

## Unsupervised Learning & Dimensionality Reduction

### Introduction

I am using a small wine dataset that is nearly balanced with 178 instances, 13 continuous features, and 3 distinct label classes. Each instance represents a wine and the 3 classes represent the type of wine or cultivar the wine is from. I normalized the data using a minmax normalization on each feature.

I am also a larger adult wage dataset with 5000 instances, 106 features (including one hot encoded categorical variables and continuous features), and 2 distinct label classes. Each instance represents an adult individual and the classes represent whether this individual makes  $\geq \$50k$  per year or  $< \$50k$  in wages. I normalized the data using a minmax normalization on each feature.

For all experiments below I used python 3.7, numpy, pandas, matplotlib, yellowbrick, and sklearn. For organizational purposes I have split the experiments on the datasets into two columns, the left column contains experiments performed on the wine dataset and the right column contains experiment performed on the wage dataset.

### Clustering

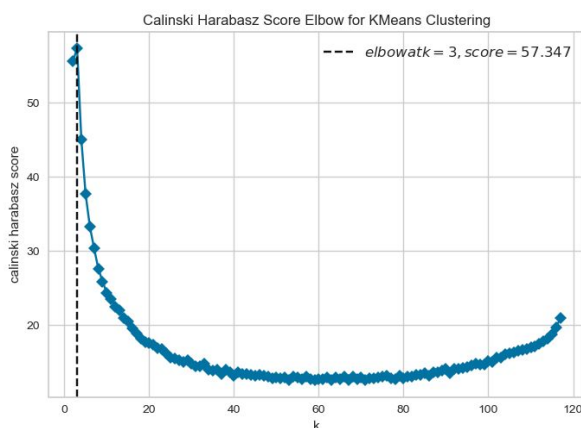
#### KMeans

KMeans clustering is a hard clustering algorithm that creates clusters in a dataset based on the distances between points in that dataset, which completes iteration when there is no more improvement in error (error: the distance of points contained within a cluster to the center of that cluster). I used sklearn's KMeans to perform these experiments.

To find the number of clusters I should use (k), I used the Calinski Harabasz Score. This metric calculates the ratio of dispersion within clusters to dispersion between clusters. This means that higher scores come from lower variance within clusters and higher variance between clusters. For finding (k) a training X (features) set was used.

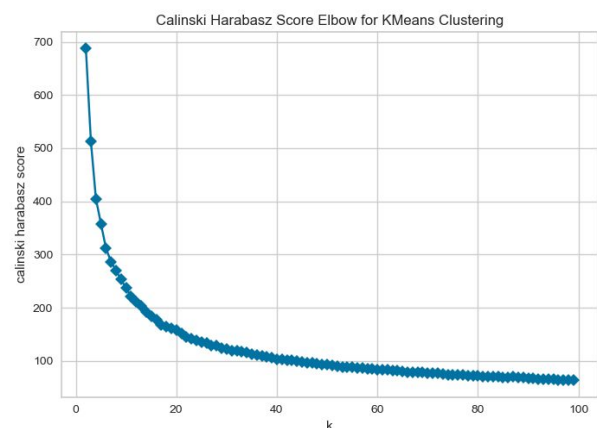
To validate the clusters, I used the homogeneity score, which measures the data in each cluster that are members of the same class. The completeness score, which measures if data of the same class are in the same cluster. Finally the v measure score, which is the harmonic mean of the homogeneity and completeness scores; a perfect score of 1.0 here means the clusters are homogenous and complete, which translates to all like labels are contained by a single cluster and all clusters contain data of a single label. For validating (k) testing X (features) and Y (labels) sets were used.

#### Wine:



**V\_measure\_score:** 0.832908  
**Homogeneity\_score:** 0.836235  
**Completeness\_score:** 0.829608

#### Wage:



**V\_measure\_score:** 0.183151  
**Homogeneity\_score:** 0.183015  
**Completeness\_score:** 0.183287

The best value for k was found to be 3 which makes sense, because for these data there are 3 distinct labels. Un-ironically this algorithm was able to discern the same amount of clusters as there are labels which is a good sign before moving onto the validation step.

I would consider these scores high because they indicate that most of the data were clustered appropriately based on their labels. This is most likely because all of the features in this dataset are continuous, and a few features may not have a strong influence on the labels.

