Min Ho Lee
#903122265
mlee432@gatech.edu

Unsupervised Learning Analysis

Introduction

In this assignment, the main focus is to investigate unsupervised learning. Two clustering algorithms (k-means clustering and Expectation Maximization) and four dimensionality reduction algorithms (PCA, ICA, Randomized Projections, and Random Subset) will be investigated. The datasets used to investigate the algorithms are *spambase.arff* and *optdigits.arff. spambase.arff* is one of the datasets I have been using and *optdigits.arff* is different since it will be more fit to unsupervised learning since the database I used before, *car.arff*, contains non-numeric instances. Weka was used as the tool to run all the algorithms, and all the data and graphs collected from it are stored in excel file *Clustering*, *Reduction,* and *Neural Network*.

Datasets Description

spambase:

This dataset contains 4601 instances and 58 attributes. Each instance denotes whether the e-mail was considered spam or not as its class, and most of the class attributes indicate whether a certain word or character was frequently occurring in the e-mail. This collection of spam e-mails came from their postmaster and who had filed spam, and Our collection of non-spam e-mails came from filed work and personal e-mails. This dataset is interesting because filtering out spam e-mail is a common interest for all email system. Since it is the most common machine learning factor we are facing daily, it is interesting to observe how each algorithm would perform one this dataset.

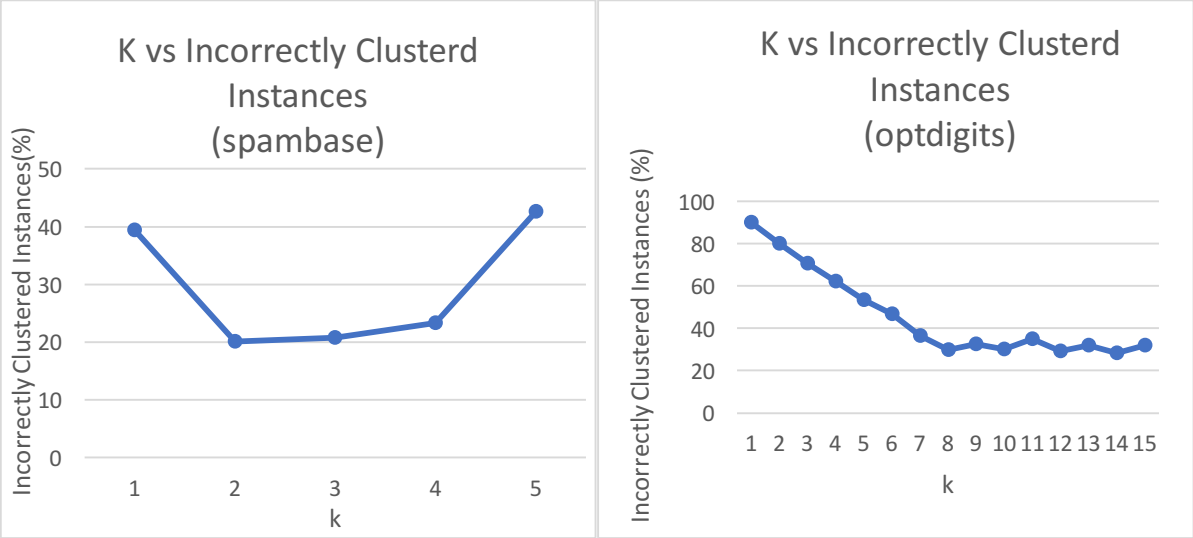
optdigits:

This dataset contains 5620 instances and 65 attributes. This is consisting of preprocessed handwritten digits from the UCI repository of machine learning databases. Handwritten digits from a preprinted form have been converted to normalized bitmaps and stored into matrix of 8x8 where each element is as an integer in the range [0 … 16]. This dataset is interesting because computer recognizing human's hand written digits is probably very close to what people would think about what artificial intelligence is like. There are so many fields where this kind of machine learning will be used, so it is interesting to see which algorithm would perform the best.

