An Analysis of Clustering and Dimensionality Reduction Algorithms

Abhinav Tirath

I. Dataset Description

A. *Credit Card Fraud Detection*

The first dataset used describes 10,000 separate credit card transactions, classified into two categories: fraudulent (represented by 1) and not fraudulent (represented by 0). The data is highly unbalanced; only 493 of the transactions are classified as fraudulent, as indicated in Figure 1.

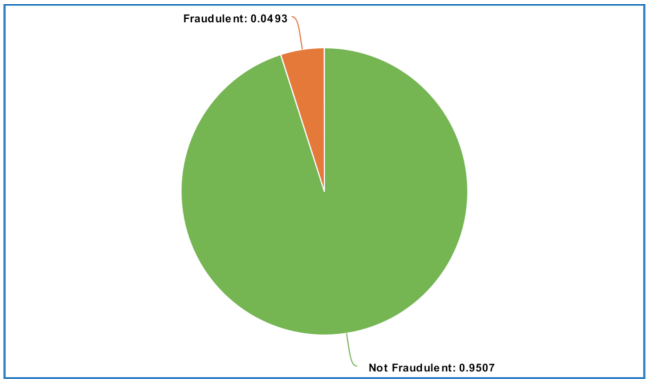


Figure 1. Displays the imbalance of the credit card data.

The data were collected and analyzed by a research collaboration of Worldline and the Machine Learning Group of ULB. This dataset has 28 features that describe each transaction, and no cell is blank. The number of instances in the dataset was reduced from about 243,000 to 10,000 in order to make the data more balanced and reduce time of computation. The features are all numerical inputs labelled V1, V2,... V28, and they are the result of a PCA transformation. Unfortunately, the original features and a further description of the data cannot be provided, due to confidentiality issues.

Credit and debit card transactions account for over 50% of all transactions in today's world. For this reason, it is essential that payments be assessed as fraudulent or not fraudulent as soon as possible. If we fail to do so, customers will be charged for purchases that they did not make, and credit and debit card companies will fall to the wayside with lawsuits and chaos. A solution to this problem will ensure the safety of consumer money and credit card use.

B. *Pima Indians Diabetes*

The second dataset used describes diagnostic measurements of 769 Pima Indian females, classified into two categories: has diabetes and does not have diabetes. The data is somewhat imbalanced; 501 of the females did not have diabetes, as indicated in Figure 2.

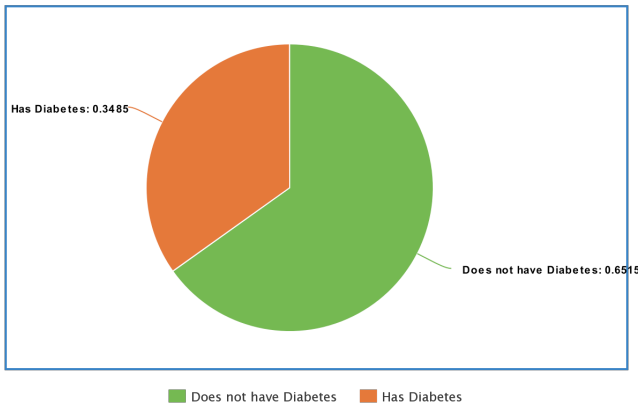


Figure 2. Displays the balance of the Pima Indian Diabetes data.

The eight features consist of age, BMI, number of pregnancies, and five diagnostic measurements, such as blood pressure. The data was provided by National Institute of Diabetes and Digestive and Kidney Diseases, but some values were left blank for some patients. These values were filled with the mean of the remaining measurements of the same feature. This was the only adjustment made to the data.

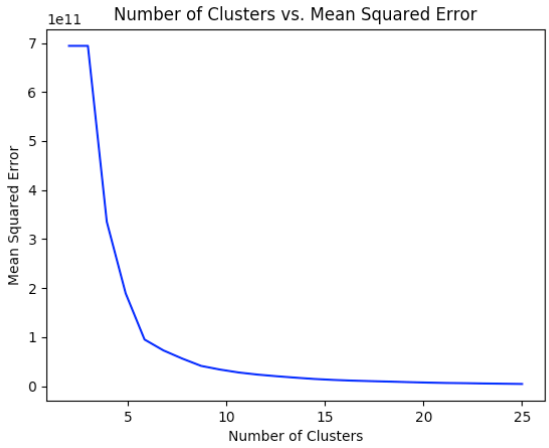
The objective of our analysis of this data is to predict whether a given person has diabetes or not, based on a few relatively simple diagnostic measures. The results can help physicians easily determine whether someone is likely to be diabetic. This has far-reaching implications. Not only could diabetic testing become much cheaper and faster, doctors may even be able to determine if someone has diabetes simply from a physical.

