ECE5970 Progress Report (Oct. 25, 2018)

Team member: Jiahao Li (jl3838)*, Quan Sun (qs84), Xiaowen Wang (xw453)

Dataset: The Alzheimer's Disease Prediction of Longitudinal Evolution (TADPOLE)

In this progress report, we show our data cleansing procedures, the baseline model and results, and initial model and results of our machine learning method in development. We describe our theoretical design for the final form of our model.

Data cleansing

The goal of data cleansing is to fill missing data, encode categorical and quantitative data properly, and select important features for training. Reading data of TADPOLE_InputData.csv as Input_Data, we found there are thousands of features, so we pick several features recommending by TADPOLE website. The features we used to train models are “PTID_Key”, “EXAMDATE”, “DX_bl”, “DXCHANGE”, “AGE”, “CDRSB”, “ADAS11”, “RAVLT_immediate”, “Hippocampus”, “WholeBrain”, “Entorhinal”, “MidTemp”, 、“FDG”, “AV45”, “APOE4”, “ADAS13”, “Ventricles” and “MMSE”. We can find there are so many missing data in them and we need to fill them out according to the existing data. Although all visits are from different patients at different time point, we make a simple assumption that with the same diagnosis each feature should have the same distribution. Therefore, we can substitute all missing data with the mean of the corresponding feature under the same diagnosis. In this way the filling of missing data will not bias the distribution of features under the same diagnosis.

To get the proper distribution for each feature of each diagnosis (label), we need to encode the labels first. “DX_bl”, “DX” and “DXCHANGE” contain the diagnosis of each visit. There are “AD”, “CN”, “EMCI”, “LMCI” and “SMC” in “DX_bl” as different values for these features. we know “EMCI”, “LMCI” and “SMC” are different stage of MCI, so we can define these three values as “MCI”. Encoding “DX_bl” by the follow map, {"CN": 0, "EMCI": 1, "LMCI": 1, "SMC": 1, "AD": 2}, which corresponds to values of target data. Upon further investigation we find that the “DX” and “DXCHANGE” have same distribution by observing the data. “DXCHANGE” data is missing when “DX” data missing, or they have same values. So, we can drop the “DXCHANGE” data. Encoding “DX” by the follow map, {"NL": 1, "MCI": 2, "Dementia": 3, "NL to MCI": 4, "MCI to Dementia": 5, "NL to Dementia":6, "MCI to NL":7,"Dementia to MCI":8}. Then we fill the missing data in “DX” by the data of last visit since it is reasonable to assume the diagnosis remains the same. Since we only want to use the three standard label currently, we ignore the trasitional information and we change the “DX” data into the status of objectives. DX=1,7 means the status is NL; DX=2,4,8 means the status is MCI; DX=3,5,6 means the status is AD.

The missing data of features are all based “DX_bl” and “DX”. If the objectives have the same values of “DX_bl” and “DX”, we assume that they have the same distribution, so that we could use the mean to fill missing data. For “APOE4” data, we use the mode of objectives with same label. The missing data of rest features are filled by the average values of objectives with same status.

Elapsing time is the most critical feature of each visit. We need to convert the given dates of visit in string type to elapsed days from the first visit. Thus we add a new feature “deltaTime” which is the difference between first visiting time and subsequent visiting.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8717 entries, 0 to 8716
Data columns (total 19 columns):
PTID_Key            8715 non-null float64
EXAMDATE            8716 non-null datetime64[ns]
CDRSB               6289 non-null float64
ADAS11              6293 non-null float64
RAVLT_immediate     6242 non-null float64
Hippocampus         5360 non-null float64
WholeBrain          6298 non-null float64
Entorhinal          5304 non-null float64
MidTemp             5304 non-null float64
FDG                 2932 non-null float64
AV45                1170 non-null float64
APOE4               8704 non-null float64
AGE                 8716 non-null float64
DX_bl               8716 non-null object
DX                  6317 non-null object
DXCHANGE            6317 non-null float64
ADAS13              6230 non-null float64
Ventricles          6150 non-null float64
MMSE                6304 non-null float64
dtypes: datetime64[ns](1), float64(16), object(2)
memory usage: 1.3+ MB
```