Clustering

***k*-means Clustering**

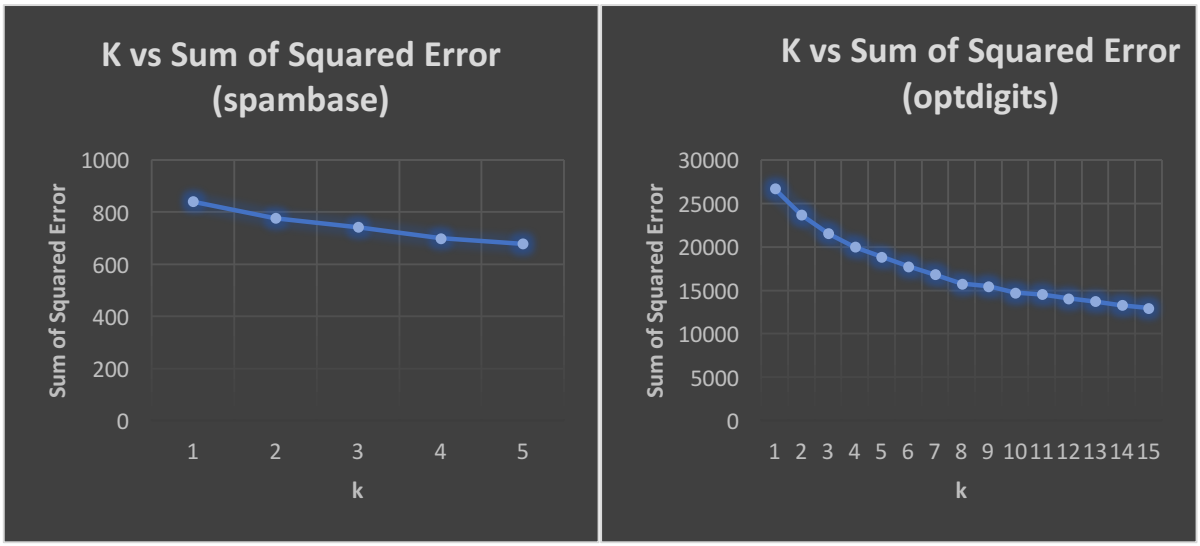
For this algorithm, Euclidian distance was used to calculated distance between instances.



To find an ideal number of clusters *k*, several times of tests were done with different value *k*. k = 1 to 5 have been input for *spambase*, and k = 1 to 15 for *optdigits*.

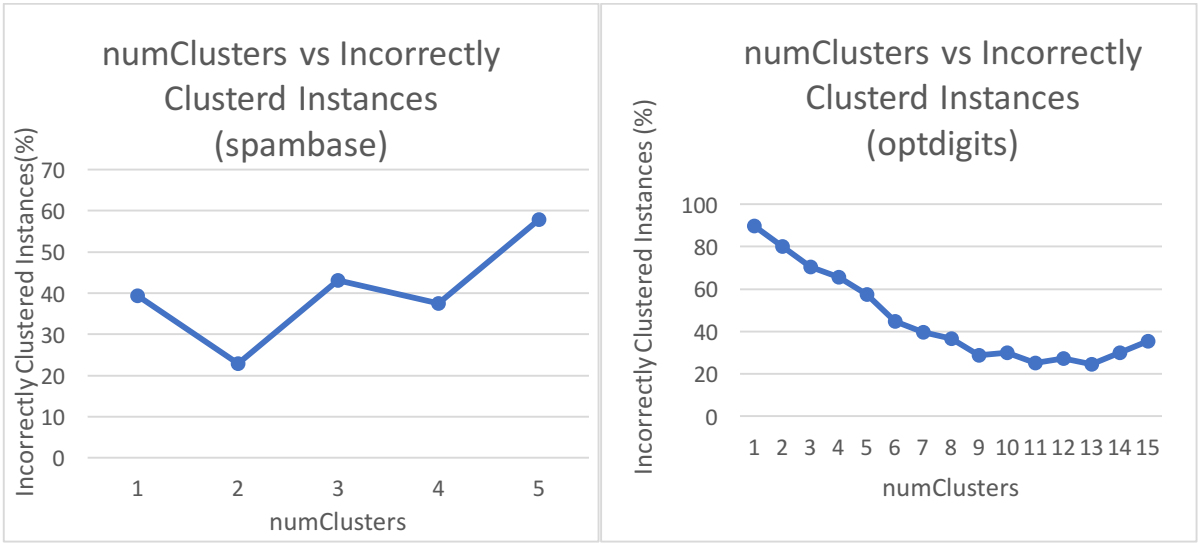
For *spambase*, as we can see from the graph above, the percentage of incorrectly clustered instances increases as the value *k* increases after *k* = 2. After it reaches that point, it drops dramatically and shows stable stage, and it increases again. Increasing number of clusters does not help it to cluster better. The result seems to be ideal since this database is binary classification.

