Project Milestone

Determining successful restaurants using Yelp Data

Team: Matt Ritter, Naho Kitade, Jason Feng

February 14, 2015

1. Project Introduction

The restaurant industry has an incredibly high turnover rate. Many new restaurants do not survive their first few years in business. In addition, the restaurant business is flooded with various factors that impact success (location, timing, atmosphere, food quality, etc), such that it can be extremely difficult to create a successful restaurant. We hope to analyze Yelp data to determine if there are certain factors that are most important for a successful restaurant. This study may be extended to ultimately look at restaurants in different cities individually, since geography plays an overwhelming role in food culture.

2. Data

We are using Yelp's academic dataset for our study. Yelp releases a large dataset for competitions and other academic uses. The page from which we requested the data can be found here.

This data contains information on 61184 businesses, 21,892 of which are classified as restaurants. The data is clustered around a few geographic areas. This will allow us to potentially compare our results across different locations, since food culture can vary wildly depending on location.

Since not all restaurants have the all features present, we have analyzed which features appear often enough to be used. On the next page is the breakdown of the features we will be using, the number of restaurants which have that feature, and the different values for each feature. 12,924 restaurants have all of the features combined. We are using some of this data (70 - 80%) as our training data and will hold out another portion of the data, randomly selected, (10 - 20%) as test data.

The success (label 1) or failure (label 0) is determined using the following function:

$$SuccessScore(stars, review\ count) = 0.7 \times stars + 0.3 \times review\ count$$

with the cut off of success being if the above success score is greater than 18.5.
We have 4881 restaurants that are classified as successful, and 8043 restaurants that are classified as failures.

Feature Table

Table of used features, number of examples in our data with those features, and the values that each feature can take:

Restaurant Attributes	# of example with attribute	Values that each attribute can take
Take-out	19769	[True: 1, False: 0]
Good For: dessert	18757	[True: 1, False: 0]
Good For: latenight	18815	[True: 1, False: 0]
Good For: lunch	18815	[True: 1, False: 0]
Good For: dinner	18815	[True: 1, False: 0]
Good For: breakfast	18825	[True: 1, False: 0]
Good For: brunch	18765	[True: 1, False: 0]
Caters (omitted)	12818	[True: 1, False: 0]
Noise Level	16613	['quiet: 1', 'average: 2', 'loud: 3', 'very_loud: 4']
Ambience: romantic	16591	[True: 1, False: 0]
Ambience: intimate	16578	[True: 1, False: 0]
Ambience: tourist	16578	[True: 1, False: 0]
Ambience: hipster	16401	[True: 1, False: 0]
Ambience: divey	15727	[True: 1, False: 0]
Ambience: classy	16578	[True: 1, False: 0]
Ambience: trendy	16578	[True: 1, False: 0]
Ambience: upscale	16468	[True: 1, False: 0]
Ambience: casual	16578	[True: 1, False: 0]
Parking: garage	18683	[True: 1, False: 0]
Parking: street	18681	[True: 1, False: 0]
Parking: validated	18470	[True: 1, False: 0]
Parking: lot	18681	[True: 1, False: 0]
Parking: valet	18681	[True: 1, False: 0]
Has TV	17274	[True: 1, False: 0]
Outdoor Seating	19370	[True: 1, False: 0]
Attire	19824	['casual: 1', 'dressy: 2', 'formal: 3']
Alcohol	18007	['none: 1', 'beer_and_wine: 2', 'full_bar: 3']
Waiter Service	18404	[True: 1, False: 0]
Accepts Credit Cards	20413	[True: 1, False: 0]
Good for Kids	19643	[True: 1, False: 0]
Takes Reservations	19262	[True: 1, False: 0]
Delivery	19173	[True: 1, False: 0]
Good For Groups	19893	[True: 1, False: 0]
Price Range	20430	Scale: [1, 2, 3, 4]
Combined	12924	

As the "Values that each attribute can take" column suggests, we have divided up truly categorical attributes such as "Ambience" into a Boolean feature for each categories, and represent other spectrum based categorical attributes to a scale of 1 ~ 3 or 4.

