Determining Restaurant Success or Failure

Jason Feng* Naho Kitade† Matthew Ritter‡

Computer Science Department, Dartmouth College §

Abstract

This paper describes the usage of an Artificial Neural Network to determine the success or failure of a potential restaurant. We analyze the Yelp Academic dataset [Yelp 2015] which includes all the data and reviews of the 250 closest businesses for 30 universities. Our approach is to utilize stochastic gradient descent and back propagation to built a model for determining success or failure of a restaurant given a set of business characteristics. We compare performance among a variety of neural network architectures and utilize dummy variables to increase the size of our usable dataset. Finally, we discuss performance of our best performing neural networks compared to our baseline models of logistic regression and support vector machines.

1 Introduction

The restaurant industry has an incredibly high turnover rate. Many new restaurants do not survive their first few years in business [Parsa et al. 2005] 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. Our goal is to construct a classification system for restaurants based on the Yelp data to determine given a set of restaurant attributes, whether or not that restaurant will be successful.

2 State of the Art

Currently in the commercial space, there exists a wide range of consumer facing applications for determining restaurant applications. Companies like Yelp, Foursquare, and Urbanspoon have created platforms that determine what restaurant you will enjoy based on your past history. In the academic space, there are a number of papers that discuss restaurant success [Youn and Gu 2006], [Kim and Upneja 2014] using various machine learning classifiers. In our research combine these two avenues by using commercial data to help business owners. Instead of using consumer data from Yelp to create a recommendation system, we help business owners determine the success or failure of a potential business enterprise.

3 Defining Success or Failure

Conceptually, we realized that a successful restaurant has a high Yelp rating and a high number of reviews. Ideally, a successful restaurant is both highly-rated and widely known. We came up with our success classification formula by using the median values for Yelp rating and number of reviews for restaurants. Ultimately, we decided that: **a successful restaurant has more than 37 reviews and at least 3.5 stars on Yelp**.

*jason.feng.17@dartmouth.edu

†naho.kitade.16@dartmouth.edu

‡matthew.p.ritter.15@dartmouth.edu

§Special thanks to Lorenzo Torresani

4 Data

4.1 Yelp Data

We are using the Yelp Academic Dataset for our study. Yelp releases a large dataset for competitions and other academic uses.

This data contains information on 61,184 businesses, 21,892 of which are classified as restaurants. The data is clustered around a few geographic areas. Areas included are Edinburgh, United Kingdom, Karlsruhe, Germany, Montreal and Waterloo in Canada, Pittsburgh, Charlotte, Urbana-Champaign, Phoenix, Las Vegas, and Madison in the United States.

For our analysis, we are using business objects within the dataset that are classified as restaurants. Although these business objects have more fields, in our analysis we only utilize the attributes JSON object. The object fields are organized like below:


```
{
    "_id" : ObjectId("54efdfbd48c7aed9c7400436"),
    "business_id" : "mVHrayjG3uZ_RLHkLj-AMg",
    "full_address" : "414 Hawkins Ave\nBraddock, PA 15104",
    "hours" : {
        "Tuesday" : {
            "close" : "19:00",
            "open" : "10:00"
        },
        "Friday" : {
            "close" : "20:00",
            "open" : "10:00"
        },
        "Saturday" : {
            "close" : "16:00",
            "open" : "10:00"
        },
        "Thursday" : {
            "close" : "19:00",
            "open" : "10:00"
        },
        "Wednesday" : {
            "close" : "19:00",
            "open" : "10:00"
        }
    },
    "open" : true,
    "categories" : [
        "Bars",
        "American (New)",
        "Nightlife",
        "Lounges",
        "Restaurants"
    ],
    "city" : "Braddock",
    "review_count" : 11,
    "name" : "Emil's Lounge",
    "neighborhoods" : [ ],
    "longitude" : -79.8663507,
    "state" : "PA",
    "stars" : 4.5,
    "latitude" : 40.408735,
    "attributes" : {
        "Alcohol" : "full_bar",
        "Noise Level" : "average",
        "Has TV" : true,
        "Attire" : "casual",
        "Ambience" : {
```

```
            "romantic" : false,
            "intimate" : false,
            "classy" : false,
            "hipster" : false,
            "divey" : false,
            "touristy" : false,
            "trendy" : false,
            "upscale" : false,
            "casual" : false
        },
        "Good for Kids" : true,
        "Price Range" : 1,
        "Good For Dancing" : false,
        "Delivery" : false,
        "Coat Check" : false,
        "Smoking" : "no",
        "Accepts Credit Cards" : true,
        "Take-out" : true,
        "Happy Hour" : false,
        "Outdoor Seating" : false,
        "Takes Reservations" : false,
        "Waiter Service" : true,
        "Wi-Fi" : "no",
        "Caters" : true,
        "Good For" : {
            "dessert" : false,
            "latenight" : false,
            "lunch" : false,
            "dinner" : false,
            "breakfast" : false,
            "brunch" : false
        },
        "Parking" : {
            "garage" : false,
            "street" : false,
            "validated" : false,
            "lot" : false,
            "valet" : false
        },
        "Music" : {
            "dj" : false
        },
        "Good For Groups" : true
    },
    "type" : "business"
}
```


