CS 7641 Assignment 3
Jeremy Martinez – jmartinez91@gatech.edu

Introduction to the data
The two datasets used here are the same two datasets I used in assignment 1. One is a set of restaurant review data and the other is a set of health records. The reviews are around 30-80 characters on average with a binary classification of whether the review is positive or negative. This is a natural language processing problem and so it treats every word as a new attribute. NLP problems are especially interesting in clustering and dimensionality reduction because with such a large array of words to choose from, it is essential to reduce the dimensions. Preprocessing is done on the words first (similar to assignment 1 so I will not go into detail here). The data consists of 1,000 rows, split evenly with 500 positive and 500 negative. From our bag of words, I will only use the 100 most common words when preprocessing the data. Dimensionality reduction will reduce this further, of course.

The health records data is more traditional in structure. It contains 46 attributes, some of which are discrete and some of which are continuous. It contains a binary classification of positive and negative test results for cervical cancer. This dataset is fascinating in respect to clustering because it proposes a very real, yet daunting challenge in machine learning. The ratio of positive to negative classifications is unbalanced at 55:803. This unbalance will influence our k value significantly.

We will split our data on an 80/20 ratio of training/validation. And so 800 reviews/686 health records for our training sets and 200 reviews/172 health records for our validation set. For brevity, when referring to accuracy (in text or plots) assume validation accuracy.

Choosing K
I will discuss this topic in two separate sections respective to each dataset. For the review data, we have a dataset of 1,000 rows. Given the nature of our data, I imagine any review with the words bad, gross, disgusting, etc. to strongly indicate negative reviews. Given our massive range of input attributes, I predict a relatively high number of clusters to prove to be accurate. I will choose to test an exponential set of k, and what better set of exponential numbers than Fibonacci? (disclaimer: Fibonacci didn't converge, so I shifted the scale a bit) So for the review set I will test k at [8, 13, 21, 34, 55, 89, 104, 119, 134, 159].

For the health record set, we are much more restricted in the number of k's we explore. Values of k we use (should be) restricted with an upper bound of 56. Consider the scenario where we group every positive classification as its own cluster and the negative classifications as one giant. Our training data will fit with 100% accuracy. We cannot train our data any better than this (or worse if you consider overfitting) and so we will keep this in mind when considering a value of k higher than 55. In reality, our training set will have less than 55. For k we will test values [8, 10, 14, 18, 25, 35, 45, 55, 65, 75]. It will be interesting to observe how our clustering accuracy behaves past 55. Though our sample set only has 55 positive classifications, our clustering algorithm could recognize patterns in our training data that will map well as new data is fed in.

Clustering
Initially we fit the data (as is) using KMeans (sklearn.cluster.kmeans) and expectation maximization (EM) (sklearn.mixture.GaussianMixture) functions. KMeans is a clustering algorithm that groups sets of data of equal variance using Euclidean distance. The number of groups is passed as parameter K. The algorithm chooses K random data points and then enumerates all (other) data points and assigns them to the closest (K) data point. The mean of each group is calculated, each K data point is re-centered at its clusters mean, and the algorithm repeats (starting at the enumeration step).

Expectation Maximization (EM) is a soft clustering algorithm that looks very similar to KMeans. The term "soft" refers to the fact that each data point is assigned a probability that it belongs to a group using a Gaussian distribution rather than a Euclidean distance metric.

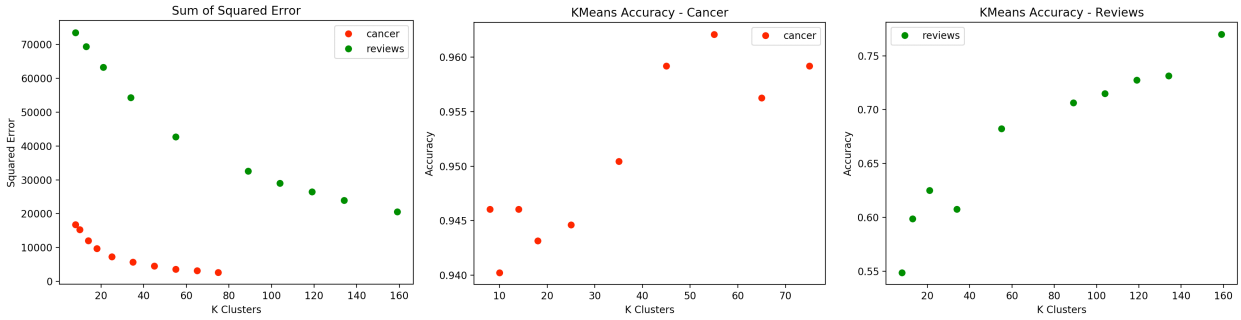
KMeans
We ran KMeans over our selected number of k parameters (defined above). In this specific experiment we have the labels to compare our clusters to the actual data and measure the accuracy. We will evaluate our parameter of K by analyzing the sum of squared errors, the silhouette score, and accuracy in comparison to the actual labels of the data.

The sum of squared error (SSE) is a measure of how internally coherent clusters are. SSE suffers from drawbacks in practice. SSE assumes clusters that are convex and isotropic and responds poorly to elongated or irregular shaped clusters. Also, SSE is not a normalized metric (though not an issue after feature scaling). In the case of both our datasets, we do feature scaling when retrieving the data (dim_reduction.get_data) to ensure it is within a given range (for our continuous attributes). It also worth noting that, Euclidean distances can become inflated in the case of high dimensionality. We will watch for this when running KMeans on our review data. We will compare the effect of this in particular when running dimensionality reduction (PCA, etc.) algorithms as well.

The silhouette score is used to measure the separation distance of clusters. It measures how close one point is to the point of neighboring clusters. It ranges from [-1, 1] where +1 indicates the point is far away from neighboring clusters, -1 indicates the sample may have been assigned to the wrong cluster, and 0 indicates it is near a decision boundary. We aim to find a value for K where the mean across our clusters is highest. Among our highest scoring, we will also consider standard deviation.

