Chris Jenne
Assignment 3
Unsupervised Learning and Dimensionality Reduction

Introduction

For assignment 3, I will delve into the concepts and analysis of using clustering and dimensionality reduction algorithms to train networks to learn based on sets of data that do not contain labels or given relativity. In order to apply these concepts, we will be using the K-Means and Expectation Maximization algorithms for clustering. For the dimensionality reduction, we will be applying PCA(principal component analysis), ICA(Independent Component Analysis), Randomized Projections, and a technique of my choice which is Low Variance Filter.

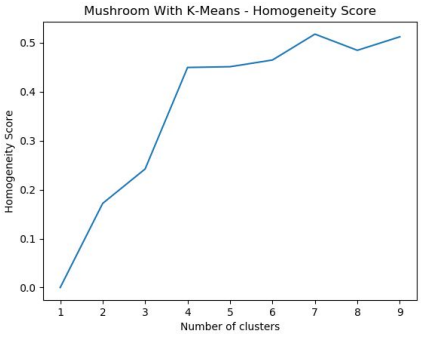
The datasets that I will be using for these techniques will be my slightly modified soccer match outcome data and the mushroom dataset. These two datasets are the ones that I have been using throughout this class and will continue to try to find the best algorithm and technique to produce the highest accuracy and performance. To reiterate the interest of these two datasets; I have always been very into soccer since I can remember. I even did an entire statistics class on predicting outcomes based on numerous variables of the modern game(home/away, W/D/L form, Goals for/against,etc). As for the mushrooms, ever since I watched “Into the Wild”, I always wanted to know the ways to differentiate between edible and poisonous wildlife. Unfortunately I was unable to find a dataset of all poisonous plants, so mushrooms will do! **NOTE:** There is a section after all reduction analysis that goes into the analysis of pushing the transformed data through a neural network.

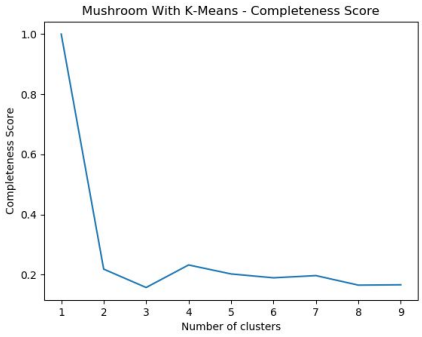
Clustering Methods

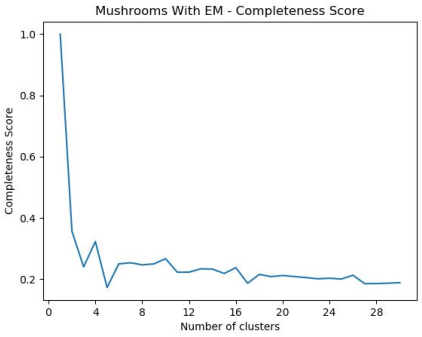
Clustering is way to find relevance from unlabeled data points within a space. This takes the assumption that points that are local to one another share some sort of similarity. K-Means and Expectation Maximization are two methods to determine which points will fall into a specific cluster. K-Means clustering uses the idea of placing ‘k’ centers in the data, and as it iterates, it slowly picks up points that are close in terms of distance. Once all the points have been assigned to a cluster,we will move the center step by step until it no longer moves (reaches convergence) or we reach the defined number of iterations.

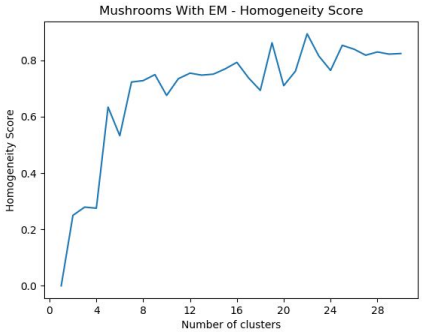
The other mentioned method is Expectation Maximization (“EM”). EM is different in that it gets structured using probability distributions within the data. To do this, EM utilizes maximum likelihood properties and alternates between estimating expectation and maximization. The one major difference between K-Means and EM, is that EM can have an infinite number of configurations since we are dealing with probability. However, EM works well with many distributions as long as we can compute our expectation and maximization. To first analyze the two clustering algorithms, we need to look at the completeness scores and homogeneity of the data. Homogeneity tells us if there will be an entire cluster of data points that are all within the same class. Completeness will tell us if all the data points that are members of a given class are elements of the same cluster. Using these two scores, we can potentially determine the ‘best’ number of clusters to produce.