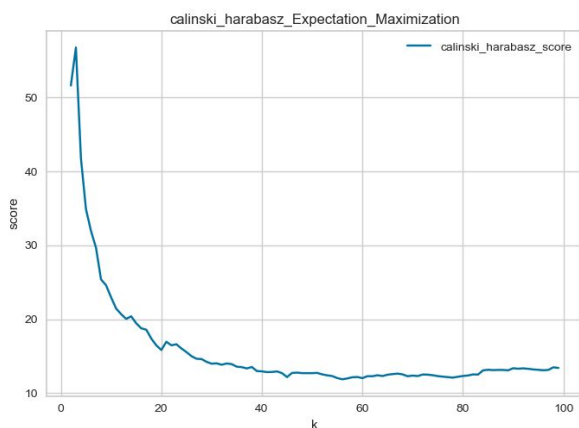
The best value for k here was found to be 2, which like the wine data is the same amount of distinct labels there are for this dataset.

These are low scores, which indicate a lot of the data were clustered inappropriately. Meaning clusters contained data of varying labels and data of a certain label were varied among the clusters. This may be due to the fact that there are many features in this data set and a lot of them are one hot encoded categorical features which adds a lot of noise. Meaning that a lot of features may not have a strong influence on the labels.

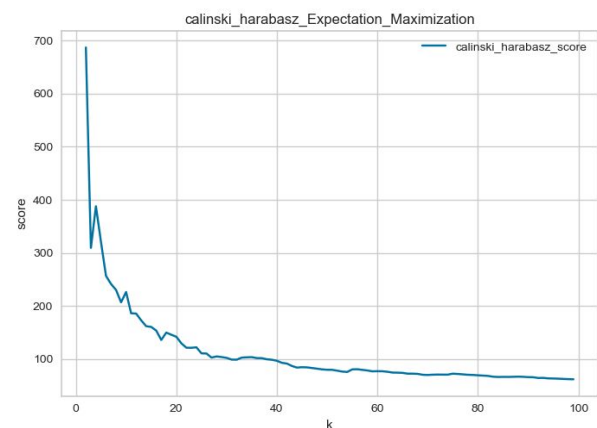
## Expectation Maximization

This is a soft clustering algorithm that calculates the probabilities that data points belong to certain clusters, then re-calculates the mean of the clusters based on the probabilities of the data points belonging to those clusters. The algorithm completes when the probabilities of data points belonging to clusters stops increasing. I used sklearn's Gaussian Mixture, which implements expectation maximization to perform these experiments.

I used the same method to choose (k) as with KMeans which is the Calinski Harabasz Score. During selection of (k), similar to KMeans I also used only a training X set. I also used the same validation metrics of homogeneity, completeness, and v measure scores with testing X and testing Y sets. The same training and testing sets were used here as in KMeans.



**V\_measure\_score:** 1.0  
**Homogeneity\_score:** 1.0  
**Completeness\_score:** 1.0



**V\_measure\_score:** 0.182940  
**Homogeneity\_score:** 0.182833  
**Completeness\_score:** 0.183047

Similar to KMeans, the best value of k was found to be 3. Again this makes sense because there are 3 distinct labels in this dataset.

Expectation Maximization was able to perfectly discern the clusters for the test set after being fit with the training set. This means all clusters contained data of the same labels and all data of the same labels were contained by distinct clusters. This scores better than KMeans probably due to the fact that data points

Similar to KMeans, the best value of k was found to be 2. Again this makes sense because there are 2 distinct labels in this dataset.

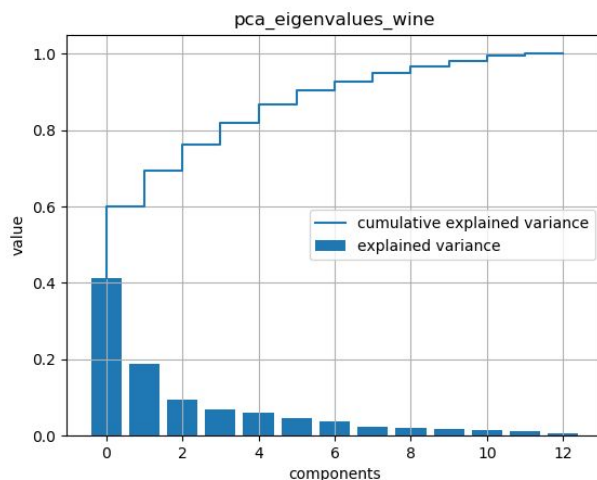
The scores here are very close to the scores for KMeans however slightly (nearly insignificantly) lower. Again this may be due to the fact that there is a high amount of noise in these data from the one hot encoded categorical features that may not provide useful information to clustering and labeling.

are not forced into clusters, but they have likelihoods of belonging to clusters which may help cluster hard to discern data points.