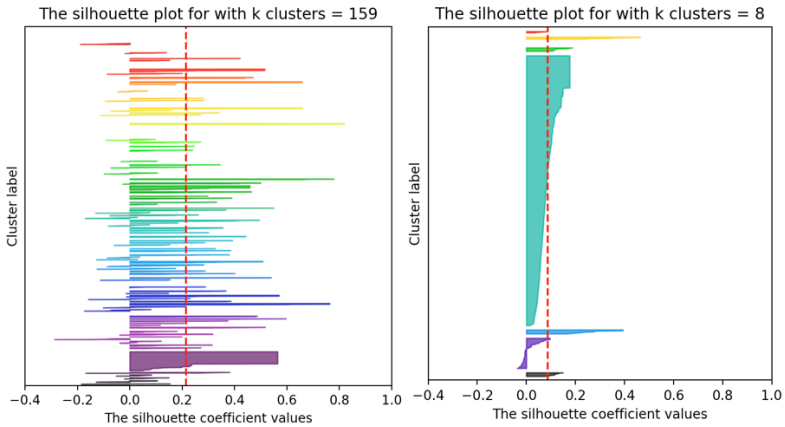
Accuracy and Error Evaluation

In the figures below we can see the SSE and accuracy of our data using KMeans clustering for our respective values of K. Both of our sets trend downward in squared error. However, we see at K=55 an interesting change in the Cancer data. After this value of clusters, the algorithm starts to overfit the data. This makes sense when we consider our argument from before with number of positive labels being 55. Since our training set will have slightly less than 55 (and our validation set has a handful of positively labeled samples), we do not consider 55 to be a hard-drawn exact calculation, but rather a rough tipping point of comparison. As for the reviews data, the accuracy continues to trend upward, though 159 clusters seems excessive. So we stop here and see if we can't use feature transformation to get more out of our data and use fewer clusters (generalize). These accuracy curves suggest that for our Cancer data, at K=55 clusters, the model accurately predicted 96.2% of the data points. For Reviews, at K=159 clusters, the model accurately predicted 77%.



Cluster Characteristics – Reviews

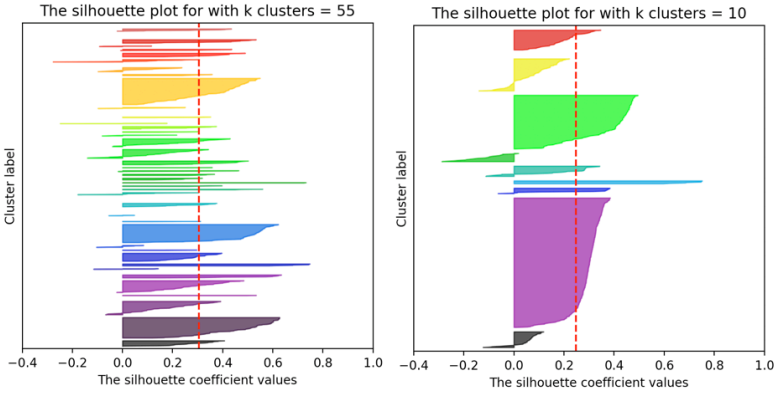
In Figures 4 and 5 we analyze the cluster for k = [159, 8]. K=8 performed very poorly and k=159 performed better, as we have explained above. When we take a look at each of the silhouette plots/scores we see why. The silhouette score of K=159 is 0.21 and the silhouette score of K=8 is 0.08. Neither of these scores are very good, since we consider a 1 to indicate decisive clustering and -1 to indicate incorrect clustering (as defined above). Also, we see that with K=159 we see many smaller clusters (displayed by the thickness of each bar), many of which scored below 0. This means that when averaging the points claimed by those clusters, they were incorrectly placed. However, we do see that several other clusters scored quite well. Looking at K=8, we see our 5th cluster (again, bar thickness) grouping the majority of our data. This indicates a strong correlation between the data here, with the



clusters on the ends indicating much sparser data points. This most likely means most of our points share similar values for a large number of attributes. Considering the characteristics of the data, most attributes (words) have a value of 0 (not present) so this is most likely why we see this behavior in our silhouette scores. Dimensionality reduction will be able to help with this significantly.

Cluster Characteristics – Cancer

In Figures 6 and 7 we analyze the cluster for K = [55, 10]. Both K=55 and K=10 performed much better with a silhouette score of 0.304 and 0.247, respectively. For K=55, we see several clusters of decent size, 3 relatively large clusters, a few medium and several small clusters. There are a few narrower clusters that range from high to low values indicating sparse data. For K=10 the majority of the data points land in the 2nd and 7th clusters. So to visualize this we would see one massive cluster, a second fairly large cluster, followed by 7 smaller clusters, with a couple really small ones. These silhouette scores reassure that the review data is certainly suffering from the curse of dimensionality. This dataset represents significant clustering and that there is a strong resemblance between these two (two largest clusters) sets of data points. The rest of the data is relatively sparse, and I predict that this is due to many irrelevant features that we will sift out in features selection. It is likely that many of our features are irrelevant/useless and they are just adding noise to the data rendering it difficult to cluster. It is important to remember also, the nature of this data having only 6.4% positively labeled samples. After running dimensionality reduction, we do still hope to see large clusters of data (clustering parts of the 93.6% negatively labeled samples). But we also expect to hone in on features that will help us accurately cluster the 6.4% better and accentuate the features that are at play with those positive classifications.

