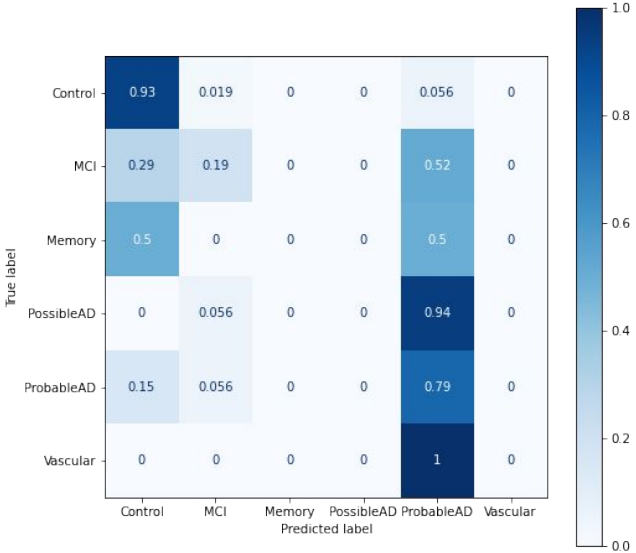
Binary:
max_depth= 9,
min_samples_split= 0.005



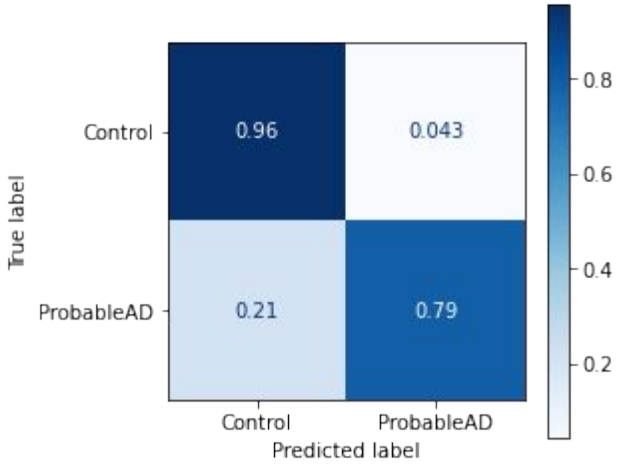
Classifiers - K Nearest Neighbours

We kept the number of neighbours low to get better recall accuracy - to reduce false negatives.
Following are the confusion matrices:

Multi-Class:
Number of Neighbours = 10



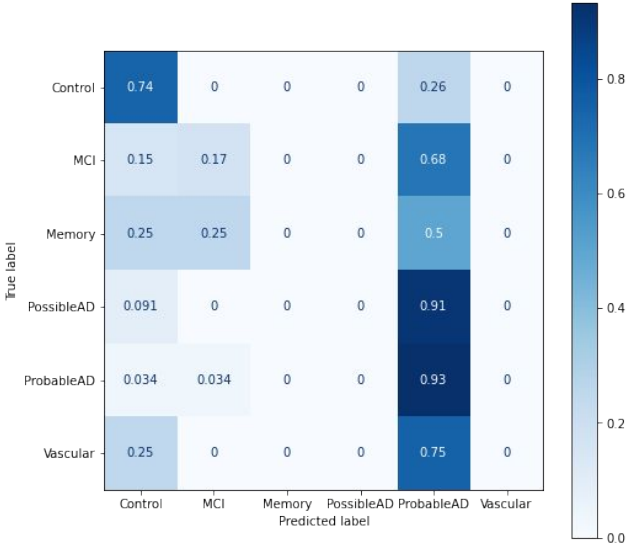
Binary:
Number of Neighbours = 10



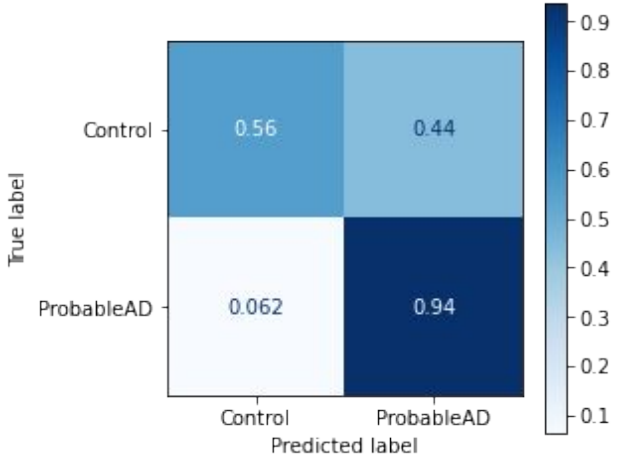
Classifiers - Random Forest

The tuned hyperparameters and confusion matrices are:

Multi-class:
max_depth= 16,
min_samples_split= 0.010,
n_estimators= 20,



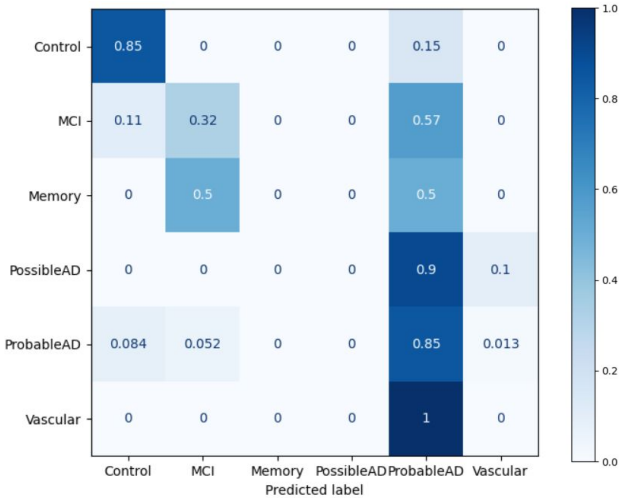
Binary:
max_depth= 8,
min_samples_split= 0.022,
n_estimators= 240



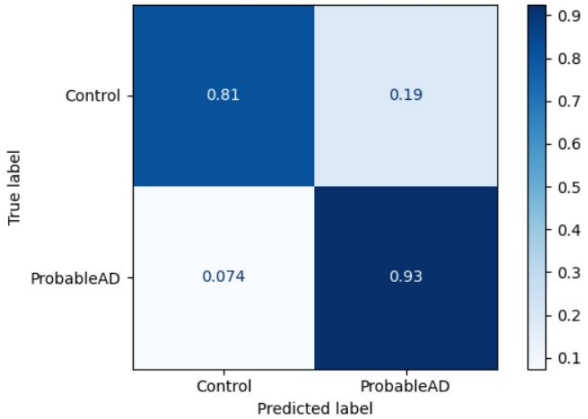
Classifiers - Logistic Regression

We tuned the regularisation strength using GridSearchCV, and obtained the following tuned hyperparameters and confusion matrices:

Multi-class:
C = 2.0



Binary:
C = 5.0



Classifiers - Neural Networks

Multi-class classification architecture

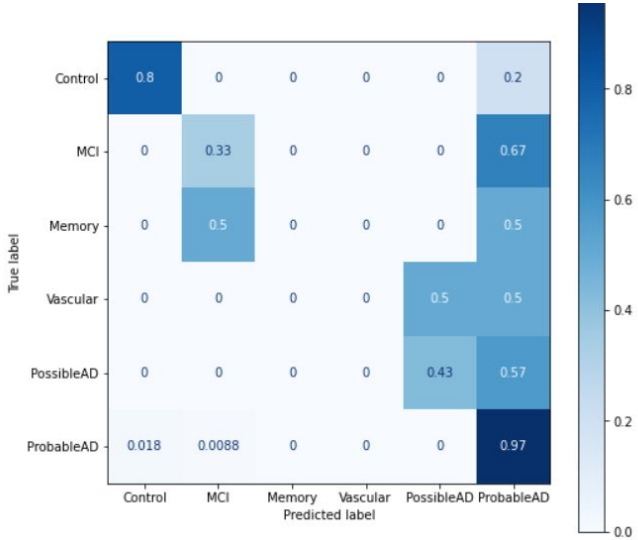
```
model = Sequential()
model.add(Dense(units = 64, activation = 'relu', input_dim = 49))
model.add(BatchNormalization())

model.add(Dense(units = 128, activation = 'relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(BatchNormalization())

model.add(Dense(units = 256, activation = 'relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(units = 256, activation = 'relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(units = 256, activation = 'relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(units = 128, activation = 'relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(BatchNormalization())

model.add(Dense(units = 64, activation = 'relu'))
model.add(BatchNormalization())

model.add(Dense(units = 6, activation = 'softmax'))
```

2-class classification architecture

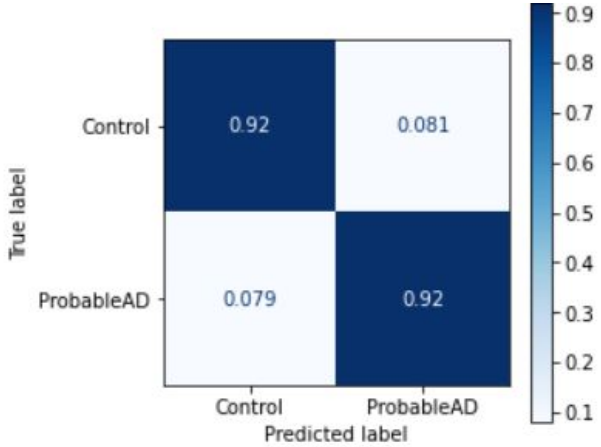
```
model = Sequential()
model.add(Dense(units = 64, activation = 'relu', input_dim = 49))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(units = 128, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Dense(units = 256, activation = 'relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(units = 256, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Dense(units = 256, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.4))

model.add(Dense(units = 128, activation = 'relu'))
model.add(Dropout(0.4))
model.add(BatchNormalization())

model.add(Dense(units = 64, activation = 'relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(units = 2, activation = 'softmax'))
```

The tuned hyperparameters and confusion matrices are:

Multi-class:
Training time - 40 epochs,
Optimizer - Adam,
Learning rate - 0.004 initially, decreased to 0.8 times its current value every 17 epochs.



