Impact of Feature Selection and Feature Transformation on Neural Network Models

Jenny Hung

[jehung@gatech.edu](mailto:jehung@gatech.edu) (alias: jhung34@gatech.edu)

1. Two Classification Problems Considered

**Freddie Mac mortgages loan-level performance data**

Freddie Mac, one of the government-sponsored enterprises went through a nearly [$200 billion government bailout](http://en.wikipedia.org/wiki/Federal_takeover_of_Fannie_Mae_and_Freddie_Mac) during the financial crisis, caused in large part by losses on loans that it guaranteed. Further, Freddie Mac began reporting loan-level credit performance data in 2013 at the direction of their regulator, the Federal Housing Finance Agency. The stated purpose of releasing the data was to “increase transparency, which helps investors build more accurate credit performance models in support of potential risk-sharing initiatives.”

This financial melt-down and the subsequent bailout has motivated my research into the Freddie Mac loan-level performance dataset. This dataset lends naturally to the classification problem of mortgage defaults, and the goal of this classification problem is to analyze multiple-period mortgage risk at loan level using the Freddie Mac dataset prime and subprime mortgages originated in the United States in 2016, which includes the individual characteristics of each loan, and monthly updates on loan performances over life of a loan.

The entire Freddie Mac dataset is immensely rich, in that it encompasses a 10-year span and contains millions of loan-level mortgage performances and default information. Freddie Mac has created a smaller dataset, which is a random sample of 50,000 loans selected from each full vintage year (defined as the calendar year in which the loan was originated). Each vintage year has one origination data file and one monthly performance file, containing the same loan-level data fields as those included in the full dataset. In this implementation, we have located the sample\_2016.zip file from the full dataset package, and used this zip package as our data source for this iteration. The “2016” in the file name indicates that the loan information was recorded in the year 2016, but the loan could be originated in an earlier year (namely, the vintage year could be an earlier year). The dataset in this implementation has more than 203,000 mortgages performance records.

The 2016 zip packages has two files: sample\_orig\_2016.txt and sample\_svcg\_2016.txt. The .txt files do not come with headers but instead, we refer to the User Guide (<http://www.freddiemac.com/research/pdf/user_guide.pdf>) to grab the name of the columns. We then join the two data files together by the loan number.

It is expected that as we progressed further, we will be using larger and larger datasets. But for this first iteration, this is what we have chosen.

Missing values can be found in the dataset. Key features that are missing are more likely to be the result of reporting errors by the originator or the servicer, or incomplete information provided by the borrower. Similar to the Deep Learning paper we are reading, we have insisted that an observation must have no missing values in any of the following:

* FICO score
* LTV ratio
* Original interest rate
* Original balance

Samples missing one of the above variables are removed.

Compared to the first iteration, we have removed mas (Metropolitan Statistical Area) as a feature. This feature does not carry a lot of additional information as the geographical location can be identified through both the state and zipcode variables.

We also examined other variables where missing values exists. Good examples of these are Super Conforming flag (exceed\_conform) and First Time HomeBuyer Flag (first\_time). Our code would set any missing values to zero first. In cases of categorical variables like these, this action will yield 3 values: Y, N, and 00. These values will then be coded as dummy vairalbes / indicator variables.

In the case of a numerical variable with missing values, the missing values would still first be converted to zero. Columns of numerical variables will then be scaled while preserving the sparse structures in the next step.

To further process the data, we have taken the following steps:

* Get the delinquency status that is associated with the loans and last observed month,
* Remove the curr\_delinq from our feature space
* Use curr\_delinq as our taget
* For the categorical variables, we convert them into dummy/indicator variables

After processing the dataset, there are 203,642 loan performance observations and 109 variables.

This problem is interesting because there exists a highly nonlinear relationship between the variables and the default prediction. In particular, many of the existing academic research studied deep neural networks, which have multiple layers of hidden nodes. In this implementation, I am more interested in finding out the relative strengths, time complexity, sample complexity and mistake bounds associated with using each one of the classifiers in question.

**Blood donation prediction**

The [UCI Machine Learning Repository](https://archive.ics.uci.edu/ml/index.html) is a great repository of data science-related projects. This dataset was originated from a mobile blood donation vehicle in Taiwan. The Blood Transfusion Service Center drives to different universities and collect blood as part of the blood drive. In this dataset, we want to predict whether or not a donor will give blood the next time the vehicle comes to campus. Given my interest in discovery our mission, we're interested in predicting if a blood donor will donate within a given time window. This is considered a beginnier’s dataset.