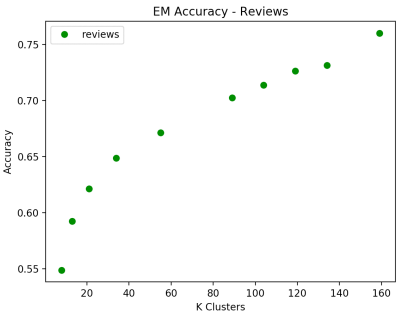


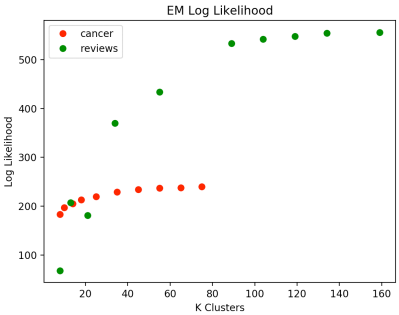
Note on silhouette scores

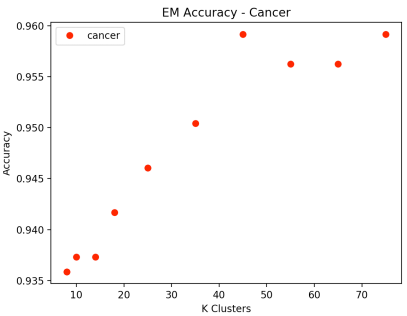
When comparing silhouette scores, it is worth nothing that (generally) with the higher we set our K value the higher our score will be. This should match intuition and is similar to overfitting our data. Consider the scenario where each point gets its own cluster (K = number of samples) then every point will be correctly placed, and our silhouette score would be 1.

Expectation Maximization

We ran EM using GaussianMixture over [our selected number of k parameters like with KMeans]. We evaluate EM clusters by plotting the maximum log-likelihood (ML) of our data. This means we are finding the ML mean of the Gaussian M by taking the sample mean of all our data points. We also compare the generated clusters of EM to the actual data, like how we did for KMeans. We can analyze our clusters in EM using the Bayesian Information Criterion (BIC). The BIC is concerned with covariance type as well as the number of components in the model and penalizes for complexity. BIC chooses the optimal combinations of number of components with covariance by assigning the lowest value.



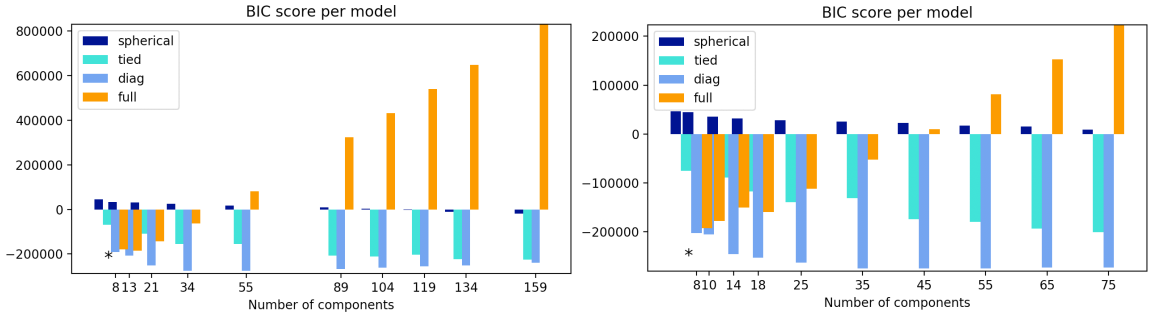




The log-likelihood is always increasing and we see it converge for both of our datasets here. Though we see similar indication of overfitting with the Cancer dataset, though it overfits at 45 components, rather than 55. For

Reviews we see accuracy continue to climb (though as we add clusters, we are weary of overfitting as well). I continued to run EM for reviews up through 215 clusters and we see it converge at n_components = [175, 185, 195, 205, 215] with accuracy of [0.765,0.765,0.77125,0.7725,0.775]. These results were run with diagonal covariance. It is worth noting that we do not have an overabundance of data samples in either case here. It seems like for clustering it is safer to detect overfitting when more samples are available rather than < 1,000.

The BIC score for the Review data gives us insight as to what the clustering looks for a given number of components and covariance. This plot suggests that diagonal covariance is far superior for this data set. It also suggests that 104 components is optimal (in combination with diagonal covariance). The BIC score for the cancer data suggests that the diagonal covariance is most optimal for our data set as well, along with number of components set to 35. Rerunning the clustering model from before on these number of components with type of covariance shows little to no change.



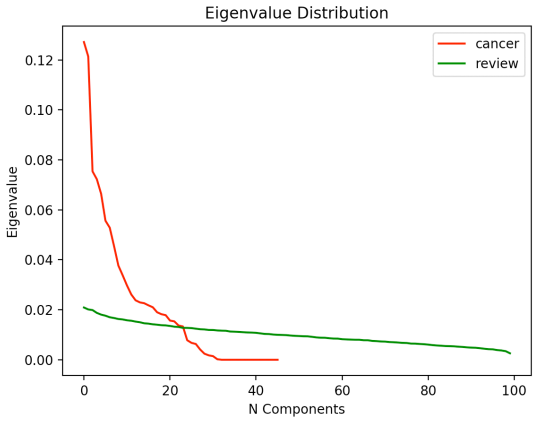
Take Away – BIC/Silhouette Scores
The BIC and Silhouette scores may not have always optimized our best parameters. However, in analyzing both of these plots, we were able to come up with the (almost) best set of parameters. This is truly incredible! After plotting both of these scores we can determine, with a reasonable level of confidence, what set of parameters to pass a clustering algorithm to model data without ever having seen a single label! If this were a real-world application, perhaps we would consider taking the set (2-3) of lowest scoring parameters and apply them to see how they compare (a sort of A/B testing you could say).

Dimensionality Reduction
Principal Component Analysis (PCA)
Principal Component Analysis is a dimensionality reduction algorithm that projects data into dimensions that contain the most variance. These projections are measured as vectors (eigenvectors) which are mutually orthogonal. This is intuitive to visualize up to three dimensions. PCA incrementally captures vectors (projections of the data) that maximize variance, 1st showing the most, 2nd showing 2nd most, and so on.