## Dimensionality Reduction

### PCA

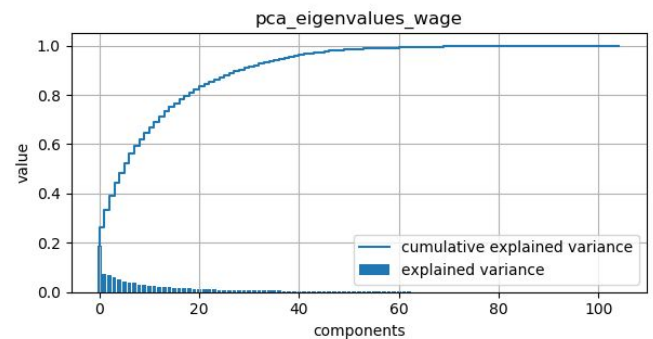
Principal component analysis is an eigenproblem that is similar to filtering, as it finds new orthogonal axes to represent the data in a way that maximizes variance. Here I am hoping to maximize variance of the new axes and reduce the amount of features by dropping components that contribute little to no variance, in order to remove un-useful information and possibly reduce noise which may increase the performance of my neural network. It can improve performance by reducing the amount of iterations from the reduction in features and increasing accuracy from the reduction in noise. I am getting the variance of each component from the `_explained_variance` attribute in sklearn's PCA which holds the eigenvalues of the model, then I divide by the sum of variance to get a percentage of variance contributed. I choose (k) when increase in variance with more components is negligible.



#### Dataset: Wine

With the wine dataset it is hard to say what the threshold for variance should be when selecting the number of components (k). This dataset has very few features to begin with at 13, and the eigenvalues seem to show that that all components contribute at least some variance except for the last component which has a value of nearly 0.

