# Deadline Prediction using Ordinal Regression

Joshua Cook, Byoungwook Jang, Aditya Mahara

March 15, 2015

## 1 Background

StudentLife was a study conducted by Dartmouth College's computer science department that collected passive and automatic sensing data over a 10 week period [3]. The goal of this study was to assess students' mental health based on their behavior. The students' behaviors were determined by processing the collected data using machine learning algorithms. However, for this data set to be useful, additional studies need to be conducted that attempt to predict metrics that can be used by the school's leadership and faculty to make real-time adjustments throughout the term.

## 2 Scope and Goal

The goal of our study is to implement the artificial neural network that can accurately predict the number of deadlines from the set 0, 1, 2, or 3 based on the student's behavior on the previous day. This way, no compliance is required from students other than simply downloading an application. We first treated this as a classification problem proposing Naive Bayes and SVM methods. However, since the ordering of classes has a relationship to one another, we reformulated our objective as an ordinal regression problem.

## 3 DataSet

The dataset is collected from 48 undergraduate and graduate students at Dartmouth over the 10 week spring term (March 27, 2013 to June 5, 2013). Within the StudentLife sensor datasets there are ten different data fields such as physical activity, audio inferences, conversation inferences, bluetooth scan, light sensor, GPS, phone charge, phone lock, WiFi, and WiFi location [3]. All sensor data were available as csv files and were organized by participants. First, we imported these datasets in a meaningful way into MATLAB. The timestamps in the raw datasets were in Unix time stamp format so the time information we obtained had a resolution of 1 second. To process the information associated with these timestamps we wrote a code to convert the Unix timestamp into month-day-year within a period from March 27, 2013 to June 5, 2013 (i.e 71 days). We also wrote codes to extract example sets and feature sets using these datasets which is described in detail below.

To test our algorithm, we used a final set of training set consisting of 7000 examples, and a test set with 2600 examples. Some information on the examples and features along with

the data processing necessary to create these example sets and feature vectors are presented below:

# 4 Examples and Features

To create an example set we use information about deadlines per day for each student. The StudentLife dataset has information from 44 students for 71 days with deadline information. Since our algorithm is trying to predict the deadline for the next day, we will be able to use information from 44 students for 70 days as examples. Therefore initially we have a total example set of 3080 (44days $*$ 70 students). We refer to this set as 'Dataset I.' Then, we scanned over the examples, and duplicated examples for class 1, 2, and 3, so that there are same number of examples for each label. This increased the number of examples to 9600 examples, of which we chose 7000 examples as a training set, and the remaining examples as test set. In order to avoid any numerical errors related to NaNs, we investigated averaging methods in order to fill in the missing data (represented by NaN) that had incomplete feature vectors.

Once we took care of the missing data by the averaging values, our data set was normalized with respect to each column, and the final numerical values ranged from 0 to 1. The following subsections will provide descriptions for our features.

In addition to replicating examples, next we will explore modifying learning objective per label to create a better classification for labels with low occurrences.

## 4.1 Features

Features were extracted to represent daily activity using sensor information through the duration of the study for those specific 44 students for whom we have deadline information available. For our algorithm analysis we have used 8 feature sets available from the StudentLife Dataset.

To construct a feature vector, we used a simplistic way to capture information about frequency of occurrence of a certain classifier per sensor. Brief descriptions on what these features represent and how we extracted them are given below.

***Audio***
The raw data file for audio has two columns. First column has timestamp information and the second column was information on audio inference where audio inference is classified as 0, 1, 2, or 3 that represents silence, voice, noise, or unknown respectively. The audio classifier runs 24/7 with duty cycling. It makes audio inferences for 1 minute, then pauses for 3 minutes before restart. If the conversation classifier detects that there is a conversation going on, it will keep running until the conversation is finished. It generates one audio inference every 2 to 3 seconds [3].

***Physical Activity***
The raw data file for physical activity has two columns. First column has timestamp information and the second column was information on activity inference where activity inference is classified as 0, 1, 2, or 3 that represents stationary, walking, running, or unknown, respec-

tively. The activity classifier runs 24/7 with duty cycling. To avoid draining the battery, it makes activity inferences continuously for 1 minutes, then pauses for 3 minutes before restart collecting activity inferences again. It generates one activity inference every 2 to 3 seconds depending on Smartphone's accelerometer sampling rate [3].

***Conversation***
The raw data file for conversation has two columns. First column represents a timestamp where a conversation began and the second column is the timestamp when the conversation ends.

