

ECE 5970 Machine Learning with Biomedical Data

Final Report

Team Member:

Xinyu Wang(xw474)

Chengjie Lin(cl2445)

Deepak Agarwal(da475)

Nov 26, 2017

**1. Introduction**

Machine learning techniques has been widely used in medical applications such as predictive model for health status prediction using the electronic medical records(EMRs). In this article, we participated in the the Alzheimer's Disease Prediction Of Longitudinal Evolution (TADPOLE) Challenge and developed different predictive models to predict the future health status of the patients, using the medical records sampled from patients’ visits of irregular frequency.

**1.1 Machine Learning Problem**

The machine learning problem is based on a typical clinical scenario where the patient data might or might not be available at all visits. The disease on which is the problem is based is Alzheimer’s Disease and dataset is derived from an ongoing data science challenge called TADPOLE which stands for “The Alzheimer's Disease Prediction Of Longitudinal Evolution”. The goal is make an accurate prediction of clinically relevant variables based on the historical data of the patient. The three variables to be predicted in the problem are MMSE test score, ADAS13 score and head-size normalized ventrical volume. After predicting these three continuous variables, we have to make the final prediction by categorizing the subject into three categories as healthy, mild cognitive impaired (MCI) and AD (diagnosed with AD’s disease).

**1.2 Data Description**

The data for this project is taken from a publicly available large-scale data collected on Alzheimer’s disease called ADNI and is available at adni.loni.usc.edu. The complete dataset is divided into three categories: training, test and validation. As is the case with a usual machine learning problem, we would be training the model using training dataset and evaluating the performance with the test dataset. The final submission is then based on the prediction of a validation dataset. Like a typical clinical scenario, the input data is not uniformly filled i.e. the data is not present for all visits. Also, for each subject, there are multiple visits at different timestamps and intervals of the those visits is not regular. Moreover, the number of visits for each subject is also not same. In a nutshell, the patient data has many irregularities which brings out the need of good data preprocessing techniques. Also given the property of the dataset, we have come up with enhanced versions of main prediction model i.e. LSTM to handle the longitudinal aspect of the dataset. This would be explained in more detail in subsequent sections.

**1.3 Model Overview**

Different machine learning algorithms are applied and compared to get different models and benchmarking result. Started with Support Vector Machine(SVM) algorithm as the first attempt, we then migrated to Long-short Term Memory(LSTM), considering the built-in time attribute of the medical records and the widespread use of LSTM in time-series prediction. However, the traditional LSTM is not customized to deal with the (i)irregular time intervals and (ii)various timesteps in datapoints, which are two main problems in our case since the time gaps of the visits are different and different patients visited different times. To solve irregularity of time problem, we proposed a data-modified LSTM model and a architecture-modified LSTM. The data-modified LSTM model eliminated the time irregularities problem by transforming the time difference between the input data and output result into a feature of input data. For example, for input with date of 1/21/2015 and output with date of 6/15/2015, the time difference △t is around 5 months and will be inserted into this input data as a new feature in the unit of month. In this way, the time difference seemed to be taken care of, however, it can be a suboptimal method because it simply uses the time difference as new feature, which lacks solid explanation and proof of concept from the theory point of view. The architecture-modified LSTM handled sequences of time irregularities with modified architecture. Many novel LSTM models have been developed for this purpose. Phased LSTM [1] has a time gate K(t) controlled by timestamp t, where the cell value C(t) and the hidden output ht can only be updated during an “open” phase, otherwise, the previous values are maintained. DeepCare [2] extended the forget gate is extended to be a function of irregular time gap between consecutive time steps, and introduced two new forgetting mechanisms: monotonic decay and full time-parameterization. The decay mimics the natural forgetting when learning a new concept in human. The parameterization accounts for more complex dynamics of different diseases over time. Here we adopted a very up-to-date model called Time-aware LSTM (T-LSTM) [3]. TLSTM has forget, input, output gates of the standard LSTM, but the memory cell is adjusted in a way that longer the elapsed time, smaller the effect of the previous memory to the current output. For this purpose, elapsed time is transformed into a weight using a time decay function. T-LSTM learns a neural network that performs a decomposition of the cell memory into short and long-term memories. The short-term memory is discounted by the decaying weight before combining it with the long-term counterpart. is subspace decomposition approach does not change the effect of the current input to the current output, but alters the effect of the previous memory on the current output [3].

