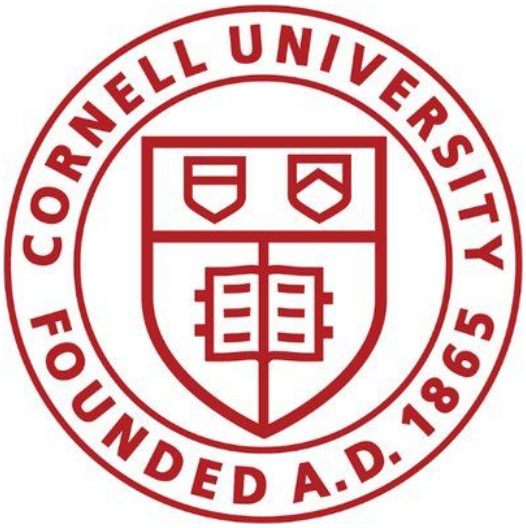


ECE 5970 Machine Learning with Biomedical Data

Final Report

Team Members:

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 4 predictive models, which Carry-forward baseline model, SVM, Data-modified LSTM and Architecture-modified LSTM, to predict the future health status of the patients, using the medical records sampled from patients' visits of irregular frequency. The performance of these 4 model are compared.

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 datapoint, which are two major 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 $\triangle t$ 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 such as Phased LSTM [1], DeepCare [2] and Adaptive

Computation Time(ACT) [4]. 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 effect 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 effect of the current input to the current output, but alters the effect of the previous memory on the current output [3].

To solve various timesteps in datapoint 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

Phased LSTM 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 [1].

DeepCare: A Deep Dynamic Memory Model for Predictive Medicine

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 [2].

Adaptive Computation Time for Recurrent Neural Networks

Adaptive Computation Time (ACT), an algorithm that allows recurrent neural networks to learn how many computational steps to take between receiving an input and emitting an output [4].

Patient Subtyping via Time-Aware LSTM Networks

T-LSTM is proposed to incorporate the elapsed time information into the standard LSTM architecture to be able to capture the temporal dynamics of sequential data with time irregularities [3].

3. Methodology

3.1 General Data Preprocessing

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 is used for data preprocessing including filtering and imputation.

Python is sed 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	TargetData_Train.csv	8	2506
	TargetData_Test.csv	8	867
After processing	TargetData_Train.csv	8	1682
	TargetData_Test.csv	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 strategies 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 $A(1|2)$ 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 $S(1,2) = 0.5 \times (n-k) \times (m+n+m+1+k)$
- For k = 0, the formula would shrink down to $A(1|2) = 1$
- Similarly we calculate A(2|1) and then $\widehat{A} = 0.5 \times (A(1|2) + A(2|1))$, repeat it for all pairs
- Then we apply the formula given in the above link to calculate final mAUC

3.3 Support Vector Machine

3.3.1 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.

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.

3.3.2 Support Vector Classification for Categorical Variable

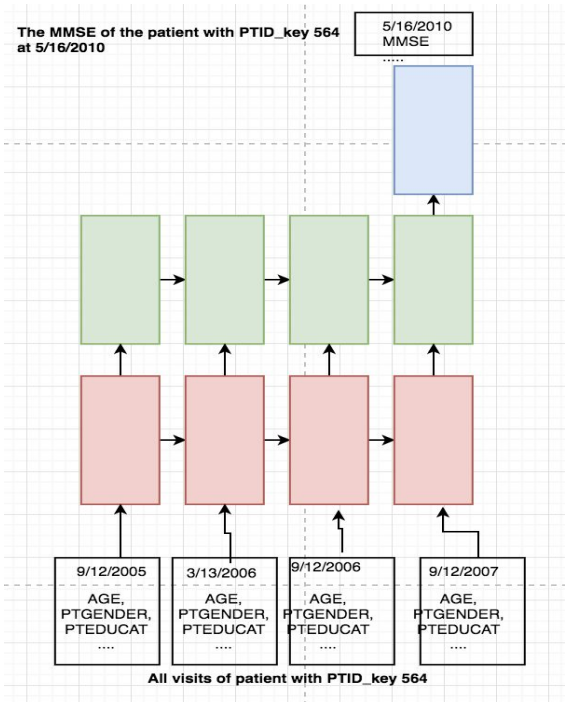
Once we get 3 continuous variables (ADAS13, Ventricles_Norm, MMSE) from Support Vector Regression from above section, we can use these 3 variables as 3 features, along with other 7 selected feature, to generate new input data to predict the categorical variable using Support Vector Classification. The other 7 features are as follows: DXCHANGE, AGE, PTGENDER, PTEDUCAT, PTETHCAT, PTRACCAT, PTMARRY,which remain unchanged with time. The specific implementation can be found in our Github.