After processing the dataset, there are 574 loan performance observations and 3 variables: number of donations, months since first donation, months since last donation.

The choice of having a beginner’s dataset is quite deliberate. Contrasting with the mortgage default prediction problem, the Blood Donation prediction problem is interesting because it motivates the comparison of classifier performances, and thus highlight the fact that different classifiers have different computational complexity, sample complexity as well as mistake bounds: what worked for one problem does not necessarily transfer to the next problem.

1. Feature Transformations Considered

We consider two broad types of feature transformations: feature selection by means of clustering and other methods, and feature transformation by means of PCA, ICA and random methods.

2.1 Clustering: KMeans and Expectation Maximization Model

Both KMeans and Expectation Maximization methods takes an integer input, n\_component, and iteratively seek the clustering of data observations into the given number of “groups”. They do not seek to classify the data, but rather, they seek to find the internal structure of the data, and group the data a way that are “similar” – in ways that can be defined precisely in a mathematical way using expert domain knowledge.

In this analysis, we have used Scikit-learn’s KMeans() and GaussianMixture() functions. Further, the default distance metrics in both functions were used in order to find the best, or the most reasonable integer, n\_component, to be used for our analysis.

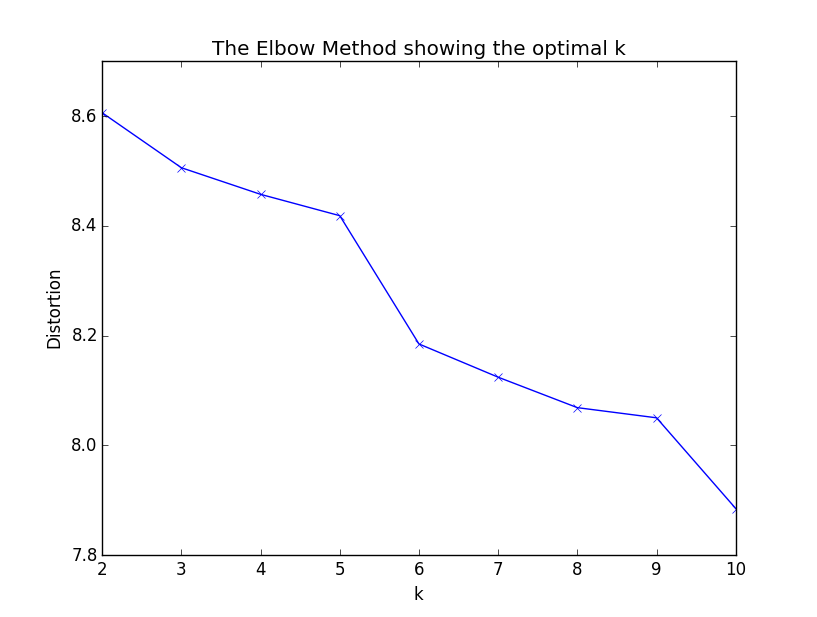
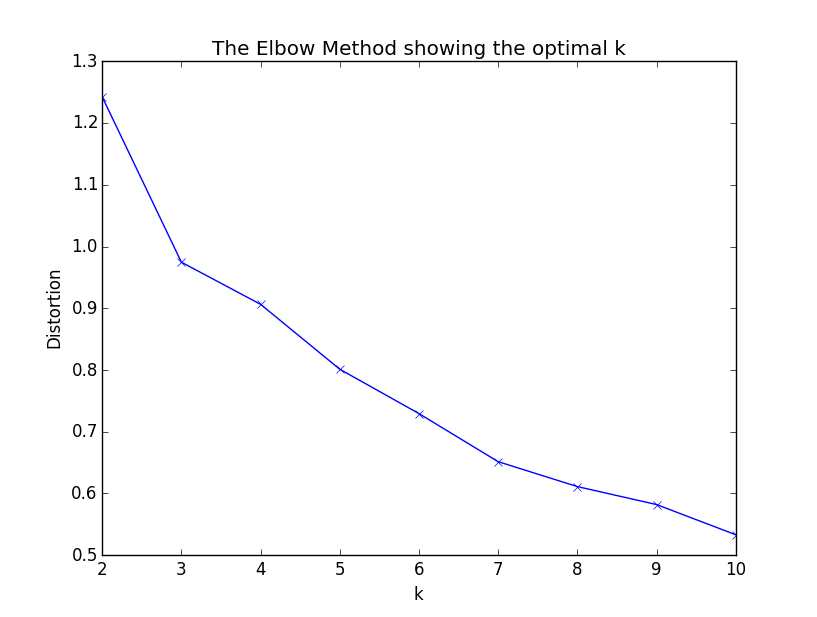
Since KMeans and Expectation Maximization requires a predetermined number of clusters as a starting point, this is our starting point for our investigations. In our implementation (see clustering.py), we iteratively implemented both methods by setting the number of clusters to be from the set of and extracted the following information from each model:

* Sum of squares for K-means
* Log-likelihood for Gaussian Mixture
* Cluster accuracy for both
* Adjusted mutual information for both

The following table summarized the results:

From the above analysis, we have chosen the value of …. And ….

In addition, for K-means, we have used the Elbow method to visualize the ideal number of clusters to use in the algorithm:

(Left: Optimal k for Freddie Mac dataset; right: Optimal k for Blood Donation dataset).

The

* 1. Feature Selection using Other Methods: Random Forest Model
  2. Feature Transformation: PCA and ICA

Both PCA and ICA have the parameter, n\_component, which denotes the number of axes in the resulting space that came from having performed dimensionality reduction. This n\_component parameter is chosen through an iterative process by setting it to be an element form the set of and extracted the following information from each model (see pca.py:

* Percentage of explained variance and eigen values for PCA
* Kurtosis for ICA

From the above analysis, we have chosen the value of n\_component = 10 for PCA and n\_component = 20 for ICA for the Freddie Mac dataset, and n\_component = 2 for PCA/ICA for Blood Donation dataset.

Below are the visualizations for both datasets after projecting into the new space using PCA:

Similarly, the visualizations for both datasets after projecting into the new space using ICA are shown below:

PCA fits an ellipsoid to the data. The ellipsoid can be described by the directions and lengths of their principal (semi-)axes. No matter how the ellipsoid is turned, the eigenvectors point in those principal directions and the eigenvalues give you the lengths. The smallest eigenvalues correspond to the thinnest directions having the least variation, so ignoring them (which collapses them flat) loses relatively little information. Thus, when we showed the most important features in PCA, we are equivalently showing the distribution of eigenvalues. This is done here:

Similarly, we show the distribution of kurtosis using ICA:

#TODO: TruncatedSVD???

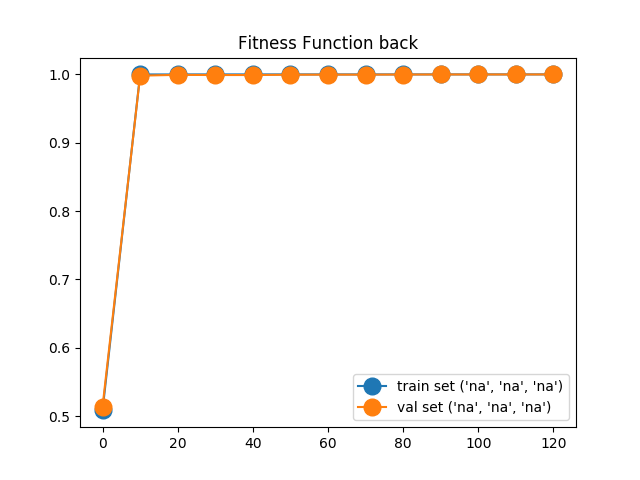
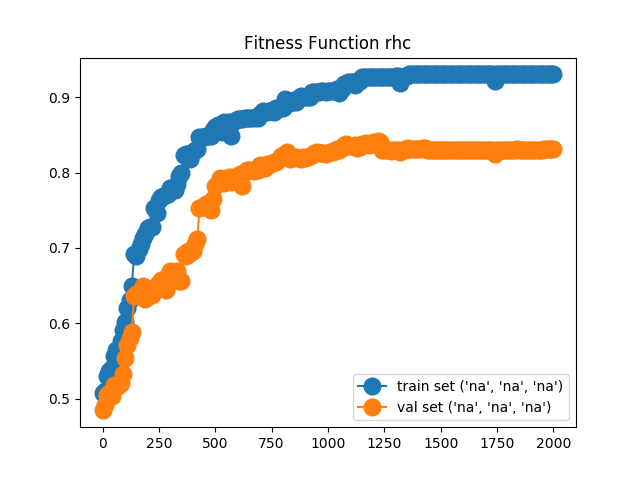
* 1. Feature Transformation: Random Projections