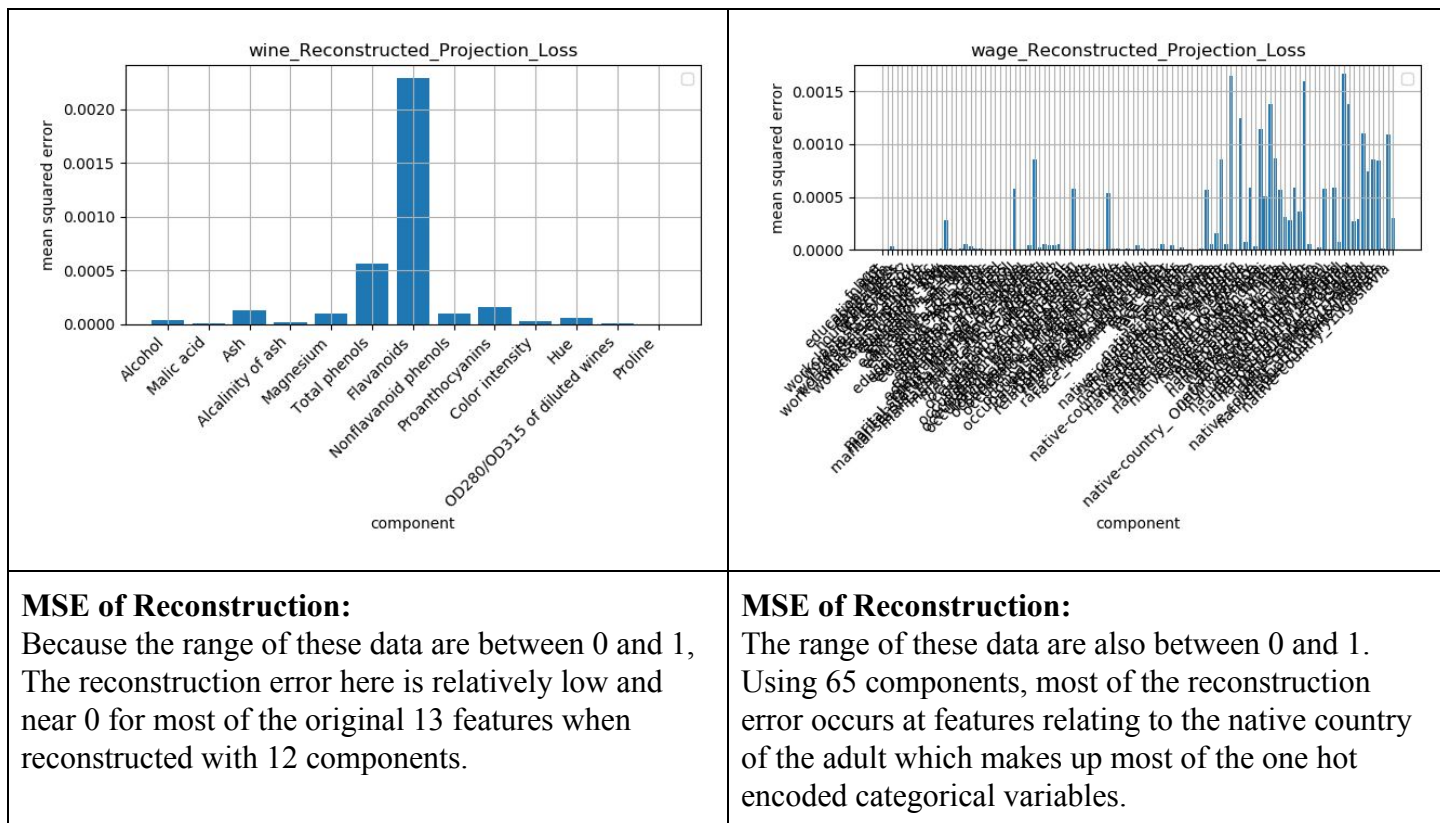
Based off of these eigenvalues I will be using 12 components, 1 less than the original feature count. This is because the last component here contributes near 0 variance to the data.