3. Methodology

Overview
We have decided to use a feedforward Artificial Neural Network (ANN) model with multilayer perceptrons. We use stochastic gradient descent with a backpropagation training algorithm to optimize our least mean squares learning objective. We use sigmoid neurons rather than simple perceptrons in our model. Similar analyses of the restaurant industry have been conducted in past using ANN models as well. [3]

Backpropagation
Backpropagation is a method for computing the gradient of the error function with respect to the weights of each node in the neural network. The backpropagation algorithm is used to train a multi-layer feedforward network by determining how changing the weights and biases changes the behavior of the neural network. The goal of backpropagation is to adjust the weights of the neural networks to minimize the weights that are contributing to the greatest error, and maximizing the weights that are contributing to our desired output.

The main problem with the backpropagation algorithm with that there is no guaranteed method to reach the global minimum because there are various local minimums in the error function. Thus, it is important to average the test error over multiple runs to determine an accurate representative performance of a specific neural network architecture.

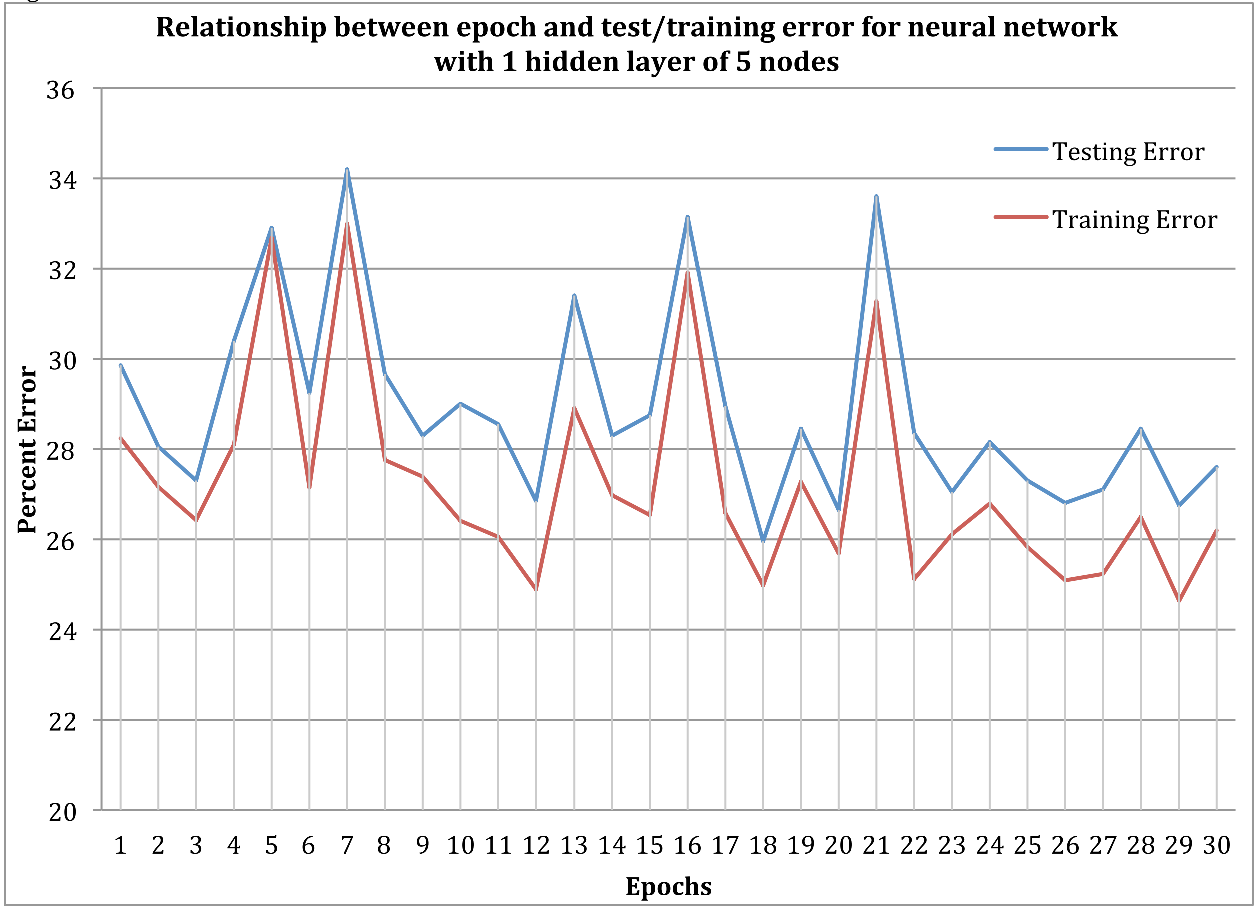
Model Selection: N-fold Cross Validation
Determining an optimum neural network architecture require significant human intervention, and there is not much more elegant of a procedure for model selection than taking a simple trial and error approach. Thus, we have implemented N-fold cross validation that would take in multiple different neural network architectures and output a csv file to visualize the performance of each model. The output contains the training and test error of the cross validation phase, as well as the training and test error averaged over multiple trials. This output enables us to easily compare the different models' performance taking into account issues of under/overfitting. An example output of the cross validation appears in the results section.