3.4 Data-modified LSTM

3.4.1 LSTM dilemma

The input data are all historical (pre cutoff date) and the objective is to predict these variable at some future data, we plan to 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 network, and the problems are explained as below.



What we want:

Each patient stands for a datapoint. The total visits of a patients are the total timesteps of this datapoint. As shown on the left, for patient with PTID_key 564, there are 4 visits, which means 4 timesteps in this datapoint.

The problems:

P1. the time interval between two visit is different, which is against the uniform and regular sampling rate in the LSTM model. (time of irregularities problem)

P2. For a many-to-one model, the output should has the same timestamp as the last timestep, which should be the output variables at 9/12/2007. However, the output data given is the output at the future time, which is the health status variables of the patient in 5/16/2010.

P3. Different patients has different number of visits, which means that in the training process, the timesteps in each datapoint is different, which is unacceptable.(various timesteps in datapoints)

Figure 1 LSTM network unrolled

As can be seen above, the traditional LSTM network is typically not used 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 different patients.

3.4.2 Data modification as solution to LSTM dilemma

To solve the various timesteps in a datapoint problem (P3), we order the input_data.csv by the PTID_KEY as usually, then we find the patient with the most visits N, which is PTID_KEY 337 with 13 visits, and set the time step to be the 13 and use the Masking (mask_value=-1) function to uniform the time steps even though the different patients have different visit times. These rows with all -1 values will be ignored during the training session. 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.

To solve the time irregularities problem (P1) and solve (P2) in the meantime, we transformed the time difference between the input data and output result into a feature of input data. In Figure 1, we calculated the time difference between each timestep and the output result using their date attribute and in the unit of month, then inserted the time difference Δt as a new feature of the input data. So input with date 9/12/2005 gained one more feature Δt valuing 57 months. Input with date 3/13/2006 gained one more feature Δt valuing 50 months. The rest of input data can be done in the same manner. In this way, we eliminated the irregular time interval by simply replacing it with time difference, which is a quantifiable metric.

Lastly, we noted that there are multiple outputs for each patients. For example, if there are 3 output for patient with PTID_KEY 564, which are output A at 5/16/2010, output B at 3/17/2011, output C at 8/2/2012., we can generate 3 training pairs. In each training pair, we have different output, and different input since the Δt changes in the datapoint as the output date changes.

3.5 Architecture-modified LSTM

To solve the problems presented in 3.4.1, one way is to hack the input data as 3.4, which leads to data-modified LSTM, another way is to hack the architecture of LSTM, which can be implemented in multiple ways. Here, we adopted a relatively new architecture-modified LSTM, also called Time-aware LSTM.

3.5.1 Architecture modification as solution to LSTM dilemma of 3.4.1

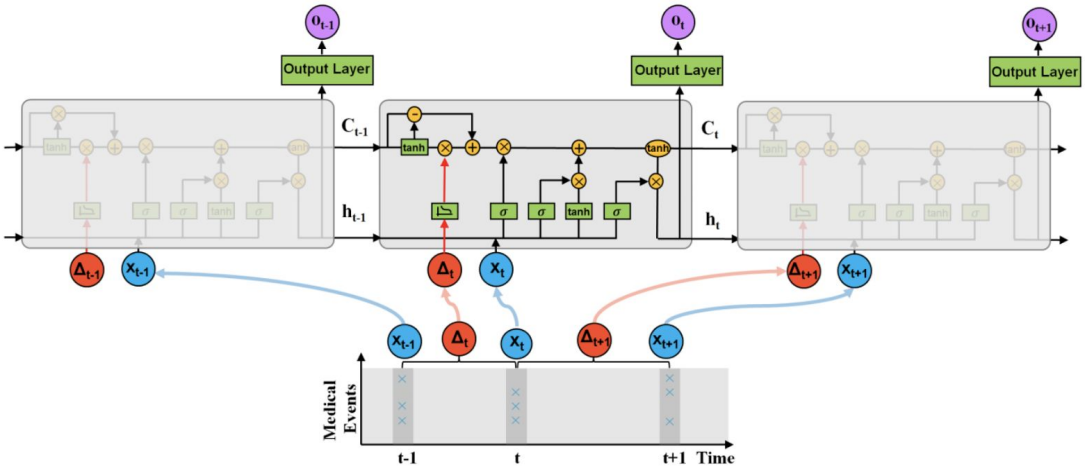


Figure 2: T-LSTM takes two inputs, input record and the elapsed time at the current time step. e time lapse between the records at time t −1, t and t +1 can vary from days to years in healthcare domain. T-LSTM decomposes the previous memory into long and short term components and utilizes the elapsed time (Δt) to discount the short term effects.[3]