For *optdigits*, the percentage of incorrectly clustered instances decreases gradually until *k* reaches to 8, and it shows stable stage after that point as we can see from the graph above. It showed the best result when *k* = 14 with 28.3808 % of incorrectly clustered instances. This dataset originally has 10 labels, which is [0 … 9]. However, the number of clusters *k* showed the best result is bigger than the actual number of labels. It is because people have different way to write numbers, and the data is just pixels of an image. For example, some people write 7 with a line in the middle and some do not. Since there are several ways to write numbers by hand, the *k*-means will probably put those 7 without a line in the middle and 7 with a line in two different categories. That could result higher *k* than actual labels.



As we can see from the sum of squared error (SSE) graphs, it shows the decrease as the number of clusters increases. This is the same as what I expected before running the test. It is obvious since more number of cluster will shorten the distance between center of each cluster and instances. I expect SSE would be 0 if we put k = number of instances of a dataset since each instance would have its own cluster so the Euclidian distance would be 0.

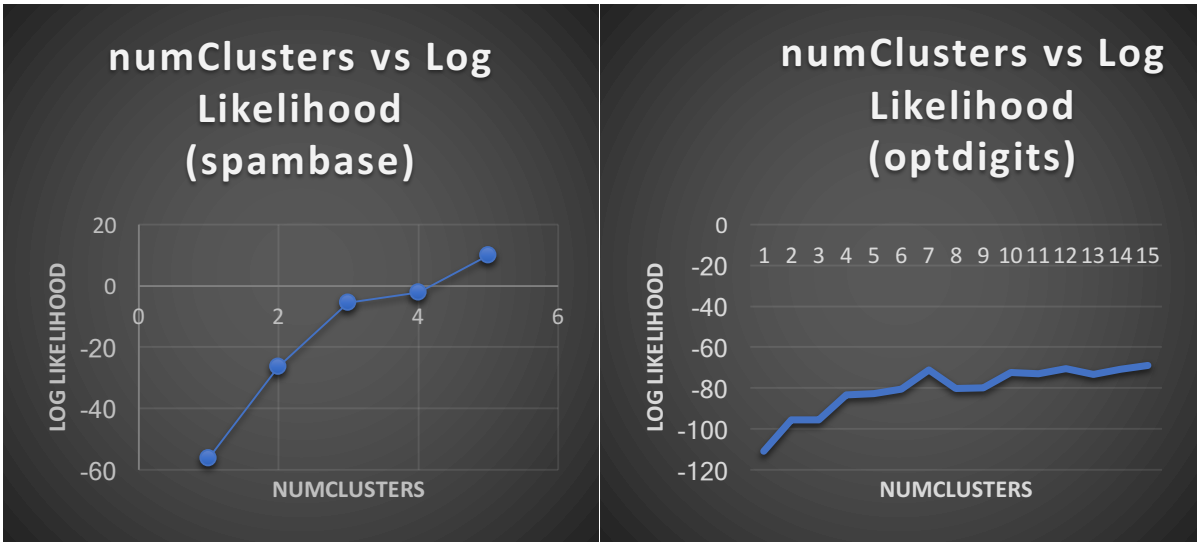
Expectation Maximization



With the similar approach as k-means clustering, different number of clusters have been input to find an ideal number. For *spambase,* as we can see from the graph above, it shows the best result when the number of cluster is 2, which is the same as k-means clustering. It is an ideal value for this dataset with the same reason as in k-means clustering.

For *optdigits,* the graph looks very similar as k-means clustering. Similarly, it gave the best result when the number of clusters was 13 with 24.6796 % of incorrectly clustered instances, and started perform badly one it passed the point. The number of clusters is bigger with the

same reason as mentioned in *k*-means clustering.