II. Clustering Algorithms

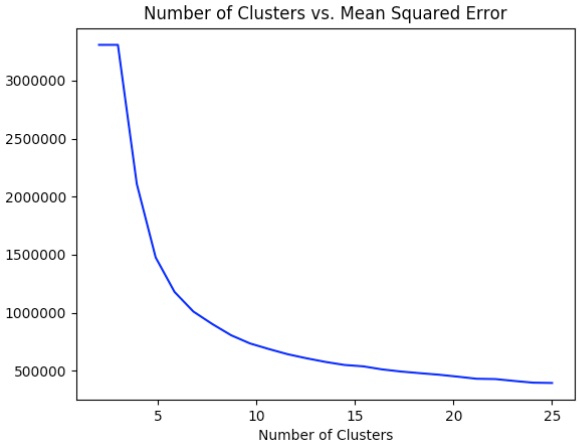
We begin our analysis by applying two types of clustering algorithms, k-means and Gaussian mixture models via expectation maximization, to both datasets.

A. K-means

Figure 3. Displays the number of clusters in k-means vs. the Mean Squared Error. We omit the result with just one cluster, which causes the initial flatline in the graphs.



a. Credit Card Data



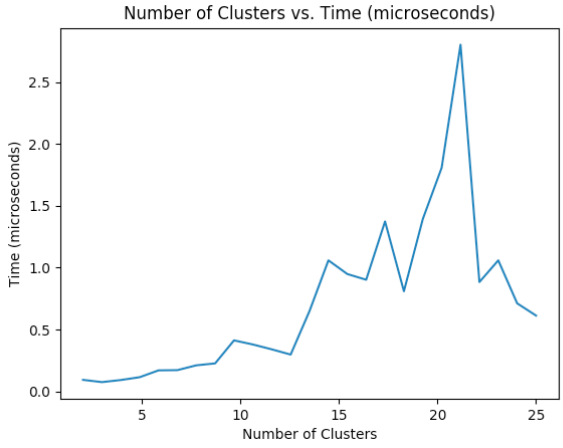
b. Pima Indians Diabetes Data

As expected, the mean square error only decreases as we add more clusters for both graphs in Figure 3. This makes sense because the more clusters we have, the more we can minimize the distance to the nearest cluster. Moreover, the mean squared error in both graphs decrease very quickly for the first few clusters added and subsequently begin to taper off. This is also expected; as we add more and more clusters, the clusters will become closer and closer together. This results in decreasing marginal gains as we increase the number of clusters.

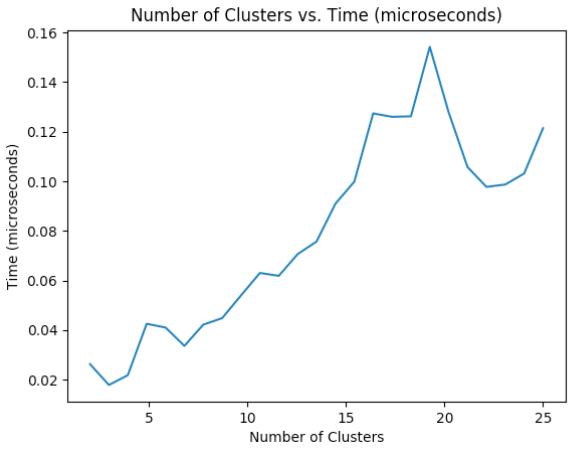
It is clear that the magnitude of error of the credit card data (Figure 3a) is much greater than that of the diabetes data (Figure 3b). This could be for due to a few reasons. This is because Euclidean distance tends to inflate as we have higher dimensional input spaces due to the curse of dimensionality. One other difference to note is that Figure 3b decreases more gradually than does Figure 3a. This implies that the credit card data likely has more "defined" groups of inputs.

Based on Figure 3, we would select a k (number of clusters) of 6 for the credit card data and a k of 7 for the diabetes data. This is because the graphs clearly have elbows at these points, signifying decreasing marginal gains.

Figure 4. Displays the number of clusters in k-means vs. the running time in microseconds.



a. Credit Card Data



a. Pima Indians Diabetes Data

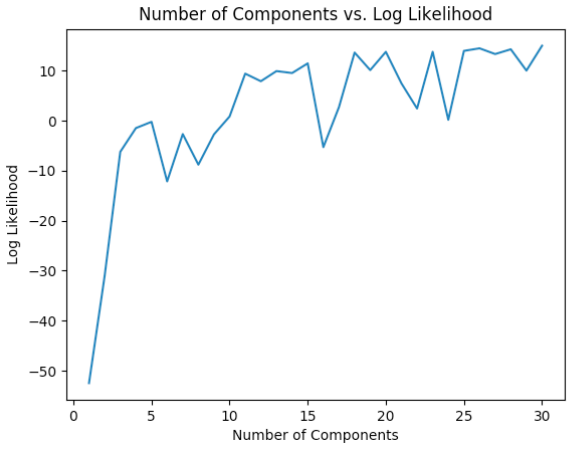
As we expect, the running time of both graphs increase as we increase the number of clusters. It is difficult to

explain exactly why both the graphs are 'jumpy.' We believe that these timings are strongly affected by where each cluster is initialized, which is somewhat random.