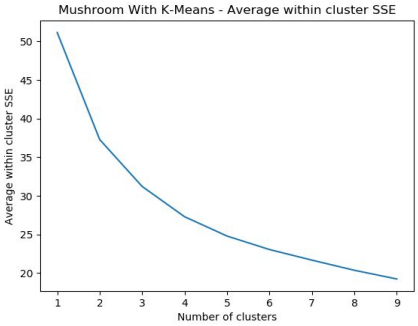
Mushroom - K-Means and EM





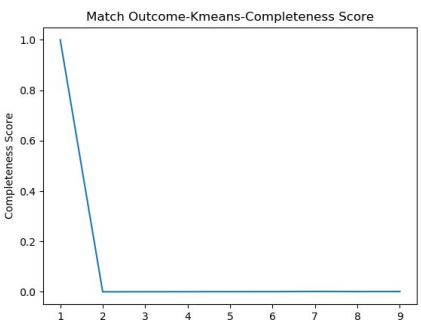


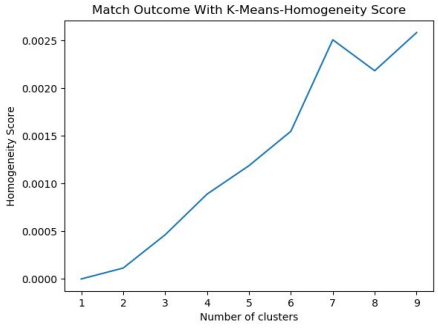


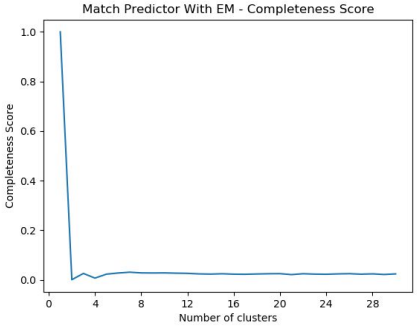


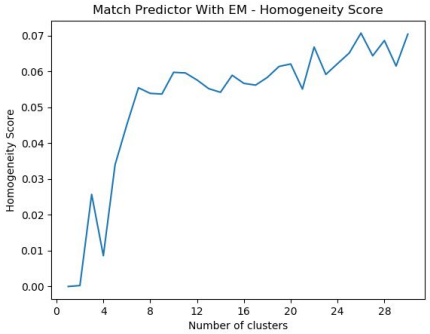
What we can tell from the above graphs is what our desired cluster count should be. Doing some research of K-Means, I learned of the 'Elbow Method'. What this method tells us is that we can identify the number of clusters based on an "elbow" found within the graph. Looking at the above graphs the first elbow we see is when k = 2, which tells us the best number of clusters would be two(i.e one cluster per class; edible or poisonous).

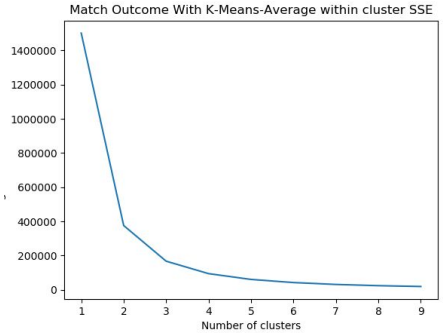
Match Predictor - K-Means and EM











Doing similar analysis here there are similar “elbow’s” being formed in the graph. However, since our match predictor dataset is a lot more robust we also see some funky lines being made. However, using the elbow method once again it comes to be that the ideal clusters is also coming out to be two. However, since we know we have three classes (Win, Draw, Loss), I would like to think that the ideal number would be at three. Looking into the completeness score we notice a very flat line after cluster = 2, for K-Means, and a slight rise at cluster = 3 for EM, which is what I expected to see. However, I would have liked a higher completeness score with more clusters. This could be a sign about the complexity of the data as well as added noise.