Below is a table of the various features a restaurant can have. Every feature has a set of possible values. Features that have a Boolean type will be True or False. Features that have a String type will hold various string values. These string values can be encoded to be a spectrum. For example, the "Attire" feature can be "casual", "dressy", or "formal". These values are a gradation, and we can encode them as 0, 1, or 2.

Features	Types	Occurrence	Percent with feature
Price Range	Number	20430	93.32176137
Good For Groups	Boolean	19893	90.86881052
Attire	String	19824	90.5536269
Take-out	Boolean	19769	90.30239357
Good for Kids	Boolean	19643	89.72684086
Outdoor Seating	Boolean	19370	88.47980998
Takes Reservations	Boolean	19262	87.98647908
Delivery	Boolean	19173	87.57993788
Good For.breakfast	Boolean	18825	85.9903161
Good For.dinner	Boolean	18815	85.94463731
Good For.latenight	Boolean	18815	85.94463731
Good For.lunch	Boolean	18815	85.94463731
Good For.brunch	Boolean	18765	85.71624338
Good For.dessert	Boolean	18757	85.67970035
Parking.garage	Boolean	18683	85.34167733
Parking.lot	Boolean	18681	85.33254157
Parking.street	Boolean	18681	85.33254157
Parking.valet	Boolean	18681	85.33254157
Parking.validated	Boolean	18470	84.36871917
Waiter Service	Boolean	18404	84.06723917
Alcohol	String	18007	82.25379134
Has TV	Boolean	17274	78.90553627
Noise Level	String	16613	75.88616846
Ambience.casual	Boolean	16578	75.72629271
Ambience.classy	Boolean	16578	75.72629271
Ambience.intimate	Boolean	16578	75.72629271
Ambience.romantic	Boolean	16578	75.72629271
Ambience.touristy	Boolean	16578	75.72629271
Ambience.trendy	Boolean	16578	75.72629271
Ambience.upscale	Boolean	16468	75.22382606
Ambience.hipster	Boolean	16401	74.91777818
Ambience.divey	Boolean	15727	71.83902796
Wi-Fi	String	14118	64.48931116
Wheelchair Accessible	Boolean	10732	49.02247396
Drive-Thru	Boolean	2553	11.66179426
Dogs Allowed	Boolean	2311	10.55636762
Good For Kids	Boolean	2194	10.02192582
Smoking	String	2125	9.706742189
Happy Hour	Boolean	1858	8.487118582

Coat Check	Boolean	1806	8.249588891
Good For Dancing	Boolean	1784	8.14909556
Music.dj	Boolean	1547	7.066508314
Music.jukebox	Boolean	1182	5.399232596
Music.live	Boolean	1179	5.38552896
Music.video	Boolean	1113	5.084048968
BYOB	Boolean	848	3.873561118
Music.karaoke	Boolean	771	3.52183446
Music.background_music	Boolean	762	3.480723552
Corkage	Boolean	645	2.946281747
Order at Counter	Boolean	373	1.703818747
Open 24 Hours	Boolean	254	1.160241184
Dietary Restrictions.dairy-free	Boolean	164	0.749132103
Dietary Restrictions.gluten-free	Boolean	164	0.749132103
Dietary Restrictions.halal	Boolean	164	0.749132103
Dietary Restrictions.kosher	Boolean	164	0.749132103
Dietary Restrictions.soy-free	Boolean	164	0.749132103
Dietary Restrictions.vegan	Boolean	164	0.749132103
Dietary Restrictions.vegetarian	Boolean	164	0.749132103
Ages Allowed	String	25	0.114196967
By Appointment Only	Boolean	22	0.100493331
Payment Types.amex	Boolean	12	0.054814544
Payment Types.cash_only	Boolean	12	0.054814544
Payment Types.discover	Boolean	12	0.054814544
Payment Types.mastercard	Boolean	12	0.054814544
Payment Types.visa	Boolean	12	0.054814544
Music.playlist	Boolean	2	0.009135757
Accepts Insurance	Boolean	1	0.004567879