$$C^S_{t-1} = \tanh\left(W_d C_{t-1} + b_d\right) \quad \text{(Short-term memory)}$$
$$\hat{C}^S_{t-1} = C^S_{t-1} * g\left(\Delta_t\right) \quad \text{(Discounted short-term memory)}$$
$$C^T_{t-1} = C_{t-1} - C^S_{t-1} \quad \text{(Long-term memory)}$$
$$C^*_{t-1} = C^T_{t-1} + \hat{C}^S_{t-1} \quad \text{(Adjusted previous memory)}$$
$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \quad \text{(Forget gate)}$$
$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \quad \text{(Input gate)}$$
$$o_t = \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \quad \text{(Output gate)}$$

T-LSTM applies the memory discount by employing the elapsed time between successive elements to weight the short-term memory content. So we transforms the time lapse into an appropriate weight. First, we extract the short-term memory component and adjust it by the elapsed time weight to obtain the discounted short-term memory . Then we add the adjusted short-term memory back with original long-term memory to get the new Ct-1

$$\tilde{C} = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad \text{(Canditate memory)}$$
$$C_t = f_t * C^*_{t-1} + i_t * \tilde{C} \quad \text{(Current memory)}$$
$$h_t = o * \tanh(C_t), \quad \text{(Current hidden state)}$$

Δt is the elapsed time between xt−1 and xt and g( ) is a function such that the larger the value of Δt , the smaller is short-memory component, and less the effect of the short-term memory

The key point of the T-LSTM architecture is the subspace decomposition into long-term memory and short-term memory applied on the memory of the previous time step. It is for sure we need to modify the previous memory, however we do not want to lose the global profile of the patient. So the long-term memory should not be discarded entirely, but the short-term memory should be adjusted proportional to Δt between t and $t-1$. If Δt is huge, it means there is no new information recorded for the patient for a long time. therefore, the short-term memory is less important to the prediction of the output.

In this way, we solve the time irregularities problem (P1) of the LSTM dilemma. However, to solve various timesteps problem (P3) and of the LSTM dilemma, instead of zero padding, same length sequences are put in the same batch. For example: L is the list containing all the batches with a length of N. L[0].shape gives [number of samples x sequence length x dimensionality]. [441*2*336] means there are 441 patients with 2 visits and 336 features. In this architecture-modified LSTM, (P2) is solved by inserting one extra timestep in the datapoint with the same date as the output result and all the features copied from last timestep. The way to handle the multiple output is same as the 3.4.2, where one output leads to one training pair.

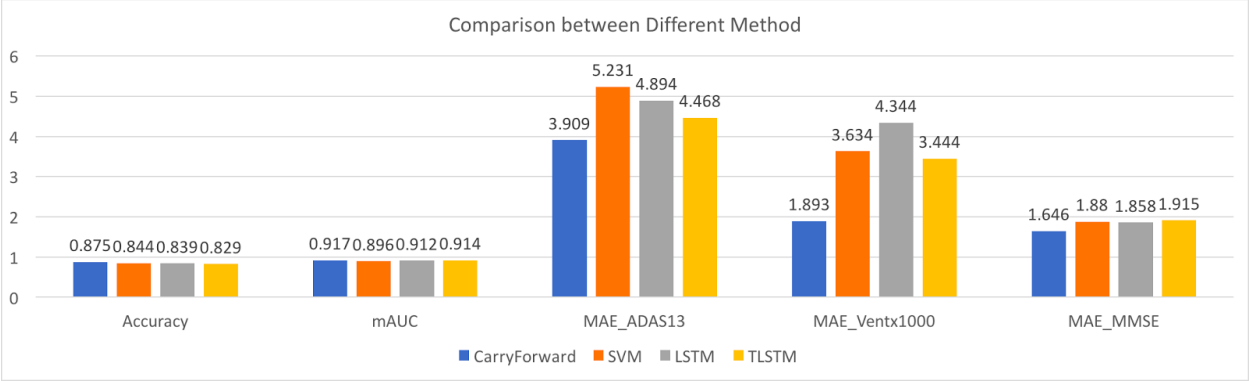
3.5.2 Architecture--modified LSTM conclusion

Instead of directly eliminating the irregular time interval problem during the data processing stage, the Time-aware LSTM modified the architecture of the gates thus doesn't corrupt the original input data and handled the time elapse naturally. The detailed coding can be found in our Github.

