Dong Son Trinh
CS 4641
Assignment 3

Unsupervised Learning

Datasets (both datasets used in Assignment 1):

Adult Data Set:

Predict whether income exceeds $50K/yr based on census data. Also known as "Census Income" dataset. This is dataset has binary output/labels, but the features are mostly discrete, so I use One Hot Encoding on them, which significantly increases # feature. This could be an interesting dataset for dimensionality reduction since from assignment 1 I got that many features could be dependent or useless from feature selection. Overall, there are ~32000 data instances and 108 one hot encoded features. This dataset will be used for parts 4 and 5.

Wild Data Set:

This data set contains data from a remote sensing study that involved detecting diseased trees in Quickbird imagery. There are few training samples for the 'diseased trees' class (74 instances) and many for 'other land cover' class (4265 instances). Compared to the Adult dataset, this one has a much more unbalanced number of instances belonging to each class (74 vs 4265). Also, since all features have continuous data, there is no one hot encoding, so we end up with only **5** features. This will prove to be not quite effective for dimension reduction, which is why this dataset will not be used for rerunning Neural Network tests from assignment 1.

Some things before analysis:

Adjusted Rand Score evaluates the similarity of 2 clustering, output is in range [-1, 1]. 1 means that they are exactly identical, 0 means that they are independent of each other and below 0 means (in my understanding) that there is some pattern to mismatching.
K-Means 2D projection visualization code
http://fromdatawithlove.thegovans.us/2013/05/clustering-using-scikit-learn.html
Expectation Maximization 2D projection visualization code taken and modified from:
http://scikit-learn.org/stable/auto_examples/mixture/plot_gmm.html

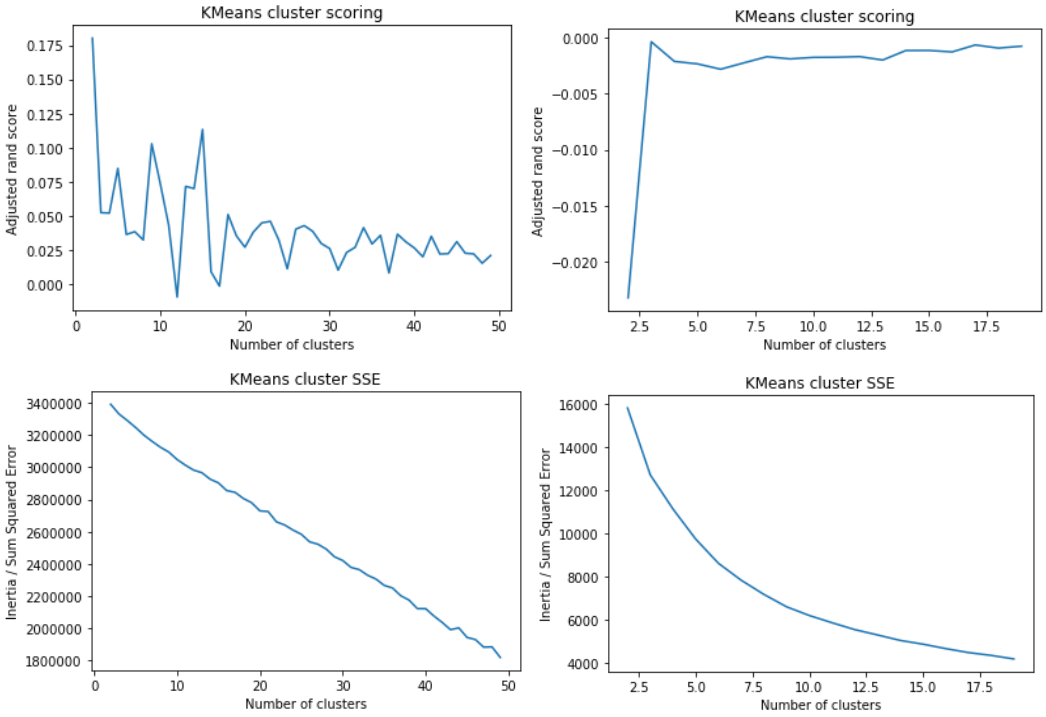
Clustering Algorithms

To start off, I ran couple of experiments on clustering algorithms K-Means and Expectation Maximization. For both, I tweaked # of clusters/components hyperparameter and # maximum iterations. As a metric for the performance (actually used throughout the assignment) I used Adjusted Random Score, which calculated similarity between two clustering labels. As the “other” clustering I used the classification labels of my dataset. Also, in clustering experiments I used inertia (Sum Squared Error) and log likelihood respectively for KMeans and EM. While they do not show the exact performance of clustering, any divergence from “normal” outputs of those metric showed incorrect or interesting behavior of clustering.
Unfortunately, when changing maximum iterations did not yield any interesting results because when I put a constant random seed into the algorithm, it always converged in < 50 iterations. This being so, I decided to leave the graphs out as I already have too many graphs to include.

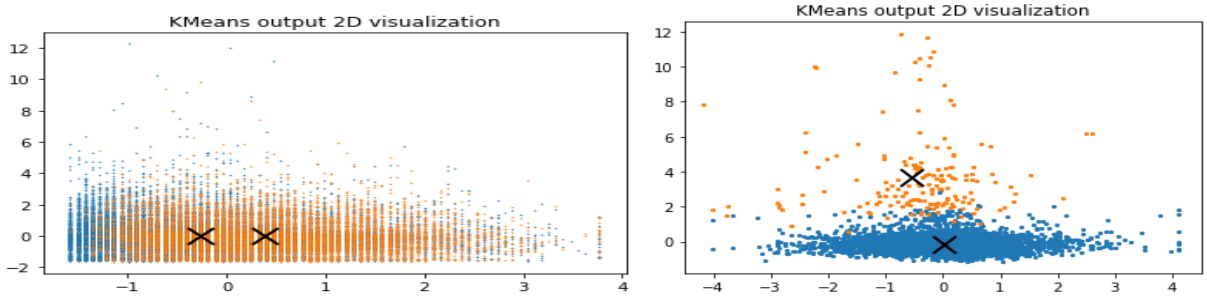
K-Means

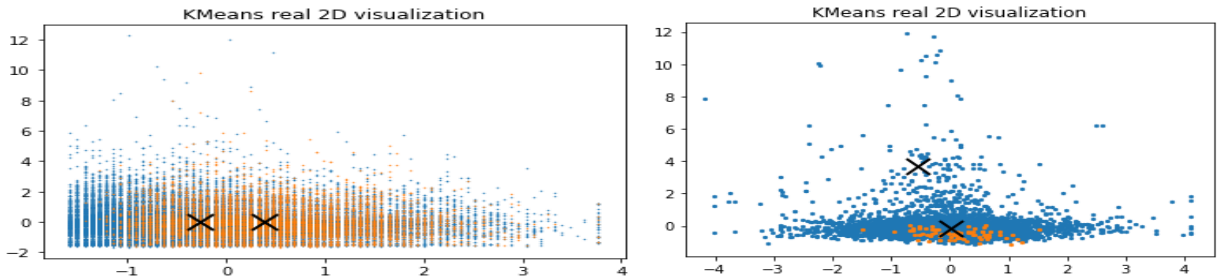
To start off, I want to remind that both of my datasets have binary classification, so I expected optimal # of clusters to be around 2. Interesting thing about Adult dataset is that most of 108 features are created from