***GPS Location***

Features related to GPS were constructed using accuracy, latitude, and longitude measurements. Twenty-four features were constructed for each of these measurement categories corresponding to 24 different hours of one day. Accuracy features for each hour were constructed by taking the sum of accuracy measurements. Latitude and longitude features were made by taking the sum of the differences between measurements taken in any given hour.

***Dark***
Dark data files record when the phone was at a dark environment for a significantly long time ($\geq$1 hour). There are two fields in each data file: start and end timestamp [3].

***Phone Lock***
The phone lock data files record when the phone was locked for a significant long time ($\geq$ 1 hour). There are two fields in each data file: start and end timestamp [3].

***Phone Charge***
The phone charge data files record when the phone was plugged in and charging for a significantly long time ($\geq$1 hour). There are two fields in each data file: start and end timestamp [3].

***NaN***
As we mentioned before, we replaced NaNs with the average value of the features that it belongs to. As we wanted to retain the information of whether or not a given feature vector had NaN values before the replacement with the average value, we added an additional feature with the number of NaNs that the given example had.

***Normalization***
With the pre-processed data set, we performed a $max - min$ normalization, which led the feature values to range from 0 to 1.

### 4.2 Feature Implementation

In Fig. 1 we have a histogram representation of a feature vector for an example feature set (i.e. Audio). Using 'Audio' feature we compute the frequency of occurrence for silence, voice, and noise, for every hour per student. The histogram represents an example of a feature

vector for 1 day for 1 student as we can see which parts of the day he/she was mostly silent and which parts of the days were mostly in noisy environments or where he/she was talking. Using this technique, we extracted 72 features (24 hours x 3 labels) for audio data.

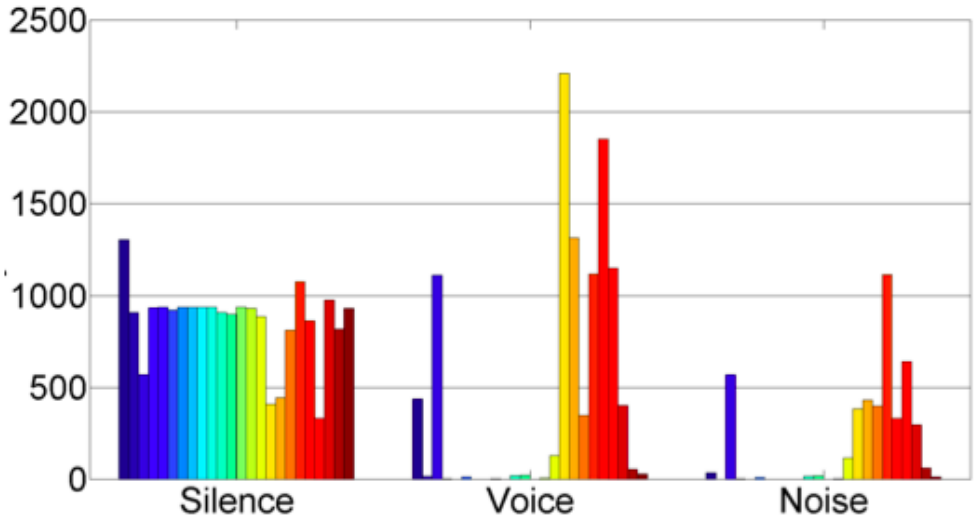


Figure 1: Feature vector profile for Audio Data

Similar techniques are used to extract 96 features (24 hours x 4labels) for physical activity and 6 features for conversation. Next steps during feature extraction will involve extracting information not based solely on frequency of occurrence, but using more elaborate information. Some of these features to be explored will be parameters such as duration of time between events, distance travelled per unit of time by a student, hours spent in the library, usage of the gym, and so on. All these information can be extracted from the StudentLife dataset that's available to us.

## 5 Implementation

Jianlin Cheng's paper, *A Neural Network Approach to Ordinal Regression*, implements the artificial neural network (ANN) to perform the ordinal regression task [1]. In order to implement the neural network, the algorithm modularized into two major parts: 1) forward propagation and backpropagation, and 2) batch gradient descent. The detailed implementation tutorial can be found from Andrew Ng's coursera course [2].