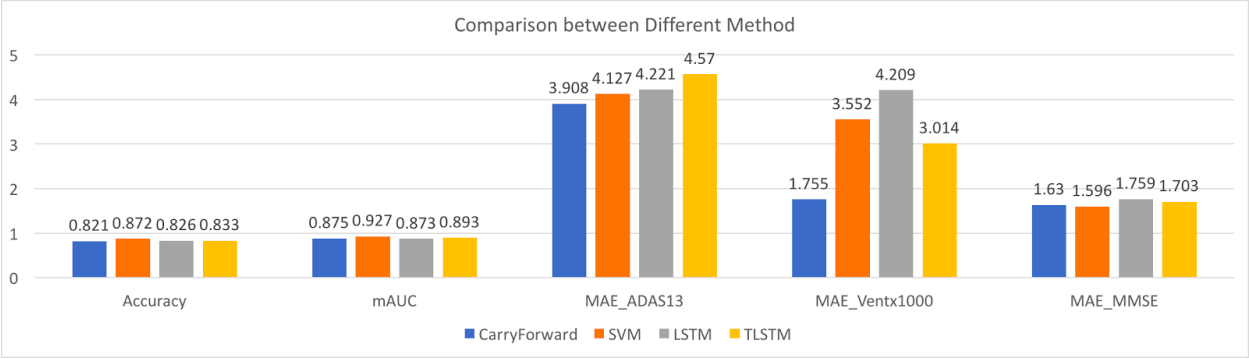
4. Result and benchmark

		Accuracy	mAUC	MAE_ADAS13	MAE_Vent	MAE_MMSE
Baseline	Train Testing Validation	89.4% 82.1% 87.5%	0.9329826 0.8750110 0.9174015	3.9413003 3.9088472 3.9090909	0.0015498 0.0017558 0.0018934	1.6186152 1.6306069 1.6467391
SVM	Train Testing Validation	91.0% 87.2% 84.4%	0.9466 0.9273 0.8969	3.3502392 4.1270754 5.2308611	0.0023526 0.0035526 0.0036349	1.4160017 1.5963041 1.8806234
Data-modified LSTM	Train Testing Validation	87.7% 82.6% 83.9%	0.9467 0.8734 0.9115	2.7966281 4.2218429 4.8935955	0.0021866 0.0042097 0.0043442	1.0751938 1.7599335 1.8582475
Time-aware LSTM	Train Testing Validation	89.6% 83.3% 82.9%	0.9398 0.8931 0.9139	3.5952806 4.5702834 4.4681689	0.0025809 0.0030140 0.0034441	1.4126004 1.7030040 1.9152606

More specific visualization of training, testing and validation result comparison can be found in Appendix.



We compared the prediction result of four different methods. For the three continuous variables, the evaluation metric is MAE while for classification problem, we use Accuracy and mAUC to evaluation. As a result, a good method needs to have lower Accuracy and mAUC and higher MAE. To make the difference between the four methods more obvious, we amplify the MAE of Vent 1000 times. In training process, the common problem is overfitted model. At first, the accuracy on train data is almost 99% but performs poorly on validation data, so we used cross-validation for SVM and add some dropout layers to LSTM to reduce the overfitting. From above Figure, the performance of CarryForward method is the best in classification and regression. The reason is that the data of most patients may keeps stable for a long time so in reality, the history data is the most important part for diagnose. The performance of other three methods are worse than carryforward method but close to it. The most important goal of this project is to predict the status of patients righly. If we consider the mAUC, our final implementation T-LSTM which transformed irregular time visits into uniform visits, gave the best performance among all the three learning models, with mAUC of 0.914. However, we also evaluate our model in test dataset. As shown in below Figure, the T-LSTM has the best performance in Accuracy and mAUC. We believe if the T-LSTM is fully exploited, it is promising that it can perform better.