One can run PCA with an expected number of components, though when little is known about the value of individual columns of your data, this is equivalent to a shot in the dark. Instead, we will enumerate values [.60, .70, .80, .90] as the percentage of variance we wish to retain from our PCA. This differs because we are not explicitly selecting the number of components to project onto. Rather, we will continue to select principal components until our combined variance reaches our input parameter. At this point, we will stop selecting components and we will have our final number. This feels less arbitrary than choosing a number of components, regardless of what their variance is.

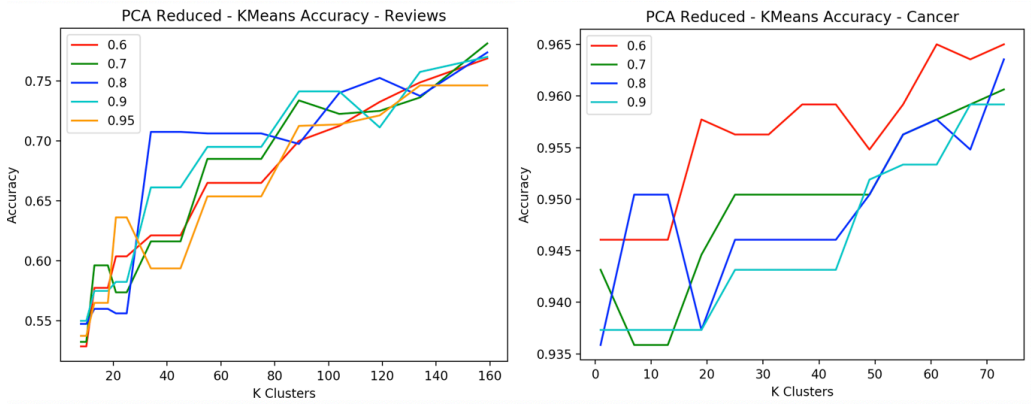
These values being summed up are the individual eigenvalues of the eigenvector. A sum of our eigenvector will give us 1.0. If we pass (any float) 0.8 as the number of components to our PCA, the eigenvector sum will equal 0.8. It is worth noting that our eigenvalues do not change with each iteration. Each iteration (each smaller number) worth noting that our eigenvalues do not change with each iteration. Each iteration (each smaller number) is simply cutting the length (end) of the vector off by some margin. Also note that data after PCA transformation loses no information when setting n_components to original value.

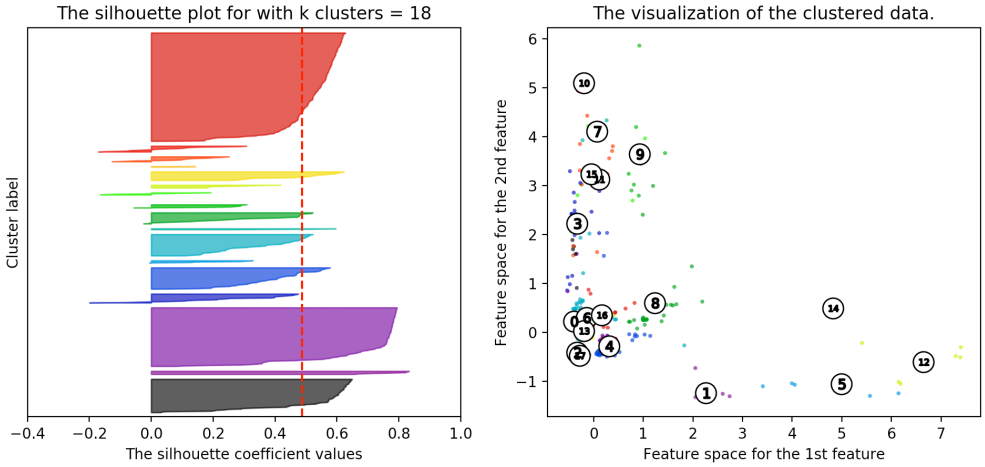
The distribution for our eigenvector can be seen in the figure to the right. What our distribution is telling us is that (in our cancer dataset) our last 15 components contribute no information to our model. This is truly incredible to discover in our data! We can see that in the cancer data our first few components have high variance, at component 12 we see it start to dip quite a bit and continue to gradually decrease until 31 when the components stop contributing information. Attributes 31-46 are therefore irrelevant. We also learn something interesting about our review data given its eigenvalue distribution. The first few words are slightly more useful and the last few less so. However, our attributes in this dataset are all relevant rendering all of them of low relevance. Note again, that 100 (most common) words were used for review data.

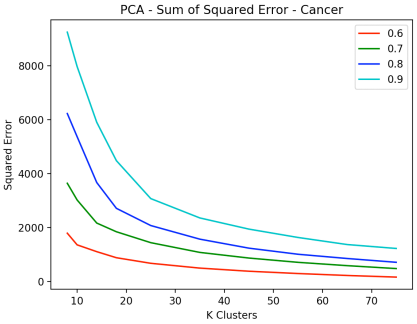


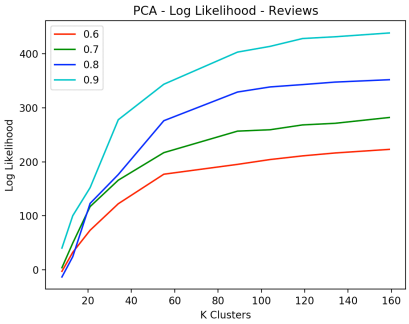
When we ask ourselves how well was the data reconstructed, the answer is it depends. Base case: setting n_components in PCA to same number of dimensions will yield in no data lost. Our eigenvalue is directly correlated to how well our data was reconstructed in that it measures the variance of PCAs output. If our first 5 components cover 80% of the co-variance then we have our answer in terms of information that our PCAs output yields. Now we will plug our dimension reduced data into our clustering algorithm from before.

