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**

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.

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. 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. Unsupervised Learning Algorithms Considered

We consider two broad types of unsupervised learning techniques: clustering algorithms by clustering and expectation maximization, and dimension reduction / feature transformation by means of PCA, ICA, random projections, and random forest.

2.1 Clustering: KMeans and Expectation Maximization Model

Both KMeans and Expectation Maximization methods takes an integer input, n\_clusters, 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\_clusters, 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 for Freddie Mac dataset:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | KMeans | | | Expectation Max | | |
| k value | Accuracy | Mutual Info | Sum of Squares | Accuracy | Mutual Info | Log Likelihood |
| 2 | 0.699934182 | 0.234484008 | 32743090.09 | 0.699934182 | 0.234484008 | 116.9068767 |
| 3 | 0.699934182 | 0.192183989 | 29931134.37 | 0.755609258 | 0.290114549 | 215.1502225 |
| 4 | 0.755609258 | 0.205830807 | 28641744.08 | 0.755609258 | 0.178449574 | 241.6277362 |
| 5 | 0.755609258 | 0.191564364 | 28641744.08 | 0.755609258 | 0.164316142 | 351.1252698 |
| 6 | 0.792943731 | 0.166099774 | 25886260.33 | 0.768588648 | 0.220322143 | 364.2010726 |

The following table summarized the results for Blood Donation dataset:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | KMeans | | | Expectation Max | | |
| k value | Accuracy | Mutual Info | Sum of Squares | Accuracy | Mutual Info | Log Likelihood |
| 2 | 0.538990826 | 0.00363415 | 1635.999136 | 0.516055046 | 0.000128851 | 3.176510333 |
| 3 | 0.650229358 | 0.04948042 | 1109.939417 | 0.634174312 | 0.037990178 | 1.901190868 |
| 4 | 0.653669725 | 0.040147433 | 837.0399988 | 0.620412844 | 0.031193957 | 1.671492233 |
| 5 | 0.658256881 | 0.034359431 | 662.3435296 | 0.655963303 | 0.042039301 | 1.568953527 |
| 6 | 0.668577982 | 0.038556442 | 520.7940537 | 0.650229358 | 0.037209373 | 2.365953306 |

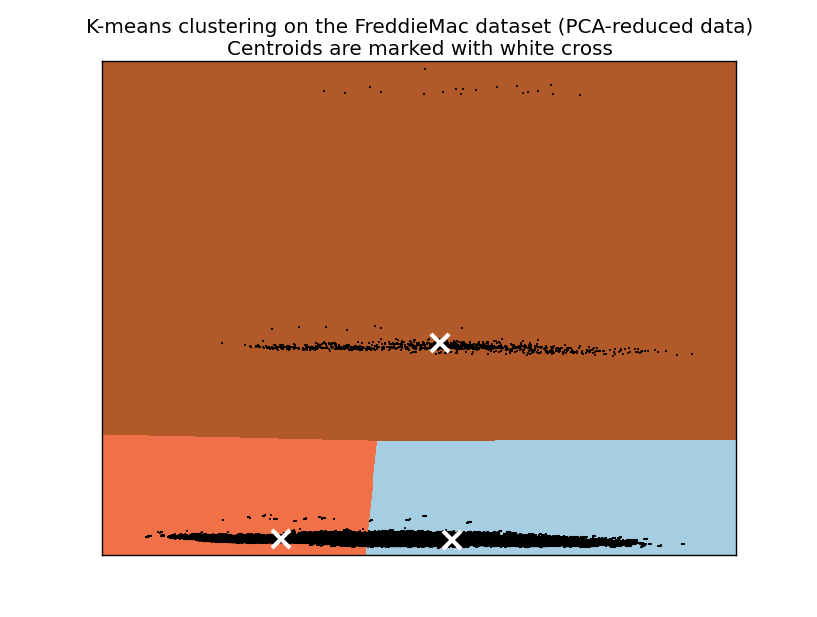
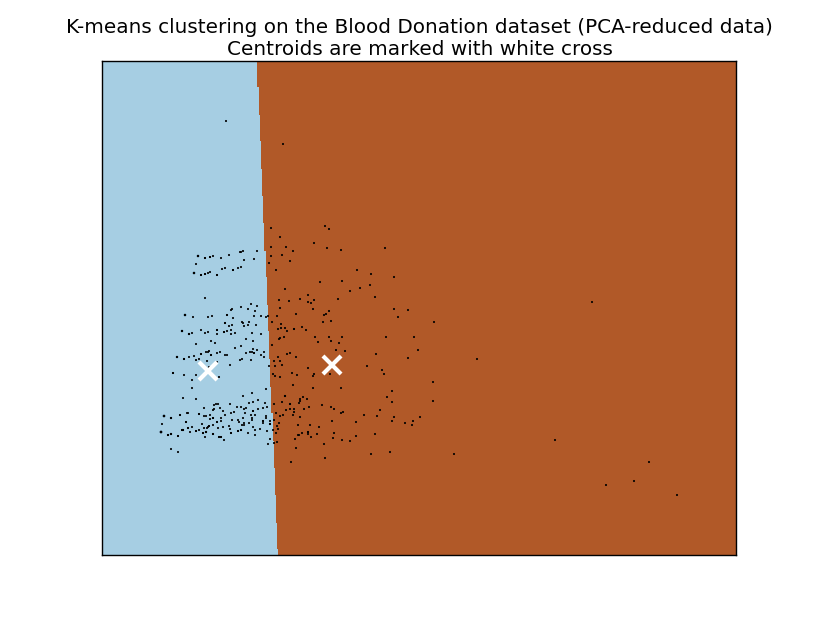
From the above analysis, we have chosen the value of k = 3 for both K-means and EM for Freddie Mac dataset.