one-hot-encoding a feature about person's geographical location. So either those features will all be crucial or useless. From the graphs below, **2 clusters** have best score, meaning that the data seems to be linearly separable. Later the score stabilizes around 0.04. Spikes can be easily explained by random initialization of centroids even though it reinitializes 10 times for each instance of KMeans. Meanwhile, the scoring of Wild dataset is very bad, below 0 (inconsistent with other, but with pattern). ~ -0.023 as an absolute value is still very close to 0 which shows independence of clustering. Interestingly, inertia still decreased as expected on both datasets which makes sense because with more clusters it is possible to cover small with more areas, basically meaning that sum of distances to centers will be less (really far points will go another cluster instead of creating more error for cluster). Back to rand score, low score on Adult dataset can also be explained by that **all** features contribute to distance, meaning that many useless features will influence badly the clustering. Also, both datasets are not balanced: Adult is ~32000:4000, Wild is ~4200:76.



For further analysis, I projected the data into 2D space to look at actual clustering of data. First, considering that Adult is imbalanced, the clustering is **very** good because the clustering seemed to balance out the datapoints belonging to 2 clusters, while the real labeling (painted according to dataset classification data) basically does the same, just covering points. From this much, having ~0.18 rand score I would consider very good, which later will score high for supervised learning. On the other side, Wild completely misclassifies datapoints. While I can't tell whether the coloring actually correspond to each other in 2 pictures, in both cases poorly because **most** point reside ~0 on y-axis, where on the real visualization getting correct clustering is literally impossible because the centroid would much more points that expected.



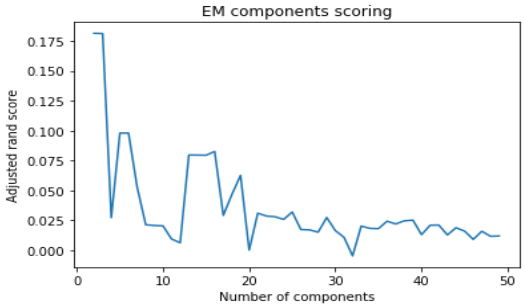


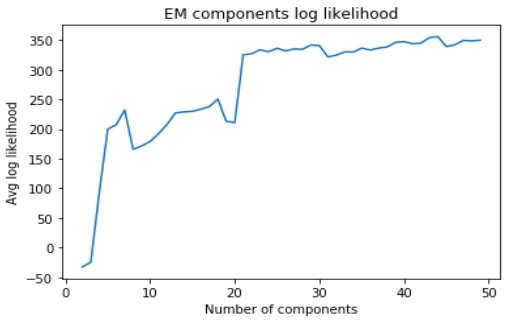
To sum up, KMeans works quite well for Adult despite bad score, while Wild dataset is just incompatible with KMeans no matter how many clusters are put (need to consider that this is 2D and in actual dimensions it might look different, but the projection onto 2D does a good job of reflecting whether it is actually possible to cluster in high dimensions).

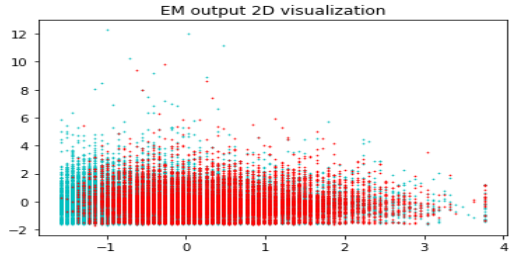
Expectation Maximization

Main difference between KMeans and EM is that EM is soft clustering and KMeans – hard clustering. It is of no great relevance to Adult because clustering in KMeans already created a somewhat ellipsoid form of points in its cluster and the problem it had was not the bordering point between clusters. As expected, it performed similarly, ~0.175, again best on 2-3 clusters. However, EM is much better for Wild because flatter ellipsoid form restricts fatal misclassification of more datapoints. In the 2D picture of Wild output clustering much less points (teal) are around ~2 y-axis. Also, since the center/mean is calculated in a probabilistic way in EM, the means of 2 cluster end up being **very** close to each other ➔ EM deals with insanely many bordering points better than KMeans. All this contributes to the ~0.12 rand score for Wild on 2 clusters. Obviously, more clusters here will not work because in 2D (fairly correct representation) most data (yellow) is centered around same place.

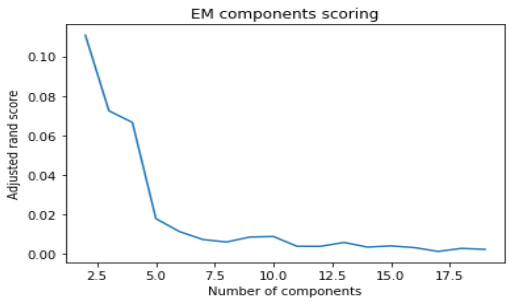
Adult

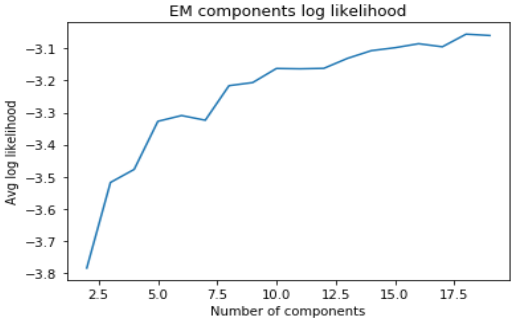


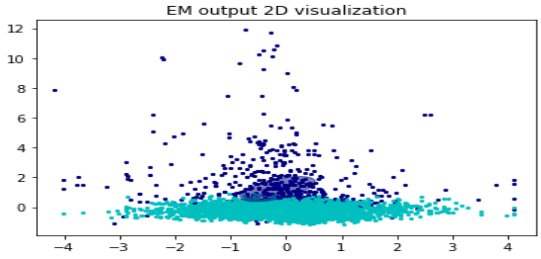


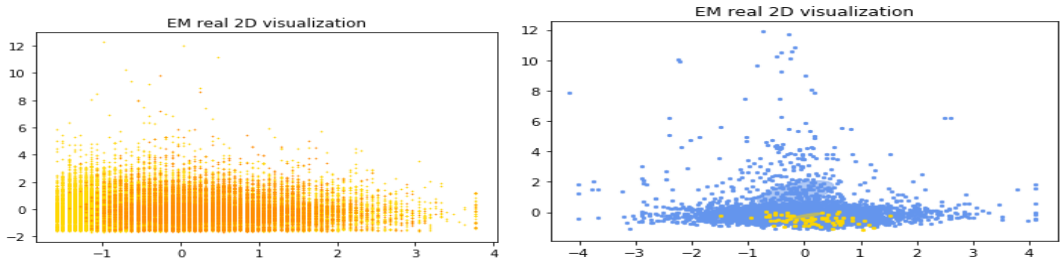


Wild









Similarly to decreasing inertia of KMeans, average log likelihood always increases with more clusters used because more clusters will decrease the distance to closest cluster ➔ higher likelihood to belong to it. And if we consider that likelihoods of points are usually likelihoods of closest cluster, we obviously increase average log likelihood with more clusters.