As we can see in the figures to the right, setting our PCA n_components total eigenvalue to 0.7 and 0.6 for review and cancer data, respectively, does not impact our ability to accurately predict the data. In the figures we can see how the clustering of our data changed as well. PCA was not able to find any components that were very overpowering and this showed in the silhouette score as we did not see much improvement there. On the contrary, we saw a drastic improvement in the algorithms ability to cluster the cancer data. With the original data we were only able to get a silhouette score of 0.304 and that was with 55 clusters. After PCA we were able to achieve 0.487. In the figure to the right we can see the actual clusters which is much less noisy than what we were seeing for K=55.









Expectation maximization performed just as well if not slightly better too. Our analysis above will more or less sound repetitive for EM and BIC so we will not dive into it at this time. It is worth analyzing briefly how our data was clustered more precisely as well. Our sum of squared error as well as log likelihood both went down significantly. This is interpreted as how precise our clusters are. After PCA the clusters are more certain and place the proper data points in their corresponding clusters, with lower standard deviation. We can see a comparison between PCA parameters in the figures to the left showing SSE for all values of n_components and log-likelihood on the right. The SSE for reviews and log-likelihood for cancer was similar in that it also decreased. For brevity I will not include those here.

Independent Component Analysis (ICA)

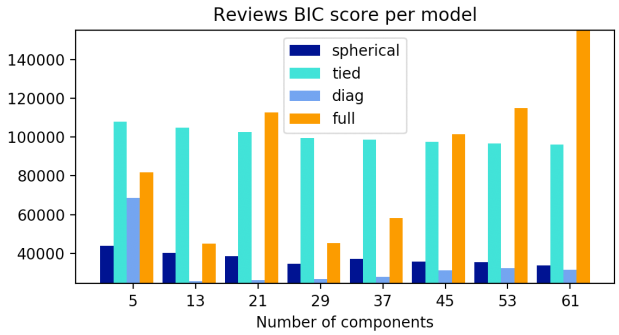
ICA is a linear transformation that finds a new feature set that is statistically independent. ICA typically performs well on noisier data or non-Gaussian data (where PCA fails). Whereas PCA maximized variance between attributes, ICA transforms a feature space X into a new feature space Y where every component is statistically independent, meaning they have no mutual information. ICA also attempts that. ICA also conducts this transformation while trying to maximize mutual information between X and Y (maximize I(Y, X)). Simply put, the algorithm attempts to retain as much information of the original feature space as possible.

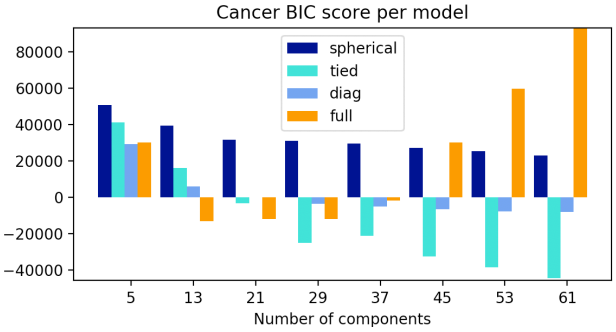
For intuition in regard to our datasets at hand, I will attempt to rationalize ICA's role on our reviews and health data. Altering the example with three microphones from the lectures, we will rationalize how exactly ICA attempts to interpret a bag of words (reviews). ICA will ingest words as a feature space, then project these words into another space for a new "bag of words" where none of these new word groupings share any information. Our hidden variables here are which (mutually independent) *combination* of words are indicative of each separate review/review label. What groupings of these words isolates one review from another, while still retaining all of the information from the original bag of words (feature space).

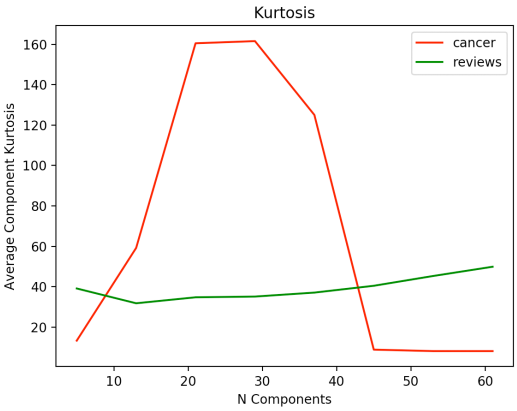
For our health data, we will use a similar argument as above. The hidden variables here are what ICA is trying to find. One should interpret these as transformations/combinations of features from the original space in a way that accurately represents the sample at hand. We want to transform the health record of a single person so that we can reproduce all of their original information, prove that we cannot derive any one feature from another (no mutual information), all while uniquely identifying the sample from which our new feature space was drawn.

These in-depth overviews of PCA and ICA are an attempt to highlight their differences. We can conclude from how ICA treats each sample of our data that it solves the blind source separation problem. Meanwhile, PCA does not attempt to solve this problem but rather uses variance in features to condense our data. ICA is also highly directional whereas PCA is agnostic and performs the same regardless of the direction of your data.

The data in ICA took on a very different structure and we see a strong example of this when plotting the Gaussian Mixture Models using Bayesian information-theoretic criteria (BIC) (see plots on next page). For the review data, the clusters in EM were represented much better with a spherical covariance type. Cancer is best represented with a full covariance type in lower clusters and tied covariance in higher numbers of clusters. This is a strong indication as to how the shape of our clusters have changed. In EM after running PCA on the feature set, components in both the review and cancer data were represented best by diagonal covariance matrices. After running ICA on the feature set, this changed. Reviews data is now represented well (still not best) when each component has its own single variance (diag). Cancer, when components is set to 13, is best represented when each component has its own general covariance matrix (full). When n_components is set high, it does better when all components *share* the same general covariance matrix (tied).