From the above analysis, we have chosen the value of k = 2 for both K-means and EM for Blood Donation dataset.

We have chosen these specific values of k after reviewing accuracy (defined as within-cluster classification accuracy), adjusted mutual information score between clusters, and SSE/Log-likelihood for K-means and EM, respectively. More concretely, we seek to find the k-value that maximizes both accuracy and adjusted mutual information for both K-means and EM. In addition, we have also chosen to maximize log-likelihood for EM.

We have also ignored the SSE metric for K-means. This was because SSE is defined as sum of squared distances to the closest centroid for all observations in the dataset. This metric necessarily is a non-increasing function as the value of k increases, namely, as more and more clusters were built (hence more and more centroids were obtained). The SSE is exactly zero when the number of centroids is equal to the number of data points. Hence, we have determined that this is not a valid choice metric for k.

With these values chosen, we are now able to visualize the clusters using K-means on both datasets. We show this below:

The clusters make a lot of sense. It is here, in Freddie Mac data set, where the power of finding hidden structures embedded in the data, truly comes into focus. From the exploratory data analysis on Freddie Mac dataset, we note that although there are only two elements in the target space (default and no default), the reality is that there could be a long transition from not defaulted to defaulted classification, as the loan necessarily need to go through the stages of being late, which could either catch up on its late payment and become current later, or not able to catch up on late payments and eventually result in default state. Below on the left, we have shown the visualized data (after it has been transformed to 2-dimensional new space using PCA). The three clusters are easily depicted – although the data are only classified into two categories.

Comment on the ability to reconstruct?

* 1. Feature Transformation: PCA, ICA, Random Projection and Random Forest

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 for PCA
* Kurtosis for ICA

PCA fits an ellipsoid to the data. The ellipsoid can be described by the directions and lengths of their principal 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 eigenvectors. Further, the percentages of variance explained for each of the eigenvectors gives the relative importance of each eigenvectors (namely, each component) and these percentages are also, as it turns out, proportional to the eigenvalues. Therefore, a good way of showing the distribution of eigenvalues is to show the percentage of variance explained by each component. This is shown below:

|  |  |
| --- | --- |
| Axis number | N-component |
| 1 | 2 |
| 2 | 5 |
| 3 | 10 |
| 4 | 15 |
| 5 | 20 |
| 6 | 30 |
| 7 | 40 |
| 8 | 50 |
| 9 | 60 |