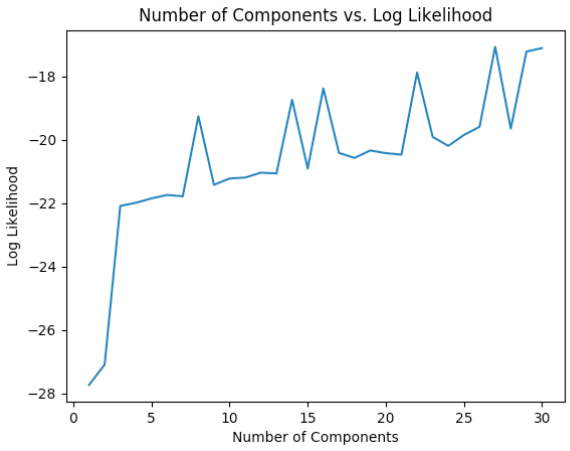
We observe that the algorithm takes much longer on the credit card data. This happens for two reasons: the higher dimensionality of its input space and the larger number of instances.

B. Gaussian Mixture Models (GMMs)

Figure 5. Displays the number of components in GMM vs. the log likelihood.



a. Credit Card Data



b. Pima Indians Diabetes Data

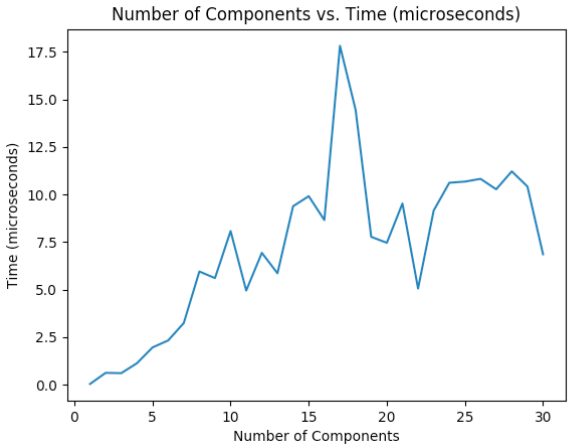
First, we establish that log likelihood is a measure of how well the data points fit the clusters to which they are assigned. Therefore, a higher log likelihood is ideal. Components are very similar to clusters, and they must be predefined for k-means clustering with scikit-learn.

Figure 5 shows that both algorithms tend to improve as the number of components increase. We also observe that the log likelihood increases quickly initially, but it tends to taper off as we add more components. These results are very similar to what we had found for k-means clustering and can be explained by the same reasoning.

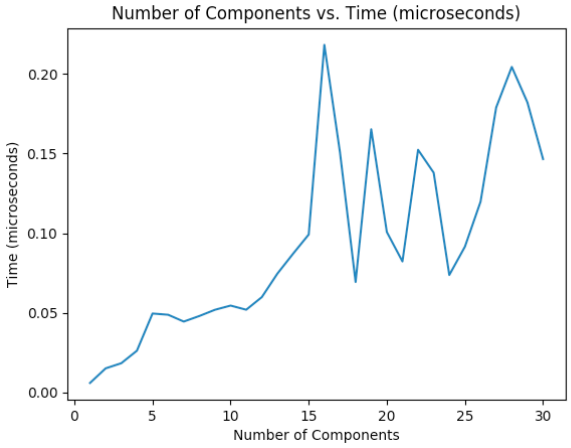
To determine the amount of components that we would use in an actual implementation, we apply similar logic to what we did in k-means: look for diminishing marginal gains. For the credit card data, we would choose 5 components; for the diabetes data, we would choose 8 components.

We clearly see that the log likelihood of the credit card data is much greater than that of the diabetes data. As a result, we determine that GMMs have been much more effective in clustering the credit card data than the diabetes data.

Figure 6. Displays the number of components in GMM vs. the running time in microseconds.



a. Credit Card Data



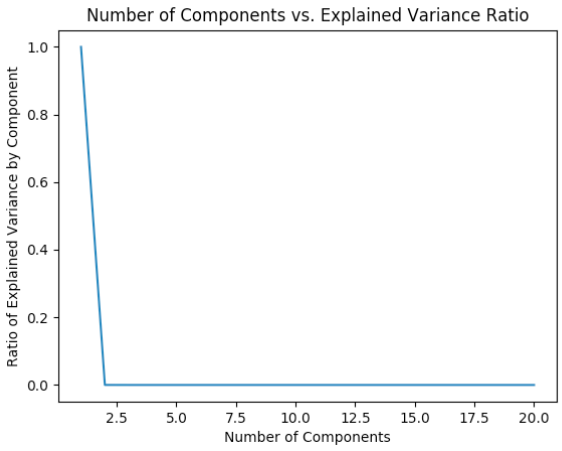
b. Pima Indians Diabetes Data

These results are, once again, generally quite similar to our results for k-means. Both the figures in Figure 6 increase when you increase in time as we the number of components and are quite 'jumpy.' The reasoning for these are exactly the same as what we previously had described for k-means.