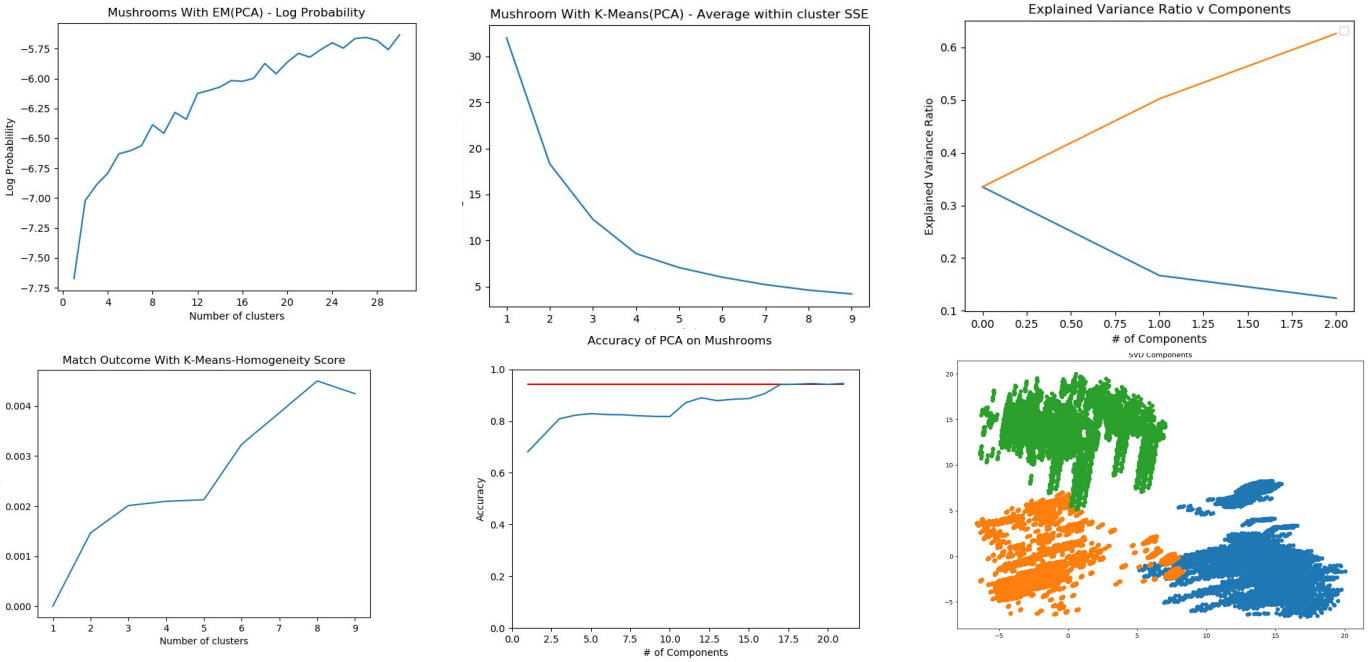
Dimensionality Reduction

The use of Dimensionality Reduction is to help move the data into simpler terms or dimensions, so that clustering can be done more effectively. There are several methods of reduction, and the ones that I will be exploring in this write-up are Principal Component Analysis (PCA), Independent Component Analysis (ICA), Random Projections (RP), and Low Variance Filter (LVF).

Principal Component Analysis

To first start the PCA, I will need first identify the amount of dimensions to break the data into. Using sklearn, I return the variance percents of my components to identify the number I can reduce the data down to. I was only able to generate Explained Variance Ratios using python, but using this method, we can come to some understanding of what the eigenvalues would tell us.