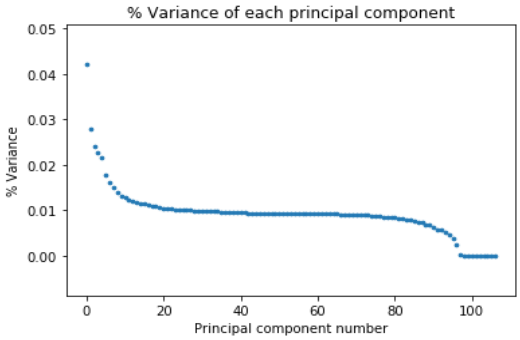
Finally, clustering algorithms on **raw** data do not show representation of data (especially that matrix-like Adult). In next experiments, I will not include inertia and likelihood graphs because they pretty much always increased and decreased respectively without any anomalies to avoid using too much space in the report. In general, EM runtime was much longer than KMeans, so I had to tune the hyperparameters not on every value of parameter. This makes sense as KMeans just calculates the distance functions, while EM does heavier probability-based calculations. Additional things I would consider to explore clustering would perhaps trying to change around the convergence rate to increases or lessen the sensitivity or stopping condition, increase number of initializations to get the best starting condition (best cluster centers). For EM, looking at and visualizing the covariances matrixes.

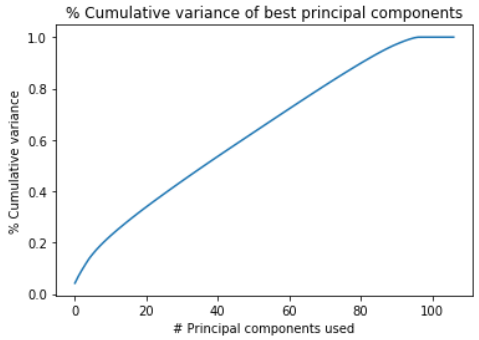
Dimensionality Reduction

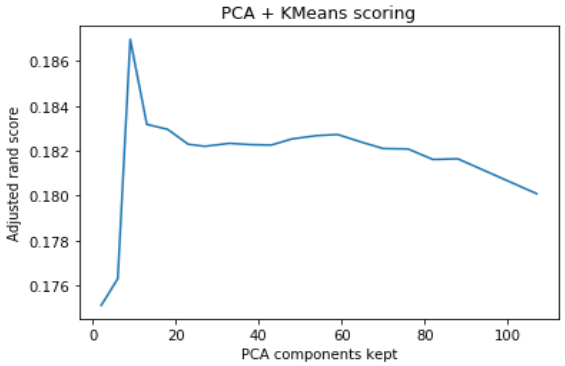
On all algorithms I ran the transformation and then clustering because that is a more efficient (and sometimes sole in sklearn) way to describe it. Except for PCA, the output models did not offer much to play with in sklearn. For all algorithms, I changed the # components/target dimensions as a hyperparameter and evaluated them based on rand score and 2D visualizations.

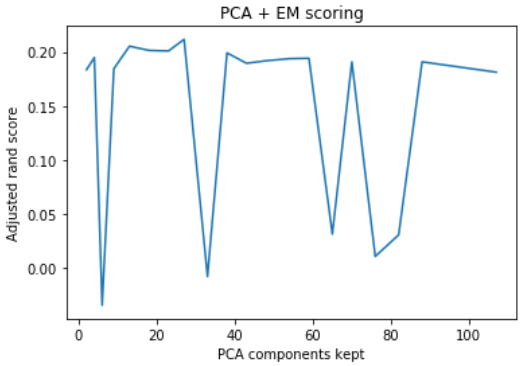
PCA

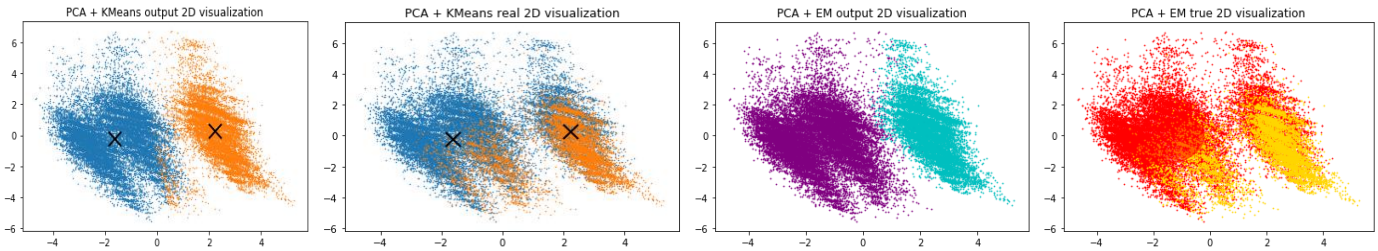
First off, what I did was run PCA to get all the components that account up to 100% variance (eigenvalues). As seen for Adult dataset, there are no principal components that account for more than 5% of variance. Interestingly, most of PCs are ~0.01 ➔ responsible equally for variance. This makes sense because many features were generated one-hot-encoded and directions of PCs onto which data is projected show that quite well. But the catch is that from KMeans scoring, optimal #PCs is 9, which accounts for ~20% of variance. Surprisingly, EM performed quite consistently higher than on raw EM. Now, if we look at the 2D projections with **20% variance** retained in transformed data, it turns that awful matrix into great circular clusters! The real clustering on 2D captures the clustering similarly to output despite some errors so I would say that PCA did a great job. This shows that some PCs, despite not highest variance distribution, are quite significant and some would not improve the clustering on 2D projections. Especially EM was able to utilize the bordering points more efficiently after transformation and dimensionality reduction of data as seen more vividly on these projections than on projections of raw data.