### 5.1 Notations

The following notations are going to be used in the cost functions.

$$(x^{(i)}, y^{(i)}) = \text{i-th training example} \tag{1}$$
$$L = \text{total number of layers in the neural network} \tag{2}$$
$$s_l = \text{number of nodes in layer l} \tag{3}$$
$$a_i^{(l)} = \text{activation of unit } i \text{ in layer } l \tag{4}$$
$$\theta_{ij}^{(l)} = \text{matrix of weights from j-th node in layer } l \text{ to i-th node in layer } l+1 \tag{5}$$

As mentioned in class, the cost function of the logistic regression is as follows

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta^2 \tag{6}$$

As we are using the logistic function for each activation node, we can sum rewrite the cost function for the neural network as follows

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} (\log h_\theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k)] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_l+1} (\theta_{ji}^{(l)})^2 \tag{7}$$

### 5.2 Algorithm

The steps for the neural network algorithm is as follows:

Given the training set $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(m)}, y^{(m)})\}$

- Initialize the $\theta$ matrix
- for $i = 1$ to $m$

  Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \cdots, L$
- Perform back propagation to compute the gradient of $J(\theta)$

With the gradient, we performed the bath gradient descent until we reached the stopping criteria. The stopping criteria is given as follows. Given $\delta_1$, $\delta_2$, and $\delta_3 > 0$, we need to satisfy the following three criteria to stop the gradient descent

$$||\theta - \theta_{new}|| < \delta_1 \tag{8}$$
$$||J(\theta) - J(\theta_{new})|| < \delta_2 \tag{9}$$
$$||\nabla J(\theta)|| < \delta_3 \tag{10}$$

## 6 Results

Using the Artifical Neural Network modified for the Ordinal Neural Network(ONN) case, firstly we use 100 training and testing examples with equal number of all deadline labels examples with a architecture of 1 layer and 10 nodes. Next we used 4000 training examples and 1696 testing examples to test two architecture: i. 1 Layer 10 nodes, ii. 2 Layers with 10 nodes each. For each arrangement we plot the error per example as a function of the lambda value we used. In all cases the training error was less than the testing error.

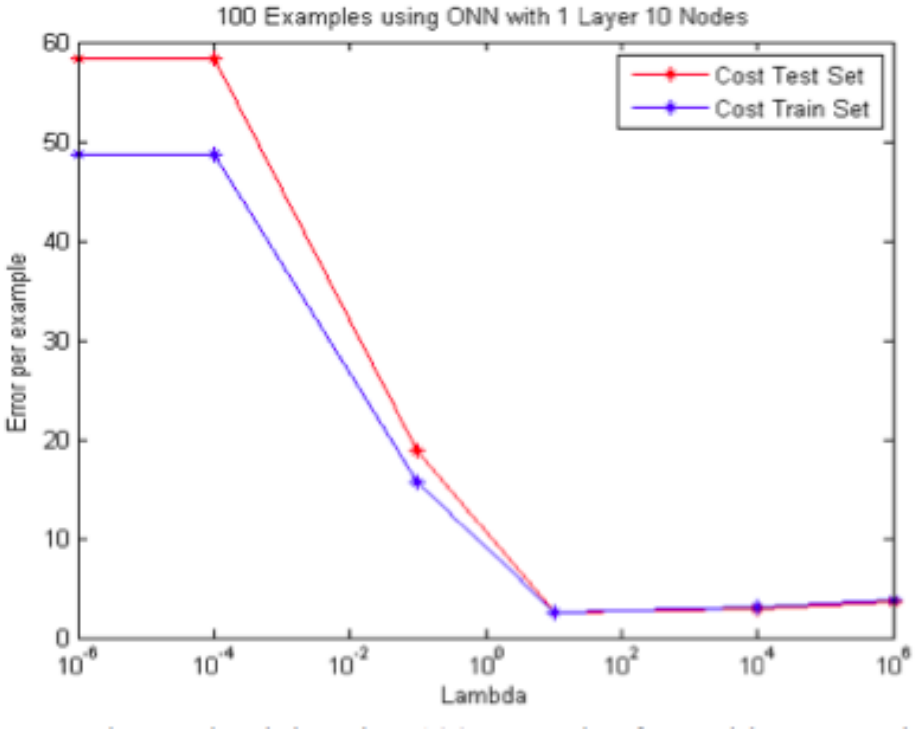


Figure 2: Error per examples vs. lambda using 100 examples for architecture using 1 layer and 10 nodes

As seen in Fig. 2. using 100 training and testing examples with architecture of 1 layer and 10 nodes we see that with increasing lambda, the error per example goes down. It seems like to lower values of lambda ($10^{-2}$ to 10) there is over fitting. Also, as seen in Fig. 3. when we use all examples and use the identical architecture, the error per example goes down; however the error seems to be scaled down. This happens since we use many more examples the average error per examples. In both cases we see over fitting for lower values of lambda. In both cases we do not see issues with under fitting.