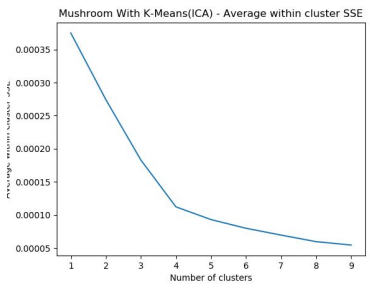
Mushroom Analysis - PCA

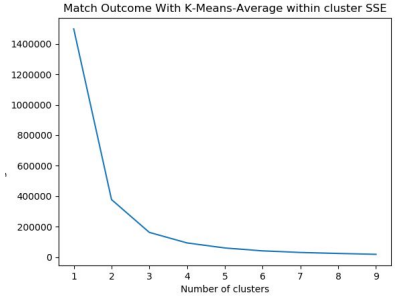


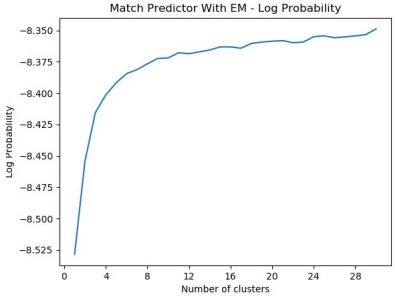
Looking at the top graphs, we can safely say that if we reduce the data down to three components, we will be able to explain about 63% variance of our dataset! Looking at the SSE

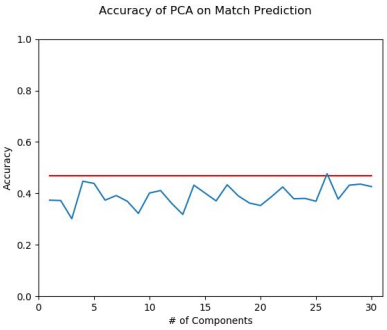
and Log Probability graphs, we can see that using PCA we lowered the SSE value and the log probability slightly increased. This tells us that using PCA helps our clustering algorithm cluster the data better. Furthering the analysis, I looked into what an SVD scatter would look like. SVD uses Eigenvalues and Eigenvectors to determine three matrices of redundant features.

Match Predictor Analysis - PCA









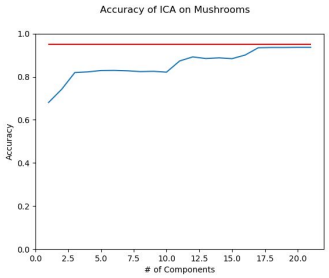
Taking a look into the Match Predictor data, running PCA has some slightly noticeable effects. First, we see that the graphs are much more smoother than before. We also see the same behavior as we saw in the mushroom dataset, where SSE is dropping as we increase the cluster count and the log probability is increasing. However, when it came time to run analysis on our eigenvalues, I noticed that the first component had a resounding 89.42% and the second came in at 3.23%. This tells us that there is a very high correlation in the dataset!

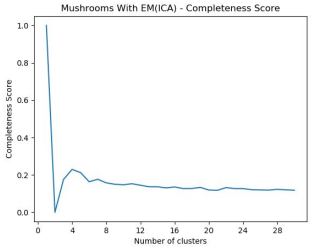
Independent Component Analysis

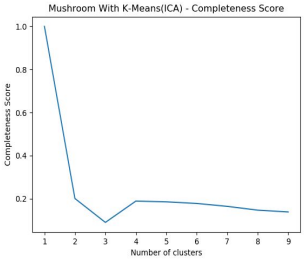
ICA works by attempting to change the structure of the data by maximizing the difference between components, and tries to find independent components of the original data. This is different from PCA because, ICA takes the additional step to maximize independence rather than just developing components that are unrelated.

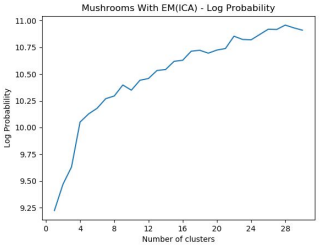
Mushroom Analysis - ICA

Running with the same number of components used for PCA (three), we can see a few key differences between our results. Since ICA and PCA share some similarity, we would expect to see some similar behavior.