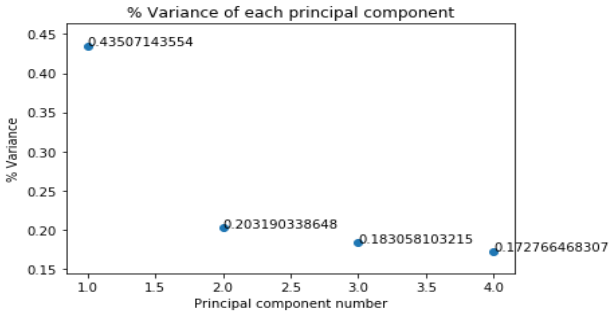


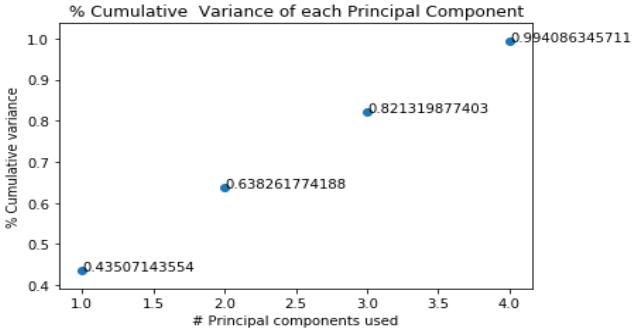


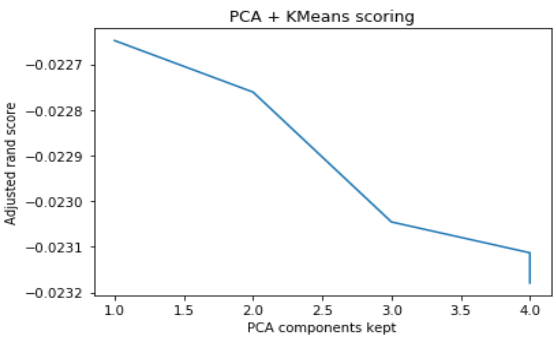


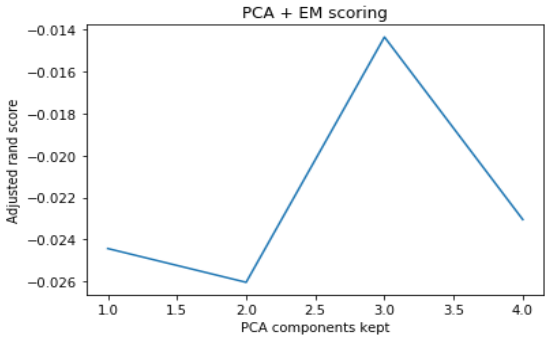


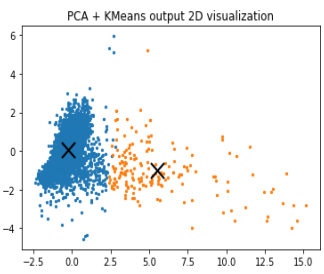
Next on the Wild dataset (which has 5 features), PCs obviously occupy much more variance than on Adult dataset. Unfortunately, as I tried changing #PCs used for 2D projection, the rand score did not improve and the projection and clustering looked similar to each other. Remembering that this dataset did terrible on KMeans, this does not change at all in this algorithm as PCA does not do well on data that cannot be linearly separated by the factors of variation/eigenvectors. In this example it also shows that PCA maintains “best” projections, but its best is not always consistent with the experiment PCA is used on. This can be seen in this example as transformed data actually does poorly on EM as well as seen from negative close to 0 rand scores while originally EM on raw data achieved ~0.1 rand score. Overall, PCA shows possible difference of performance based on the structure of datasets which is one of weaknesses of the algorithm.