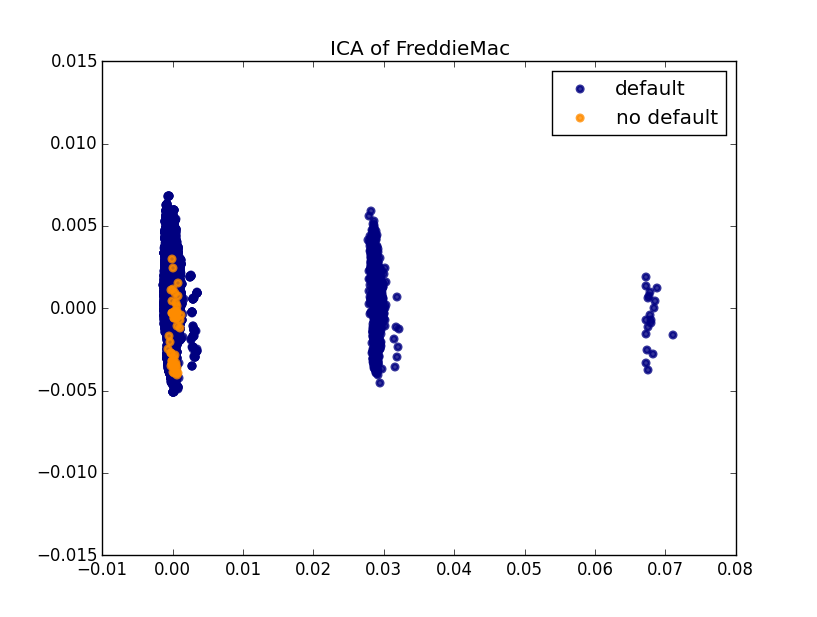
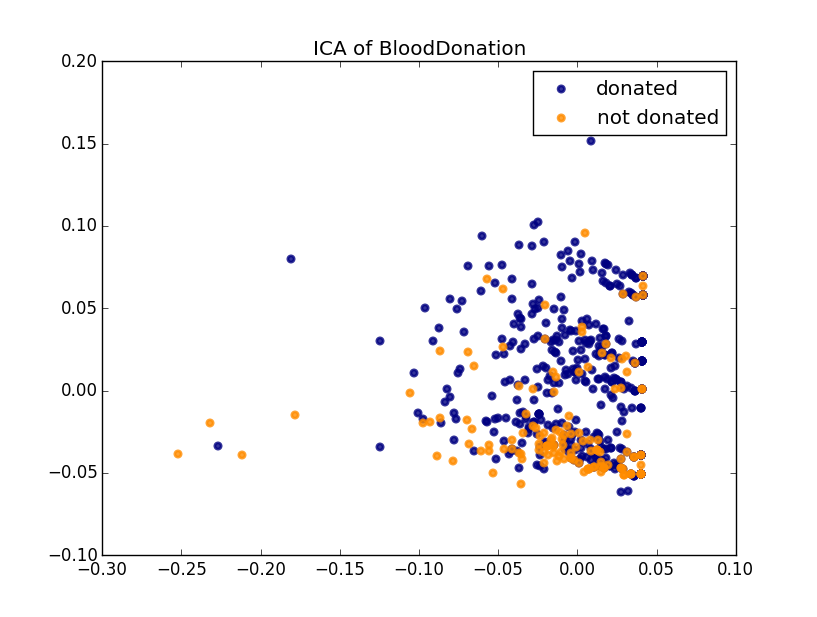
This table shows the mapping of horizontal zis to the parameter N (n-components) in PCA.

From the above analysis, we have chosen the value of n\_component = 60 for PCA and n\_component = 20 for ICA for the Freddie Mac dataset, and n\_component = 2 for PCA/ICA for Blood Donation dataset.