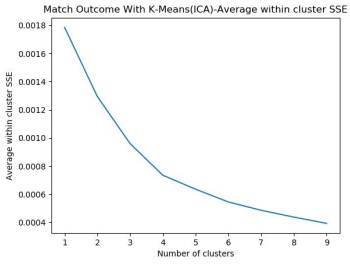


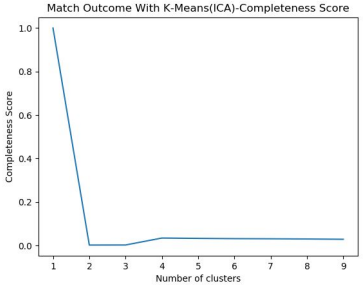


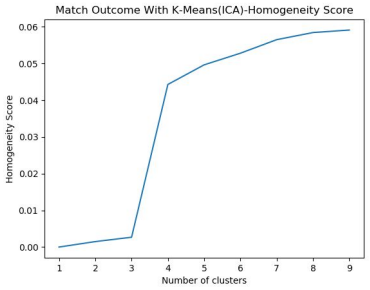


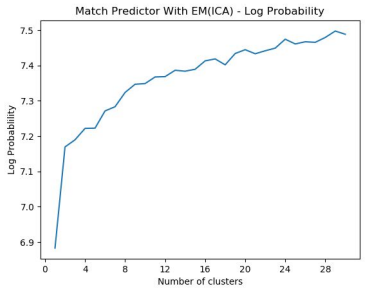
One thing we can tell is that the lines are pretty smooth, with some exception to the log. However, there is definitely the trend of raising log probability with decreasing SSE values. It seems that any value of Clusters that is greater than four, will start containing noisy information or that the clusters would not tell us much about the data. Running through the Neural Network, we

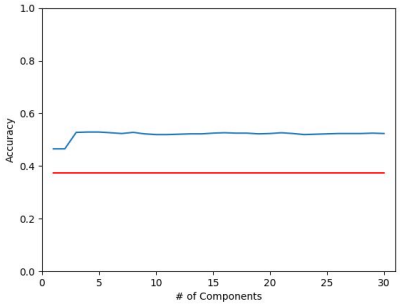
Match Predictor Analysis - ICA











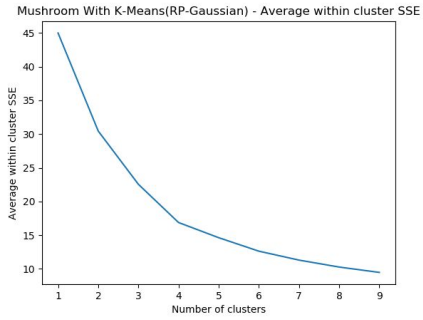
Looking into the Match Prediction dataset, we see some funky results in the completeness scores. We have the similar behavior of dripping SSE values with increasing Log Probability. I would have expected to see our graphs telling us that around three/four cluster mark we would have our best scores. I assumed this behavior since we know that we have the three classes. One thing to note is that we are well above the baseline in accuracy, meaning this reduction helped our clustering a good amount!

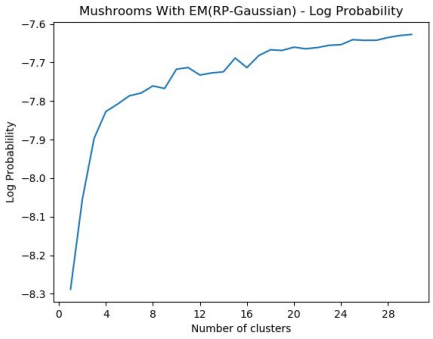
Random Projections

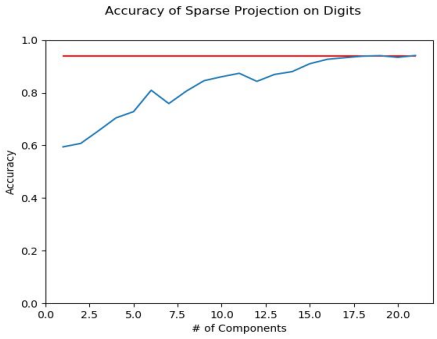
Random projections, perhaps my favorite one of these dimensionality reduction methods, due to the potential “randomness” that can ensue, attempts to plot all attributes to a lower dimension space. Random Projection puts the non-reduced data in a randomly generated gaussian matrix to attempt to put points in relational spaces for clustering. It also manages to preserve distances

nicely! It acts quite similarly to PCA, but works in a much faster time. Using sklearn I was able to apply both Gaussian and Sparse random projections.

