Assignment 3 Analysis **Chris Farr, gid: cfarr31**

Introduction

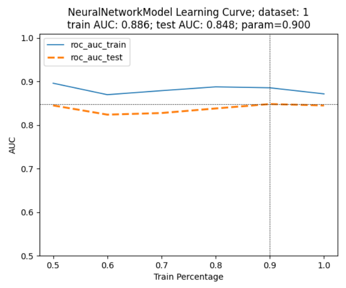
In prior assignments the focus was on various modeling algorithms and techniques. This gave me a good foundation in many different algorithms while using the input dataset largely unaltered. However, there is potential opportunity for improving model performance further through feature engineering, feature selection, and more generally dimensionality reduction. This analysis will focus on various dimensionality reduction techniques through unsupervised learning.

Datasets Introduction

The first dataset is originally from the National Institute of Diabetes [1]. It is a smaller dataset that contains 768 examples. Moderate class imbalance with ~35% of examples containing the positive target. This data is a clinical dataset with 8 clinical descriptors for features including pregnancies, glucose, blood pressure, skin thickness, insulin, BMI, diabetes pedigree function (which is a clinical diabetes risk score), and age. All features are either continuous or ordinal. An interesting trait is that the patients contained in the dataset are all female and from similar heritage, lending to a homogenous set of examples in that regard. The shorthand name for this dataset is **ds1**.

The second dataset is from the Behavioral Risk Factor Surveillance System [2] and collected by the CDC. This is a large dataset with ~253K examples. This dataset has more extreme class balance with only ~16% positive examples. The patients contained in this dataset are from a variety of demographics and all of the data points are self-reported by the patient. The dataset contains 21 features which include a variety of clinical, demographic, lifestyle data points. The shorthand name for this dataset is **ds2**.

These two datasets are interesting together because they have similar targets but are different in how the data is collected, the underlying population of patients, and the features available. These differences should provide many opportunities for analyzing the impacts they have on predictive performance and model behavior. [The descriptions above were largely copied from my Assignment 1 dataset descriptions]



Benchmark SL Performance for DS1

Without any feature engineering or transformations the learning curves for ds1 is shown above. Comparisons can be made between test performance and over or underfitting after various feature engineering and feature projection steps, especially for dimensionality reduction, are made below. Through my experiments my goal is to find a set of transformations that would improve the performance for ds1 by reducing overfitting while limiting the reduction in test performance.

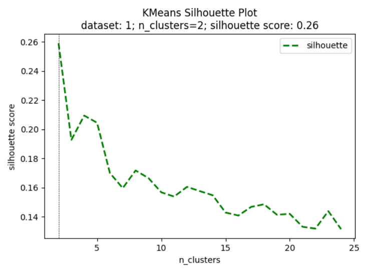
Introduce Clustering Algorithms: Experiment 1

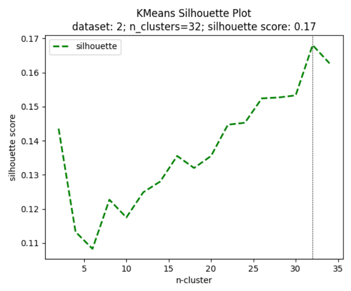
KMeans

The first experiment is K-Means clustering. For all KMeans experiments I'm using 10 random initializations with 300 max iterations. Having multiple random initializations ensures that the final output is not limited to a poor local minima which may be observed if the random initialization caused the algorithm to get stuck in the local minima. The random start involves a centroid for each desired cluster before iteratively assigning examples to the centroid, updating the centroid, and then reassigning clusters based on best fit per example. In this case, fit is a measure of inertia or within-sum-of-squares. Essentially for each datapoint this measure can be computed as a sum of the squared differences between the example and a centroid, the example is then assigned to the centroid with the lowest inertia. It is important to scale the data points so that all data points errors are handled evenly. If two data points are on different scales of magnitude, then the higher magnitude will dominate the error metric and the results will likely be meaningless or heavily skewed towards the higher magnitude features. For these experiments, I have min-max scaled the data points so that all values range from 0 to 1.

silhouette score: The silhouette score is computed on a per sample basis and it is a measure that describes how well clusters are defined in a set of examples. The metric increases if examples within a cluster are more similar and examples outside a cluster are more distinct. Possible values range from -1 (the worst) to 1 the best [3]. Below I will perform multiple experiments that involve clustering the examples for each dataset. I will use the silhouette score to determine which number of clusters to use for further downstream experiments as well as to better understand how well clusters are defined as the number of clusters increases.

K-Means Clustering





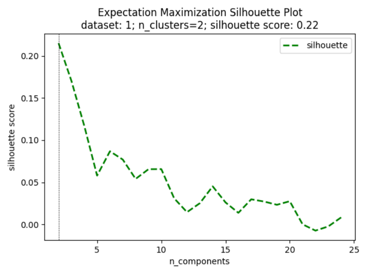
When running KMeans on ds1 the best silhouette score is for 2 clusters and the score progressively decreases as more clusters are added. This could be an indication that ds1 does not naturally have well defined clusters. Perhaps before this dataset can be clustered into more than 2 well-defined clusters it needs some type of transformation first.