Similarly, we show the distribution of kurtosis using ICA:

Similarly, we show the distribution of reconstruction error using random projections:

The visualizations for both datasets after projecting into the new space using PCA is shown in the earlier section (Section 2.1). Similarly, the visualizations for both datasets after projecting into the new space using ICA are shown below:

A significant issue when choosing the n\_component value for PCA and ICA for the Freddie Mac dataset comes from the large number of one-hot encoded variables. As PCA implicitly requires numerical, continuous values, PCA encounters significant difficulty when trying to find the axis that yields the highest variance. In practice, what we have found is that the n\_component parameter needs to go to as high as so as for us to account for at least 80% of the variance. ICA, on the other hand, fares much better, when n\_component was chosen to be 20 when considering the maximum of kurtosis. This suggests that using ICA, we are able to find more meaningful axes, and the original data can be transformed and projected onto the new space in a much more meaningful way.

When it comes to Random Projections, we use the Johnson-Lindenstrauss lemma, which states that any high dimensional dataset can be randomly projected into a lower dimensional Euclidean space while controlling the distortion in the pairwise distances. The projection p is an epsilon-embedding defined by:

1 - eps) \|u - v\|^2 < \|p(u) - p(v)\|^2,

Where u and v are any rows taken from a dataset of shape [n\_samples, n\_features], epsilon is in , and p is a projection by a random Gaussian matrix with shape [n\_components, n\_features].

The lemma tells us that with an increasing number of samples, the minimal number of dimensions (n\_components), increased logarithmically in order to guarantee an epsilon-embedding.

Further, the lemma also tells us that an increase of the admissible distortion, represented by epsilon, allows to reduce drastically the minimal number of dimensions (n\_components) for a given number of samples (n\_samples). According to the JL lemma, projecting 500 samples without too much distortion will require at least several thousand dimensions, irrespective of the number of features of the original dataset.

Our experiments indicate that dimension reduction procedure has an effect on the number of clusters that is required in subsequent machine learning workflow. After projecting the original data onto the new spaces (namely, after PCA and ICA), we are able to choose the number of clusters. Using the same selection criteria as listed above in 2.2, we are now able to choose 2 clusters for the Freddie Mac dataset, and 2 clusters for the Blood Donation dataset using PCA. However, when using ICA, the number of clusters had remained as 2 clusters for Freddie Mac, but has not increased to 3 for the Blood Donation dataset. This further gives us evidence that using ICA, we are able to find more meaningful axes for the data.

3.1 Dimension Reduction before Neural Network Classifier

In this section, we investigated the impact that dimension reduction algorithms have on a neural network classifier, and retrained the neural network (the weights from assignment 1 were NOT re-sued). To solve for the optimal structure of the network, Scikit-learn’s Pipieline() feature as well as GriddSearchCV() was used. See code in PCA.py, ICA.py and RP.py for implementation details.