Mushroom Analysis - Gaussian Projections

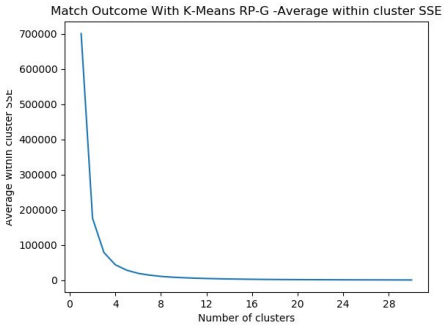


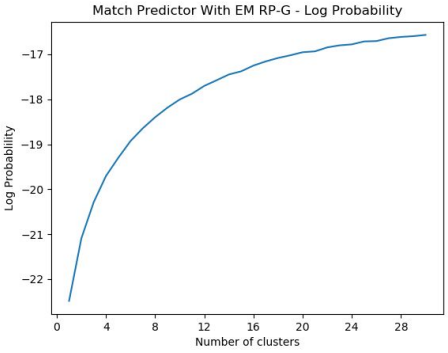


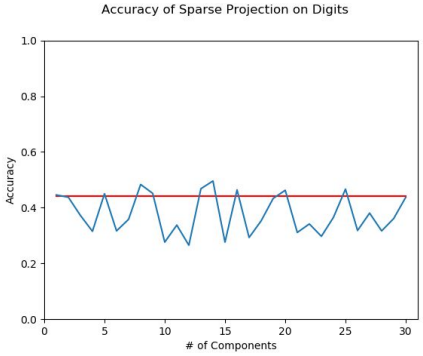


To continue with the same analysis, we look at the log probability and SSE values. They are working in the same direction that tells us that we successfully made clustering more effective with this reduction. The third graph, shows us our accuracy based on the Neural Network baseline. As we are getting around to 6 components, we have a decent peak of accuracy. This helps tell us the number of dimensions we can reduce to in order to still reach high accuracy, and in the graph at 16, we can reach baseline accuracy.

Match Predictor - Gaussian Projections





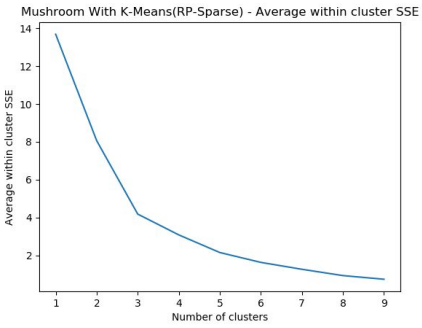


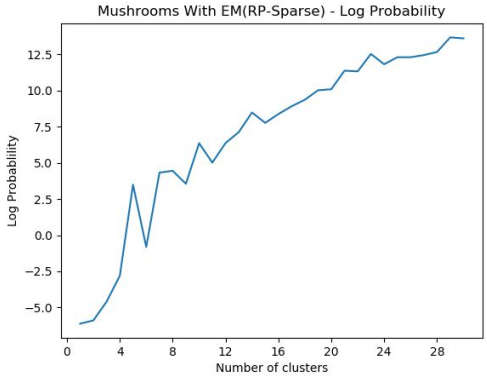
For Gaussian projection, there are nice smooth lines for the SSE and Log probability values, with SSE dropping incredibly and Log Probability increasing by a very small margin. What this is telling me is that we had a better K-Means clustering after performing this reduction. Looking into the accuracy, we get a really funky looking line! It reaches the peak at 14 components, but hits a decent accuracy around Components = 8.

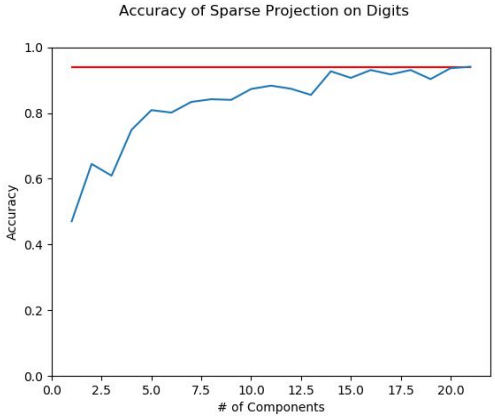
Mushroom - Sparse Random Projections

Looking into the Sparse version of random projections we see much lower values of SSE, and the log probability increases well outside of the negatives. Another point to note, is that our accuracy projection increases faster at lower number of components, while it more or less matches the higher numbers. This is mainly because of the randomness of the method, and if