When we use architecture with 2 layers with 10 nodes each, we get a error per example plot as shown in Fig.4. This doesn't make a lot of sense to us since there seems to be one value of lambda for which the error is maximized and there seems to be no issues caused by over fitting and underfitting. Further analysis for architectures with additional layers and nodes will need to be conducted before we can conclude anything from these preliminary results.

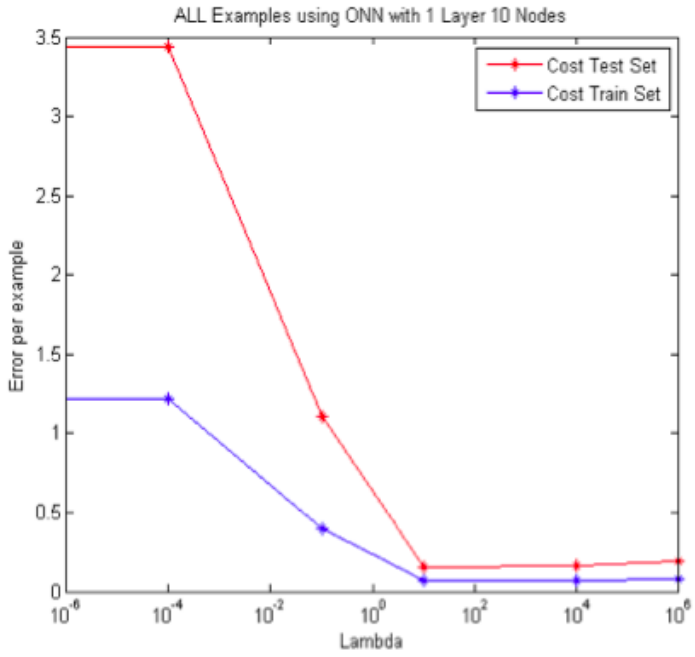


Figure 3: Error per examples vs. lambda using all examples for architecture using 1 layer and 10 nodes

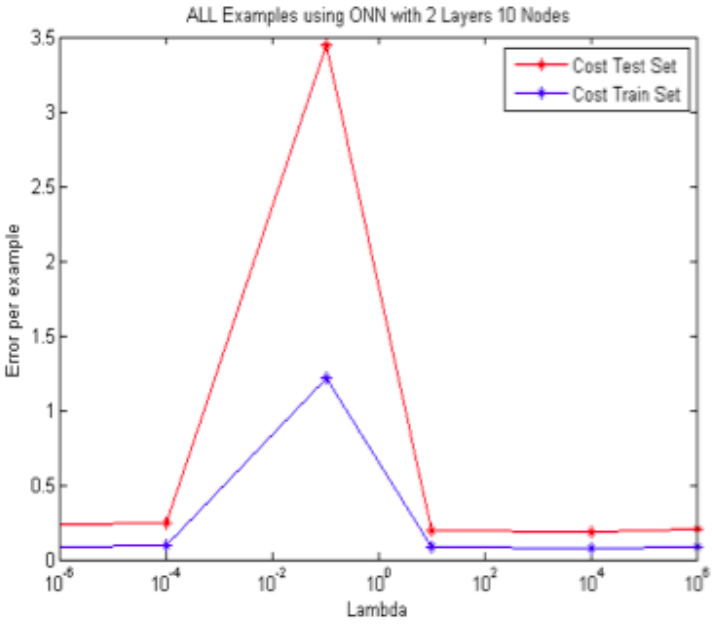


Figure 4: Error per examples vs. lambda using all examples for architecture using 2 layers and 10 nodes each

Some of the things we plan to do next are analyze the propagation of error as a function of system architecture (nodes/layers) to get a sense of which architecture might perform the best for this application. In addition to that we plan to analyze the error for each deadline

label separately to see how the non uniformity of distribution of examples (per label) is affecting the performance of our algorithm.