We quickly learned that most restaurants are missing many features. To handle this, we tried three different paradigms for training our neural network. First, we studied only restaurants that have all features that appear in 50% or more of the restaurants. This was by far the smallest dataset, with 3,758 successful restaurants and 2,746 unsuccessful restaurants. Second, we studied every single restaurant. For every missing feature, we replaced that feature with the midway value between the possible outcomes. For example, Price Range, which has possible values of 1, 2, 3, and 4, would be replaced with 2.5. This set had 5,352 successful restaurants and 16,540 unsuccessful restaurants. Third, we studied every single restaurant, but we replaced missing features with the average value for that feature. For example, in this case Price Range would be set to 1.8, the average value across all restaurants.

We use 70% - 80% of the total data as our training data and hold out the remaining 20% - 30% as test data. The divisions are made randomly.

4.2 Using MongoDB

Given that Yelp provided their academic dataset in the form of five large JSON files, it made sense to store the data with MongoDB. MongoLab, a cloud storage system for MongoDB, was used to host the database. This made it easy to run the neural network from any computer.

The database ended up holding only the JSON file corresponding to the Yelp businesses data, which was about 550MB.

Using MongoDB allowed the team to quickly analyze the data. Because JSON has nested keys, it is not trivial to find all of the possible keys for a given restaurant. Fortunately, using MongoDB and an open-source tool called VarietyJS allowed us to find all of the possible features with the command:

```
mongo ds049181.mongolab.com:49181/new_yelp_data -u naho -p naho --eval "var collection='businesses', query=
    {'categories': 'Restaurants' }" variety.js
```

MongoDB also allowed us to quickly calculate data metrics. In particular, it simplified determining the average value for every feature. This task was necessary for implementing dummy variables, and it could be accomplished with a small amount of math and a simple query:

Finding the Count of Every Value for a Given Key:

```
db.businesses.aggregate(
[{
    $match: { "attributes.Ambience.romantic" : { $exists: "True" }}},
    {
        $group: {
            _id: "$attributes.Ambience.romantic",
            count: { "$sum": 1 }}},
            { $sort: { count: -1 }
    }
])
```

Ultimately, our storage system let us iterate quickly and focus on the neural network itself, rather than data scraping, processing, and manipulation. MongoDB provided us with the tools to run our neural network multiple items in order to optimize our model selection over many trials.

5 Methodology

5.1 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 [Youn and Gu 2010].

5.2 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 challenge with neural networks is that there is no guaranteed method for reaching the global minimum because there are various local minima in the error function. Thus, it is important to average the test error over multiple for accurate predictions. We take this problem into consideration by averaging the performance of multiple hold out validations. Below is a summary of the equations of backpropagation:

$$\delta^L = \bigtriangledown_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T delta^{l+1} \odot \sigma'(z^L)$$