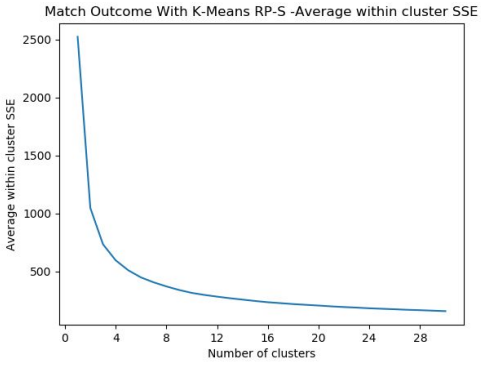
ran multiple times we will see different answers each time. Unfortunately I am running out of pages and can only say that around the 16 cluster mark do both of these perform very well.

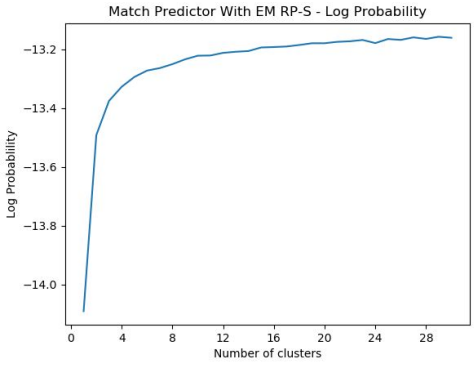


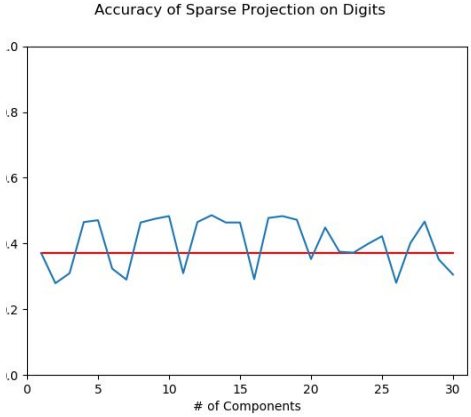




Match Predictor - Sparse Random Projections





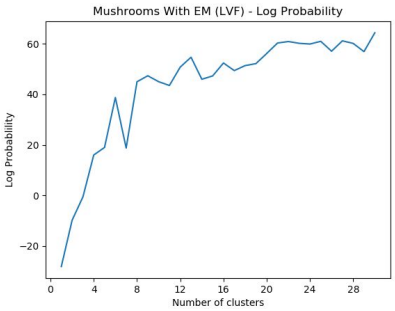


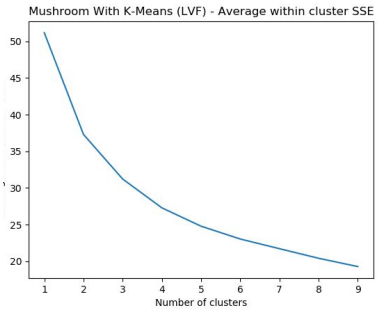
Comparing the Sparse projections with Gaussian I noticed a couple of key differences. One being the speed as well as the decrease of the SSE as we increased our cluster count. SSE drops very low which shows we have an effective ability to predict with smaller dimensions of our data!

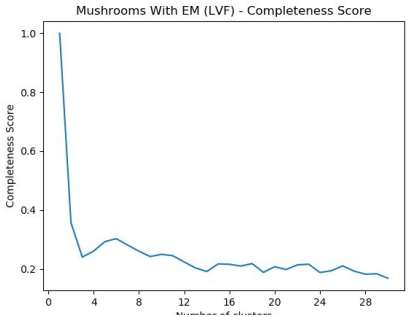
Variance Threshold

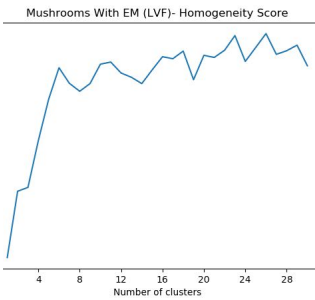
The low variance threshold method is a feature selector that removes all the low-variance features in the dataset. This method will look only at the features within the data, and not the desired outputs.