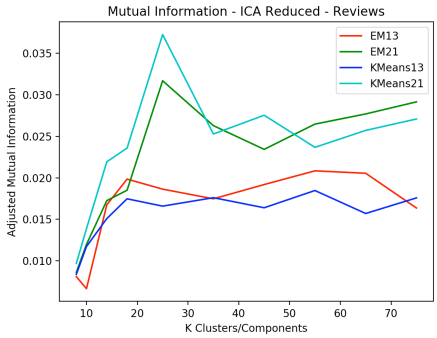


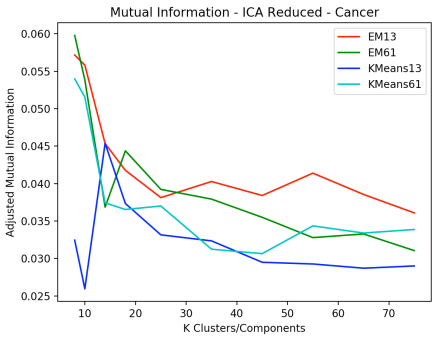


Another measure of the shape of our data in respect to ICA is kurtosis. The kurtotic behavior can be observed in the figure to the left. Cancer takes on a higher kurtosis in the middle of the curve which suggests that more of the variance is the result of infrequent extreme deviations, as opposed to frequent modestly sized deviations. The line for Cancer here indicates high kurtosis. This makes sense given the nature of the cancer data (contains continuous attributes). On the other hand, the review data is all discrete and sparse, so we are not surprised to see low kurtotic behavior.

Kurtosis is not a measure of standard deviation. It is a measure of which values in our dataset are causing the standard deviation. In our cancer data, there is high kurtosis because the variation in the data is due to an influx in central/extreme values. Our review data has low kurtosis because the variation is (in larger part) to intermediate values. You will find this with all discrete values. Since a discrete value (in the review dataset) can only have a binary classification, there is no central/extreme values, just 1/0. Continuous input (which our cancer data has more of) can range in value which will yield higher central/extreme values and, therefore, higher kurtosis.

One of the main goals of ICA is to project the data so that it is statistically independent of the original data set. One way we can measure this is through adjusted mutual information. Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. MI is a measure of the similarity between two clusterings. In the graphs below, that measure is between the predicted and actual labels for both reviews and cancer. We reran our EM models from before using a 'diag' covariance for the review dataset and 'tied' covariance for cancer. Studying the figure to the left, the blue line represents KMeans for review data when the feature set was reduced using ICA with n_components set to 13. This line shows very low MI which is why we would most likely use this model or EM13 moving forward. The lines in the cancer MI plot are a bit closer but we would use the same argument to derive a model for it (KMeans 13).





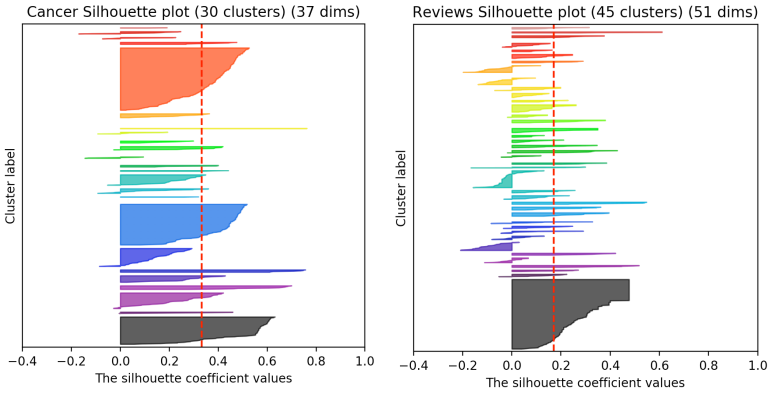
Random Components Analysis/Randomized Projections (RP)

RP is a dimensionality reduction algorithm that leverages Euclidean distances when transforming the data, however, it works on a different basis that makes it much faster than PCA. PCA selects projections based on maximizing variance. RP selects these based on random directions. Since there is no calculation needed in finding directions with high variance, it is much faster than PCA. RP still retains a strong accuracy despite its randomness. *When your*

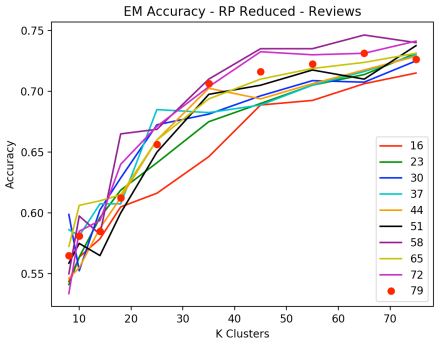
data has high dimensionality, is sparse, or just does not contain a lot of samples, projecting (with RP) from n dimensions to m (where m may not be as low as it would be with PCA) can prove to have high accuracy.

Having these prerequisite characteristics to your data can perform best under RP because finding directions of high variance may not indicate exactly what you expect in the first place. Often times you won't project to as few dimensions as you would with PCA, but RP will find some correlation with your data, especially when *it is sparse, contains high dimensions, and few samples*. With data such as this, the direction with highest variance (found in PCA) may not significantly differ from the 2nd, 3rd, 4th (etc.) directions of highest variance. Meaning, PCA may be believing the data too much by looking for something (directions of high variance) that are not there, which sounds a lot like overfitting. One interesting example to explore would be to run PCA on the data of which RP produces. When given a dataset as described above (and PCA having little ability to derive directions with significant variance) would it be useful to first run RP, then run PCA on the data generated by RP, then fitting your clustering algorithm? (Data => RP => PCA => KMeans/EM) We shall see!