To solve various timesteps in datapoints problem, we proposed the masking technique in input data. The results of these three models are compared along with baseline algorithm.

The rest of the paper is organized as follows: related literature review in Section 2, technical details of the data preprocessing and proposed models are explained in Section 3, experimental and benchmarking results are presented in Section 4, and the study is concluded in Section 5.

**2. Related Literature Review**

**Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences**

**DeepCare: A Deep Dynamic Memory Model for Predictive Medicine**

**Adaptive Computation Time for Recurrent Neural Networks**

### **Patient Subtyping via Time-Aware LSTM Networks**

**3. Methodology**

**3.1 General Preparations**

Before building the models, we made some general preparations that applied to all of the models. The general preparations includes coding environment setup, input data filtering, output data filtering and input and output data imputation. The data preprocessing here is very general, however, when it comes to the data-modified LSTM model and architecture-modified LSTM, the data will be further processed and modified accordingly.

**3.1.1 Coding Environment Setup**

MATLAB: Used for data preprocessing including filtering and imputation.

Python: Used for actual implementation of the machine learning codes. The libraries includes Keras with TensorFlow backend, sklearn, numpy, pandas and other supporting packages.

**3.1.2 Input Data Filtering**

I. Observing the input data, we find that some columns are missing many data, which in our opinion, should not be used for imputation. Since the result of imputation on an inconsistent and inaccurate data can worsen the overall quality of data, we decided to remove those columns using well-thought strategy. Using MATLAB, we first calculated the percentage of blanks or NaN present in a column and compared it to a threshold. If the percentage exceeds the threshold, we termed that column as not meaningful and invalidated it. After some empirical experiments, threshold was set to 70%.

II. We also noticed that for some columns, the data of that column is the same for all samples, which means all the samples have the same value for this feature. That is some redundant information we can get rid of as it might affect the performance of the model with respect to accuracy as well as computations.

Using Matlab, we filtered the columns which had single values, for example column TEMPQC\_UCSFFSL\_’ had ‘PASS’ values 99% of the time without any useful meaning.

III. The result: After above steps, we filtered around 1500 columns and total no of valid columns are around 337(exclude the PTID\_KEY and EXAMDATE), which will be our starting point for algorithm testing.

IV. If we utilize the PCA for the result columns, we will get around 200 columns. However, we think 300+ features are acceptable.. So for now, we are using **337** features.

|  |  |  |
| --- | --- | --- |
| Input\_Data.csv | Columns | Rows |
| Before processing | 1892 | 8715 |
| After processing | 338 | 8715 |

**3.1.3 Output Data Filtering**

For TargetData\_train.csv and TargetData-Test.csv, just like input\_data csv, we firstly ordered the rows by PTID\_KEY. Then we followed these two strategies: 1. Delete the sample which has NaN in all columns for all visits (example patient id : 1603) and 2. Delete the sample which has at least one column as NaN for all the visits of a patient (example patient id : 1418)

|  |  |  |
| --- | --- | --- |
| TargetData\_Train.csv | Columns | Rows |
| Before processing | 8 | 2506 |
| After processing | 8 | 1682 |

|  |  |  |
| --- | --- | --- |
| TargetData\_Test.csv | Columns | Rows |
| Before processing | 8 | 867 |
| After processing | 8 | 825 |

**3.1.4 Input and Output Data Imputation**

After we are done with the filtering, we can then impute the missing value in the processed input csv file. We tried 3 different imputation strategy() provided by scikit-learn and the linear interpolate imputation in MATLAB.