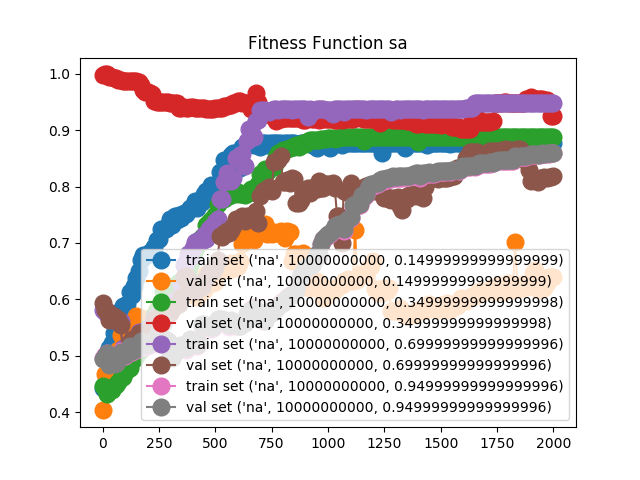
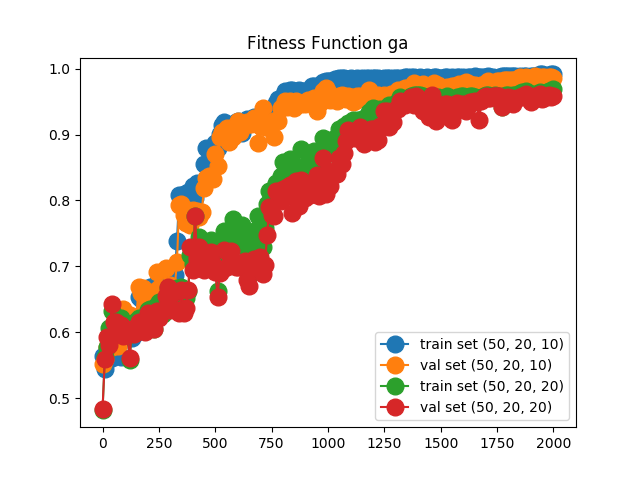
3. The neural network

In this part, the Freddie Mac dataset using neural network from assignment 1 is analyzed using randomized optimization algorithms. Based on Assignment 1, the best neural network had an input layer equal to the number of attributes, an output layer, and five hidden layers with 109, 76, 76, 76, 76 units for each respective hidden layer. The weights within the network were determined using back propagation, and in Assignment 1, achieved a classification rates of 92%.

In this Assignment, I have performed the same analysis using the back-propagation implementation in the JAVA library ABAGAIL to establish a baseline, again which the randomized algorithms will be compared to. I used the standard minimization of the sum of squared distances (SSD) of the error. The weights were initialized randomly and each optimization algorithm was run 5 times. The mean results are given in the following table. Note that in this assignment, a split of 0.1, 0.1 was used to split the data into training and test set, and cross-validation has not be been implemented.

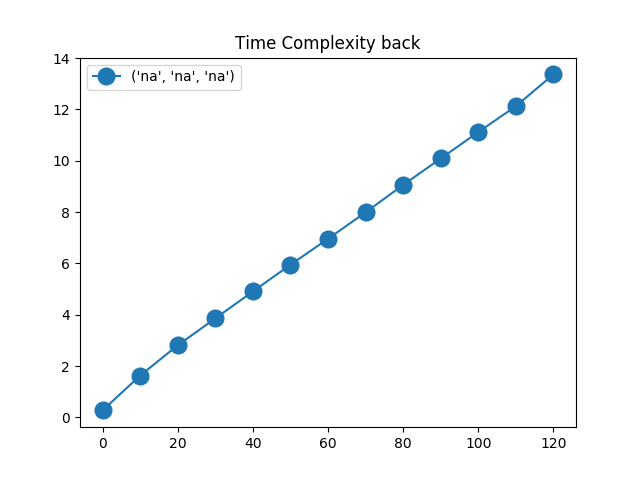
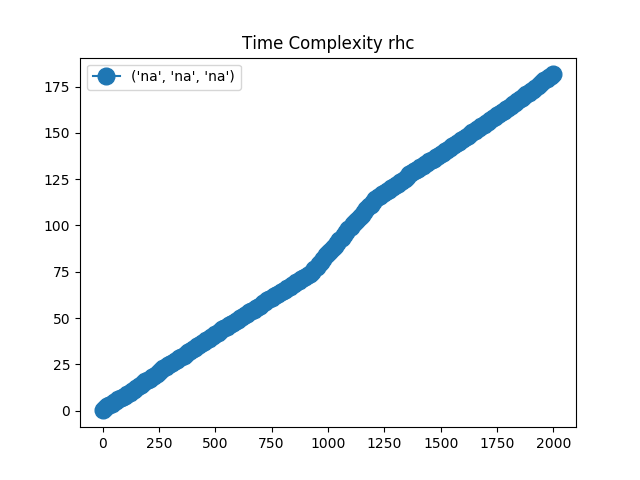
Running time were plotted against number of iterations below.