4. Initial Results

General run result

Here, you can see a graph displaying the improvement of a given neural network run over each epoch. We trained our neural network using 30 epochs, which is a number that was arbitrarily chosen to obtain initial results.

Figure 1

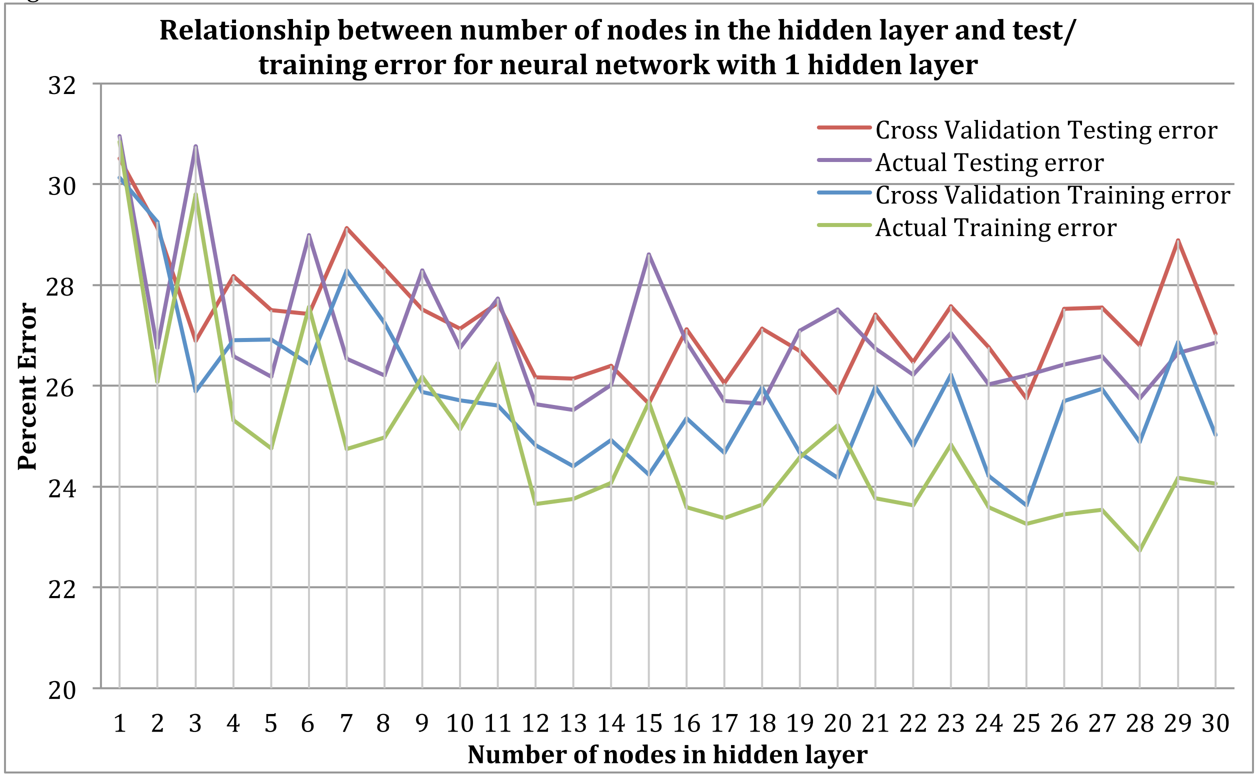


This graph shows how the test error and training error change similarly per epoch, with a downward trend. This is reassuring, since we can be confident that the network is actually learning after each epoch. The training error is consistently lower than the testing error, which is to be expected, but we can start to see some signs of overfitting from the errors of epochs 24 onwards since the difference between the training and testing error is starting to increase. This result suggests that after determining the optimal network architecture, we should try to determine the optimal epoch number.