And as we can see from the log likelihood graph above, the log likely hood increases as the number of clusters increases. Since increasing log-likelihood means better results because we are more sure that our parameters for clusters are correct, the graph seems to show correct result.

k-means clustering (spambase)				
k	**numIteration**	**Incorrectly Clustered Instances (%)**	**Sum of Squared Errors**	**Build Time (s)**
2	12	20.0826	777.5437052	0.07
EM (spambase)				
numClusters	**numIteration**	**Incorrectly Clustered Instances (%)**	**Log Likelihood**	**Build Time (s)**
2	3	22.8429	-26.39379	0.91

k-means clustering (optdigit)				
k	**numIteration**	**Incorrectly Clustered Instances (%)**	**Sum of Squared Errors**	**Build Time (s)**
14	31	28.3808	13274.25589	0.86
EM (optdigits)				
numClusters	**numIteration**	**Incorrectly Clustered Instances (%)**	**Log Likelihood**	**Build Time (s)**
13	1	24.6797	-73.2539	8.45

This table above shows comparison between *k*-means clustering and Expectation Maximization (EM). As we can see from the table, *k*-means performed slightly better than EM on *spambase,* and EM performed slightly better than *k*-means on *optdigits.*

For *spambase, k*-means clustering performed better which suggests that this dataset can be simply determined whether it is spam or not by simple subset of keywords.

For *optdigits,* EM performed better since the way EM handles whether an instance is in a cluster or not is soft unlike *k*-means clustering. *K*-means clustering is hard assignment which explicitly sets an instance to a cluster. On the other hand, EM calculates the probability that each instance is in each cluster and chooses the most likely one. Since this dataset contains many blurry instances (i.e. 7 example given in *k*-means), giving probability would perform better than explicitly putting in different categories.

I expect both algorithms would perform better with a dataset which contains attributes with small variance and small covariance with other attributes, since redundant attributes will not be in the dataset. This will shorten the time to compute distances or probability.

Dimensionality Reduction Algorithm

Principal Component Analysis

For this part, I used 4 different variances covered value to observe how this various input would influence on the result. For *k* and number of clusters value, I used the values found from Clustering section, which gave us the lowest incorrectly clustered instances.

spambase (58 attributes, Number of Instances: 4601)					
k-means clustering (k = 2)					
variance covered	numIteration	numAttribute	Sum of Squared Error	Incorrectly Clustered Instances (%)	Build Time (s)
0.25	15	6	61.6439779	19.0828	0.02
0.5	17	17	206.738441	17.9092	0.04
0.75	13	32	323.2676252	18.5829	0.08
0.95	16	49	566.3211482	20.7564	0.09
Expectation Maximization (numClusters = 2)					
variance covered	numIteration	numAttribute	Log Likelihood	Incorrectly Clustered Instances (%)	Build Time (s)
0.25	16	6	-7.52479	48.1852	0.22
0.5	15	17	-21.31495	47.3375	0.63
0.75	24	32	-37.4544	43.1645	1.06
0.95	20	49	-55.24697	42.2734	1.76

As we can see from the table above, *spambase* shows better performance on *k*-means clustering. For *k*-means clustering, it performed the best when the variance covered was 0.5, with 17.9092% inaccuracy (20.0826% without PCA). It seems the dataset contains a lot of redundant or irrelevant attributes. Thus, it seems performs better with the lower number of attribute than original because it makes *k*-means clustering easy to cluster those attributes without redundant. On the other hand, Expectation Maximization (EM) performs poorly with PCA. The lowest percentage of incorrectly clustered instances with PCA was 42.2734% (22.8429% without PCA). Lowering number of attributes does not seem to make it performs better. EM calculates the probability that each instance is in each cluster and chooses the most likely one, but it seems like it has difficulty with putting probability to instances with lower number of attributes. Statistically, probability is more accurate with more samples so the result seems to be correct.

optdigit(65 attirubutes, Number of Instances: 5620)					
k-means clustering (k = 14)					
variance covered	numIteration	numAttribute	Sum of Squared Error	Incorrectly Clustered Instances (%)	Build Time (s)
0.25	13	4	256.4338836	80.2847	0.01
0.5	11	9	522.6436151	80.3203	0.02
0.75	12	20	644.8544224	80.3203	0.05
0.95	39	43	991.7709505	35.3025	0.72
Expectation Maximization (numClusters = 13)					
variance covered	numIteration	numAttribute	Log Likelihood	Incorrectly Clustered Instances (%)	Build Time (s)
0.25	13	4	-6.39994	40.694	4.46
0.5	45	9	-14.05572	42.3488	4.5
0.75	70	20	-26.80944	41.6904	9.72
0.95	55	43	-47.24881	44.3238	18.73

For *optdigits*, it performed very poorly on both *k*-means clustering and EM as we can see from the table above. For *k*-means, the lowest percentage of incorrectly clustered instances with PCA was 35.3025% with variance covered 0.95 (28.3808% without PCA). It shows around 80% of inaccuracy when the variance is low. For EM, the lowest percentage of incorrectly clustered instances with PCA was 40.694% with variance covered 0.25 (24.6797% without PCA). Both algorithms with PCA on this dataset performed poorly because this dataset has 10 labels and there are only few redundant or irrelevant attributes in the dataset.