5. Conclusion

The project development started with a simple persistence model which is based on LOCF (Last Output Carry Forward) strategy. This is a very good choice of a base model since it doesn't use any historical data and can be evaluated against other models which do male use of patient historical data. The models that we chose were promising and aligned well with the nature of the problems, and can be improved further to better solve this problem. We believe that the choice of our final model i.e. T-LSTM aligns with the nature of the problem very well and its performance can definitely be improved. Given the time constraints, we couldn't work more on further enhancement but we certainly believe that T-LSTM can prove to be a very reliable model for such kind of dataset. Each member's contribution can be found in the Appendix B.

Reference:

[1] Daniel Neil. Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. *arXiv:1610.09513*. Oct 2016

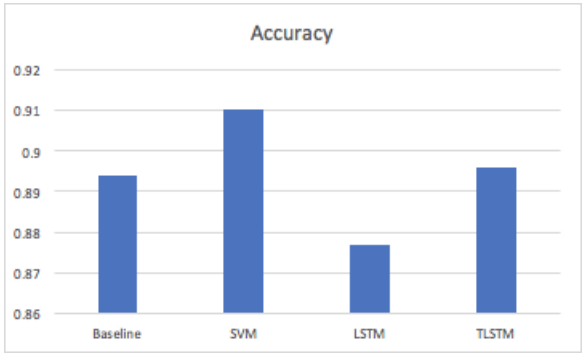
[2] Trang Pham, Truyen Tran, Dinh Phung and Svetha Venkatesh. DeepCare: A Deep Dynamic Memory Model for Predictive Medicine. *arXiv:1602.00357*. November 10, 2017

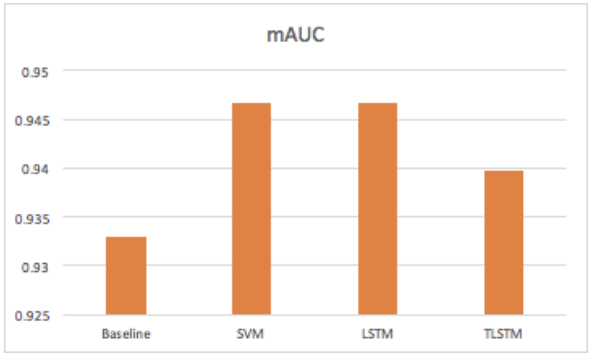
[3] Inci Baytas. Patient Subtyping via Time-Aware LSTM Networks. *KDD 2017 Research Paper*. August 2017. Retrieved from:
http://www.kdd.org/kdd2017/papers/view/patient-subtyping-via-time-aware-lstm-networks

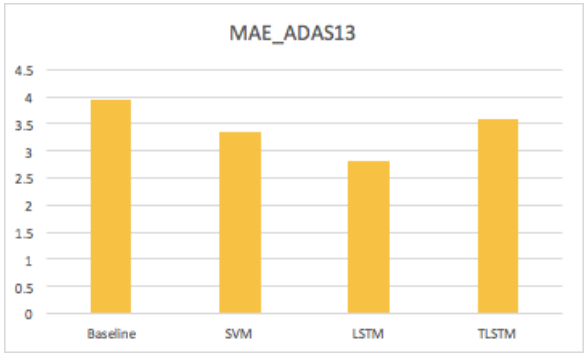
[4] Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv:1603.08983*. February 2017