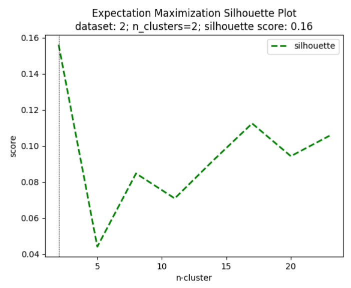
Somewhat opposite of the ds1 results, after 20 clusters the silhouette_score reaches a new high and this continues till at least 32 clusters. It seems that with more clusters in ds2 it leads to better defined clusters while for ds1 fewer were better defined. Maybe the types of features are contributing to this since ds1 has more continuous features and ds2 has more binary features, I would imagine that binary features allow for easier definition of clusters. For instance it would be easier for a single binary feature to decide how to cluster, all with a 1 go in one group and all with a 0 go in the other. This is less clear for a continuous because some arbitrary cutoff is required. The runtime to complete the silhouette plot took 3537.1 seconds for ds2, which is nearly prohibitive when it comes to fine-tuning parameters.

Expectation Maximization (EM)

The next algorithm is a Guassian Mixture Model which models Expectation Maximization. My understanding of this algorithm is that the centroids are first initialized using a clustering method like K-Means or through random initialization, from there the algorithm begins to iteratively shift the centroids. However, instead of the centroids being computed as only a function of their assigned examples, a distribution is computed that allows each example to impact all centroids based on the probability that they belong to that centroid. If an example has a high probability of belonging to a centroid then it has more of an impact on the centroid calculation in the next iteration. This continues until the convergence criteria is met which basically measures how much the centroids are changing. [5]

Scaling the input data is also important for this algorithm and for my experiments below I have used min-max scaling as before.



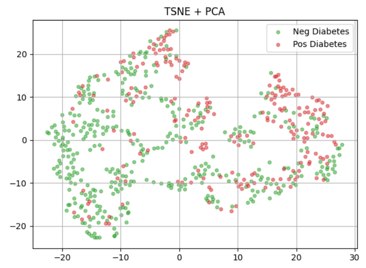


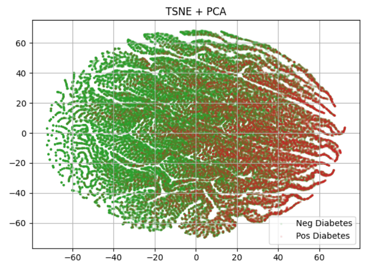
Analysis: Compared to K-Means, the silhouette score for 2 clusters is slightly worse when using EM. Otherwise, the behavior as cluster count increases is very similar. K-Means appears to be better at creating more defined clusters than EM for ds1. I see a similar line shape between K-Means and EM for ds2, although due to runtime I wasn't able to test as many clusters and in this case the best silhouette score found was for only 2 clusters. Perhaps similar to KM if more clusters were tested after >25 clusters the silhouette score would reach a new high. Based on this visual though it also seems there are more diminishing returns when given more clusters than witnessed in K-Means. For ds2 the silhouette plot completed in 3186.4 seconds. I am currently still using the default k-means initialization but for later experiments this may become problematic.

Introduce Dimensionality Reduction Algorithms: Experiment 2

The next set of experiments involve dimensionality reduction in some form or another. The first algorithm I will use is PCA, or principal component analysis. At a high level this algorithm transforms the input data into a specified number of components where a component is basically a direction of the underlying data in high dimensional space. The first component has the strongest signal coming from the dataset and each subsequent component is orthogonal to all previous components. [6] To visualize the outputs I will only be using the first 2 components along with TSNE which is a visual representation of the data that converts the data points to joint probabilities which allows structures within the data to be represented visually. [7]

Principal Component Analysis (PCA)





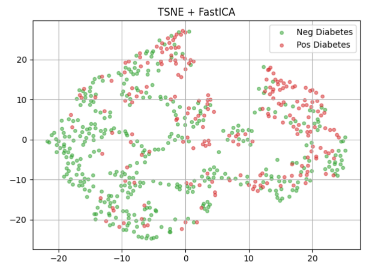
When using Tsne to visualize 2 components some clear patterns emerge which seem to generally separate negative and positive classes. Interestingly, this is able to occur without any knowledge of the targets. Considering that the inputs here are clinical and related to diabetes it is not surprising that Tsne can loosely separate the classes. There are also many instances of a positive example placed near groups of negative. Similar to how model performance can't be perfect, here, there are also challenges with separating examples mainly coming from the fact that the algorithm doesn't know which features are more important for a particular task.