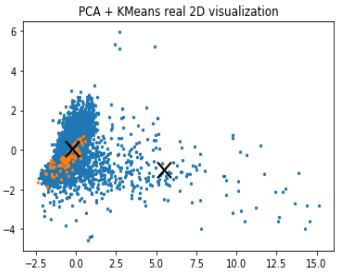


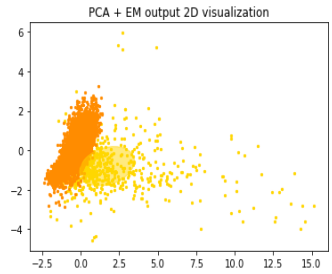


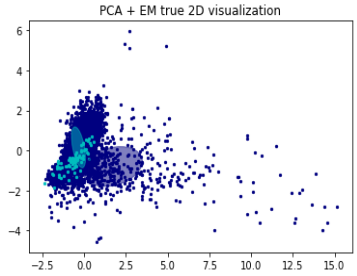






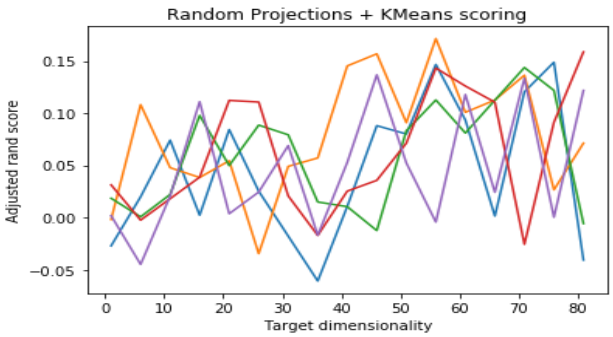


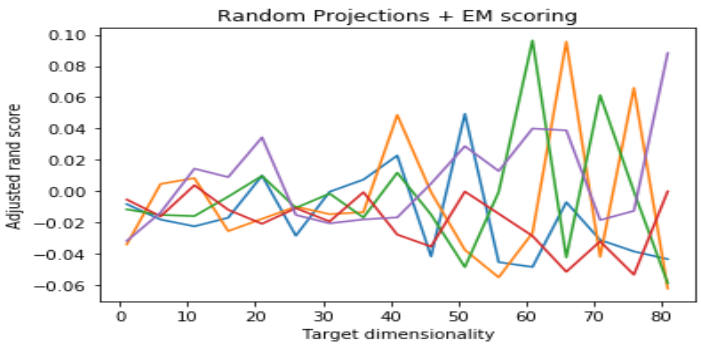


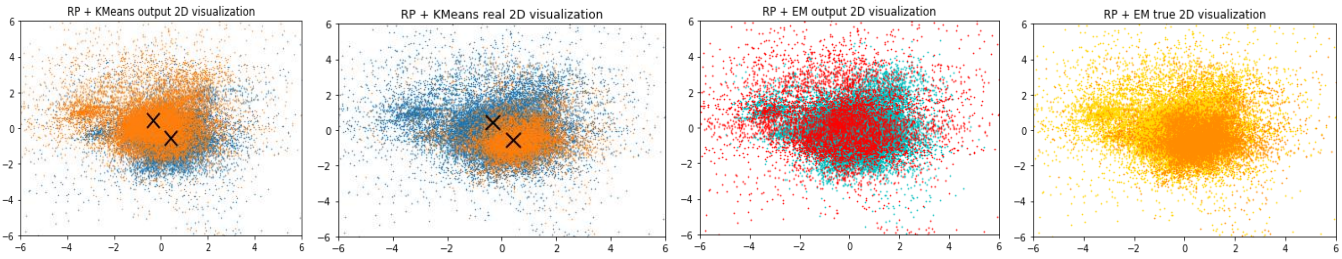


Random Projections

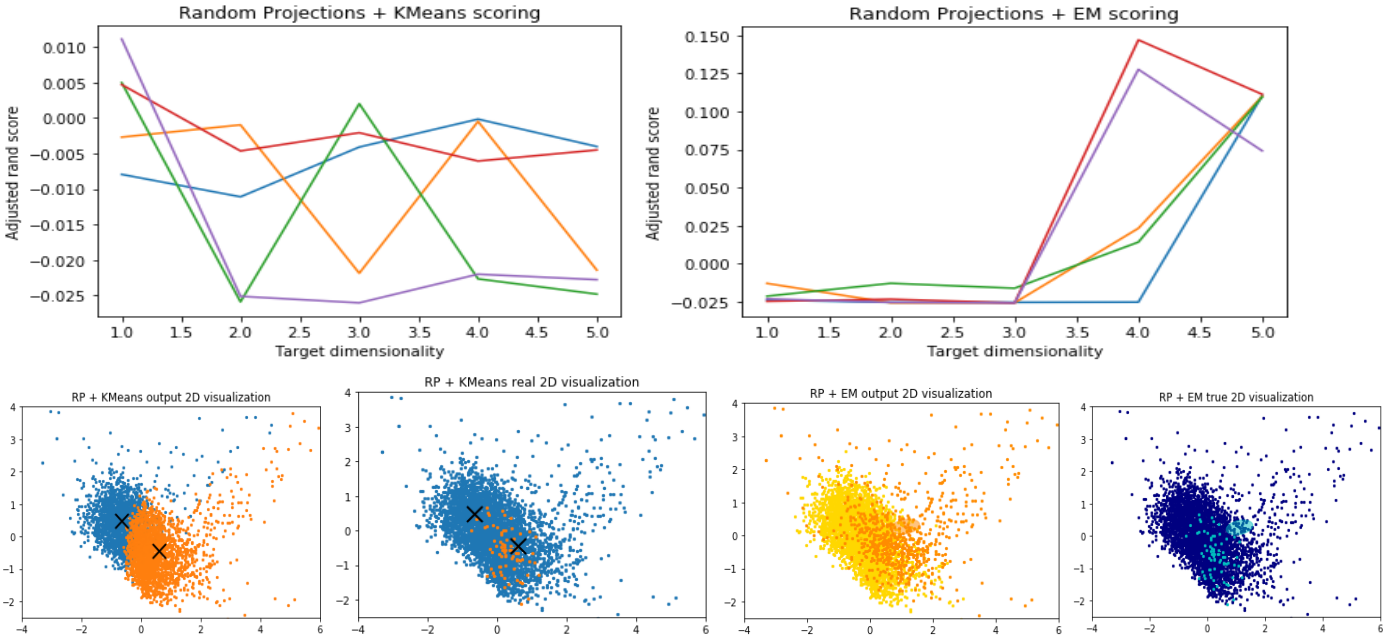
One thing to note right away is that RP is **much** faster than PCA calculation even when I ran it 5 iterations of it. That being the case, I had a lot of time to actually re-run the whole experiment on both datasets and most of times the results were truly random, but there were some patterns worth mentioning (the graphs presented are the "best" result I got from running RP multiple times). The Johnson-Lindenstrauss lemma which basically says that if data is projected onto subspace of suitably high dimensions, the distance structure will be preserved well. Looking at results of Adult on KMeans and especially EM, there seems to be a pattern of each run (1 color – 1 iteration) seems to move around some constant value (~0.05-KMeans, ~0-EM). But it seems like EM on this transformed data performs relatively poorly. If we look at the 2D projections of the data (ran on 60 target dimensionality) of output vs actual clustering, transformed data is densely concentrated in a circular data, though very evenly. Compared to PCA, it places data so that the cluster centers are **very** close to each other, making one cluster like a "subset" of another. If we remember the raw data visualization, we saw exactly something like this in the "matrix" pictures. This means that RP managed to conserve the structure of data very well. While PCA may have separated the data with the principal components, RP just concentrated data just like the raw data, but we cannot tell which is better before inserting the transformed data into Neural Net. However, on average the rand score of PCA seems better which makes sense as RP does well on average, not the "best".







On the Wild dataset, the same problem of KMeans resurfaces no matter the data transformation. As seen on rand score for KMeans, the score does not go far from 0 and this kept on happening when I reran the experiment on Wild as all 5 iterations graphs stay around the same. RP did not manage to concentrate all real datapoints densely, but the clustering does catch all of them though with too many wrong points. Key detail about RP is that it managed to conserve some structure of original data which PCA **failed to do**. If we look at the ellipsoid of the output clustering, it seems to be on the edge of the ball of data, which makes it catch all the point far from the ball itself, but the bordering points are handled well again by EM. Compared to real labeling, it does approximately well, except those far away points, which we could consider noise, seeing how they did not end up in the dense area. I need to mention I got the ~0.125 rand score after multiple runs, as the average best score often was much worse and the 5 runs were not so consistent with each other from Target Dimensionality 1-3.

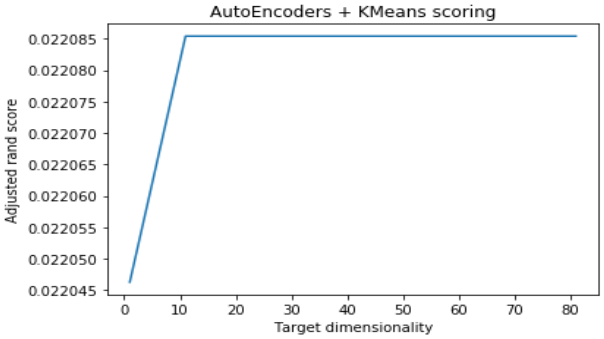


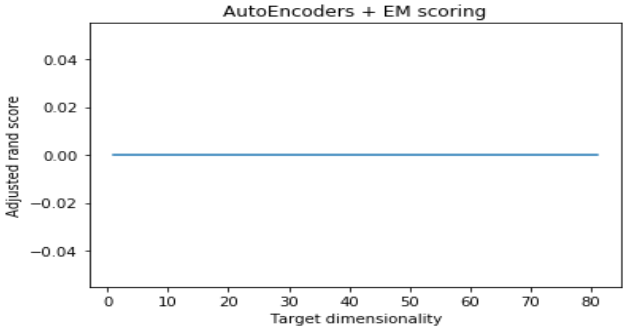
Finishing on RP analysis, the performance of the algorithm heavily relies on restricting the dimensions to high enough number to conserver structure as seen on Adult dataset. But optimization and hyperparameter tuning is very easy as the runtime of the algorithm is very small, making it more efficient in terms of time. Still, the results seem to be very random and the variance in rand score is very high from run to run and on each iteration as seen from graphs. But as we will see later, consistent average performance + good representation of original distance (compared to PCA) in lower dimensions will give better actual results in supervised setting.