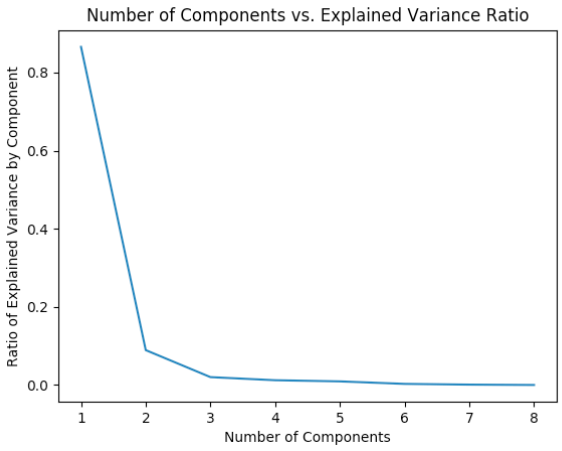
One thing that is important to note is that GMM took longer for both datasets. We see that, in particular, the training time of the credit card data with GMM greatly surpasses the same with k-means. These occur because it is creating a distribution of probabilities, rather than just diving the input data into groups.

III. PCA Transformation

Figure 7. Displays the number of components vs. ratio of explained variance.



a. Credit Card Data



b. Pima Indians Diabetes Data

Here, the explained variance ratio means the amount of variance normalized to 1 for which a single component accounts.

In Figure 7, we the components are ordered from highest explained variance to lowest explained variance, so the graphs must necessarily decrease.

In Figure 7a, we see that the first component accounts for over 99.9% of all variance. We did not expect this at all. We might determine that the dataset is somehow flawed due to this fact, but we continue to carry on our analysis. Moreover, we would only select the first component to continue with in our analysis with the credit card data. This value will also be important in the following section.

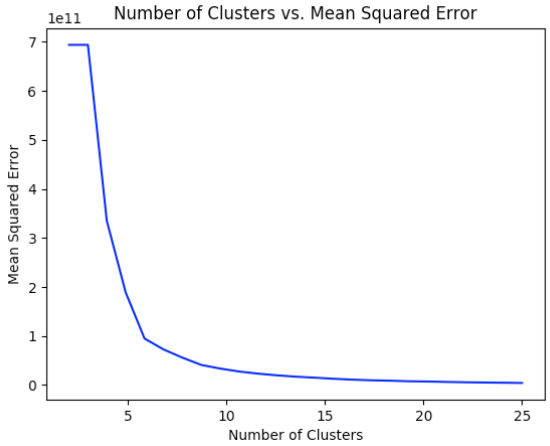
In Figure 7b, we see that the first component still accounts for about 85% of the total variance, which is still quite high but not too abnormal. However, we can clearly see an elbow when we have 3 components, so we will the first 3 components in our next section. Together, these 3 components explain about 97% of the total variance.

IV. Clustering and PCA

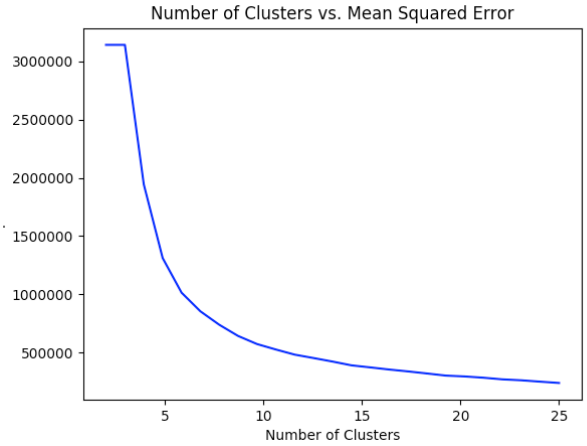
We know use the results of the last section to reduce our input dimensions accordingly and apply the clustering algorithms once again.