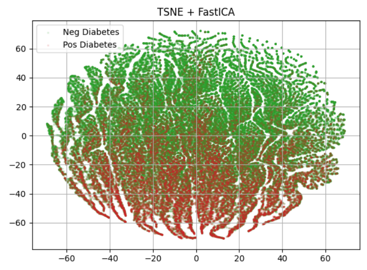
After the first iteration it was hard to see the label colors because of the density of the examples. After shrinking the size of the scatter dots and also making them transparent I can see more structure and a clear layout where patients with diabetes tend to plot on the right and patients without tend to plot on the left. There is also a considerable amount of overlap as well as some exceptions. With more time I might play around with transparency more to see the gradient of patients from left to right. For ds2, the Tsne plot completed in 226.9 seconds.

For both visuals when testing with 3 components there was less structure defined. Since these are being plotted on a 2D space it would make sense that the 3D Tsne is losing some structure. I find it fascinating that using PCA and Tsne results in separating or centralization of the target classes without having any prior knowledge. I suppose if this weren't true at all then maybe it would be an impossible prediction problem. This shows me that there is a signal of the target even without the target being present.

Independent Component Analysis (ICA)

Similar to PCA, ICA is an algorithm that converts input data into components, and in this case the components are "maximally independent" [8]. It is not clear to me exactly how it works, however, I have observed from these experiments that the results are very similar to PCA. As I show below the results are nearly identical after additional manual transformations are made.



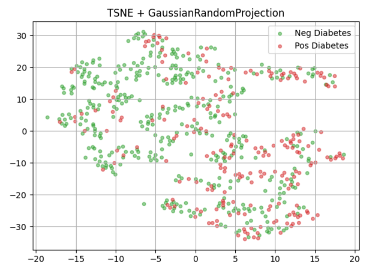


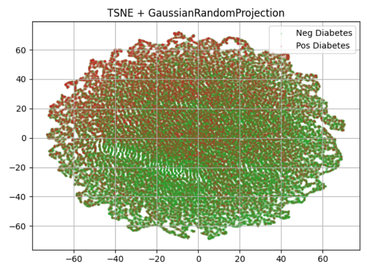
The outputs of ica for ds1 are almost a 180 degree rotation of pca, indicating very similar results. I applied a 180 degree rotation to the Tsne plot for ICA to illustrate this property. When comparing the rotated ica to the pca for ds1, nearly all of the structure looks similar except for an additional tail reconnecting the lower left area to the lower middle of the structure. This tail in PCA connects to the other side but is still visibly present. From this I would conclude that only a handful of examples are treated differently and even for those there is some consistency in how they are mapped to the projected space between PCA and ICA. If I were to use a projection in an SL model, based on this analysis it would be worth trying both methods because there may be an important exception to how some examples are projected and the desired handling may depend on the problem at hand.

For ds1 I again saw similarities between the PCA and ICA output when using the same dataset. For the first dataset this was a 180 degree rotation, here though I maybe see a mirror on the vertical axis then a 90 degree count-clockwise turn. Due to the added complexity of the projection I didn't attempt to rotate the output but it can be clearly seen from the visuals. I continued to use only 2 components for the Tsne plot to avoid losing information about structure. This resulted in very interesting visuals with clear structure.

Randomized Projections

The next algorithm is called Randomized Projections. This algorithm seems very simple in that it takes a random gaussian matrix and projects the input data into lower dimensional space. The pairwise distances between any two points in the dataset are preserved and the purpose of this algorithm is to trade accuracy for lower dimensionality just as with PCA and ICA [9].



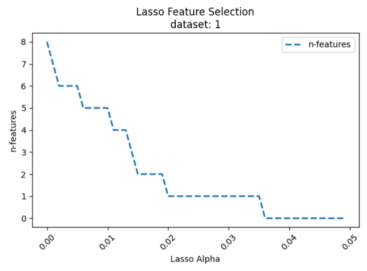


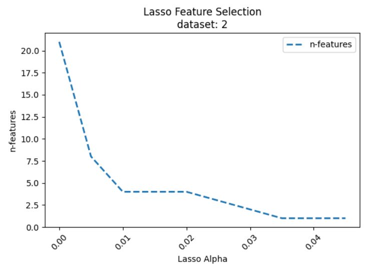
Testing both 2 and 3 components for the Tsne plot, 2 appears to provide more structure as well as more separation of the positive and negative classes. Of course, to avoid making decisions based on the label I would choose 2 because it provides more structure, although I think I biased myself by looking at the class separation first now no matter what I decide it will be in an effort to unbias myself which is impossible now.

The result of ds2 is essentially looking at a blob of points with maybe some interesting structure around the edges. When the positive labeled data appears on the front of the image it appears as if the negative labeled data may tend to one side of the graph. However once the negative is brought to the front it can be seen that the negative samples overwhelm the positive and limited separation is apparent. Based on this analysis I would conclude that random projection is not suitable for ds2 as it creates no natural separation between the two target classes. Once I reduced the opacity of the dots I could see more clearly that positive examples tend to be in the top left of the graph while negative tend to be in the bottom right. Although there is still not much separation it does appear to have found some degree of target signal in the dataset without ever seeing the target.