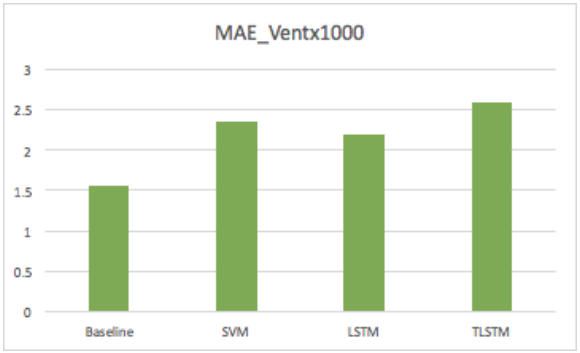
Appendix A

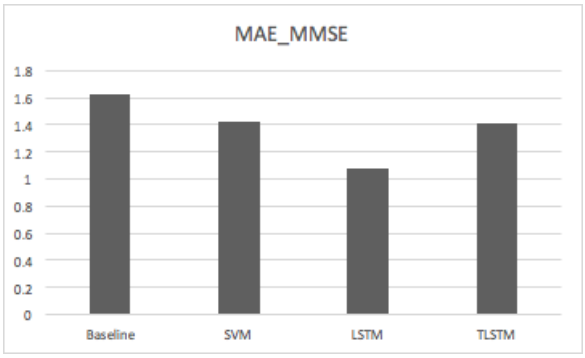
1. Training



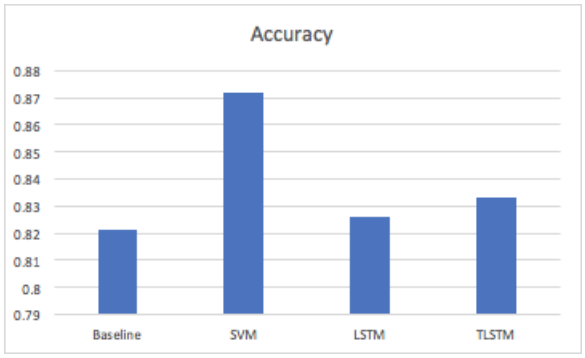


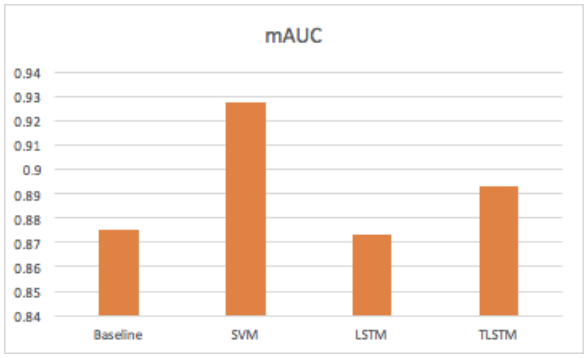


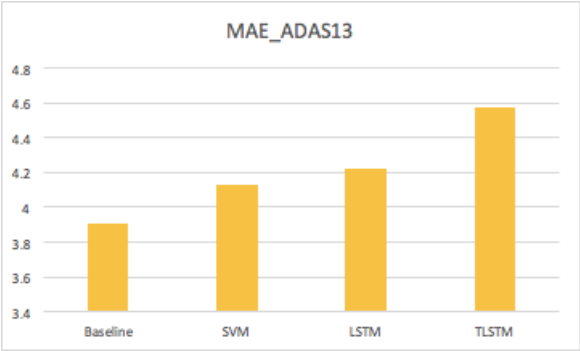


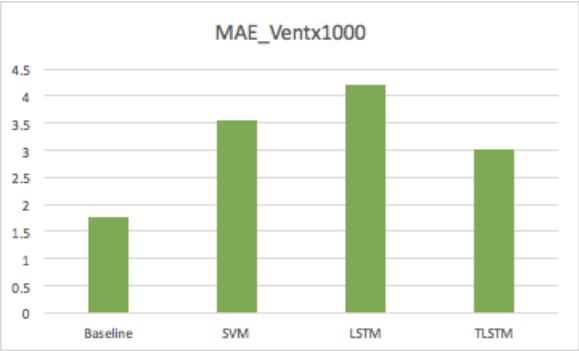


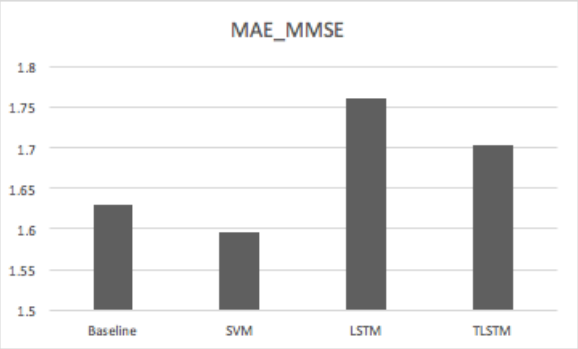
2. Testing



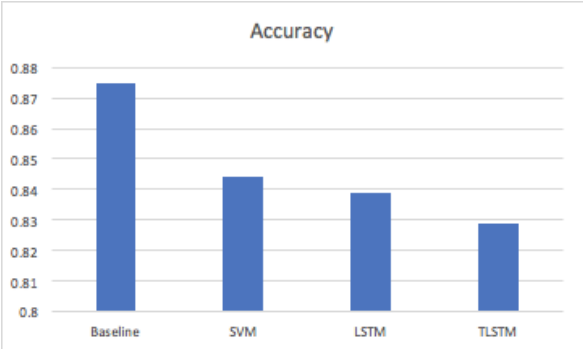


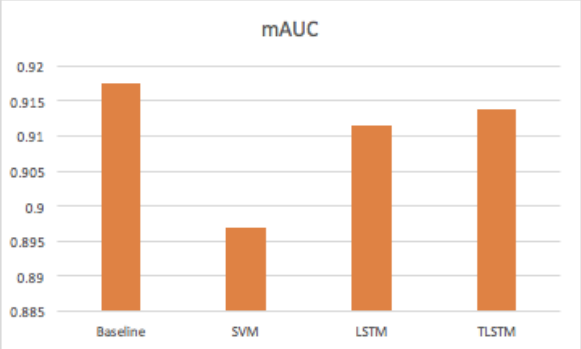


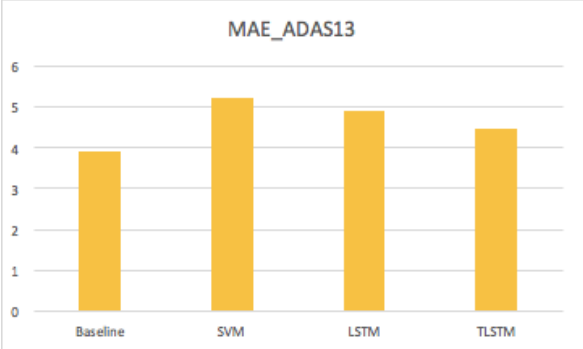


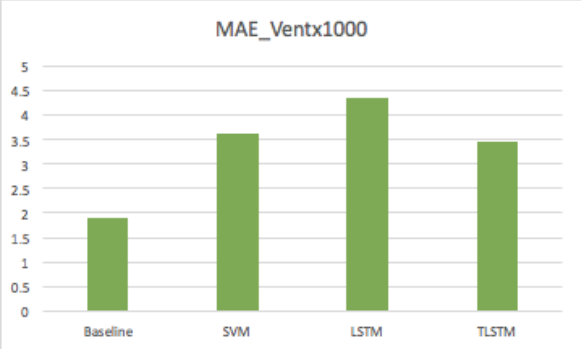


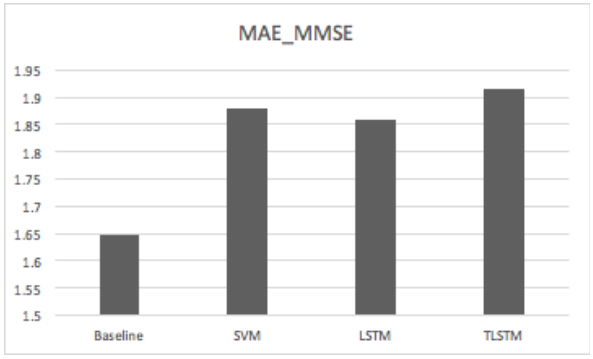
3. Validation











4. Comparison

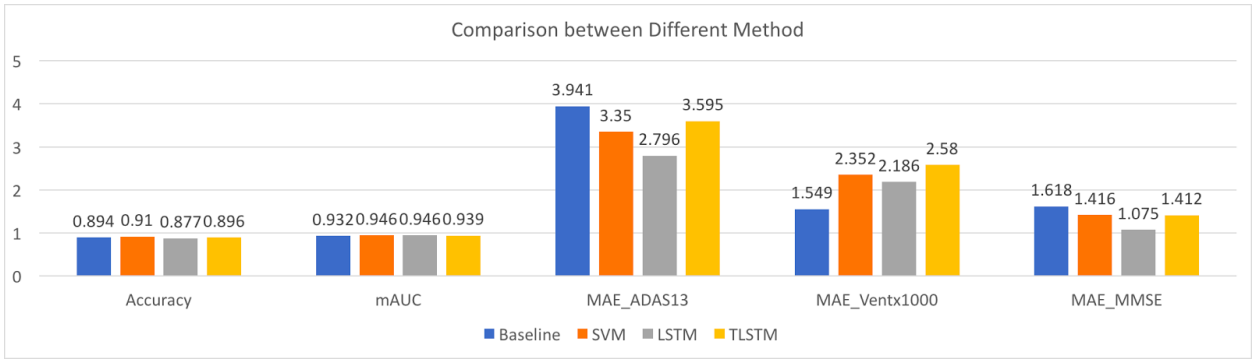


Figure 1. Training Data Result

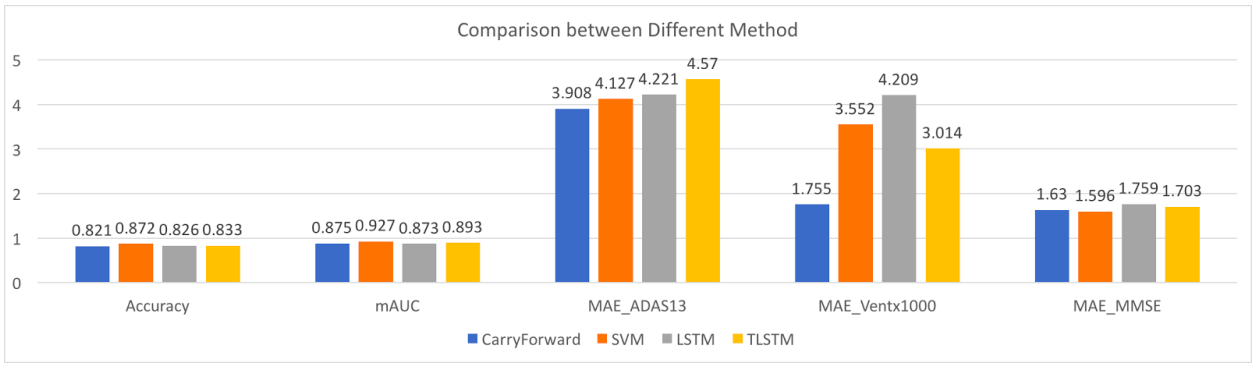


Figure 2. Testing Data Result

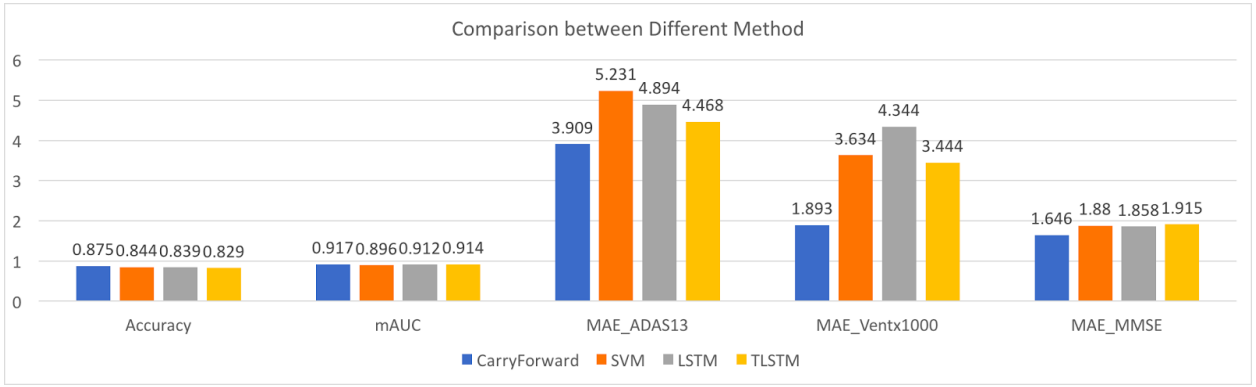