10-fold cross-validation result

With the assumption that our data is not linearly separable, we have determined that our network should have one or more hidden layers. We started with one hidden layer, and the result of the cross-validation on different number of nodes in that hidden layer is shown below. We calculated the cross validation test error, cross validation training error, overall test error, and overall training error. We plotted these errors together to find trends of overfitting and underfitting (by comparing the training and test errors), and to check that our cross-validation is reliably predicting the final performance of the neural network (by comparing the cross-validation and overall errors).

Figure 2



From above, we can see signs of underfitting in the smaller number of nodes in the hidden layer (< 6), and signs of overfitting in the larger number of nodes in the hidden layer (> 25), which is expected behavior. The optimum number of layers for a network with one hidden layer seems to be in the 12 ~ 17 hidden units range, with the best model based on "Actual Testing error" results being 13 hidden units with error of 25.5%.

One frightening thing that we can tell from the above graph is that if we had picked our model based on the cross validation results, we would have chosen the model with 15 hidden units, which gave a 28.6% "Actual Testing error" – unluckily one of the higher errors out of all of the models. Since the performance of even one architecture of neural network varies over multiple runs, it may be the case that the actual training error calculated for the 15 hidden units was based on an unusually badly performing run, although this error is calculated by averaging the errors over three runs. This may suggest that we should calculate the "Actual Testing/Training error" over more runs to get a more representative performance measure.

5. Optimization and Next Steps

Completely determining the success rating function and cut-off
Although the score was constructed through intuition, this is an arbitrary cut off and function, so we should completely solidify what this function should be before moving on any further with model selection, now that we have implemented all of the tools needed for those steps. We may take into consideration that the current ratio between the two classes is unbalanced, so we may want to balance this when modifying the labels.

Increasing the number of hidden layers
Now that we have our cross validation implemented, the next logical step is to determine the optimum number of hidden layers and the correct number of nodes in each hidden layer through trial and error. Since we have experimented with having a single hidden layer, we will explore if adding more layers to network will have a positive effect on our test error. One of the challenges with this next step is that neural networks with more hidden layers require much more computational power and time, and the number of possible neural net architecture increases drastically. We must do some research to determine a relatively small set of candidate neural network structures and possibly optimize or parallelize the cross-validation implementation for efficiency.

Increasing data set
Since filtering out for restaurants with all features of interest shrinks our data size to around half to that of the original, we may attempt to include restaurants without specific features by marking their presence as an additional feature. We want to use as many examples as possible, but do not want to decrease the number of features that we use to train the network. Thus, we can add a dummy Boolean feature set to 1 if a given feature, say feature A is present or 0 if not present in the hopes that the network will learn that feature A is unreliable when its dummy feature is set to 0. A challenge here is that we cannot apply this procedure to all of the features, as there will be too many dummy features, which will lead to other problems. We may try to see which features, when included, decreases the overall example size the most, and decide which of those features are worth including with the presence variable to deal with this trade off.

Early Stopping
Looking at Figure 1, we see that the more epochs we run, we risk overfitting. Thus, we can employ the early stopping technique to find the optimal epoch to stop our training to maintain a reasonable generalization error, especially if we decide that the optimal network architecture is multilayered and complex. This regularization technique seems promising and is very easy to employ – we can simply modify our cross-validation implementation to be able to vary epoch size, and implement an additional stopping condition. We have found literature explaining specifically on how to best employ this technique, which we plan to follow. [1]

Disconnected from improving our network's performance, we may want to implement some benchmark method like logistic regression or support vector machines to compare the performance of our algorithm.

Sources:

1. Prechelt@ira.uka.de, Lutz Prechelt. Early Stopping - but When? (n.d.): n. pag. 1997. Web. <http://page.mi.fu-berlin.de/prechelt/Biblio/stop_tricks1997.pdf>.