Lasso Feature Selection

The final dimensionality reduction algorithm I will use is a Lasso Feature Selection algorithm. This approach actually requires the label as it is used to train a lasso model which indicates which data columns to keep based on non-zero coefficients. As regularization in the Lasso model is increased, more and more columns will have a zero coefficient until all columns are removed.



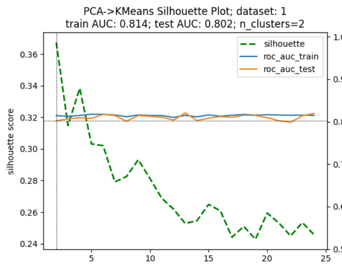


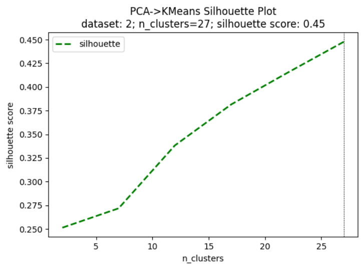
My goal here is to understand the behavior of the feature selection algorithm as I change the Lasso Alpha parameter, producing a validation curve in a way, but instead of analyzing model performance I am just looking at how many features are selected. There is a steep drop from 0.0 to 0.02 in which it goes from all 8 features down to only a single feature. Perhaps looking for longer plateaus is a natural and semi-supervised approach to selecting how many features to use. A plateau means there is a gap in alpha space between the decision to drop the next feature. Similar in concept to the elbow method, but instead of diminishing returns for inertia it would be diminishing returns for increased regularization on the number of features selected. If I were to use that approach here I would probably stop just before Alpha=0.01 and use the 5 features selected at that point. Using a similar approach to ds1, there appears to be an elbow which creates a plateau when alpha=0.01 for ds2. At this point there are about 4 features selected. Using some intuition I might opt for about .05 where there are 8 features since more than half of the original features are dropped by this point. It would be worth trying both in a SL setting to see which performs the best.

Clustering Projected Data: Experiment 3

Now that I have experimented with projecting the datasets into lower dimensionality as well as clustering the original datasets, I will experiment with a combination of first reducing dimensionality and then clustering the resulting project of the data. As was done in the previous experiments I will continue to use the Silhouette plot to determine how many clusters are best. New for these experiments I will also incorporate the target information to evaluate the quality of the resulting data transformations in a supervised learning setting. For each experiment for ds1 I performed a validation curve and overlaid it on the analysis used previously. The validation curve for the clustering algorithms has the silhouette score on the y axis and number of clusters on the x-axis (pardon the missing labels). Then on the secondary axis AUC was shown which is a metric that penalizes incorrect ordering of examples, something readily used in the healthcare industry in my experience. For the feature selection algorithm a similar approach was used except replacing the silhouette score with the number of features selected on the y-axis and the lasso alpha parameter on the x-axis.

PCA & K-Means





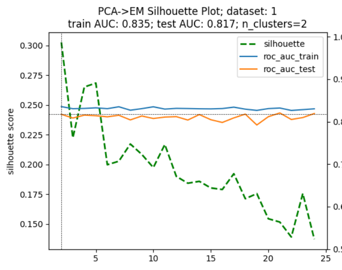
I tested these experiments with a various number of components for the PCA transformation.

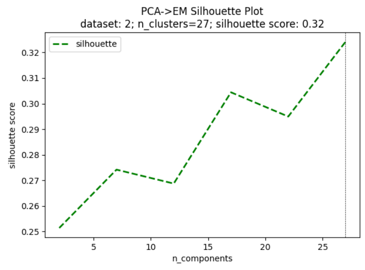
- 2 components, 2 clusters: 0.766 test and 0.77 train AUC
- 3 components, 2 clusters: 0.802 test and 0.814 train AUC
- 4 components, 2 clusters: 0.811 test and 0.829 train AUC
- 5 components: 2 clusters: 0.814 test and 0.84 train AUC

I could see that many cluster settings performed better than 2 even though the silhouette method preferred only 2 clusters. For example, 7 clusters resulted in a test AUC of 0.82. Also, in some cases 2 clusters appeared to reduce overfitting at this number of components although that is not shown above. The general trend observed here is that with more components the model is more likely to overfit the data and too few components results in underfitting. Additionally, it seems that increasing the number of clusters doesn't help with improving test performance in nearly any case. Maybe this points to the silhouette score as a reliable way to select the number of clusters without knowing the targets considering the silhouette analysis points to a recommended 2 clusters.