A. K-means

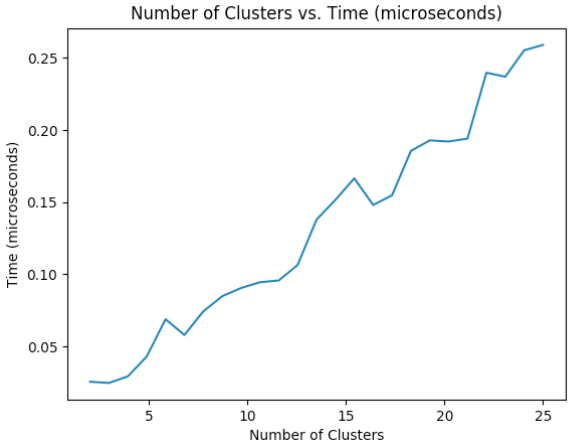
Figure 8. Displays the number of clusters in k-means vs. the Mean Squared Error on the PCA transformed data.



a. Credit Card Data



b. Pima Indians Diabetes Data



a. Credit Card Data

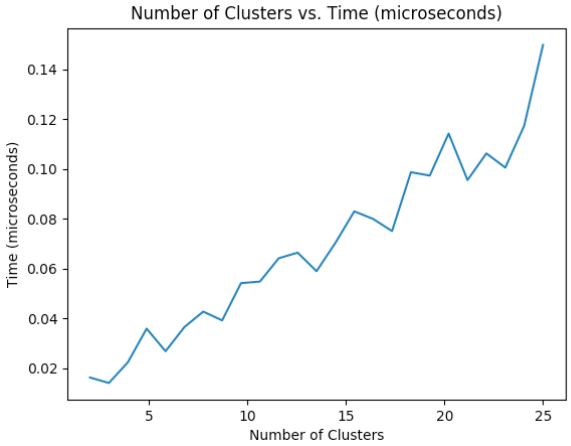
We will analyze the clusterings in Figure 8 by comparing them to their original clusterings in Figure 3.

Figures 8a and 3a, which represent the credit card data are indistinguishable. This is to be expected because our reduced number of dimensions still account for over 99.9% of all variation. Therefore, we have represented the original dataset very well, even though we have reduced it to one dimension.

Figures 8b and 3b, which represent the diabetes data are also very similar in shape. However, if we examine the y-axis in Figure 8b, it is clear that the maximum value of 3,000,000 has been shifted up when compared to the y-axis in Figure 3b. Why has our mean squared error gone down by a small amount? With the 3 components we are using for the diabetes dataset, we account for about 97% of the variance. As a result, the overall mean squared error will decrease slightly, as indicated in the figures.

The graphs are so similar, in fact, that we would not change the k from the original groupings. We would still choose a k of 6 for the credit card data and a k of 7 for the diabetes data.

Figure 9. Displays the number of clusters in k-means vs. the total running time on the PCA transformed data.



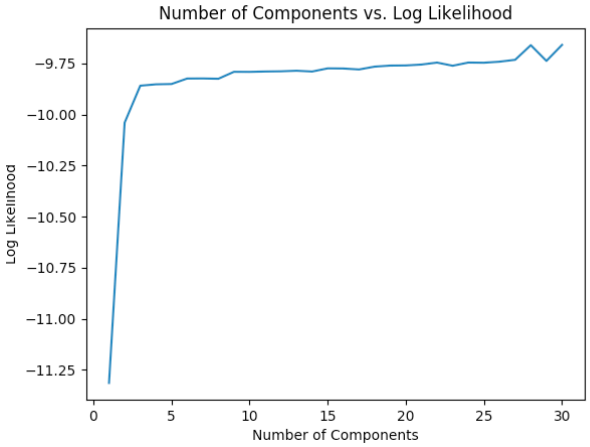
b. Pima Indians Diabetes Data

Comparing Figure 9a to Figure 4a, we see that the running time for the credit card data has drastically reduced. By transforming the input from 28 dimensions to just 1 dimension, it makes sense that the algorithm greatly reduces its runtime. Although much less noticeable, we also see that the runtime of the diabetes data decreased. It has not reduced as much because we only transformed the input from 8 dimensions to 3 dimensions.

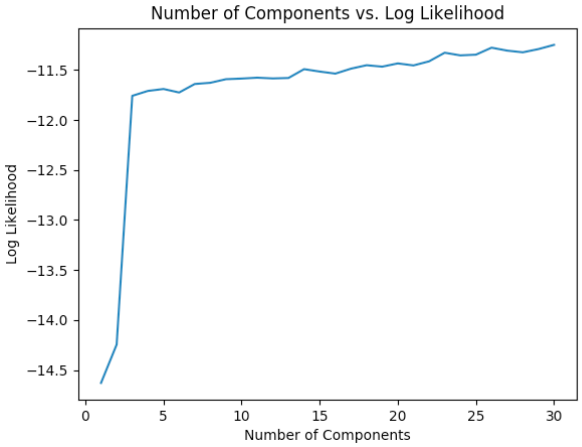
We also note that both graphs in Figure 9 increase quite linearly and are not quite as random as the previous graphs. This is likely because the reduced dimensionality makes it much easier to select good, less random starting locations for our clusters.