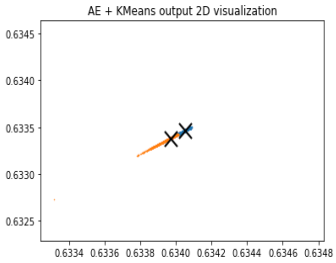
AutoEncoders

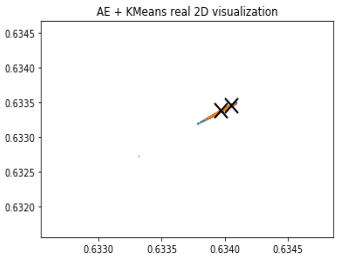
This particular dimension reduction algorithm was harder to evaluate in terms of rand score. What I did for both datasets was have the AE layers structure:
[input(#original features), const low number of nodes, varying # nodes]
I tried running with hidden layer nodes = 1, 2, half of input features, but the difference was not that different in terms of rand score on KMeans and EM. But the problem was that visualization was particularly

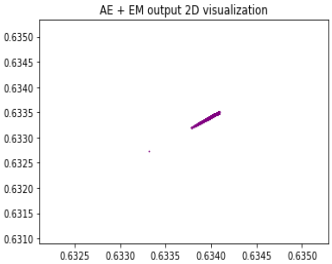
confusing. To ensure correctness of activation functions (sigmoid used on all), I made sure that all values were <= 1, so I divided whole data by max(dataset_value). This gave result of the pencil like looking projection. Considering that there are ~36000 datapoints, doing this makes absolutely no sense because as I zoomed in, I could not see anything worth analyzing. Also, on the AE + EM output projection, the clustering turns out to have colored **all** datapoints into 1 cluster (this can be seen as rand score is always 0, meaning that clustering are independent of each other). While KMeans managed to at least divide them into 2 visible clusters, the data transformation seems very densely concentrated. Therefore, I decided to insert datasets into AE without making sure that all values were <= 1 because maybe that would give some more interesting graphs to work with.

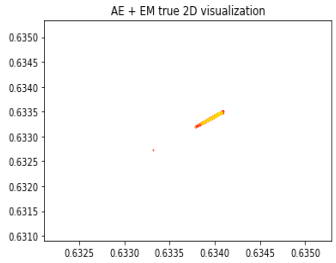




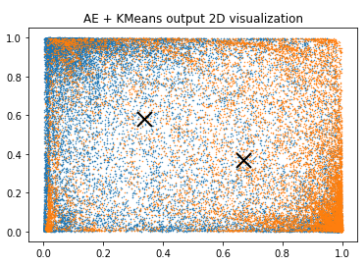


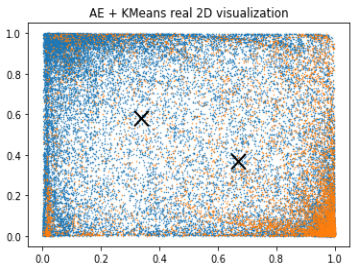


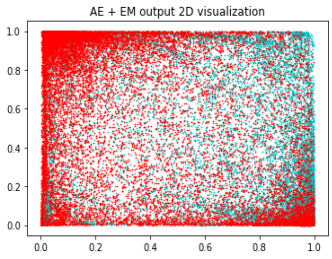


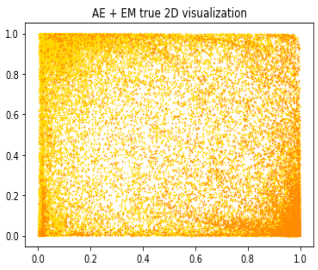


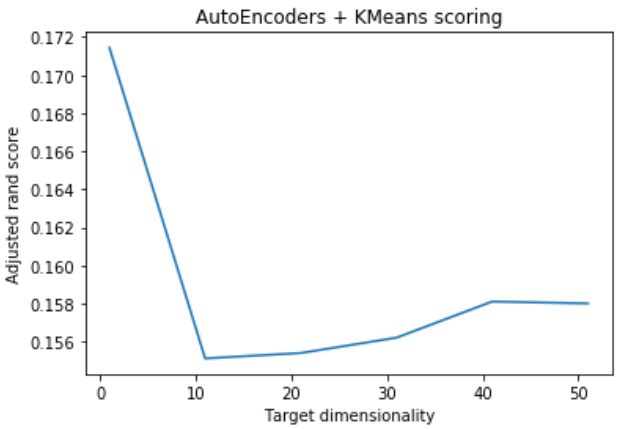
One thing worth mentioning right away on the Adult dataset is that as expected, a lot of points get accumulated on the borders of x=1 and y=1, but in defense I will say that the clustering on AE + EM is not unary as before. Later I will also prove that the accuracy of supervised learning will be higher with this approach as well. Of course, another thing to mention is that the performance on KMeans and EM are both very good and comparable with previous dimensionality reduction algorithms.