For ds2 I only took a single shot because it takes so long to run. I learned a lesson from the ds-1 experiment where 4 components were the best balance between test performance and overfitting. Results show that up to at least 27 clusters, adding more clusters is resulting in more defined clusters. The optimal number of clusters is then >=27. Now I compare the results of the silhouette analysis here to when I performed it without first the PCA transformation. Generally the results are similar, however, after transforming with PCA the shape of the plot is different and now there is only a positive trend in silhouette score as more clusters are added. I conclude then that the direction of the data that is described in components 5 and higher were the drivers in that higher silhouette score for smaller number of clusters. This approach appears to clean up the silhouette analysis and I might prefer to use PCA if I were to cluster this data.

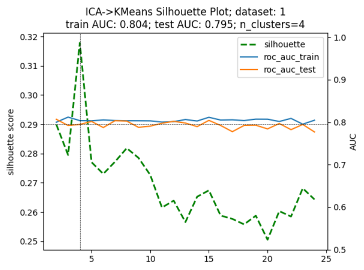
PCA & Expectation Maximization

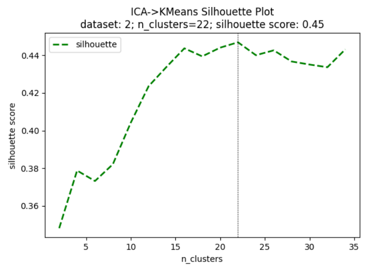




The results in this analysis were very similar to that seen in the K-Means version. A slight difference in ds1 is that I see a little more overfitting in general. It is unclear, however, if the EM feature is causing more overfitting or if the KM feature was reducing overfitting or something else. Again for ds2 I see a different shape to the silhouette curve in that again the score now only improves as more clusters are added. I can come to the same conclusion in the last analysis for KM that the PCA components left out here were contributing to the early increase in silhouette score, which is likely misleading. So I would probably prefer to use PCA if I were to perform EM for ds2 as well.

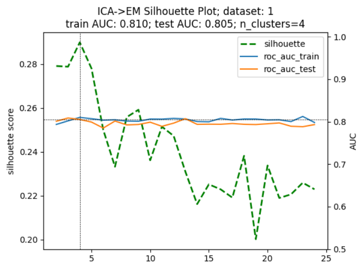
ICA & K-Means

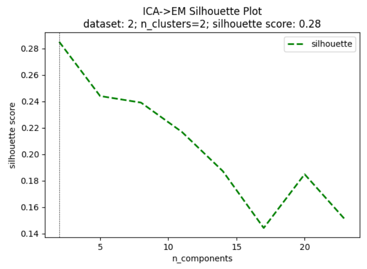




For ds1 I am generally seeing limited impact on the final score as the single new feature clusters increase. For this and all similar experiments I think that the reason is the cluster features alone do not add significant signal that wasn't already present. It is also possible that the clusters correlate with the existing features, which in this case would be ICA components used for clustering. However, there may be a better way to utilize this approach and I will try one in experiment 5. As for ds2, I imagine that this is the ideal looking silhouette plot. At first the clusters are not very well defined as I would expect for 2 clusters on a large dataset of heterogeneous examples. Then the silhouette score quickly improves as more clusters are added and eventually more clusters have diminishing returns until finally the score begins to drop. For ds2 I would likely opt to use the ICA transformation and if I were also using K-Means I think I would select about 22 clusters when using the entire set of features.

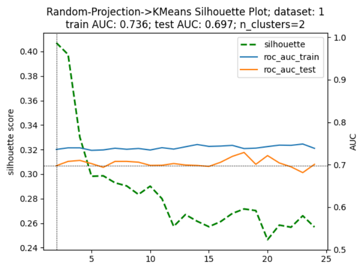
ICA & Expectation Maximization

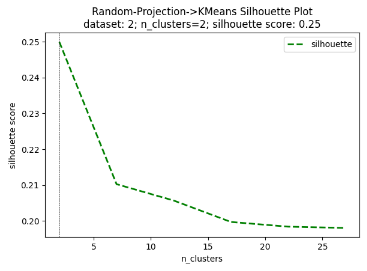




For ds1, one thing to note is the limited overfitting along with still greater than 0.8 AUC. Perhaps there is something to this combination of transformations and feature engineering for ds1. Then something to note for ds2 is that the silhouette plot is now shaped how ds1 is shaped for most curves, while typically ds2 silhouette plots had positive slopes. This tells me that ICA might not be the best dimensionality reduction algorithm to use for ds2 and as noted previously I believe the optimal shape has a positive slope with diminishing returns until a peak is reached. Therefore, I would stick with PCA for ds2 after seeing this.