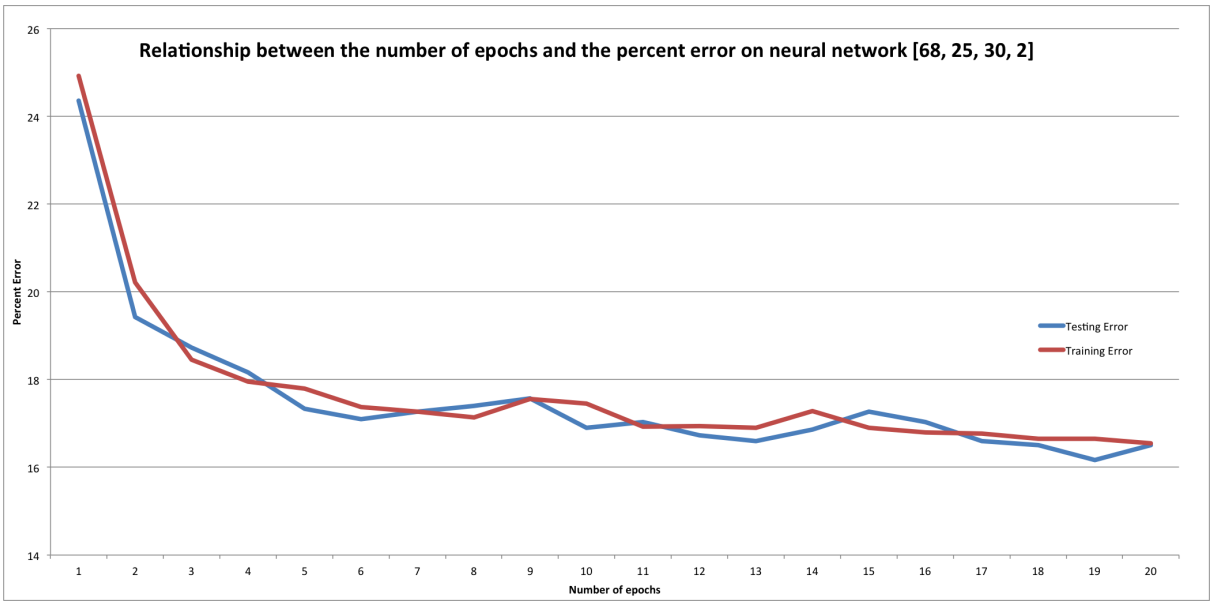
$$\frac{\delta C}{\delta b^l_j} = \delta^l_j$$

$$\frac{\delta C}{\delta w^l_{jk}} = a^{l-1}_k \delta^l_j$$

The first equation is the error in the output layer, which depends on how fast the cost is changing as a function of the output activation and how fast the activation function is changing at the current output. Next, we relate the equation for the error in layer l in terms of the error in the next layer, $l+1$ The two final equations relate the cost function to the weights and bias of each particular neuron.

We referenced chapters 1 and 2 of Michael A. Nielsen [Nielsen 2015] in our neural networks analysis and adapted both his stochastic gradient descent and back propagation algorithm for our analysis.

When we train the neural network, we also reduce the learning rate by half after 5 epochs. If the learning rate is too high, it can actually cause the error to increase over epochs, when it should generally be decreasing or staying the same. This graph displays a single neural network being trained over time, with the error slowly decreasing.