```
In [21]:
1 df_data.isnull().sum()
2 # all missing data have been filled

Out[21]:
PTID_Key            0
EXAMDATE            0
CDRSB               0
ADAS11              0
RAVLT_immediate     0
Hippocampus         0
WholeBrain          0
Entorhinal          0
MidTemp             0
FDG                 0
AV45                0
APOE4               0
AGE                 0
DX_bl               0
DX                  0
ADAS13              0
Ventricles          0
MMSE                0
dtype: int64
```

This figure shows that after filling missing data, all important features are now complete.

Baseline Model

The baseline of this problem is simply copying the labels of last known visit to all the future visits that need to predict, the so-called carry-forward baseline. This is based on the assumption that all labels remain the same. This is obviously too naive. Therefore, any improvement made in our model should outperform this one. For this simple baseline model, we measure it by using cross-entropy and MSE on prediction, classification and regression respectively. The classification cross-entropy is 20.715, MSE(ADAS13) is 166.388, MSE(MMSE) is 15.295.

ML Algorithm Developing

Our ML algorithm for prediction of possible Alzheimer’s Disease progression in TADPOLE involves with status evolving with respect to time. We build our model based on the idea that for each individual patient, the clinical status depends on the probability space of status transition, whose distribution is estimated by previous clinical status as well as previous visit data. We call it an auto-regressive manifold learning. The below chart describes the basic pipeline.

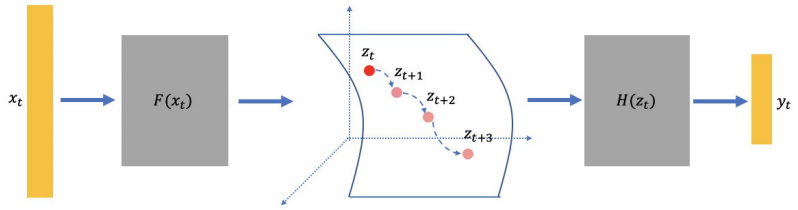

Figure 1 Auto-regressive manifold learning framework

For each patient i we are going to predict his or her future clinical status denoted by $\left\{y_{t+1}^{(i)}, \cdots\right\}$, we first take the previous visit data $\{x_t^{(i)}, y_t^{(i)}\}$ from the baseline visit $(t=0)$, to the nearest history visit t, using F mapping each $x_t^{(i)}$ to a high dimensional space where each $z_t^{(i)} \in Z$, a manifold representing the clinical status. Then an auto-regression is conducted as follows,

$$z_{t_n} = f\left(z_{t_{n-1}}, \Delta t_n\right) + \varepsilon$$

where $\Delta t_n = t_n - t_{n-1}$ .Therefore we can predict the next future visit based on trajectory on manifold from previous data. H maps the manifold point to the output space, which is $y_t^{(i)} = [CN_t^{(i)}, MCI_t^{(i)}, AD_t^{(i)}, ADAS13_t^{(i)}, MMSE_t^{(i)}]$.

The end-to-end F, H encoding-decoding model is trained using all previous visit data from training patient ID. For the manifold learning, the previous data as well as future labels from training patient ID are applied to find appropriate auto-regressive and time-variant model. Then the whole model is evaluated on validation set. In order to avoid overfitting, we use cross-validation to monitor learning process.

For now, we have implemented a simple way to find regression using the baseline status and time intervals, which implies an essential idea that every auto-regression will give a static point solution, furthermore a probability distribution remaining unchanged to time.

Current Machine Learning Model

The current prediction model we implemented is very simple but still performs better than baseline. We assume given the data of first visit and time elapsed, there exist a function maps into the desired output: the 3-status categorical diagnosis (encoded in 0, 1, and 2), and quantitative "ADAS13", "Ventricles" and "MMSE". Therefore the input data we choose is the value of "DX_bl" and "deltaTime". We construct four simple neural networks to represent the "black box" function mentioned above, three regression models for quantitative labels ("ADAS13", "Ventricles_Norm" and "MMSE", respectively) and one classification model for "DX_bl" the categorical label. TensorFlow is used to construct all models currently.

i) Regression Model

Design three hidden layers and mean square error as loss function. The initializer of parameter "W" is variance_scaling_initializer.