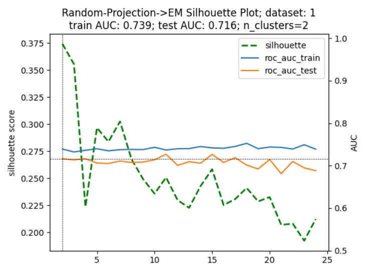
Randomized Projections & K-Means

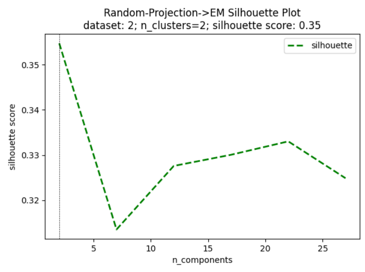




The results for randomized projections were not very impressive. Both ds1 and ds2 silhouette plots have only a generally negative slope and the SL performance on ds1 shows drastic reduction in test performance as a result of underfitting. It would seem that the random projections are creating too much noise in the data which is preventing it from fitting the data properly. Additionally, this noise seems to also enable the algorithm to more easily overfit which could indicate that the noise misleads the model away from learning generalizable signal and replaces it with random noise.

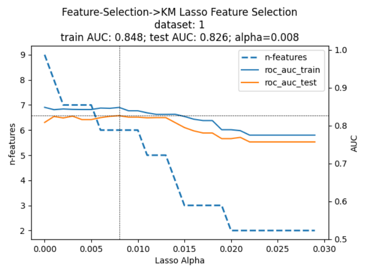
Randomized Projections & Expectation Maximization

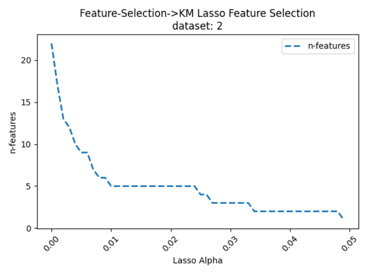




For ds1, 3 components for randomized projections when using EM seems to provide a small boost in performance but the overall result is still inferior performance to ICA and PCA. Perhaps because the random noise is gaussian, the gaussian based expectation Maximization algorithm can piece the generalizable signal back together enough to create this performance increase (compared to Randomized Projection & K-Means). Even still, the result of this experiment shows that performance dropped while also increasing overfitting, not ideal in an effort to improve performance while reducing overfitting. For ds2 the shape of the silhouette plot has changed slightly but still does not compare well against the shape I saw when using ICA & KM.

Feature Selection & K-Means

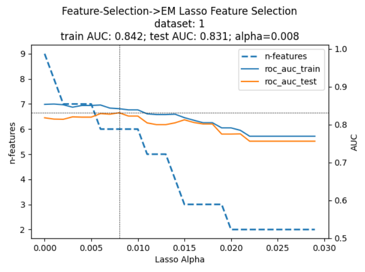




One of the more interesting outcomes this far. To obtain this result I applied lasso based feature selection to the original input data. Then I ran the selected features through the K-Means algorithm. Then I took the resulting data for a given lasso-alpha, which is the parameter that controls how much regularization is applied to the logistic regression, and performed cross-validation on the dataset. Using this approach the best test performance is observed when using an alpha of .008. As alpha increases there is a small drop in test performance while also reducing overfitting. This analysis is interesting because the alpha parameter actually impacts the train and test AUC, whereas with all of the prior combinations the scores were mostly flat. I could imagine a scenario where adjustments made to feature creation need to be properly analyzed against a feature selection algorithm and this analysis demonstrates such an approach.

For ds2 it looks like there are 5 features left when the feature selection plot hits the first large plateau. While I did not extract these features names it would be interesting to learn what they are and to consider incorporating them into additional engineered features.

Feature Selection & Expectation Maximization

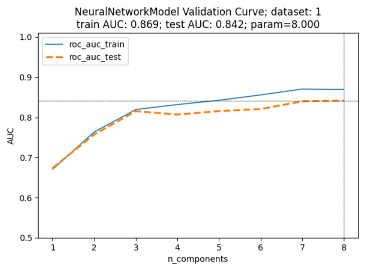


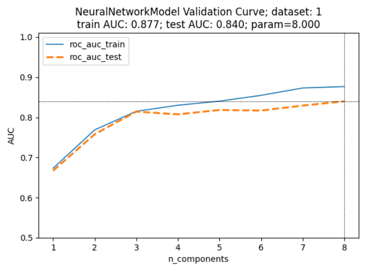
The same analysis for ds1 is repeated here and actually this looks to be the best output from the analysis thus far. The impact that feature selection had in addition to adding the EM feature has resulted in a large decrease in the difference between the train and test scores while only dropping the test AUC by roughly 1. As for ds2, the EM algorithm seemed to not be able to converge as it ran longer than I cared to wait, even after running for several hours. I tested some additional parameters such as testing out a random initialization strategy which is supposed to take less time, however, this is also at the cost of longer convergence times. Either way I was not able to get the algorithm to finish computing on the larger ds2.