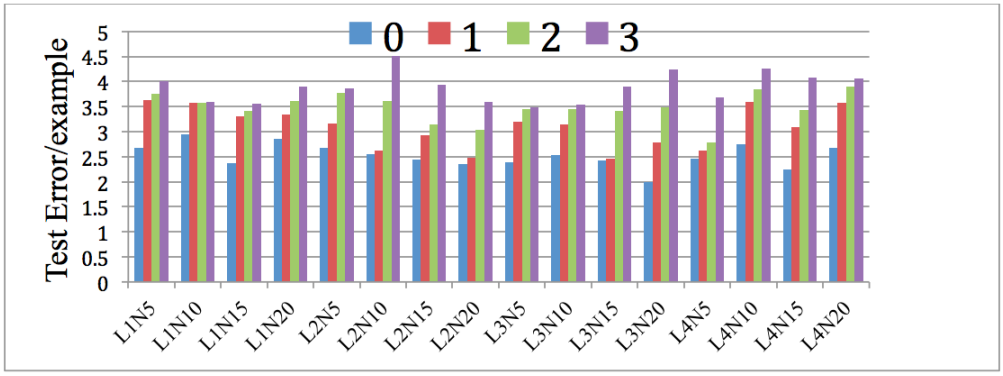


Figure 5: Test Error per label without replication

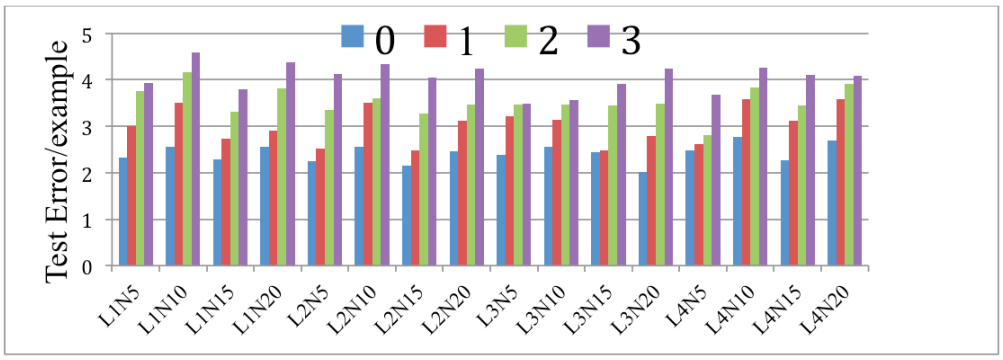


Figure 6: Test Error per label with replication

In order to visualize the test errors for each label, we implemented 16 different architectures, varying in the number of hidden layers and the number of nodes in each hidden layer. These values are plotted with replicated examples and without replicated examples as shown in Figure 5 and Figure 6. We chose our architecture to have two hidden layers, and plotted test and train errors for different number of nodes (Figure 7 - 10).

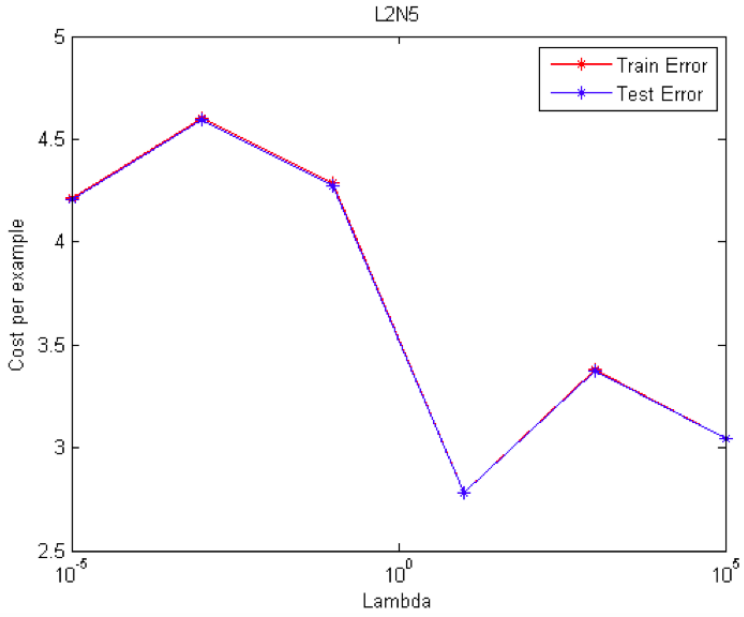


Figure 7: Test and Train error of the architecture with 2 hidden layers and 5 nodes