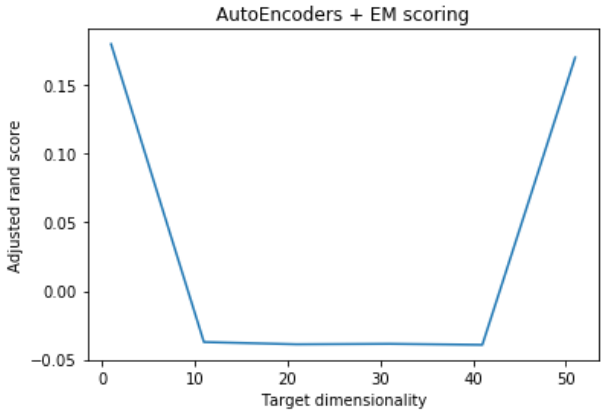




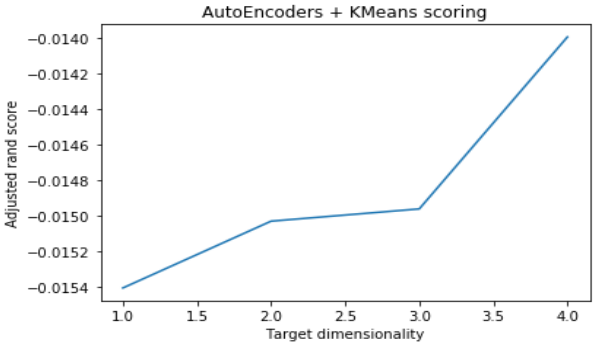


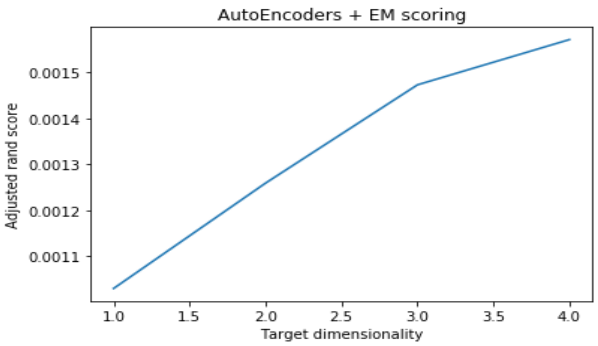


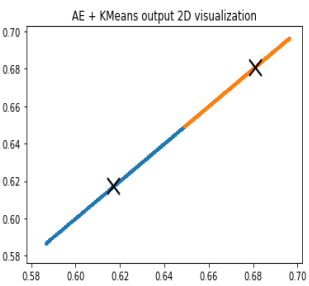


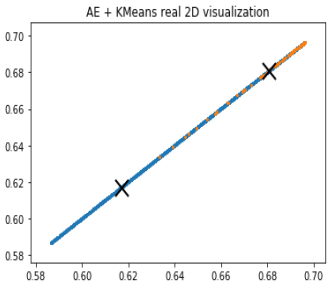


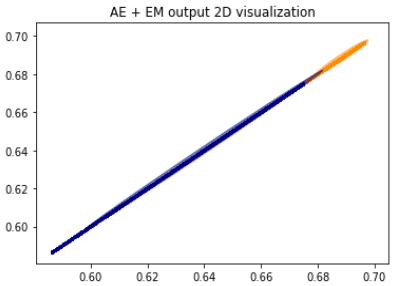
On the Wild dataset the scoring was very bad as well. This seemed to me like a sign that adjusted rand score perhaps was not compatible with AutoEncoders or, more likely, visualizing transformed data is not useful. If you look at the projections of data in KMeans and EM, they also like a thick line slope 1. No matter how many nodes I placed in the hidden and output layer, the result was always this type of line. Even if there are ~4200 compared to ~36000 datapoints of Adult dataset, the distribution of features does not seem sparse enough to make any strong conclusions. However, despite bad scores, “visually” the clustering does not seem incredibly off.

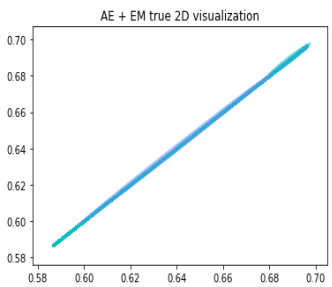








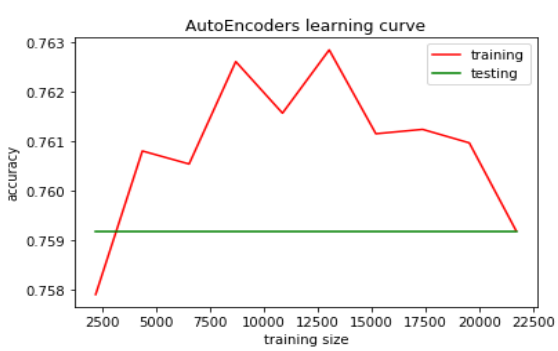


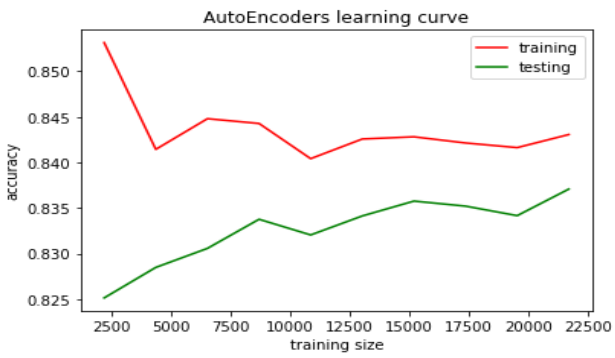


In conclusion, the power of AutoEncoders should come out later when these feature vectors into a Neural Network because the useful information encoded by AE is not human-readable without decoding. I would assume means that the “structure” for a Neural Network is stored in the transformed data and is “readable” only by another neural network. Main problem of AutoEncoders was the insanely long runtime to tune the hyperparameters. If given more time, I would have tried exploring more with the parameters, but because every partial rerun took at least an hour, it was quite discouraging.

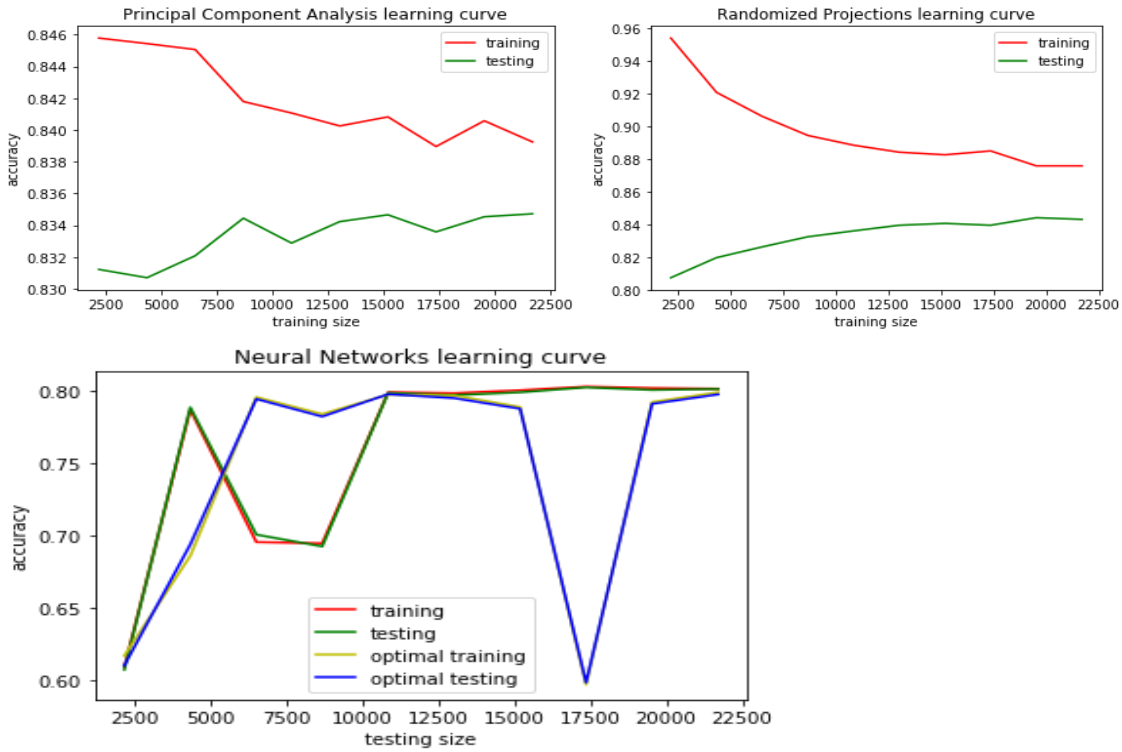
Dimensionality Reduction + Neural Network

ran all the Neural Network experiments on the Adult dataset especially because the learning curve in Assignment 1 had very interesting things to analyze. To start off, the graphs show that the learning curve on “normalized” AutoEncoders that not only is relatively worse than NN trained in Assignment 1, but also it actually does not learn anything with increasing training size as both training and testing accuracy do not change at all. Compared that with the graph on the right, there are no fluctuations in accuracy and the accuracy is off by ~6%.