Dimensionality Reduction for Supervised Learning: Experiment 4

For experiment 4 the focus is primarily on the combination of the dimensionality reduction algorithms I have introduced so far and their impact on the supervised learning performance for only ds1. For PCA, ICA, and Random Projections, the number of components controls the variance of a function approximator if the data was used for such a purpose. As mentioned previously the first component describes the strongest direction in the data and each subsequent component describes a new and orthogonal direction which is progressively weaker until every dimension in the original dataset is described. These weaker components, I might like to think, may be responsible for causing an algorithm to overfit. I believe this is an inherent trait of weaker components because their information doesn't generalize to the broader population as strongly, otherwise the component would be stronger and would have been selected in the early components. The below experiments illustrate this concept as have some of the previous experiments, such as when PCA was used in PCA & K-Means in section 3, there I tested the model performance while varying the number of components and saw that when I added too many components the model would begin to overfit.

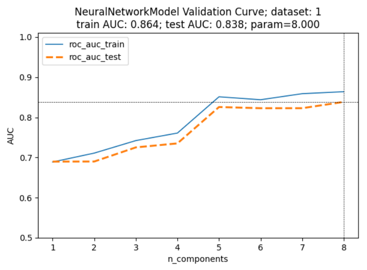
PCA, ICA





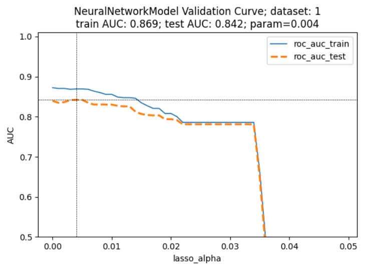
The results for both of these (PCA on left, ICA on right) are very similar. We can't even be sure they aren't identical since we are just analyzing a sample of data. The conclusion I would draw here is that 3 components, as also discovered previously, appears to lead to the highest level of test performance while still limiting overfitting.

Randomized Projections



While the overall shape of this curve is similar to those of PCA and ICA above, there does not appear to be any reduction in overfitting even when using 2 or 3 components. Again it seems that the Randomized Projections algorithm is adding noise which is not helping with overfitting like PCA and ICA and also not resulting in a higher test performance.

Feature Selection



This experiment was slightly different than with PCA, ICA, and Random Projections in that instead of analyzing the number of components along the x-axis I am analyzing Lasso's alpha which is directly tied to the number of input features as previously shown.

Based on this analysis it appears that an alpha of .004 results in the highest score. Looking back at the earlier lasso selection plots it looks like this allows 6 features to be passed to the model. Its not known what those features are but that would also be interesting analysis. A lasso of just over 0.02 appears to nearly eliminate overfitting. Something to consider is finding more and better features to add while keeping the lasso parameter high in an attempt to take advantage of higher test performance with limited overfitting.

Dimensionality Reduction & Clustering for Supervised Learning: Experiment 5

Now that I have seen the impacts of various dimensionality reductions on the DS, these experiments will focus on understanding PCA, ICA, and Random Projections and their impact on SL performance with a slightly different flavor than you might be used to so hold on tight. Additionally, I have been experimenting with clustering in a SL setting by adding a single cluster feature. Now I am going to repeatedly fit clusters to random pairs of components generated by PCA, ICA, or Random Projections. The first reason for this is to find better features through the use of dimensionality reduction and clustering algorithms, and the second is to learn about which components are more likely to be selected by the lasso algorithm when they are included in a cluster.

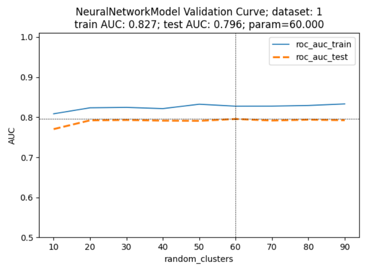
MyAlgorithm looks like this:

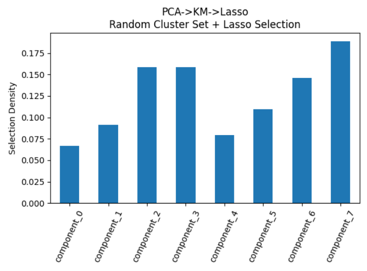
1. Transform the input dataset using PCA, ICA, or Random Projections.
2. Repeatedly select random pairs of components and fit a K-Means clustering algorithm to the data subset and add the resulting clusters as a new feature.
3. Use lasso feature selection to select a subset of the new cluster features along with the pool of original components.
4. Finally, evaluate the selection density of each component to determine its importance in the final model.

Additional notes: For the below experiments I used an alpha of 0.01. This was in an effort to learn which components are preferred by a stricter lasso algorithm and to also help keep the dimensionality lower while I added more engineered features.

Some issues with this analysis are that repeated clusters on only 8 components creates highly correlated clusters. This paired with lasso feature selection may result in the algorithm dropping correlated features entirely even if they were helpful individually. A better approach might be a stepwise algorithm that drops just a single feature at a time to allow the algorithm to decide if it wants to keep a single correlated feature.