```
Iter 0, training loss 63413.480469, validation loss 366.518433
Iter 1000, training loss 150.212921, validation loss 95.446442
Iter 2000, training loss 66.912537, validation loss 98.254021
Iter 3000, training loss 26764.464844, validation loss 95.171768
Iter 4000, training loss 50.136803, validation loss 93.804649
Iter 5000, training loss 132.285202, validation loss 94.710777
Iter 6000, training loss 114.990517, validation loss 96.888649
Iter 7000, training loss 53.917847, validation loss 97.335876
Iter 8000, training loss 57.670719, validation loss 97.218834
Iter 9000, training loss 85.005432, validation loss 96.364342
Iter 10000, training loss 74.340950, validation loss 94.363518
```

```
Iter 0, training loss 194564.656250, validation loss 782.423157
Iter 1000, training loss 40.913841, validation loss 10.493280
Iter 2000, training loss 52.233036, validation loss 9.937047
Iter 3000, training loss 19.922905, validation loss 10.107354
Iter 4000, training loss 32.897987, validation loss 9.632438
Iter 5000, training loss 20.333055, validation loss 10.227160
Iter 6000, training loss 20.098204, validation loss 9.852644
Iter 7000, training loss 22.532272, validation loss 10.330798
Iter 8000, training loss 13.341170, validation loss 10.619847
Iter 9000, training loss 14.584060, validation loss 9.885215
Iter 10000, training loss 19.309853, validation loss 10.093781
```

The validation loss of ADAS13 and MMSE are respectively around 90 and 10.

ii) Classification model

Design three hidden layers and cross entropy as loss function. Before training transferring the data into one-hot code. The initializer of parameter "W" is variance_scaling_initializer. This model approximately got the validation accuracy of 70% as shown below.

```
In [12]:
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

for iter in range(epochs):
#     x_batch,y_batch = random_batch(input_data,output_data1,batch_size)
    sess.run(train_step, feed_dict={X:X_train, y_gt:y_cls})
    if iter%1000 == 0:
        train_loss = sess.run(loss, feed_dict={X:X_train, y_gt:y_cls})
        train_acc = sess.run(accuracy, feed_dict={X:X_train, y_gt:y_cls})
        print("Iter %d, training loss %f, training accuracy %f" % (iter, train_loss, train_acc))

        val_loss = sess.run(loss,feed_dict={X:X_validation, y_gt:y_cls_val})
        val_acc = sess.run(accuracy,feed_dict={X:X_validation, y_gt:y_cls_val})
        print('Iter %d, validation loss %f, validation accuracy %f' % (iter, val_loss, val_acc))
        print("-"*200)
```

```
Iter 25000, validation loss 1.737614, validation accuracy 0.687428
----------------------------------------------------------------------------------------------------------------------------------------
Iter 26000, training loss 0.457565, training accuracy 0.816791
Iter 26000, validation loss 1.737809, validation accuracy 0.692042
----------------------------------------------------------------------------------------------------------------------------------------
Iter 27000, training loss 0.453840, training accuracy 0.816475
Iter 27000, validation loss 1.782802, validation accuracy 0.686275
----------------------------------------------------------------------------------------------------------------------------------------
Iter 28000, training loss 0.458801, training accuracy 0.816581
Iter 28000, validation loss 1.835217, validation accuracy 0.683968
----------------------------------------------------------------------------------------------------------------------------------------
Iter 29000, training loss 0.451268, training accuracy 0.817846
Iter 29000, validation loss 1.885507, validation accuracy 0.701269
----------------------------------------------------------------------------------------------------------------------------------------
```

Future Improvement

For next step, we are going to refine our auto-regressive model using more sophisticated method, specifically LSTM model to learn information flow within time series data, where the cell status could match well with our idea from previous data to refer a transition probability.

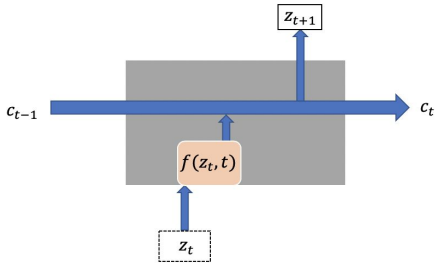

Figure 2 LSTM for auto-regressive model

An alternative model we are considering as a comparison is enlightened by personalized Gaussian Processes from last year's TADPOLE Challenge. However, the former one presumably fixes the time interval to 6 months. We are going to estimate a time-variant Gaussian Process, and put the time freedom into the kernel, which may be a good adaptation in this particular problem and may change fixed time series into a continuous domain.