#### Dataset: Wage

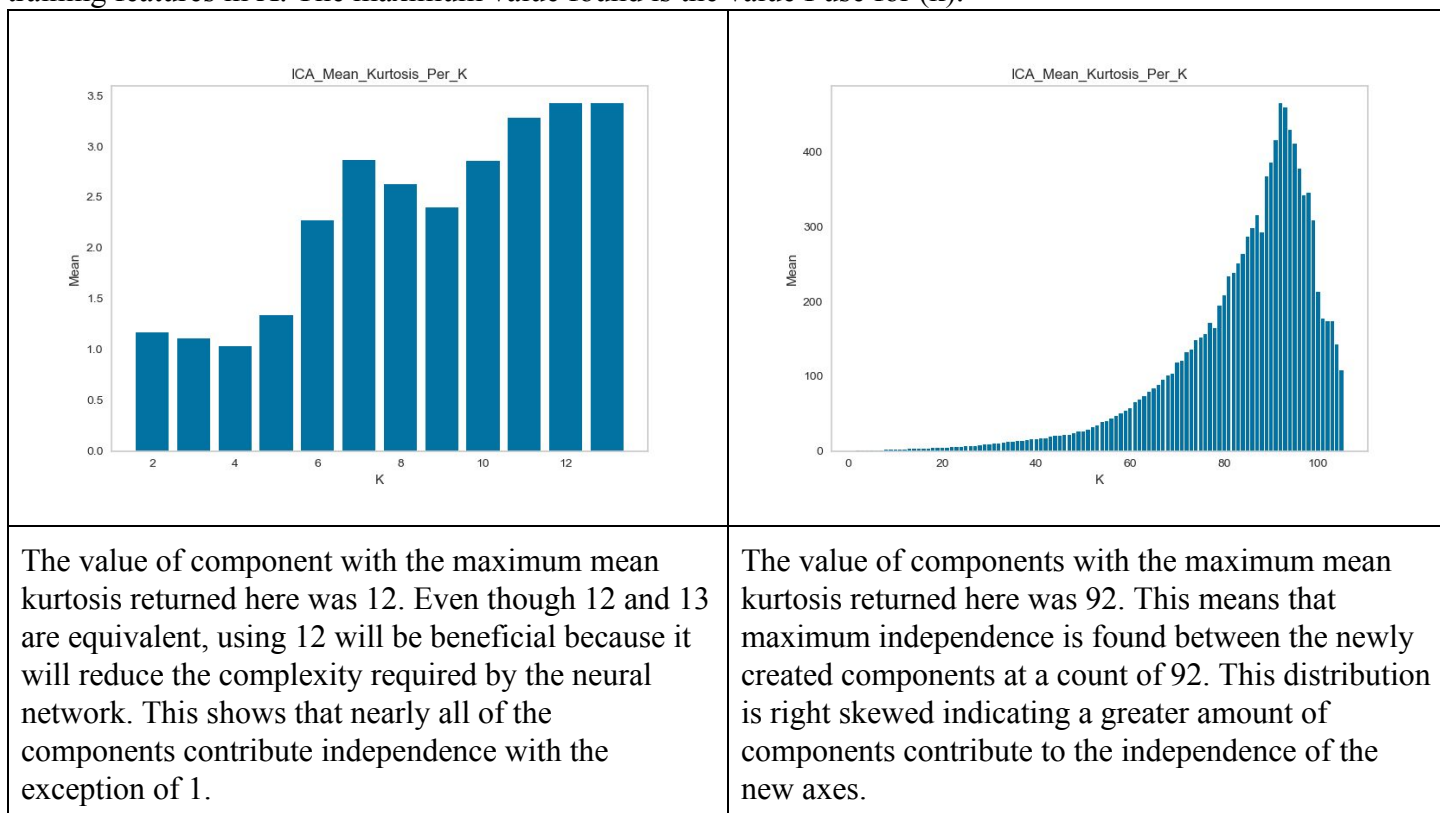
The eigenvalues here show that there is one component that explains significantly more variance than the rest. Also negligible variance gained after about 65 components. This makes sense because many of the features in this dataset are one hot encoded categorical values, specifically the feature with the most categories is the country. This seems to show that there are some categorical feature values that have an impact on the variance in the data while most of the others do not.

Because change in variance is negligible after 65, I will choose the number of components (k) to be 65. This same argument could also be made for 40, however I want to be careful not to lose useful information when possible because this would increase my reconstruction error.