B. Gaussian Mixture Models

Figure 10. Displays the number of components in GMM vs. the log likelihood on the PCA transformed data.



a. Credit Card Data

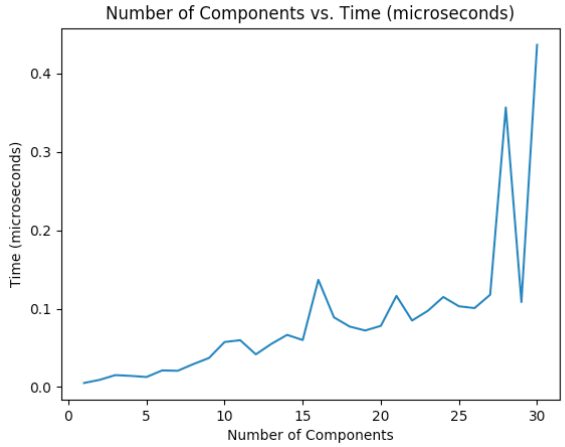


b. Pima Indians Diabetes Data

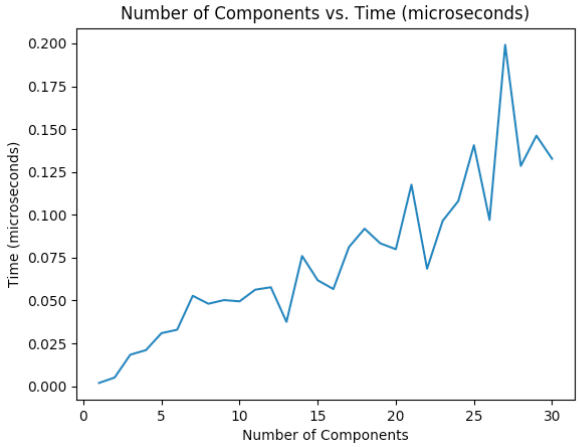
In comparing Figure 10a to its counterpart, Figure 5b, it is clear that the log likelihood of the credit card data has decreased drastically. It is difficult to truly understand why any drastic change would occur given that our reduced input accounts for over 99.9% of the variation. This reinforces the notion that there is some flaw with this dataset and that its results may not be completely understandable. One possible explanation is that another component (one that we did not include) has a small amount of variation that greatly helps the algorithm identify good cluster centers. Still, the most likely explanation is some flaw with the data.

Furthermore, Figure 10b looks to be about what we expected. It looks relatively similar to its counterpart in 5b, but the log likelihood has clearly increased. This fits the trend from our interpretation of k-means on the PCA transformed data: the algorithm gets slightly "better" because some amount of variation is reduced.

Figure 11. Displays the number of components in GMM vs. the total running time on the PCA transformed data.



a. Credit Card Data



b. Pima Indians Diabetes Data

Similar to what we observed with the k-means data on the PCA-transformed set, we see an even bigger drop in training time for the credit card data and a much smaller but still apparent drop for the diabetes data. As we stated earlier, this is due to the reduced dimensionality.

Overall, our most important takeaways are that we are able to significantly reduce runtime and achieve similar results with clustering algorithms by using PCA, provided we are able to reconstruct a large proportion of the data with a fraction of the components.

V. PCA and Neural Nets

The next step of our analysis is to use the PCA-transformed diabetes data, split into training set with 80% and a test set with 20% of the data, to train a neural network. We do this by learning a projection from the

training set, projecting the training and test set, and train a neural network on the newly projected data.

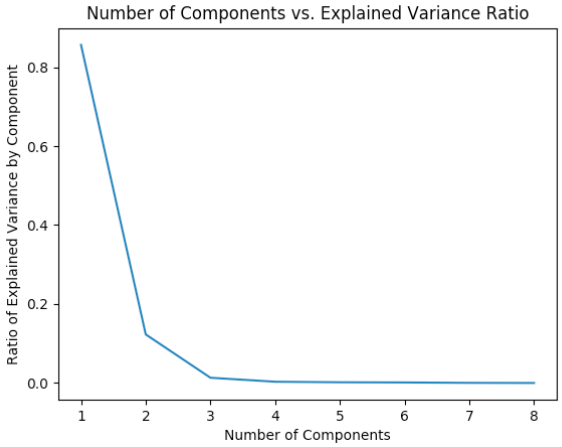


Figure 12. Displays the number of components vs. explained variance ration on the PCA transformed training set.

First, we must select how many components to use from the PCA-transformed data set. Figure 12 seems very similar to its previous counterpart, and we will choose to use the first 3 components once again.