The first thing to do is to replace all the blanks in the csv file to NaN so the missing value will be NaN in the program. Then order the input samples by its PTID\_KEY so that it will be easier for us to impute since the missing value will be estimated using linear interpolation. For example, patient of PTID\_KEY 8, there are 5 results in the TargetData\_Train.csv. Within the 5 results, two of them are missing the MMSE value, we need then impute the missing 2 MMSE using linear interpolation based on the other 3 results.

**3.2 Baseline Algorithm**

The baseline algorithm is based on LOCF technique i.e. Last Output Carry Forward, which is persistence model. For a particular patient, the three continuous variables values are taken from the most recent recorded observation of the patient is based on a simple assumption that patient’s health has not changed after the last observed data. Ventricles\_norm is computed as "Ventricles" divided by "ICV"

We don’t need to compute cross-entropy since the output will have a 0 field, instead, we use MAUC for discrete values and MAE for 3 continuous variables.Derivation of MAUC : We explored the mAUC calculation and graph mentioned at the TADPOLE website <https://tadpole.grand-challenge.org/performance_metrics/> and derived a formula for the same:

* Suppose there are n points in class 1 and m points in class 2. We want to calculate which means rank points in class 1 and class 2 based on the likelihood of those points to be part of class 1
* Suppose there are k points (k<n) which are mis-classified by our model
* So in our ranking, we put (n-k) points first, then m points, and then k wrong points
* S(1,2) which is the sum of ranks of points in class 1
* Fro k != 0, the formula would be
* For k = 0, the formula would shrink down to A(1|2) = 1
* Similarly we calculate A(2|1) and then , repeat it for all pairs
* Then we apply the formula given in the above link to calculate final mAUC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MAUC | MAE ADAS | MAE MMSE | MAE Vent |
| Baseline train | 0.9335 | 4.834774 | 2.432971 | 0.023168 |
| Baseline test | 0.8807 | 9.916541 | 3.463178 | 0.026208 |

**3.3 Support Vector Machine**

**3.3.1 Support Vector Classification for Categorical Variable**

**3.3.2 Support Vector Regression for Continuous Variables**

To transform this time series prediction into a regular supervised learning., we ordered the input samples and output with patient id (PTID\_KEY) as before, then we took cross-product of these two csv based on the same PTID\_KEY. For example, if a patient has N input sample and M output, the input file after cross-product will have N\*M samples for this patient, and for each of the input sample, we added a feature T(time interval of the input and the output), which is the Date in output minus ExamDate in input with unit of month.

However, due to the limitation of time, we only used the last visit of a patient in the input to do the cross-product. For example, we only used the last record of patient of PTID\_KEY 8 in the input and M results of PTID\_KEY 8 in output, to get the input.

|  |  |  |
| --- | --- | --- |
| Input\_Data\_SVM.csv | Columns | Row |
| After processing | 338 | 1683 |

With the aforementioned data preprocessing, we actually transformed this time series prediction into a regular supervised learning. As indicated in project proposal, we will use the Support Vector Regression (SVR) to predict the 3 continuous variables and use Support Vector Classification(SVC) and predicted MMSE from previous SVR to predict the one categorical variables. Here for testing purpose, we only test the prediction of MMSE. All in all, after the processing of data, we can just run a simple SVR model using keras.

We tried different kernels with SVC model, and apart from the performance metrics required, we additionally added our own accuracy function to evaluate the performance. If it predicts correctly, add 1 to the count, and the accuracy will be count/total # of output.