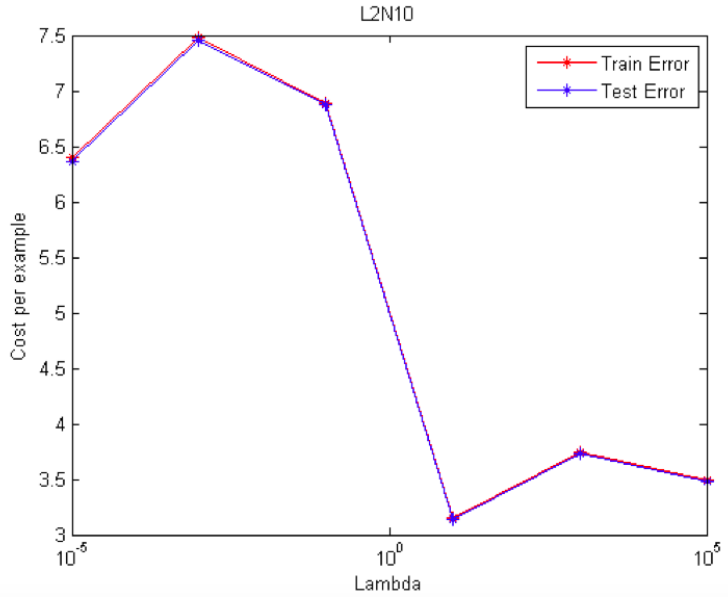


Figure 8: Test and Train error of the architecture with 2 hidden layers and 10 nodes

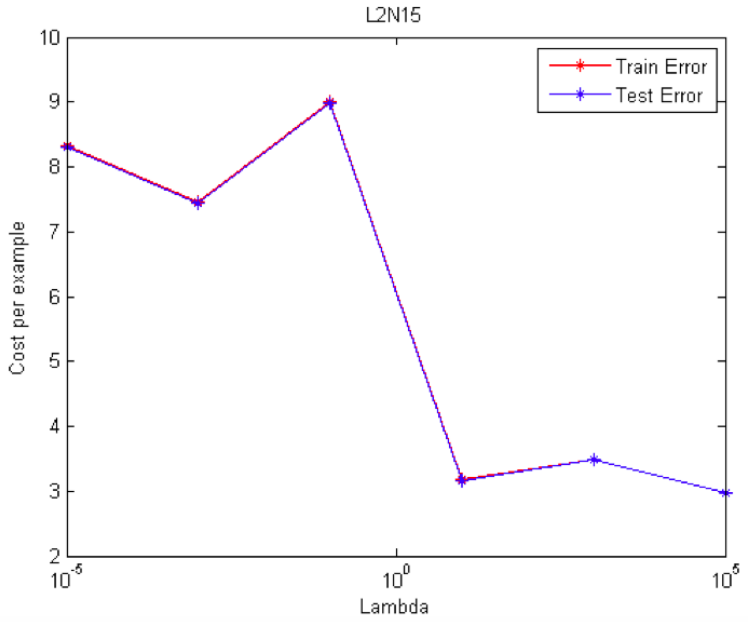


Figure 9: Test and Train error of the architecture with 2 hidden layers and 15 nodes

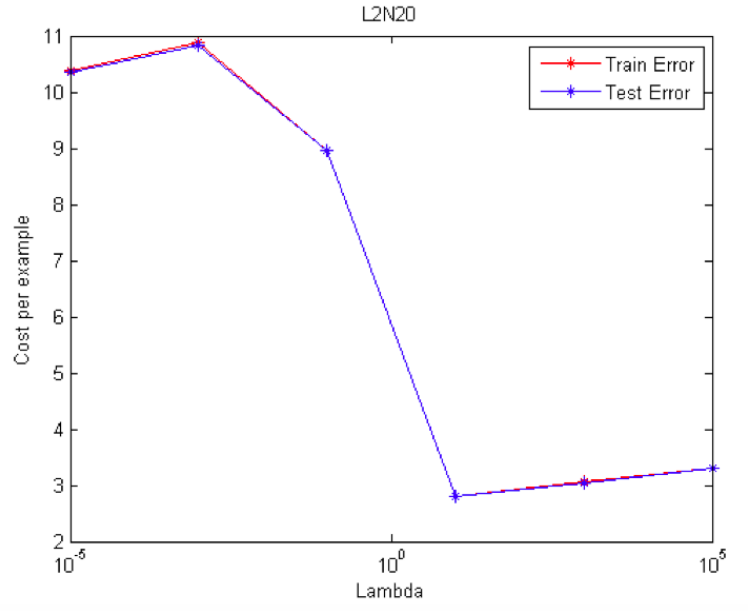


Figure 10: Test and Train error of the architecture with 2 hidden layers and 20 nodes

At the final presentation, it was suggested that there seems to be barely any difference between our train error and test error. This comes from the fact that we calculated these errors with the regularization term, which was overpowering the error calculation. Thus, the following figures show the train errors and test errors calculated without the regularization terms.