Binary:
Training time - 60 epochs,
Optimizer - Adam.
Learning rate - 0.004 (constant)



Results

The following results were obtained on the test data (20% split of the dataset):

Classifier	Accuracy (All Groups)	Accuracy (Binary)
XGB Classifier	**74.40%**	90.54%
Support Vector Classifier	70.07%	90.04%
Decision Tree	68.11%	85.57%
K-Nearest Neighbours	68.63%	82.58%
Random Forest	70.47%	86.06%
Logical Regression	72.16%	90.67%
Neural Network	71.73%	**92.05%**

Result Analysis

- When it comes to binary classification, neural networks outperformed every other model with a test accuracy of 92.05 %

- While training for all classes, XGB classifier gave the best performance with an accuracy of 74.40 %.

- From the confusion matrices for multi class classification, it can be seen that some classes were not predicted at all for the test data. A possible explanation for this is the presence of very few such examples in the dataset.

- Another notable observation from the results is that MCI was very frequently being classified as ProbableAD, which is suggestive of the fact that MCI cannot be reasonably distinguished from Alzheimer's Disease using only speech data.

  This observation can also be interpreted as MCI being an early stage of Alzheimer's disease.

Conclusion

- In this project, we primarily worked on detecting Alzheimer's disease using speech analysis. By automating the task of predicting whether someone has Alzheimer’s or not using ML and DL based models, we made an attempt to mitigate the difficulties that people affected by Alzheimer’s commonly face.

- We achieved a maximum accuracy of 92.05 % and 74.40 % while classifying the disease into 2 and 6 classes respectively.

- The models fail to perform well when there are a small number of samples in some groups. Development of artificial data (data augmentation) might be a possible solution via GANs if enough base data is acquired.

- We plan to build up from work we have done here and achieve even better results in the future by trying out some more advanced architectures like attention based LSTMs and GRUs.

Acknowledgement

We would like to express our deep and sincere gratitude towards Prof. Veeky Baths for providing the opportunity to work under his supervision and guidance in this project, which is a niche topic in the field of Cognitive Neuroscience and Medical Sciences. As an experienced individual in this field, he was able to provide great insight about the research opportunities in the field which served as a great motivating factor.

We would also like to thank DementiaBank for being generous and responsive in providing access to their valuable datasets for research purposes.

Finally, a great thanks to those part of this team. The project could not have been possible without them in the limited time frame, with continuous support in analysing the research papers and various machine learning and deep learning models.

References

1. Dataset - http://dementia.talkbank.org
2. Hassanali, Khairun-Nisa & Liu, Yang & Iglesias, Aquiles & Solorio, Thamar & Dollaghan, Christine. (2013). Automatic generation of the index of productive syntax for child language transcripts. Behavior research methods. 10.3758/s13428-013-0354-x.
3. CLAN Guide - https://labs.wsu.edu/vandam/documents/2017/01/329.pdf/
4. Fraser, Kathleen & Meltzer, Jed & Rudzicz, Frank. (2015). Linguistic Features Identify Alzheimer's Disease in Narrative Speech. Journal of Alzheimer's disease : JAD. 49. 10.3233/JAD-150520.
5. Chien, YW., Hong, SY., Cheah, WT. et al. An Automatic Assessment System for Alzheimer’s Disease Based on Speech Using Feature Sequence Generator and Recurrent Neural Network. Sci Rep 9, 19597 (2019). https://doi.org/10.1038/s41598-019-56020-x
6. Chen, Jun & Zhu, Ji & Ye, Jieping. (2019). An Attention-Based Hybrid Network for Automatic Detection of Alzheimer’s Disease from Narrative Speech. 4085-4089. 10.21437/Interspeech.2019-2872.
7. Lin, Liu & Zhao, Shenghui & Chen, Haibao & Wang, Aiguo. (2019). A New Machine Learning Method for Identifying Alzheimer's Disease. Simulation Modelling Practice and Theory. 99. 102023. 10.1016/j.simpat.2019.102023.
8. https://www.mayoclinic.org/diseases-conditions/alzheimers-disease/symptoms-causes/syc-20350447