Mushroom Analysis

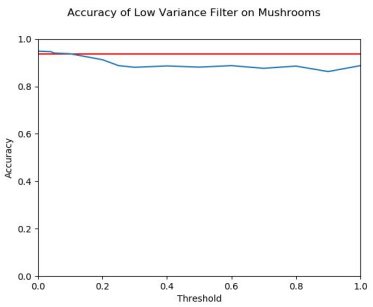




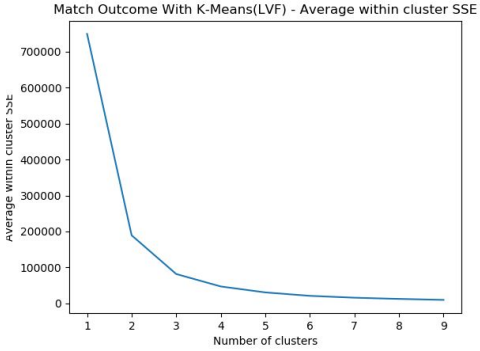


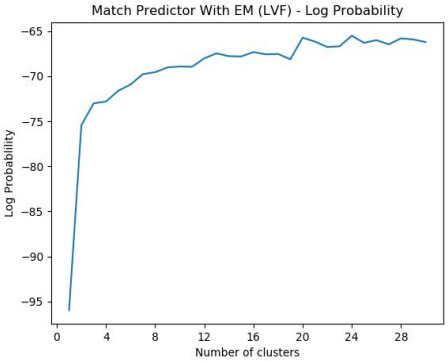


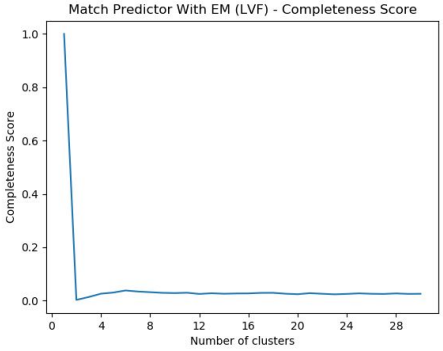
Looking at the graphs and accuracy for the LVF, I want to believe that this might be the better identifier for a good selection for the number clusters we are going to want to use. We can see some elbows in the graphs at about Clusters = 2. The accuracy with a threshold equal to zero performs the best (as defaulted too). As we remove more features we take a loss on accuracy.

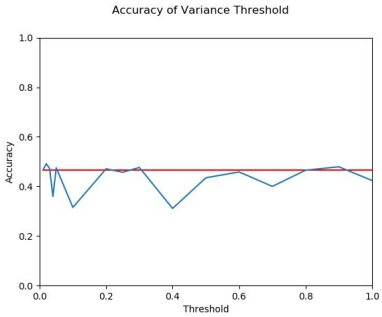


Match Predictor









After running the match predictor through the LVF reduction method, keeping a low threshold would allow us to predict with a better chance, but nothing more than .203%. Playing the elbow game one last time we see a the edge occuring around similar territories at Cluster = 2. So once again this is a successful reduction!

Clustering & Reduction With Neural Network

When comparing the run times with the Neural network, I found that the transformed data worked more efficiently, and still managed to compete with baseline in accuracy.

PCA finished in 12.821046 seconds
PCA accuracy: (0.813333)
ICA finished in 3.299504 seconds
ICA accuracy: (0.813333)
RP - Gaussian finished in 12.645850 seconds

RP - Gaussian accuracy: (0.813333)
RP - Sparse finished in 12.130835 seconds
RP - Sparse accuracy: (0.813333)
LVF finished in 17.899990 seconds
LVF accuracy: (0.813333)

Conclusion

All in all, this assignment was very beneficial for managing data and taught me a good amount of reducing the complexity of a dataset that contains lots of noise. As I come up on deadline, I kept working on the plot of all the accuracies against the neural network. I did some work with the baseline neural network in the individual analysis, and showed that most of the time using reduction methods increased the accuracy of the model.