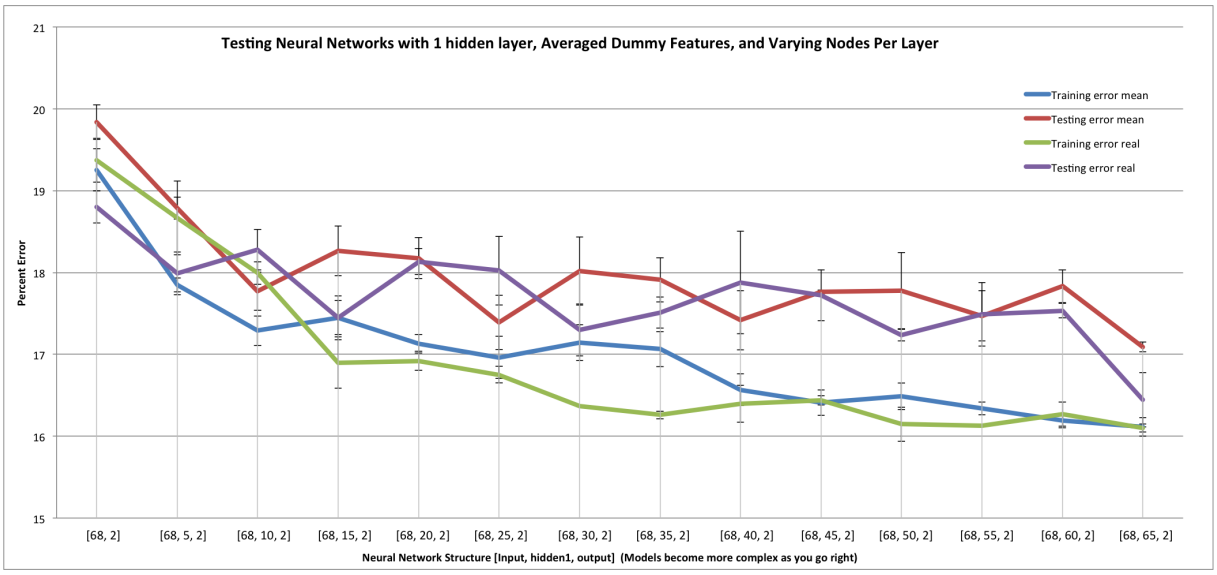
6 Model Selection

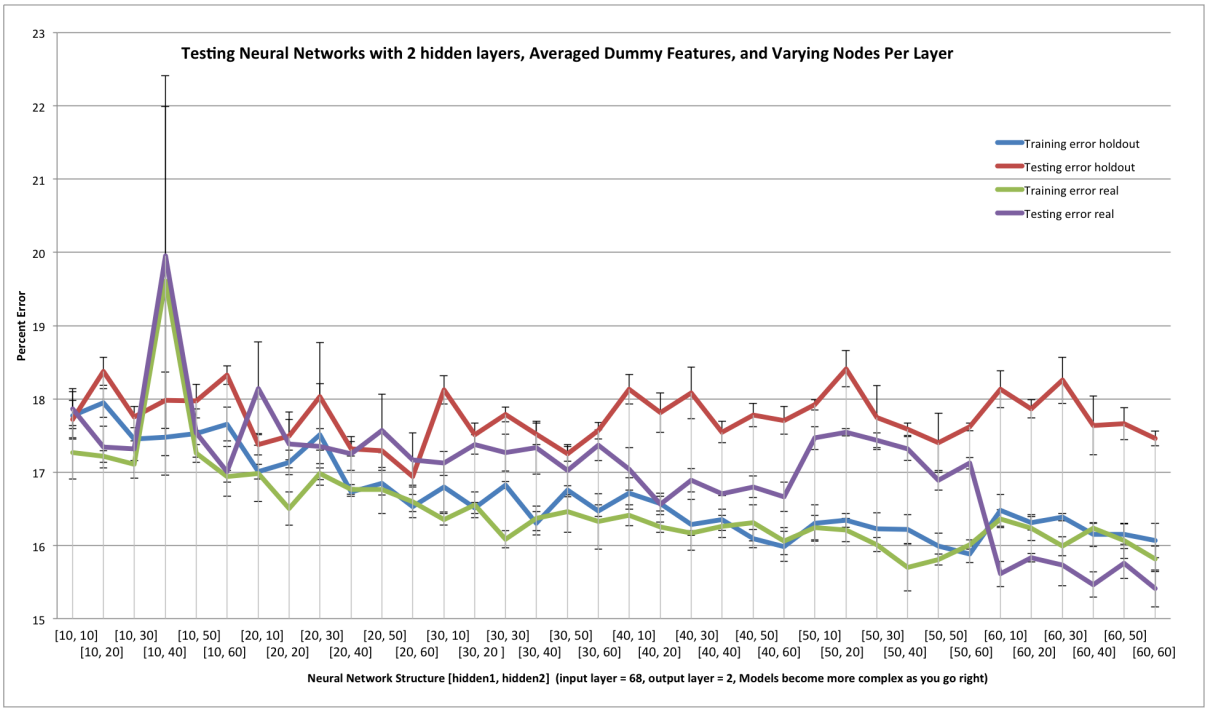
Determining an optimum neural network architecture requires significant human intervention, and there is not an elegant procedure for model selection. Thus, we have implemented holdout validation that takes in multiple different neural network architectures and outputs a csv file to visualize the performance 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 performance of various models, taking into account issues of under/overfitting. Each training of a neural network is susceptible to converging to a local minimum, which makes the network produce high errors. Since we use these error measurements to determine performance, we train and test the same network using the holdout validation technique many times and average the performance over those runs.