And if we look at the PCA and RP learning curves, they do look like neural network learning curves as well. Especially RP shows that well as training accuracy deteriorates a lot with bigger training size to overfit. If you look at the graph below with optimal testing and optimal training curves, you can see that the optimal and normal curves do not differ at all. The optimal curves were made by feature reduction and the # features I cut it down to was ~60 (! Similar to target dimensions of RP). Since the graphs look same ➔ the extra ~50 features were **completely** useless as they neither decreased or increased accuracy as the Neural Network labeled those features useless in process. The amazing thing about the dimensionality reduced data is that they didn't just reduce #features put into neural net, they transformed into more "interpretable" for the NN data. Remember that PCA reduced to only 9 features as I chose to keep 9 PCs (~20% variance) that were just linear combinations of 108 feature vectors which NN favors for learning. RP, compared to PCA, attained similar testing accuracy despite its random nature and made transformed data more interpretable. If we look back at the 2D projections of both algorithms clustering, they actually did very different things: PCA divided clusters clearly and RP preserved the whole structure of raw data clustering.

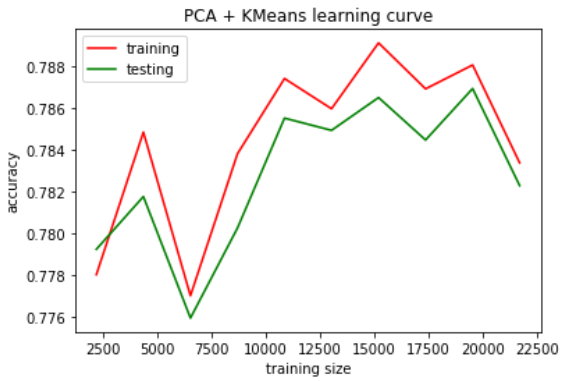


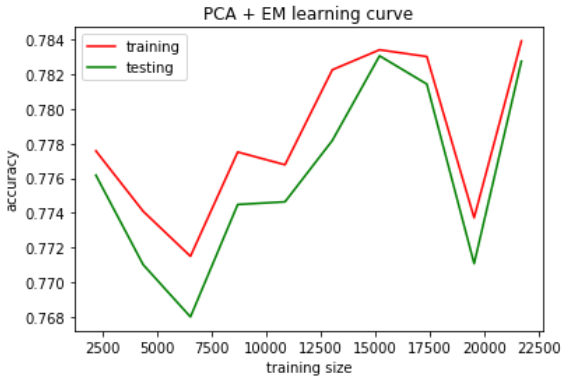
Overall, all 3 data transformations with correct parameters yielded better results than the non-reduced NN from Assignment 1. Another important thing to mention is that the runtime was **much** faster than in Assignment 1 as the # features inserted was often less, but more importantly it was faster because the data structure was easier to determine as it was: separated and linearly separable in PCA, cleaner and denser in correct places in RP and already encoded for AE.

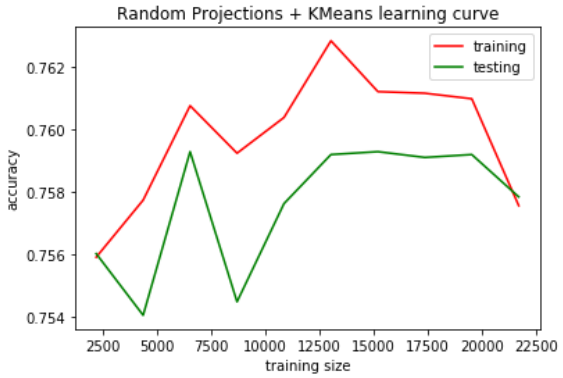
Dimensionality Reduction + Clustering + Neural Network

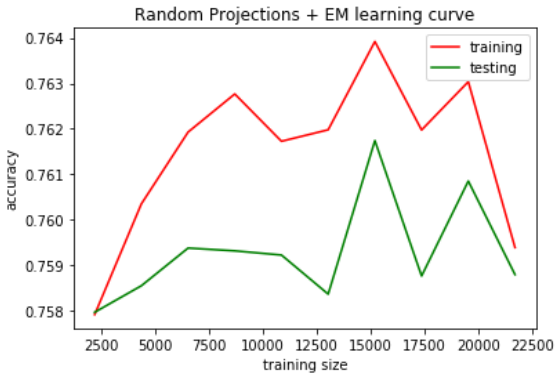
For this part, what I did was get the transformed data that I used in the NN for previous part, ran clustering algorithms KMeans and EM to get respectively centroids and means (expectation) of clusters. Then I created new feature vectors: each feature for a datapoint is Euclidean distance to existing cluster. So, new dataset looks like (# samples, # features) + (# centers, # features) → (# samples, # clusters). For all clustering, algorithms I used 2-cluster parameter, so all my data ended up being (#samples, 2).
One significant advantage of this approach was that it was lightning fast (2 features). Maybe if I tried working with this dataset like I did in Assignment 1 for Neural Network I would get much better accuracy, but still the accuracies on all 6 combinations of dimensionality reduction + clustering gave comparable

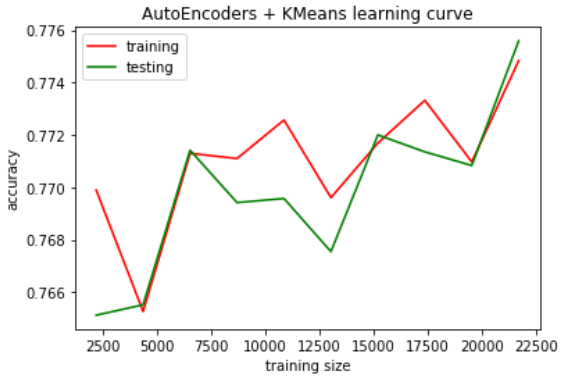
(though a bit worse) performance than NN from Assignment 1 from the learning curves.
I think this approach gave worse accuracy because having features as distances to centers of clusters does not necessarily give good representation of data: distance to cluster centers does not exactly give information whether the classification will be correct. Some cluster centers were located very off relative to concentration of most points (KMeans) as seen from 2D projection. Besides, sometimes the centers were very close to each other (RP) which shows that each dimensionality reduction algorithm handled preserving structure of data differently. Equating all of them the same oversimplifies the data and does not give an accurate representation.

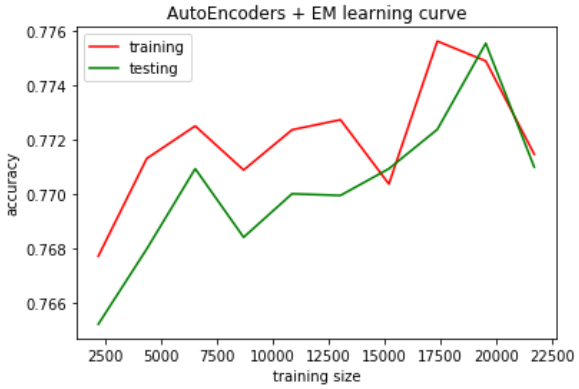












Citations:

Becker, B. (1994). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Data Mining and Visualization, Silicon Graphics.

Johnson, B. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. 2108-11 Kamiyamaguchi, Hayama, Kanagawa,240-0115 Japan, Institute for Global Environmental Strategies.

Johnson, B., Tateishi, R., Hoan, N., 2013. A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. International Journal of Remote Sensing, 34 (20), 6969-6982.