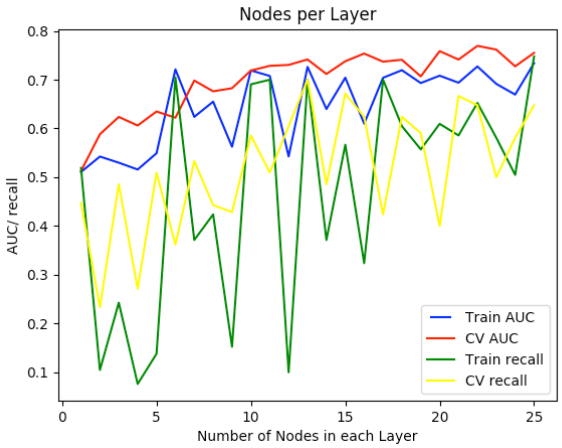
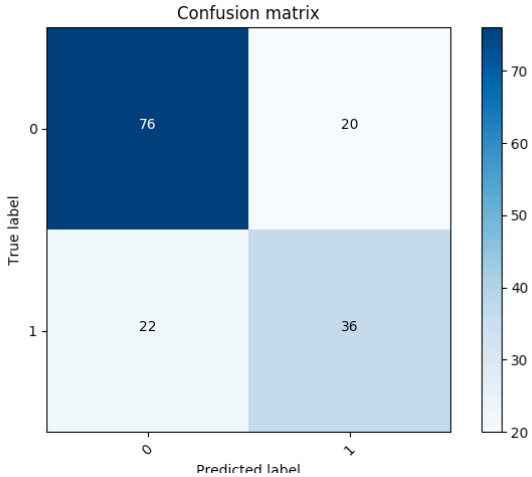


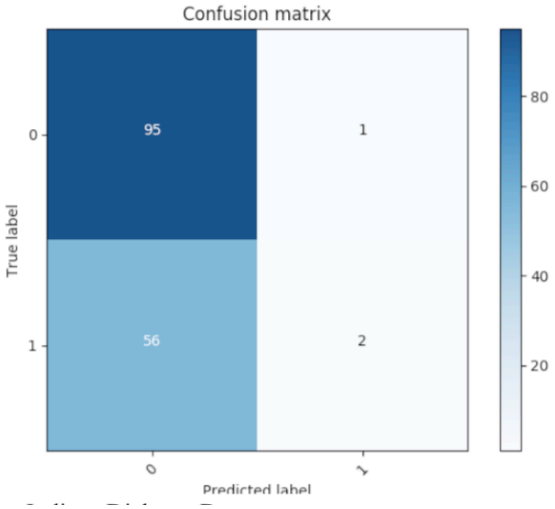
Figure 13. These metrics display recall and Area Under Curve (AUC) versus the number of nodes in each layer. AUC is a common metric used in binary classification to assess accuracy. A higher AUC is preferred.

Here, we are showing how we are pruning the number of nodes that should be in each layer to maximize accuracy. Overall, cross-validation AUC is generally rising as we increase the number of nodes in each layer. In this case, we see that 22 nodes in each layer maximizes AUC, our measure of accuracy, so we will continue forward with that.

Figure 14. Displays the confusion matrices of the neural networks with data from the PCA transformation and with the original data.



a. Neural Network with PCA-Transformed Data



b. Neural Network with Original Data

From Figure 14, we can determine that the neural network with the PCA transformed data (NN1) is about 73% accurate and the neural network with the original data (NN2) is about 63% accurate. We also observe that NN1 is much better at predicting 1, which means it much better at predicting when a person actually has diabetes.

This was somewhat unprecedented. Our hypothesis was that the information lost from the transformation would make the neural network with the PCA transformed data perform slightly worse. However, we have found that it actually performs significantly better. This must be because the PCA transformation helps get rid of some unwanted noise or complexities from the input data and gives the neural network a representation of the input data that actually improves the performance.

One way I could potentially improve this algorithm would be to increase the limit on the number of nodes per layer. In Figure 13, we see that the general trend of the AUC cross-validation is still up when we have 25 nodes per layer. More nodes per layer may or may not make the algorithm more accurate.

VI. Clustering and Neural Nets

The last piece of analysis we want to conduct is to try to use the predictions of our clustering algorithms as inputs to a neural network in place of the original inputs. We will analyze the results.

A. K-means

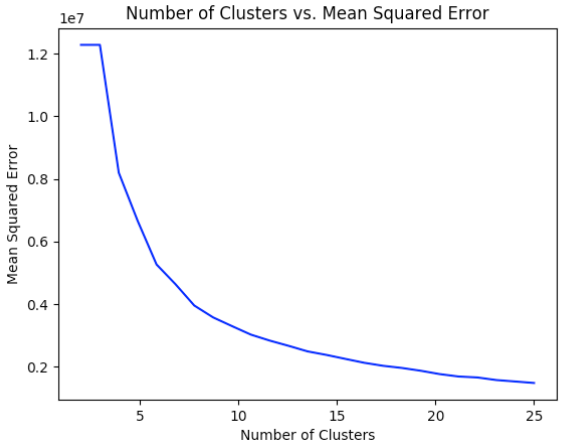


Figure 15. Displays the number of clusters for k-means vs. mean squared error on the training set.