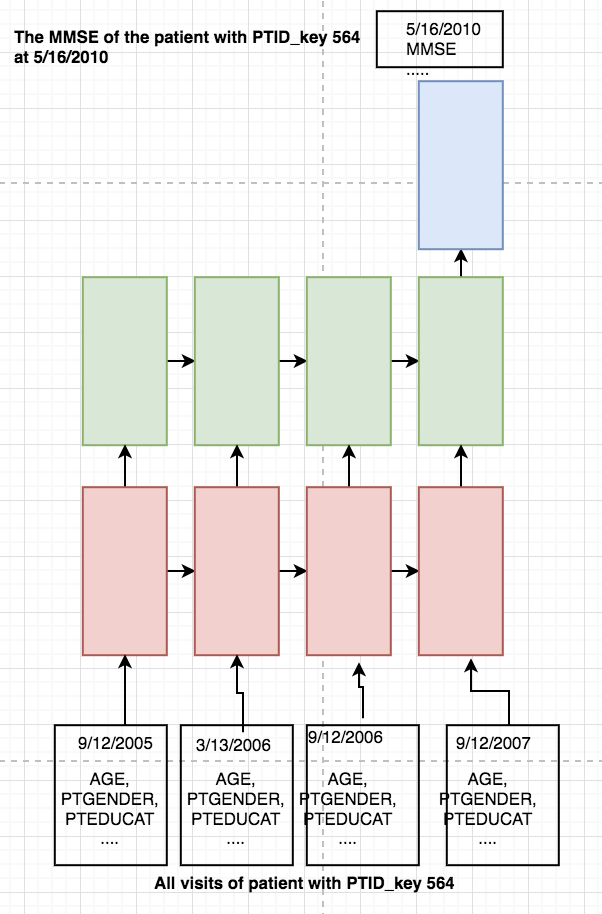
The SVM results show that SVR truly improve the accuracy of prediction of 3 continuous variables compared with baseline implement. However, the results still turned out to be poor and to be a little overfitting with train data since we haven’t modified the parameters. Consider it is a prediction problem with time series, we try to use RNN as final preliminary algorithm.

|  |  |  |  |
| --- | --- | --- | --- |
|  | MAE ADAS | MAE MMSE | MAE Vent |
| SVR Train | 1.126812 | 1.392880 | 0.004537 |
| SVR Test | 7.891946 | 3.743878 | 0.005972 |

**3.4 Data-modified LSTM**

The input data are all historical (pre cutoff date) and the objective is to predict these variable at some future data, we think we can use Long-short Term Memory(LSTM) for such time series prediction. This will be a many-to-one model since we will need to make predictions about multiple time steps in the future.

For a many-to-one LSTM model, the input is shaped into the [samples, timesteps, features] in the Keras context. The idea is that every patient correspond to a datapoint, the visits of the patient correspond to the timesteps in this datapoint. In this way, we can feed the medical records into LSTM network. However, our unorganized input data and output data have many problems that make it difficult to fit into traditional LSTM model. Unroll the LSTM and the problems are explained as below.



As can be seen above, the traditional LSTM network is typically use to handle a time series with irregular time interval between different timesteps, where the each timestep correspond to a visit. This is due to nature of the input data as well as the common sense in medical field, that a patient usually don’t visit institute and get their medical status recorded on a regularly basis. The timesteps in different data points are not consistent due to the different times of visit of the patients.

As indicated by the figure below, firstly we order the input\_data.csv by the PTID\_KEY as usually, then we find the patient with the most visits N as set the time step to be the N and use the Masking (mask\_value=-1)function to uniform the time steps even though the different patients have different visit times. Then for each output in the TargetData\_Train.csv, this time step and output makes a training pair. The batch\_input\_shape will be (batch\_size, time\_step, feature]) in the code. The We can use this model for the LSTM training.

**3.4.1 Data Preparation**

To fit the input data and output data into the LSTM model

**3.4.2 Data-modified LSTM model implementation**

**3.5 Architecture-modified LSTM**

**3.5.1 Data Preparation**

**3.5.2 Architecture--modified LSTM model implementation**

**4. Result and benchmark**

**5. Conclusion**