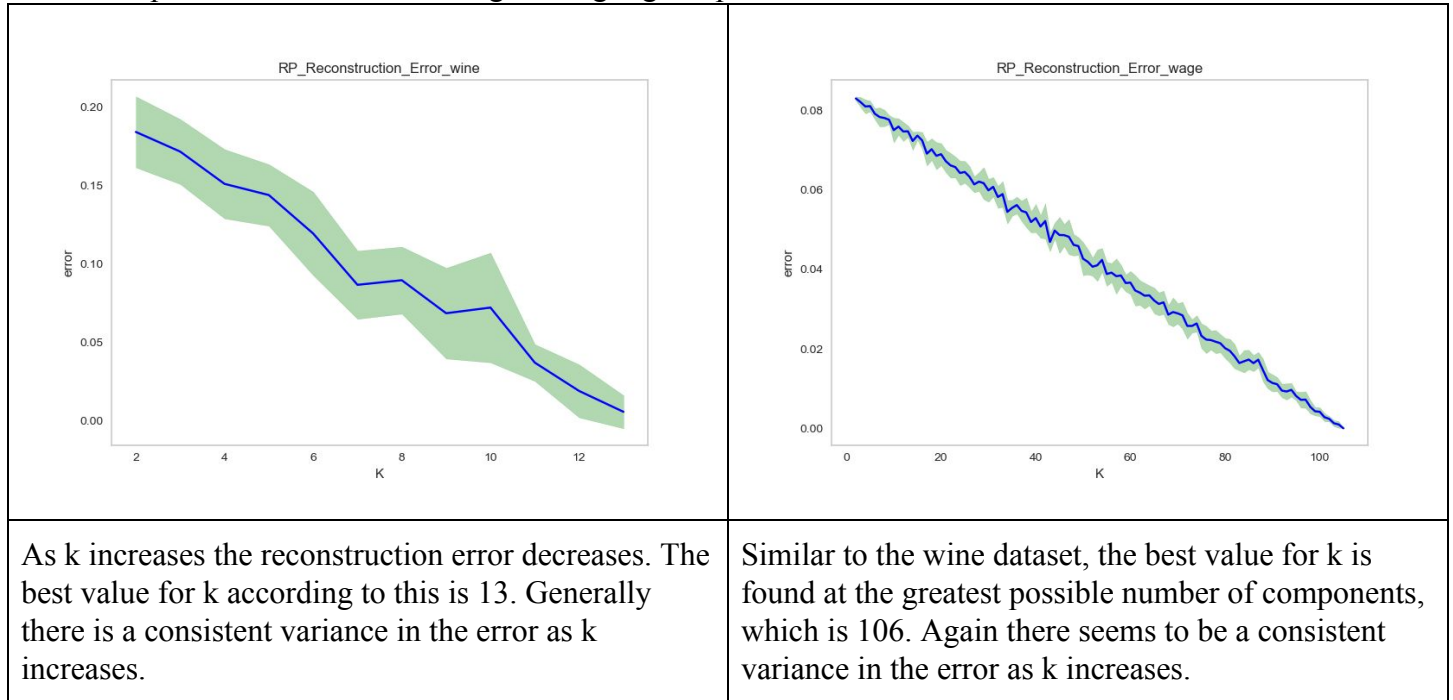
## ICA

Independent Component Analysis attempts to find linear projects, that unlike PCA are not necessarily orthogonal, with the goal of statistical independence as opposed to higher variance. The goal here is to maximize non-gaussianity which will maximize component independence. To do this, I run ICA over a training X set for each n\_component value of (k), then I use pandas to take the mean of the kurtosis of the transformed training features in X. The maximum value found is the value I use for (k).