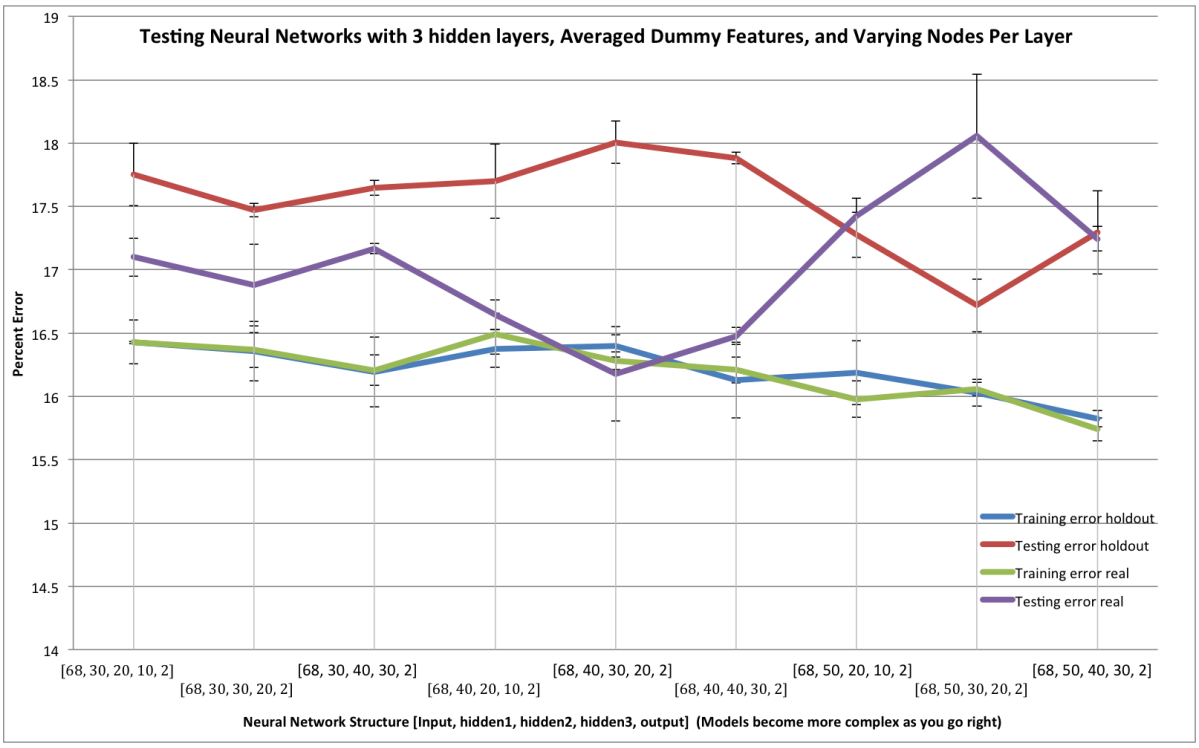
We used the same process of model selection for our three data cases: using only restaurants that have every feature, using dummy variables where each variable of the non-existent features is the midpoint of possible values, and using dummy variables where each variable of the non-existent features is the average of the possible value among all restaurants in our dataset. Since average dummy variables proved to be the best, we will show the graphs and describe the process with respect to that case.

We developed scripts that would run hold out validation on a given dataset using a fixed number of hidden layers and a varying number of nodes in each hidden layer. First, we had to determine the ideal number of hidden layers for our neural network. Too many hidden layers

may increase error due to overfitting, are computationally expensive to train, and are more susceptible to local minima. Thus, we opt for the simplest model that does not have significantly more error than a model with one more hidden layer. The following graphs describe the various error for models with one, two, and three hidden layers. (All error bars show the standard error)