Figure 3. Validation Data Result

Appendix B Individual Contribution

Xinyu Wang (xw474):

Main programmer: as the main programmer, firstly, Xinyu worked on analyzing and implementing code for mAUC with Deepak. Second, he preprocessed datasets like feature select, missing data imputation. Also for each model, Xinyu modify the data format so that datasets can be used for specific models and achieved 4 models implementation. Third, Xinyu brainstormed with Chengjie to come up with the T-LSTM model and optimized the models. He put sincere and huge effort into this project and also did a great job in documenting and reports.

Chengjie Lin (cl2445):

Project manager: as the project manager, Chengjie first proposed the main framework and overall aspects of this project, which goes from carry-forward, SVM to Data-modified and Architecture-modified. He mainly focus on literature review and benchmark different architecture-modified LSTM models including Phased LSTM, DeepCare and AC and finally settled on T-LSTM model with Xingyu. He is also in charge of report writing and documenting. He also helped Xingyu process the data, implement and debug the codes.

Deepak Agarwal (da475):

Project coordinator: As a project coordinator, Deepak was involved in the project as a helping hand to Xinyu and Chengjie. Derived mAUC calculations and simplified it further to optimise the computations. Worked on base model implementation with Xinyu while getting hands-on on scikit-learn python library. Assisted in final model evaluation, report writing and class presentation.