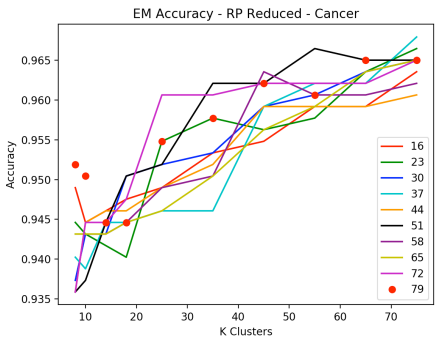
My assumption is that whether this method of chaining reduction algorithms improves your classification accuracy depends on if PCA does well in the first place or not. If RP outperforms PCA straight up, then I think running PCA after RP would find new eigenvectors that could highlight decisive separation in your clusters. On the contrary, if PCA (initially) outperforms RP, then running them in this fashion would be counterproductive. Since PCA is already finding vectors with high variance, randomly (re)projecting the data and disrupting those projections of high variance would serve little to no purpose.



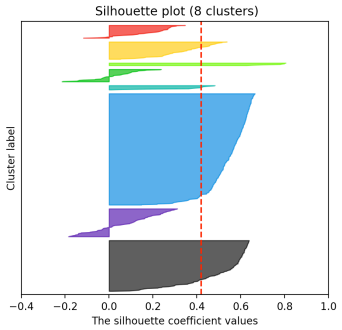
Moving forward with RP, we reduced dimensionality testing [16, 23, 30, 37, 44, 51, 58, 65, 72, 79] as inputs. Instead of approaching RP like we did with PCA, we first will plug the transformed datasets through our silhouette and BIC plots. For KMeans on cancer data, reducing (from 46) to 37 features performed best across 16, 30 and 45 clusters. KMeans on review data performed best reducing (from 100) to 51 features (features to the right). For EM, we analyzed the BIC scores and determined that cancer did really well projected onto 44 (only two less) dimensions. The BIC scores for projecting onto higher dimensions were close to the results of 44 so we will explore what happens when we run a cancer dataset of 55 features though our clustering algorithm as well. The review data did well projecting onto 51 dimensions and slowly (but not significantly) improved approaching 79 dimensions. Not showing BIC plots here for brevity. After rerunning RP multiple times, the results I get fluctuate significantly. So one should take these plots with a grain of salt and be aware that they are subject to change with different iterations of RP.

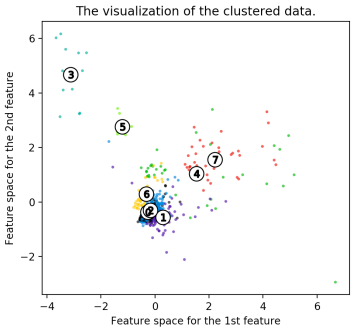


Regardless of running RP multiple times, it never matched the accuracy of PCA. When comparing dimension reduction within RP, we don't see any decisive set of dimensions (without our range) that see to perform significantly better. In fact, we see projecting to a (slightly) higher dimension (51 features) performs slightly better (though the difference is minimal).



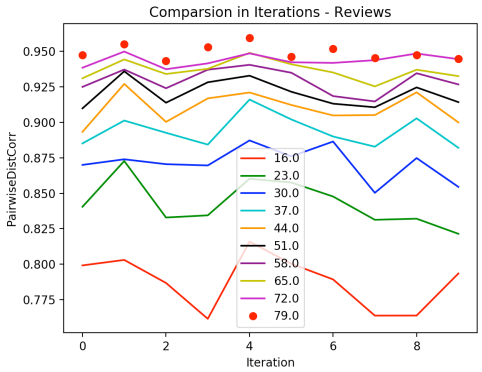
For our review data, dimensions of 58 and 72 performed the best and converged near 0.74 (cancer plotted in left figure, review in right). The accuracy plots for KMeans looked similar to these.

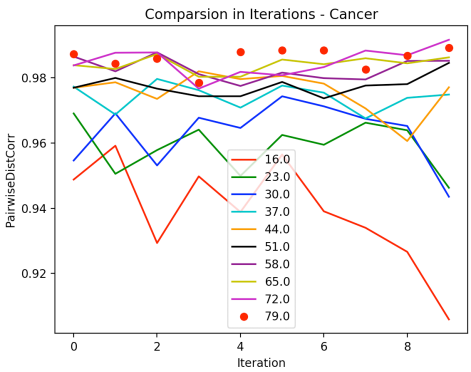




Since our accuracy here is lower than our accuracy with PCA, it seems that running PCA on our RP dataset would not perform better than PCA originally. We can test this hypothesis when we rerun PCA and plot the silhouette and accuracy (done on cancer dataset). First with our silhouette plots and then accuracy. We interestingly see PCA do quite well (not as well as before) with a silhouette coefficient at 0.41 but with 8 clusters instead of 18. In the figure to the right we can see much very clearly our either clusters. This is interesting to consider in that we were able to generalize almost just as well with fewer clusters when running our dimensionality reduction on a feature set that was already reduced by RP. PCA in this plot is retaining 0.6 covariance. We see relatively similar levels of accuracy for both the cancer and review data in this scenario as well (not showing accuracy plots for that reason). It is worth noting that as we add data samples, few clusters is preferable as it is least likely to lead to overfitting.

When running RP multiple times, we see a sporadic fluctuation in the way that it reduces our data. We measure this fluctuation by calculating the pairwise distance between the vector data X and the transformed vector data. This measure uses a Euclidean distance. It is worth noting that for some data (non-symmetric data) Euclidean may not be the most efficient. Further observation using cosine, L2 (and other distance measures) may show an interesting change in RP behavior. For both cancer as well as review data we see that with each iteration the RP transformed data vector varies in distance to the original data. Each line (16 – 79) represents the dimensions after transformation. Notice the pairwise distance varies greatest when projecting into lower dimensions in the cancer data. For this reason, we would want to avoid doing so for our final model.