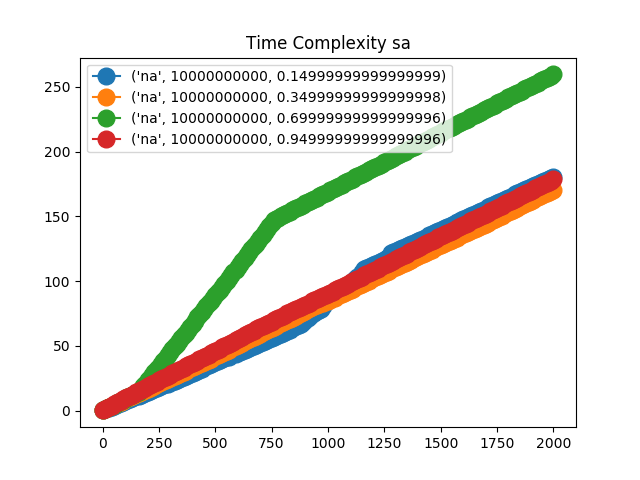
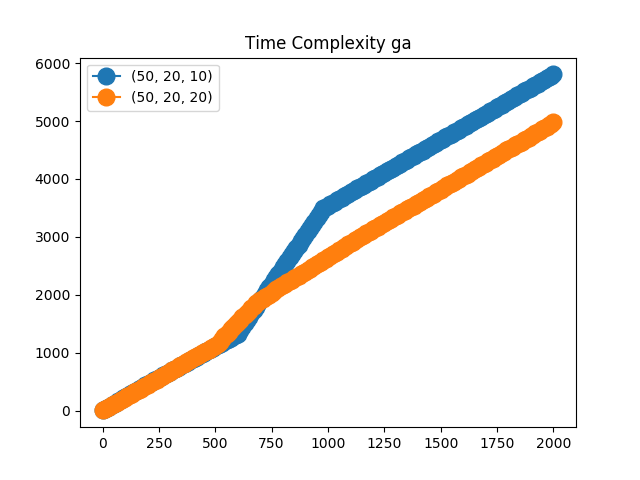
 

Not surprisingly, back propagation has the highest accuracy rate when compared to randomized algorithms. However, among the randomized algorithms, GA performed the best, defined as having the lowest sums of squares errors, in approx. 700 iterations, depending on the parameters chosen. More specifically, when we used a lower number of mutation points in GA, the accuracy rate is observed to be quite good at a low number of iterations. RHC performed second best, but it does take more iterations to converge – but there has been a larger difference between the training errors and test errors (called validation errors in the graph), something that GA does not show.

Running time were plotted against number of iterations below.

Again, not surprisingly, back-propagation converged has the lowest time complexity, followed by RHC and SA. We see that GA took a long time for each iteration. This observation is very much consistent with our observations so far.

4. Conclusion

It can be shown that smooth problems can be solved best with SA or GA (such as neural network training), while MIMIC tend to perform well in Flip Flop problems, which can be described as having discrete smooth cost functions. RHC is an exhaustive search which performs very well on smooth problems, but ran into diﬃculties with problems with discontinuities. All four algorithms perform poorly on problems like TSPs, where special variants of them are needed to obtain good results. If we have a smooth problem with a global optimum, such as in the case of a neural network with a convex cost function, then none of the algorithms can outperform an ordinary Gradient Descent method used in the back-propagation.