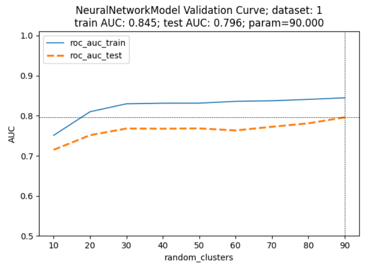
MyAlgorithm & PCA

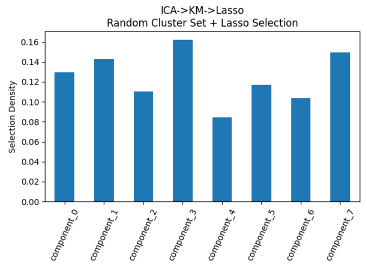




Using PCA as a transformation before applying MyAlgorithm shows that the highest test performance is reached when 60 random clusters are generated. This is of course before the feature selection occurs. The components preferred by this algorithm tend to focus in the middle and even the weakest components. This is not what I was expecting as I thought that the algorithm may tend to favor the stronger components.

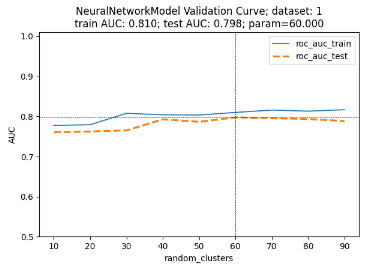
MyAlgorithm & ICA

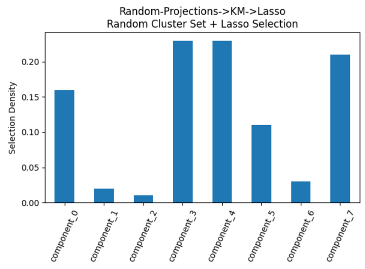




Here I see even more overfitting than when using PCA and there is almost a uniform distribution across the selected components although I can still see that component 1, 3, and 7 (counting from 0) are slightly preferred.

MyAlgorithm & Random Projection





To quite the surprise, MyAlgorithm applied with the Random Projection algorithm has resulted in less overfitting than PCA or ICA. It is also a little surprising that the test performance is comparable to both. After seeing these results my theory is that the algorithm actually benefited from the additional noise and randomness that the Random Projection provides. Inherent in MyAlgorithm, as previously mentioned, is a potential to produce highly correlated features. Then lasso might go and drop these features because of the intercorrelation. If Random Projection creates noisier components, then it might be possible that different clustered component pairs may not have the same level of correlation; this could be what is helping this version to have less overfitting. Another interesting characteristic of this version is that the feature importance plot shows clear favor in select components while all but ignoring components 1, 2, and 6. I can't be sure as to why this occurs but my guess is it is related to what is also helping the algorithm to reduce overfitting.

Conclusion

A consistent theme in this analysis was controlling overfitting through dimensionality reduction. I explored various clustering and dimensionality reduction techniques. It was clear that some algorithms, such as EM, for larger datasets quickly became prohibitive in iterating through experiments. Throughout the experiments the trade off between bias and variance was present. For PCA, ICA, and Random Projection this was controlled by the number of components. By decreasing the number of components test performance may slightly drop in exchange for a drop in overfitting as well. Finding the right tradeoff was a challenge but in many experiments it was quite obvious, for example in experiment 4. Feature selection with Lasso was another way to reduce dimensionality without also having to project the entire dataset, allowing some flexibility there if it is important to maintain a human understanding of the input data which is common in many real world applications such as healthcare. To control the bias variance tradeoff while using lasso feature selection the alpha parameter, which controls how much regularization is used in the lasso model.

Clustering was perhaps a slightly different flavor of this analysis in that it wasn't necessarily used as a feature reduction technique but instead a feature engineering technique. The same bias variance tradeoff should be present here as well, however, as fewer clusters might mean more bias and more clusters higher variance. However, this was not apparent in this analysis. When adding a cluster feature to the dataset it had little effect on the test performance or on overfitting. This is perhaps because the features were merely added to the dataset instead of replacing it. In experiment 5 clusters were the only features present but here they were also repeatedly created with random component pairs. Again, the relationship between the number of clusters and overfitting was not apparent but interesting nonetheless. If I were to recreate this analysis I might focus on passing only a single clustered feature to try and isolate its impact on overfitting.

References:

1. Dataset 1: https://www.kaggle.com/datasets/mathchi/diabetes-data-set
2. Dataset 2: https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset
3. Silhouette Score: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
4. K-Means: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
5. GMM: https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm
6. PCA: https://en.wikipedia.org/wiki/Principal_component_analysis
7. Tsne: https://scikit-learn.org/stable/modules/manifold.html#t-sne
8. ICA: https://scikit-learn.org/stable/modules/decomposition.html#ica
9. Random Projection: https://scikit-learn.org/stable/modules/random_projection.html