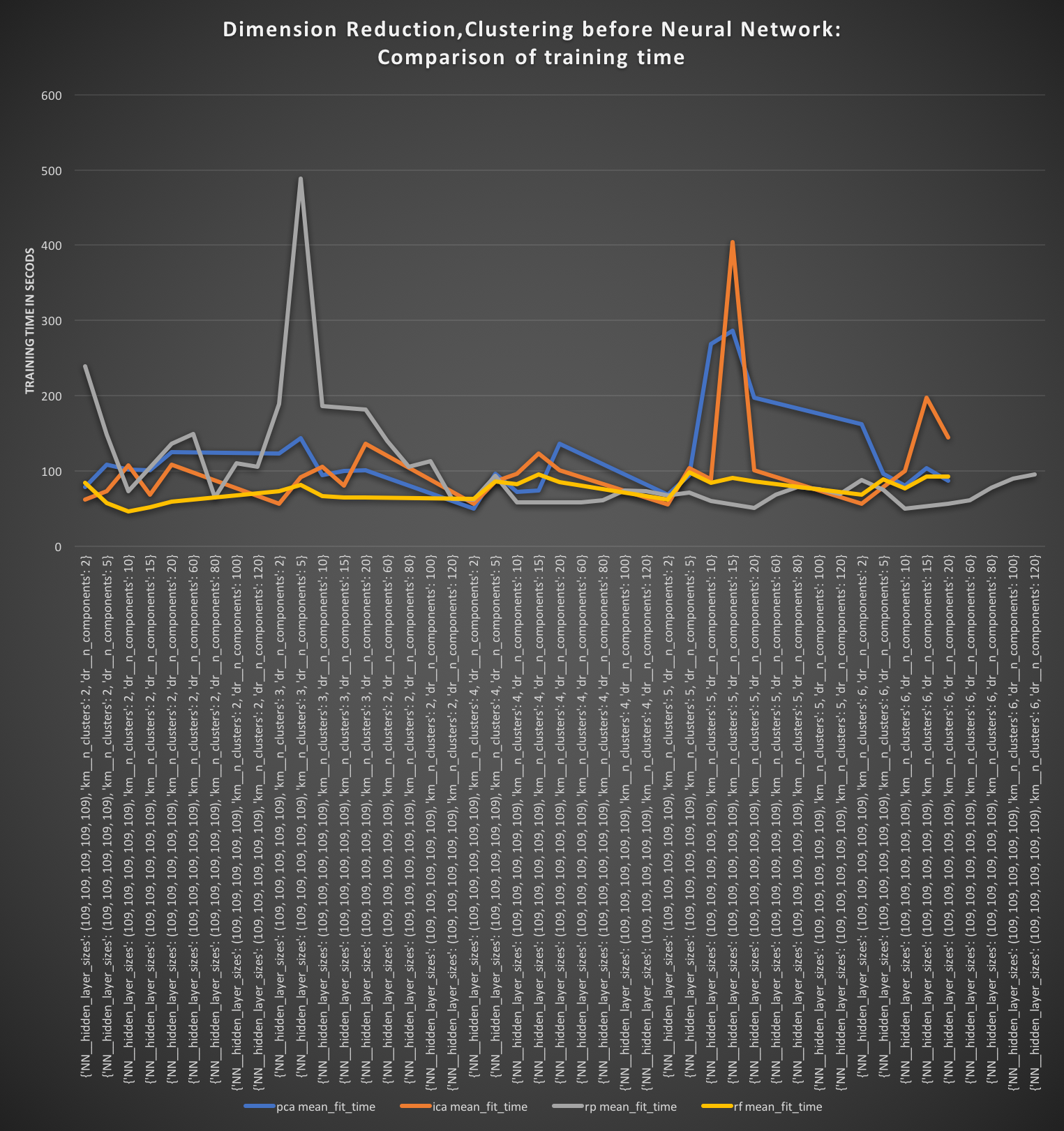
Since the sever class imbalance is present in both datasets, we have used SOMTE to synthetically oversample to address this imbalance before fitting a model. Note that this step was not performed in the original Assignment 1, hence we now “refit” the original network using synthetic oversampling as well, to ensure comparability across model performance.

The summary plot from the detailed training result, that compared dimension reduction algorithm of PCA, ICA, RP is shown below:

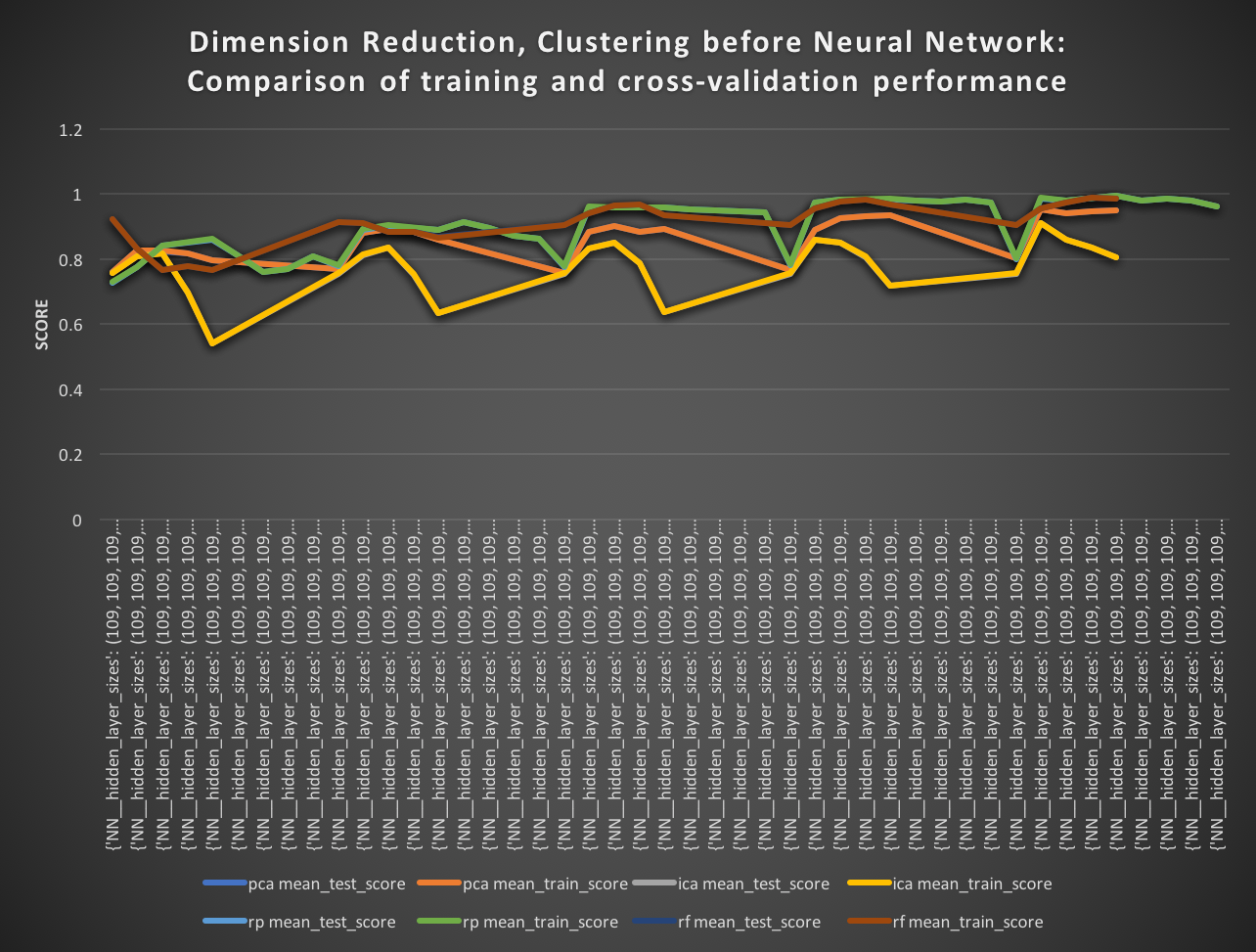
When dimension reduction was performed prior to training a neural network (on the Freddie Mac dataset), the resulting training time is longer. However, Random Forest tends to add the smallest amount of additional time to the training process. It is worth to note that Random Projection takes a long time if there is a large amount of compressions (namely, when dimensions are low), but the training time quickly decreases as there are less compression to be done.

* 1. Dimension Reduction, clustering, before Neural Network Classifier

The summary plot from the detailed training result, that compared dimension reduction algorithm of PCA, ICA, RP is shown below:



Similar to what you have noticed earlier, training time is now longer given the more processing steps involved. However, Random Forest tends to add the smallest amount of additional time to the training process. It is worth to note that Random Projection takes a long time if there is a large amount of compressions (namely, when dimensions are low), but the training time quickly decreases as there are less compression to be done.



From the above visualization of pipeline performance comparison, it is extremely interesting to note that, based on our experiments, Random Projections and Random Forest, as algorithms to reduce dimension, consistently achieve the best accuracy score, for both the training set and the validation set, across all combinations of parameters (i.e, n\_components, n\_clusters, and hidden layer notes).