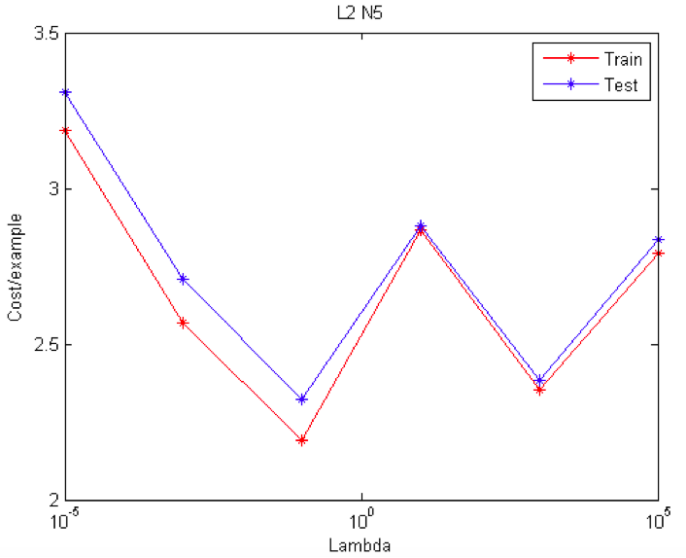


Figure 11: Test and Train error of the architecture with 2 hidden layers and 5 nodes without the regularization term

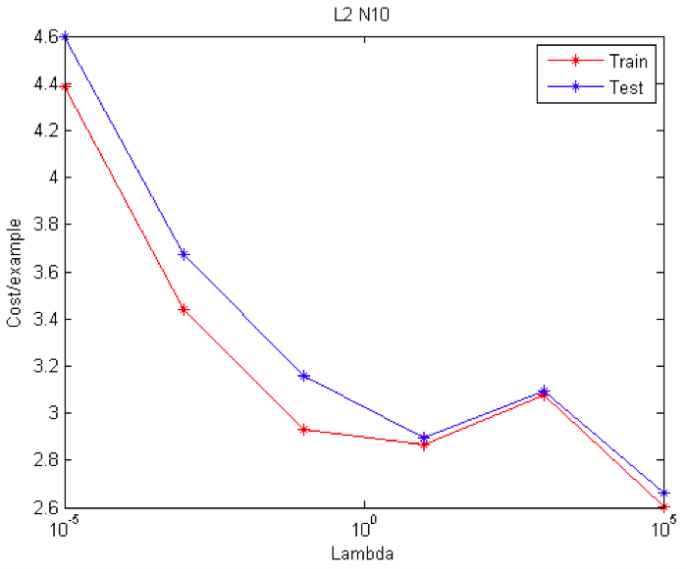


Figure 12: Test and Train error of the architecture with 2 hidden layers and 10 nodes without the regularization term

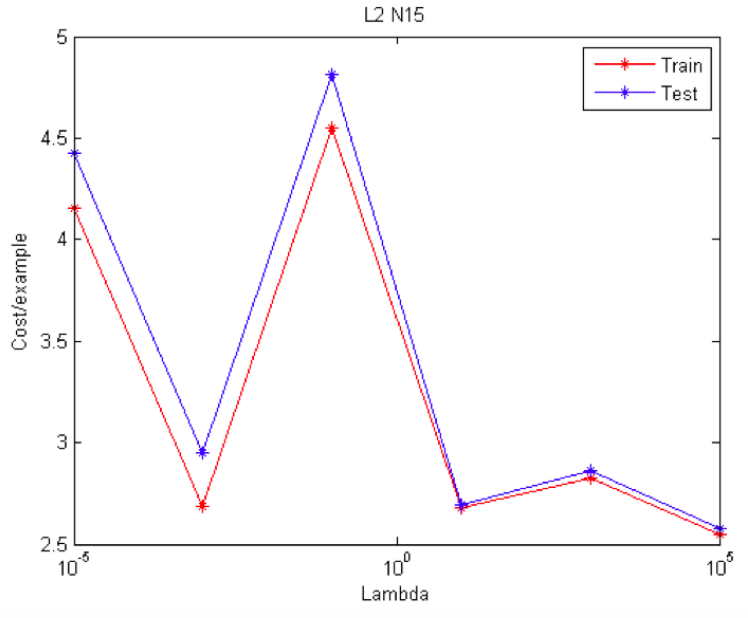


Figure 13: Test and Train error of the architecture with 2 hidden layers and 15 nodes without the regularization term