First, we must examine Figure 15 and determine what k should be given the training data. When there are 8 clusters, there appears to be a nice elbow, so k will be 8.

Now, we use this k-means model to predict the training inputs and test inputs into one of these 8 groups and subsequently use that as inputs to a neural network.

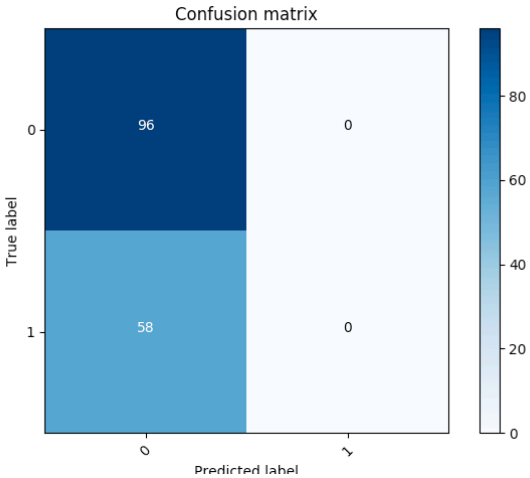


Figure 16. Displays the confusion matrix for k-means predictions as inputs to a neural network.

We can see from Figure 16 that the neural network always predicts 0. Recall that the diabetes dataset is unbalanced in favor of 0. By only predicting 0s, the neural network is 62% accurate. If we look back at Figure 14b, we can see the neural network with the original data is only 1% more accurate. Therefore, it is not at all surprising that the information loss from using just the clustering did not help very much.

B. Gaussian Mixture Models

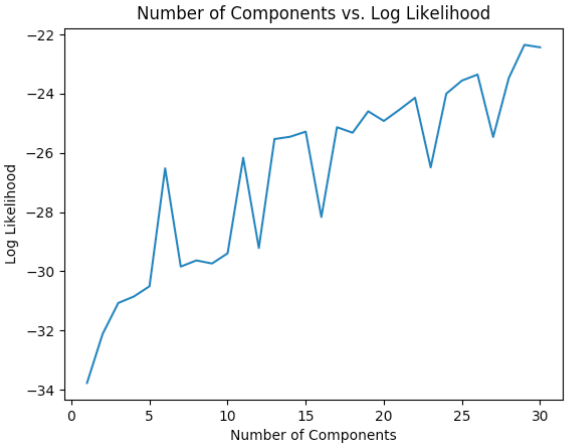


Figure 17. Displays the number of components for GMM vs. log likelihood on the training set.

Once again, we must determine the ideal number of components from Figure 17. There is a fairly high peak when we have 11 components, so we will move forward with 11 components.

Now, we use this GMM model with 11 components to predict the training inputs and test inputs. We use the probability distributions as inputs to a neural network.

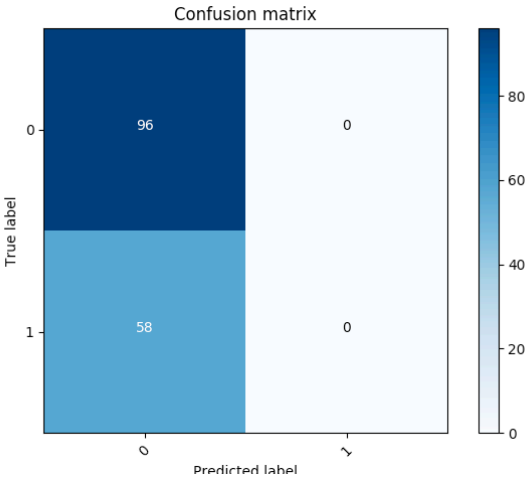


Figure 18. Displays the confusion matrix for GMM predictions as inputs to a neural network.

Once again, we see that the neural network chooses only to predict 0. Even though the GMM will likely give a lot more information that k-means did, this is still not surprising for the same reasons.

Overall, we have seen that using PCA to transform input data may actually improve the results and reduces run time. On the other hand, using the outputs of clustering algorithms as your only input to a neural network will likely give you subpar results.

VII. Conclusion

A. *K-means and Gaussian Mixture Models*

We have observed that the evaluation function of both these algorithms improves as we use more clusters or components. The tradeoff is an increase in running time and computational power necessary. Furthermore, we saw that using clustering algorithms with PCA can be helpful: the algorithm will provide similar results much faster as long as you are able to reconstruct the input data well. Lastly, we learned that using the outputs of clustering algorithms as the sole input for a neural network, or likely any learner, is likely to result in substandard results.

B. *PCA*

We learned that PCA can help reconstruct a dataset very well with only a small fraction of the inputs. We also observed how a good reconstruction with PCA improves training time and has little effect on the evaluation of clustering algorithms. Lastly and most importantly, we observed that PCA transformations can be used to not only decrease training time of a neural network (or any other learner) but also increase its accuracy.