Independent Component Analysis

Randomized Projections

For Randomized Projections, 3 different number of final attributes and 3 different seed have been applied as inputs. For *k* and number of clusters value, I used the values found from Clustering section, which gave us the lowest incorrectly clustered instances. For each number of final attributes, I took the average of 3 to observe how it performs.

spambase (58 attributes, Number of Instances: 4601)					
k-means clustering (k = 2)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	42.6864	22.5386	36.4921	33.9057	0.04
20	42.1213	36.4269	36.4486	38.33226667	0.05
35	46.0987	36.4486	36.2747	39.60733333	0.04
Expectation Maximization (numClusters = 2)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	38.5351	29.4067	31.8626	33.26813333	0.4
20	29.8631	29.6675	30.3847	29.97176667	0.71
35	45.6857	29.8413	30.7759	35.4343	0.77

optdigit(65 attirubutes, Number of Instances: 5620)					
k-means clustering (k = 14)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	42.9537	52.1352	41.9751	45.688	0.45
25	42.9181	42.9181	38.2384	41.3582	0.56
50	30.6406	30.6406	42.7758	34.68566667	0.75
Expectation Maximization (numClusters = 13)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	42.3488	50.1068	42.6157	45.02376667	5.23
25	23.6299	36.6548	38.3274	32.8707	11.91
50	22.758	27.758	26.4769	25.6643	18.76

As we can see from the table for dataset *spambase,* both *k*-means and Expectation Maximization performed poorly. We can see some of low inaccuracy on the table, but on

average it does not show good performance. Since Randomized Projection (RP) projects high dimensional data onto a lower dimension space using a random matrix, we can assume it lost some required information when is mapping to low. The inaccuracy was the lowest when the final number of attribute was 10, and increased after reached the point. The results seem to very similar as the other algorithms. For *spambase,* when the number of attributes increased more than certain point, the inaccuracy started increasing, and vice-versa for *optdigit* with the same reasons that are mentioned in previous parts.

Random Subsets

I chose Random Subsets as the fourth algorithm for dimensionality reduction algorithms because I thought it would be interesting to compare with other algorithms, especially with Randomized Projection. For *k* and number of clusters value, I used the values found from Clustering section, which gave us the lowest incorrectly clustered instances. For each number of final attributes, I took the average of 3 to observe how it performs.

spambase (58 attributes, Number of Instances: 4601)					
k-means clustering (k = 2)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	25.2336	40.1869	34.4708	33.2971	0.02
20	22.8646	34.7968	49.8587	35.8400333	0.05
35	21.93	26.1248	45.9031	31.3193	0.06
Expectation Maximization (numClusters = 2)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	44.6207	24.2991	37.5788	35.4995333	0.22
20	19.7783	40.1869	33.8839	31.2830333	0.37
35	33.0363	33.6231	32.2756	32.9783333	0.26

optdigit(65 attirubutes, Number of Instances: 5620)					
k-means clustering (k = 14)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	66.2633	59.4662	54.7331	60.1542	0.23
25	43.0249	41.8149	33.4342	39.4246667	0.36
50	28.0605	26.8683	27.669	27.5326	0.55
Expectation Maximization (numClusters = 13)					
	Incorrectly Clustered Instances (%)				
attribute	seed: 42	seed: 71	seed: 100	Average	Average Build Time (s)
10	65.7473	53.8968	58.6655	59.4365333	2.59
25	46.4235	46.9929	45.6584	46.3582667	5.89
50	33.9858	31.6904	27.4377	31.0379667	8.47