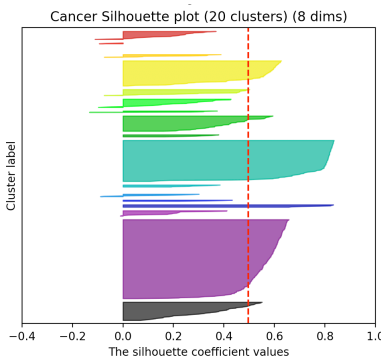




Truncated Singular-value Decomposition (SVD)

For our fourth and final dimensionality reduction algorithm we used truncated SVD. This decomposition performs linear dimensionality reduction by means of truncated singular value decomposition. This technique differs from PCA in that it does not center the data prior to running the decomposition. This means that it works especially well with sparse data, which we find with our review data.

Truncated SVD uses a form of feature transformation called latent semantic analysis (LSA). It tends to be computationally heavy. Typically, SVD tends to do well when dealing with synonymy and polysemy. LSA seems to address many of the drawbacks of NLP problems and for this reason is interesting to try out on our review data (and why not our cancer data as well).

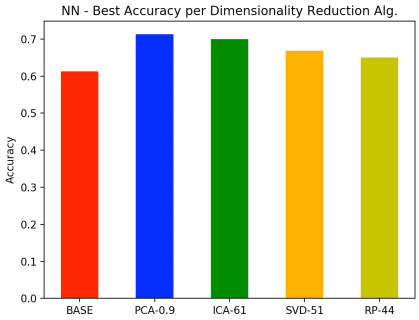


SVD performed comparable to PCA and even did slightly better in some areas. We have plotted the highest silhouette score with SVD at 0.496 (figure to the right) with a dimensionality of 8 and 20 (KMeans) clusters. Other curves such as SSE, log-likelihood, validation accuracy, etc. can be plotted

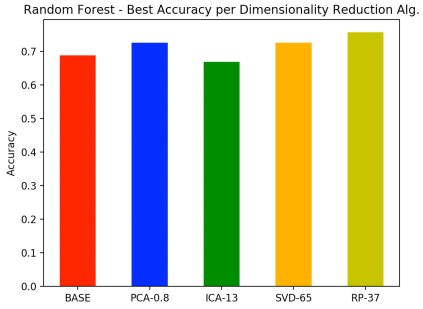
with the code but will skip showing them here since our analysis will seem repetitive to PCA.

Neural Network with Dimensionality Reduction

Now we will run neural network on our reduced review data. We use the original dataset as a base case (BASE) which has 100 attributes. We used the same 80/20 split on our train/validation set. And in the bar graph to the right we can see our results for each bar at 0.6125, 0.7125, 0.7, 0.66875, 0.65 (0.6125 as our controlled BASE case). In comparison to our BASE case PCA does very well here. We can conclude that with NN we can see significant gains in model accuracy from dimensionality reduction. An interesting question would be to pose this question against other types of models.



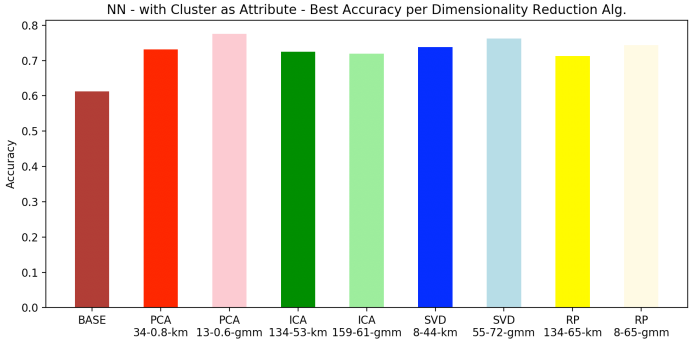
In the next bar graph, we see how a Random Forest classifier reacts to dimensionality reduction. Now when re-running this classifier multiple times, we see a fluctuation in results. RP consistently does well, though sometimes BASE does just as well or better. We use entropy as our information gain. This is interesting to plot as well because it suggests that dimensionality reduction may not be a do-all solution across all models. And more importantly, that our model could generate slightly different results on subsequent iterations



Since we are reducing the number of dimensions, I would expect NN to run much faster. However, fitting the classifier runs in relatively the same amount of time (sometimes slightly slower, sometimes faster, though nothing significant). This is strange, and I wonder if this is just because we are really only reducing from a small number of dimensions. In assignment 1, our NN ran much slower because we were using 1500 dimensions. It would be interesting to restart this entire assignment using 1500 (most common words) dimensions and reducing from there

Neural Network with Dimensionality Reduction and Cluster as Attribute

When running KMeans clustering and Expectation Maximization (gmm) on the reduced data, we also added the cluster that each row belonged to as another attribute of the data. This way, each sample can leverage the knowledge of our unsupervised learner and train our supervised learner model on that information. This was extremely successful (which surprised me very much to be honest…) at increasing our accuracy. Simply running the data as is (BASE case) gave us an accuracy of 0.6125 (same as above). However, our PCA EM dataset reduced to 0.6 of its original attribute size and adding the unsupervised learners prediction at 13 clusters yielded an accuracy of 0.775. This is an increase of 0.1625 over our BASE case and 0.0625 over our best NN performer above (PCA-0.9). This is absolutely incredible and is the best example of how powerful a model can be when combining unsurprised methods with supervised learners



Tying it All Together

After analyzing all of our learners, we have walked away with an arsenal of tools for analyzing and improving a learner regardless of the accuracy of our input data, or sample size. Silhouette curves and BIC plots can show us efficacy of our clusters. Using these plots alone we can derive a set of parameters that generalize our data well without having access to the labels of our data samples. SSE and log-likelihood indicate how accurate our clusters have been drawn and show a mathematical representation of how much error respective of our data points we can expect from our clusters. PCA and RP are useful dimensionality reduction techniques that work well in maximizing variance in our data and finding correlations between attributes. ICA works differently in that it tries to reduce our data based on features that share little to no information. SVD is another algorithm that can work well with sparse data. And we have also seen the power of combing both unsupervised methods with supervised methods.