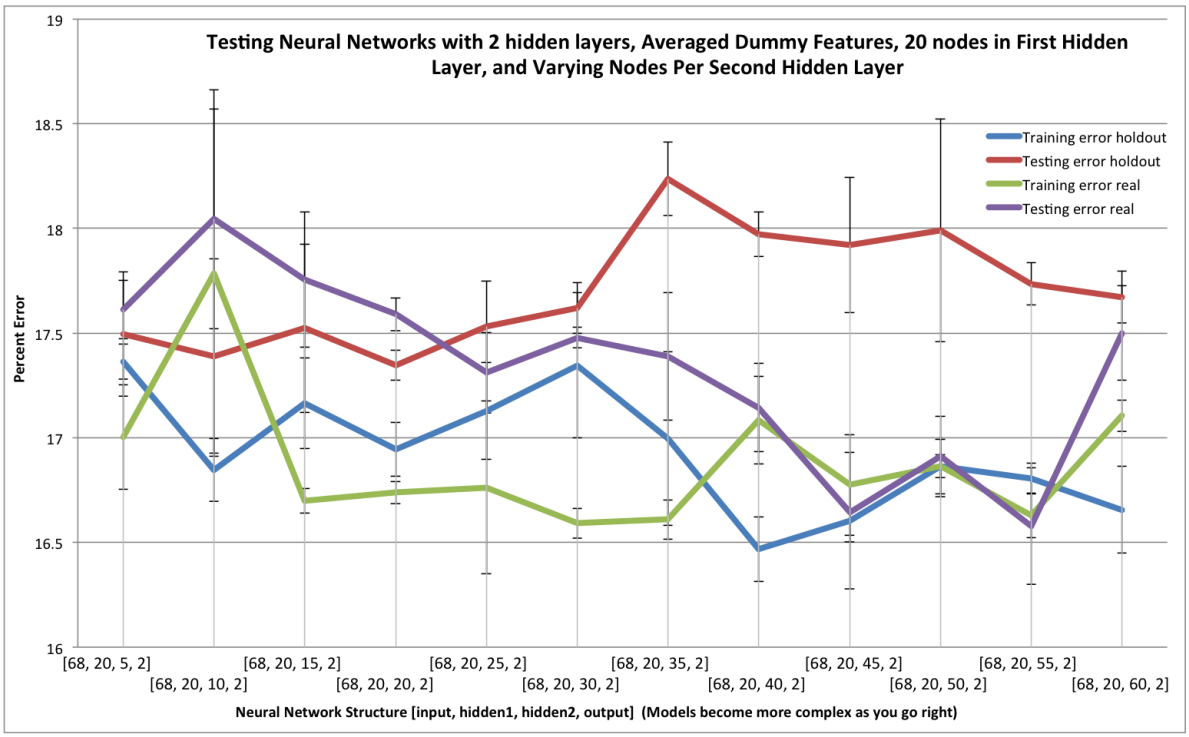






Evidently, networks with two hidden layers seem ideal. There is a downward trend in the percent error as make the model more complex by adding nodes in the hidden layer. The error decreases when going from one layer to two layers, but adding a third layer does not seem to significantly decrease the error.

The next step in our process was to find the ideal combination of nodes per hidden layer. Looking at the two hidden layer graph, the mean test error is clearly lowest somewhere in the neighborhood of twenty nodes in the first hidden layer and sixty nodes in the second hidden layer. From there, we ran a more fine-grained model selection that examined node values specifically within that tightened range.



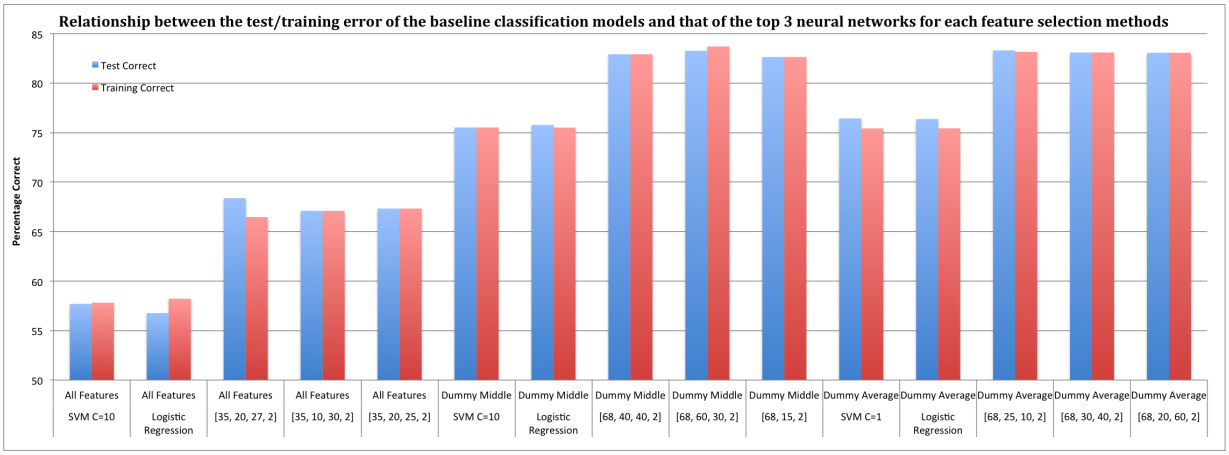
This graph is the equivalent of "zooming in" on the overall two hidden layers examination to the networks that have around twenty hidden nodes in the first hidden layer. Now, we can easily pick our three best models based on the mean test error of the holdout validation (note that we do not pick our networks based on the real test errors. We only look at the mean errors from the holdout validation). They are:

- 2 hidden layers, 25 nodes in first layer, 10 nodes in second layer
- 2 hidden layers, 30 nodes in first layer, 40 nodes in second layer
- 2 hidden layers, 20 nodes in first layer, 60 nodes in second layer

We repeated this process for the other two methods of feature selection to find the three best neural network models on a per case basis. This allowed us to create our final results.