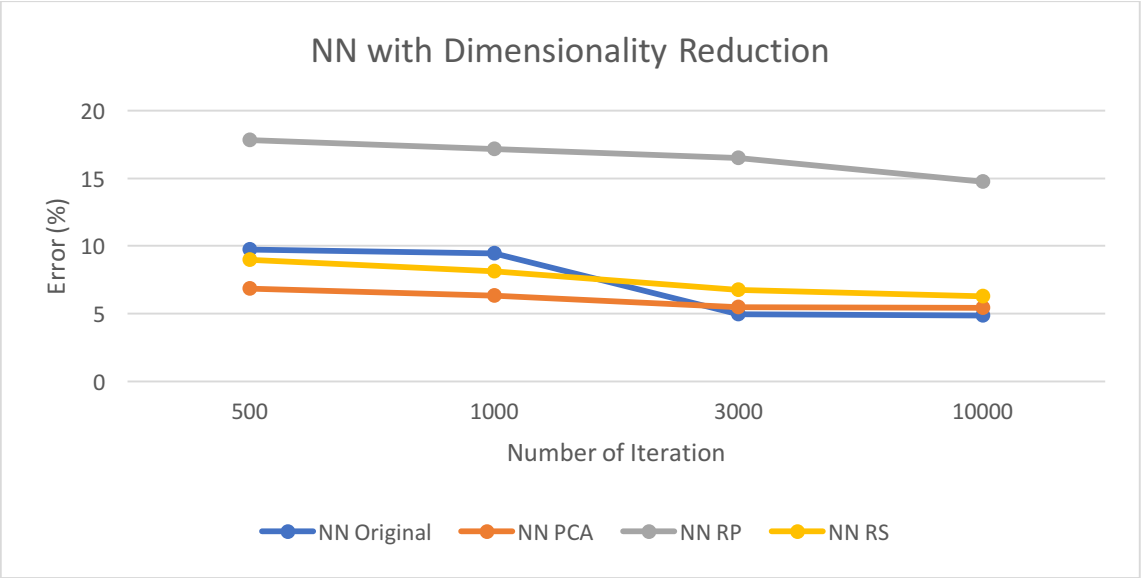
Disappointingly, or as expected, it did not perform any better than Randomized Projection.

Some of lower inaccuracy can be seen from the table above, but it is not significantly better. However, it is somewhat expected because it is hard to imagine that randomly picking attributes to keep does a better job than reducing dimensionality with a random matrix. Sometimes it shows really accurate result for *spambase,* when seed is 42 and the number of attribute is 35 for both *k*-means and EM, but it shows poor performance when it is averaged out.

Neural Network

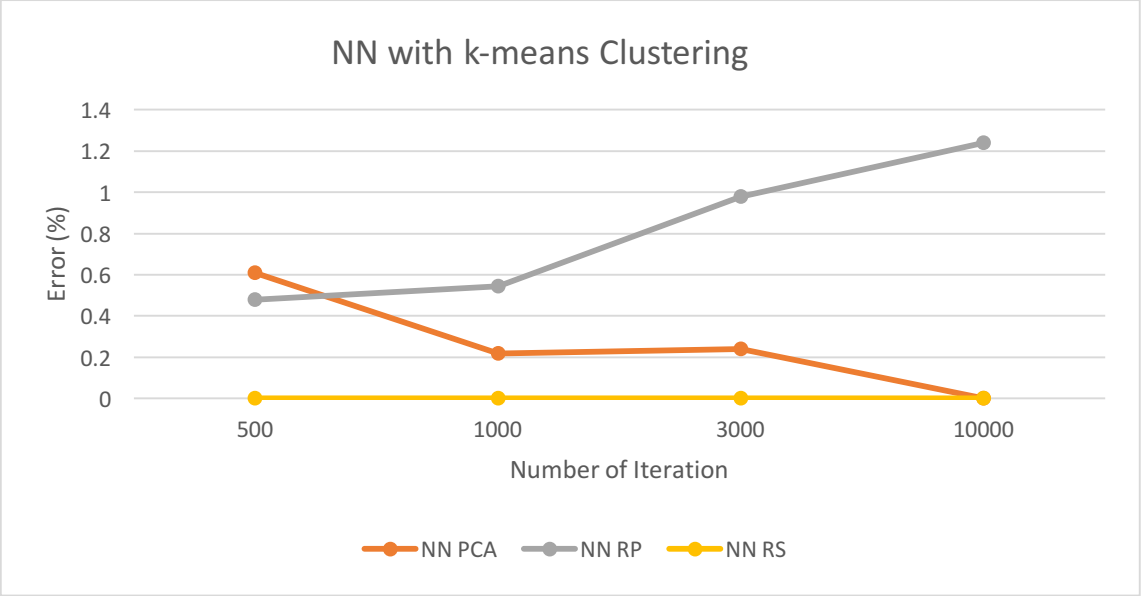
For Neural Network, *spambase* was used for the dataset and I used L = 0.1 and M = 0.2 for the input since I got the best result with those inputs in Supervised Learning assignment. I could not find how to run ICA, so it was excluded from experiment.