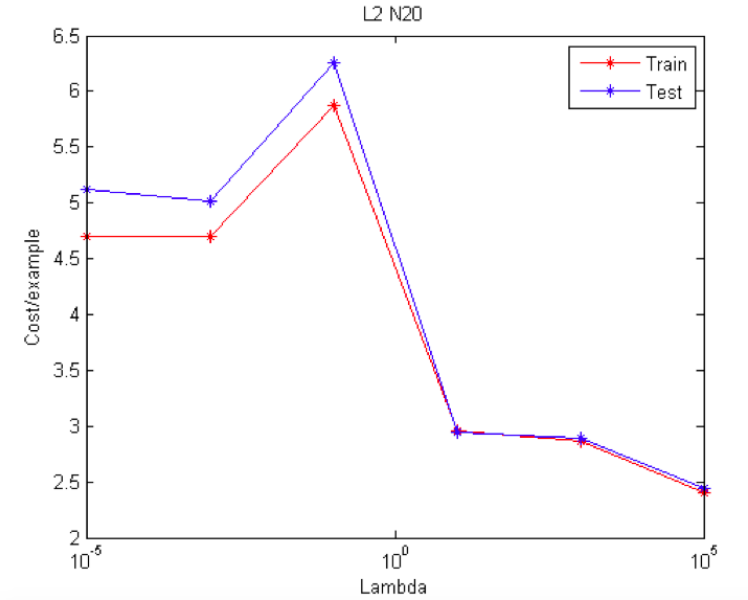


Figure 14: Test and Train error of the architecture with 2 hidden layers and 20 nodes without the regularization term

## 7 Conclusion and Discussion

The final poster and report includes plots of architectures that seemed to have the best results. The final report includes additional error plots that do not include the model pa-

rameters as part of the cost. It was expected that larger differences between training error and testing error would be seen once terms including model parameters in the cost function were removed but this was not the case. This was caused by the normalizing process used in the preprocessing of our data to construct features. Since the features were between 0 and 1, this caused our model parameters to be on orders of magnitude that were between 10-7 and 10-8. Furthermore, this normalization of all features to the same scale is also what probably caused us to converge to very poor local minima. If we were to retrain with the same features, using a very small value for the learning rate would likely yield results with lower error rates.

Another factor that had a huge impact on the algorithm performance was the sensor data used. The algorithm was built for regression to predict whether or not students had 0, 1 ,2 or 3 deadlines. There were large differences in the number of training examples that were available to us for each of these classes. In addition, many of the examples that we did have did not have complete feature vectors. As discussed previously, examples were replicated and some averaging methods were used to try and create a data set with an equal number of examples for each label and fill in empty features. However, even though these methods helped patch up some of the issues with the original data set, it made a lot of training and testing examples too similar to make large distinctions between training and testing error rates. If more time was available, it may be possible that using N-fold cross validation would give better results than using the hold out validation results shown in the figures.

Furthermore, if this algorithm were to be used for targeting advertising it may be better to treat this problem with a binary classification approach. These would alleviate any need to replicate examples since the number of examples labeled 0 would be equal to the number of examples for categories 1,2 and 3 combined.

### 7.1 Implementation Details

In order to translate the timestamps in our data, which were in Unix time, we imported 'unixtime.m' MATLAB function online, which translates the unix time stamps to regular calendar date and time. Furthermore, in order to read in the cvs files, we also adopted 'mfcvsread.m' from online to read in the cvs file to MATLAB. The portions of the code that was implemented by the group is provided in the submission on Dartmouth Canvas.

## References


[1] Jianlin Cheng, Zheng Wang, and Gianluca Pollastri. A neural network approach to ordinal regression. *Neural Networks, 2008. IJCNN 2008*, pages 1279–1284, 2008.

[2] Andrew Ng. Coursera - machine learning. *https://www.coursera.org/course/ml*.

[3] Fanglin Chen Zhenyu Chen Tianxing Li Gabriella Harari Stefanie Tignor Xia Zhou Dror Ben-Zeev Wang, Rui and Andrew T. Campbell. Studentlife: Assessing mental health, academic performance and behavioral trends of college students using smartphones. *In Proceedings of the ACM Conference on Ubiquitous Computing*, 2014.