7 Results

Using the best models for each case, we were able to comfortably beat benchmark tests of SVM and Logistic Regression. We ran a simple model selection on the SVM to fine tune the hyper-parameter C from the L1-norm soft margin SVM model, and we also tried using a polynomial kernel for each of the feature selection methods. This graph compares the test errors for each of the different feature selection methods and models.



Evidently, using dummy variables proved extremely helpful - we can see clearly that without any dummy variables, the neural network had a 67% success rating. This difference in performance is probably not due to the dummy variables themselves, but due to the fact that we can use all restaurants. In fact, the training set size with dummy variables was roughly 4 times larger than that with no dummy variables. Taking into account this drastic increase in training set size, the outcome is not very surprising since neural networks suffer greatly with smaller datasets in comparison to other models. As the networks becomes more complex, this issue is aggravated. Additionally, it is possible that restaurants with incomplete features are generally less successful, since Yelp may not be as interested in those establishments. This gives additional information to the neural network, since it can learn to correlate many mocked features with failure. Interestingly, there was not much of a difference in performance between when we mocked non-present features with the average feature value instead of the middle feature value.

Although the neural networks that did not use dummy features underperformed those that did, our neural network outperformed all benchmarks for all three methods. Our neural network easily outperformed Logistic Regression and SVM by around 7-8% across the board. Our best neural network classifies restaurants with 83.3% accuracy, while the best performing Logistic Regression had 76.4% accuracy, and SVM had 76.4% accuracy. Given that we even used model selection to pick the best performing SVM, we are impressed by the performance of our networks.

8 Contact Information

If you have questions or suggestions regarding this paper, please contact Jason Feng at "jason.feng.17@dartmouth.edu", Naho Kitade at "naho.kitade.16@dartmouth.edu", or Matthew Ritter at "matthew.p.ritter.15@dartmouth.edu".

Acknowledgements

We would like to acknowledge Yelp for providing us with the dataset that made this experiment possible. We would also like to acknowledge Professor Lorenzo Torresani for all of his advice.

References

Kim, S. Y., and Upneja, A. 2014. Predicting restaurant financial distress using decision tree and adaboosted decision tree models. *Economic Modelling 36*, 354–362.

Nielsen, M. A. 2015. *Neural Networks and Deep Learning*. Determination Press.

Parsa, H. G., Self, J. T., Njite, D., and King, T. 2005. Why restaurants fail. *Cornell Hospitality Quarterly 46*, 3 (August), 304–322.

Yelp, 2015. Yelp's academic dataset. `https://www.yelp.com/academic_dataset`.

Youn, H., and Gu, Z. 2006. Predicting restaurant bankruptcy: A logit model in comparison with a discriminant model. *Tourism and Hospitality Research 30*, 4, 474–493.

Youn, H., and Gu, Z. 2010. Predict us restaurant firm failures: The artificial neural network model versus logistic regression model. *Tourism and Hospitality Research 10*, 3 (jul), 171–187.

Implementation Details

All of our code is hosted on Github here. All of the files present in the codebase were written exclusively by the team, with the exception of parts of network.py and variety.js. Network.py was based on code snippets provided by *Neural Networks and Deep Learning*, found here. This file was initially meant to be used in an experiment to assess handwritten numbers, but we have modified it for our Yelp dataset. In the codebase, functions provided are annotated as External Software. Contributions from our team are labeled as such as well.

Variety.js is a file that helps us analyze MongoDB data. Its use is described above for learning about the data features. It is also available on Github here.

In addition, we relied heavily on some very helpful packages:

- Pymongo: Used for interacting with MongoDB and MongoLab
- Numpy: Used for creating vectors and matrices in Python
- VarietyJS: Used for learning about restaurant features
- SciKit-Learn: Used to provide SVM and Logistic Regression for benchmarks