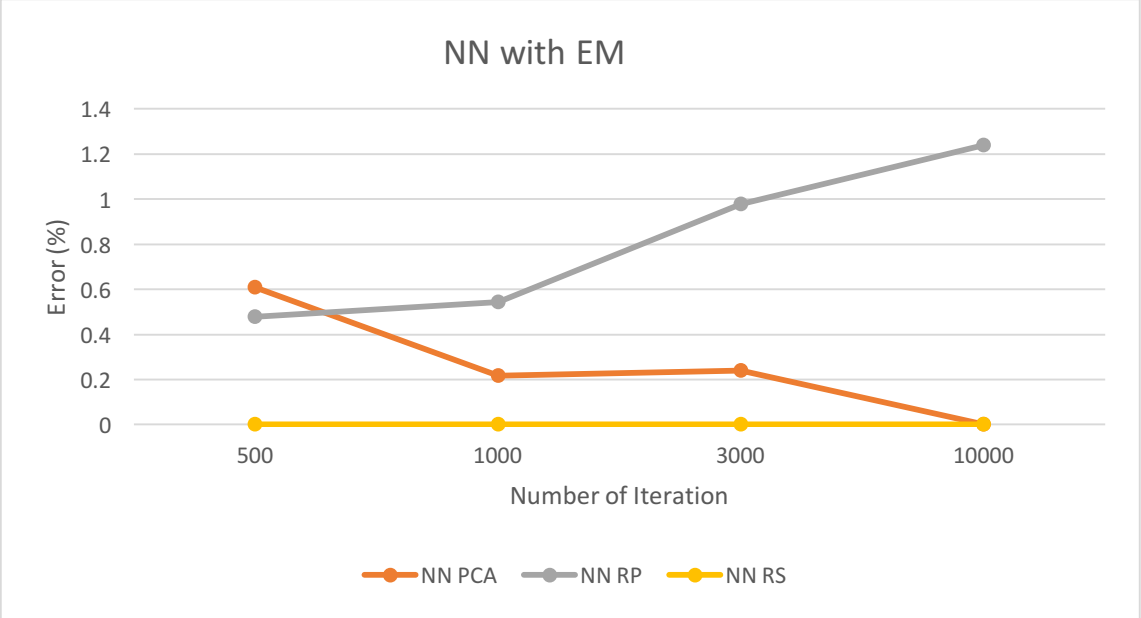
Dimensionality Reduction



As we can see from the graph above, running neural network with dimensionality reduction, most algorithms showed very accurate results. Running neural network after dimensionality reduction is an interesting idea. Since training time decreases as the number of resultant attributes decreases after dimensionality reduction, it shortened the training time. It could perform better since it needs less data to fit a function. It seems PCA and RP work better when the epoch is small, but when 3000 was given as an input the original result, without dimensionality reduction, showed better accuracy over all the algorithm. However, it is still impressive because the time taken to run neural network with dimensional reduction got shortened out a lot. While PCA, RP, and RS are showing find performance, RP is performing poorly. It is interesting because RP and RS performed similarly in the previous part. It seems there was a problem when RP projects high dimensional data onto a lower dimension space using a random matrix.

Clustering





As we can see from the graph above, neural network with clustering performs very well. Not only it shortened the time of the build time, but also it shows very low inaccuracy. However, we can observer overfitting on RP. As the number of epochs increases, the error percentage goes up. It is interesting because I thought I would not see any overfitting because the dataset has been clustered already. It might be there was a problem when RP projects high dimensional data onto a lower dimension space using a random matrix, and that might have affected on the dataset after the dimensionality reduction. In Dimensionality Reduction part, RP performed very poorly compared to other algorithms.

Overall, I think running neural network after dimensionality reduction and clustering is very efficient way to achieve a goal. It tremendously decreased the time to build, and it shows very find performance other than RP. Especially after clustering, the error rate was

decreasing very close to 0 as the number of epoch increased. Therefore, I think it is a good approach not only for time efficiency not also for accuracy.