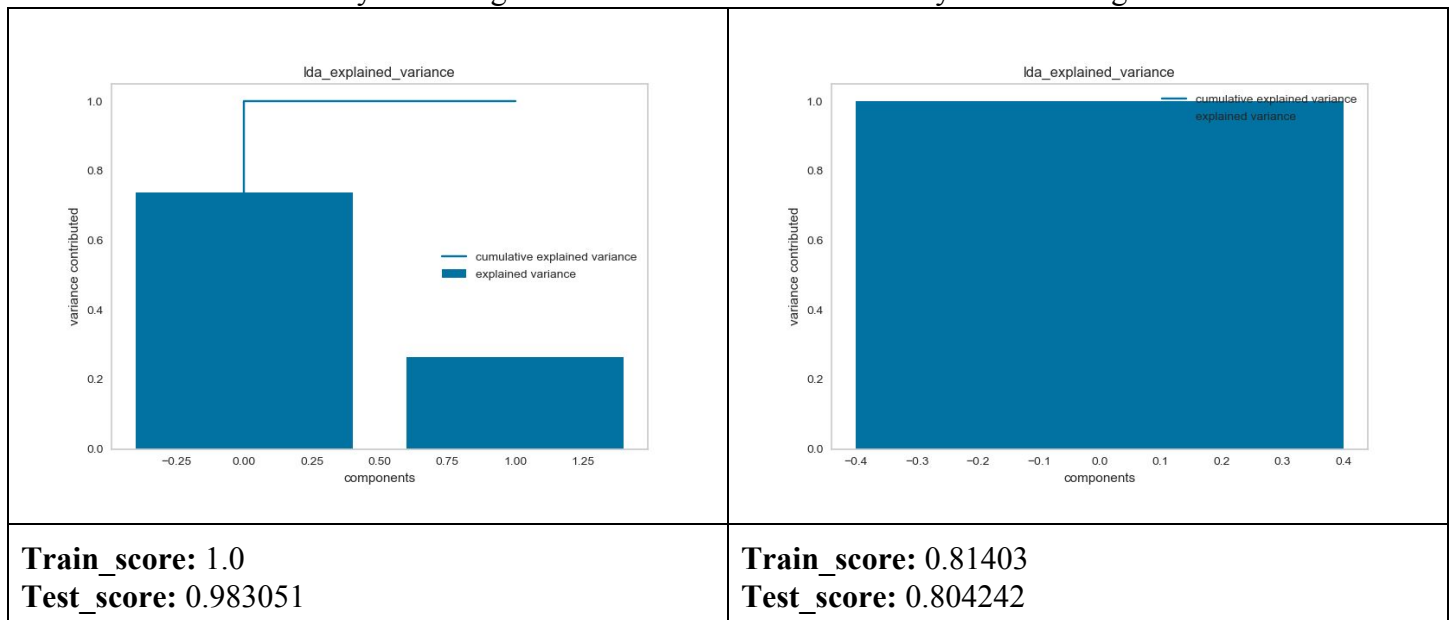
## RP

Random Projection generates random axes directions and projects the data onto them. I use the lowest reconstruction error to select the best value of (k). To get the reconstruction error, I used the projection matrix generated by fitting the model to reconstruct the training data, then I found the mean of the squared differences of the training data and reconstructed data. For these plots I ran the projection 10 times for each value of k, the blue line represents the mean and the green highlight represents the standard deviation from the mean.



## LDA

Linear Discriminant Analysis is a supervised version of dimensionality reduction that finds discriminating lines based on the labels of the data. This means LDA creates components based on one less than the number of distinct labels there are for the data points. Therefore this algorithm drastically reduces the number of components, however this is understandable because of the supervised nature of the algorithm. This algorithm has the advantage of know how many unique classes there are and their features. This is also similar to the dimensions created by clustering if it were used as a dimensionality reduction algorithm.



In this dataset there are 3 distinct labels so 2 discriminating lines are created. This method was able to achieve a very high score on the testing data by how it split the data. Reducing the components to 1 wouldn't make sense here because the lower variance component contributes about 20%.	Because there are only 2 labels for this dataset, a single discriminating line was created which accounts for all of the variance in the data. With this discriminator the train and test data were classified properly about 80% of the time, which is actually surprisingly high. Although this can be explained by
--	---

### Clustering & Dimensionality Reduction Experiments

Here I am rerunning the KMeans and Expectations Maximization clustering algorithms using the data from the dimensionality reduction algorithms with the selected number of dimensions for each algorithm for each dataset. The table below shows the v\_measure, homogeneity, and completeness scores, which indicate how well the algorithms were able to cluster the transformed data.

**V = v\_measure\_score | H = homogeneity\_score | C = completeness\_score**

		PCA	ICA	RP	LDA
wine	KMeans	V: 0.832908 H: 0.836235 C: 0.829608	V: 0.686110 H: 0.662513 C: 0.711449	V: 0.699347 H: 0.700606 C: 0.698092	V: 0.935949 H: 0.936952 C: 0.934949
	EM	V: 0.935949 H: 0.936952 C: 0.934949	V: 0.497143 H: 0.499634 C: 0.494677	V: 0.492286 H: 0.480965 C: 0.504154	V: 1.0 H: 1.0 C: 1.0
wage	KMeans	V: 0.183151 H: 0.183015 C: 0.183287	V: 0.002190 H: 0.001227 C: 0.010178	V: 0.175159 H: 0.174820 C: 0.175500	V: 0.293224 H: 0.289250 C: 0.297308
	EM	V: 0.182940 H: 0.182833 C: 0.183047	V: 0.001422 H: 0.001243 C: 0.001661	V: 0.164921 H: 0.164008 C: 0.165843	V: 0.295227 H: 0.287089 C: 0.303839

Also the Calinski Harabasz Score yielded different results for ICA to KMeans and ICA to Expectation Maximization on the wage dataset as well as LDA to KMeans and LDA to Expectation Maximization on the wage and wine datasets. Below I look into the different cluster values found for the dataset after running ICA.

Using the newly found cluster count for KMeans and Expectations Maximization on the wage dataset yielded a vastly better homogeneity score (clusters containing data of a single class). This can be explained by the fact that the number of clusters was greatly increased, which means for the low number of unique labels in this dataset (2) there are bound to be clusters that contain data points of only one of those labels. Because the completeness is still very low and did not increase very much I can say that increasing the clusters in this case did not have a positive effect on segmenting the data and is most likely a result of the spread of the data created by ICA. Therefore it is beneficial to keep the original cluster counts found at the beginning of the report.

For RP, the clustering was generally worse than the un-transformed data, however the scores varied for each run. The benefit of using random projection is that it is faster, while sacrificing some accuracy. In this case the data are small enough that this is not the most useful dimensionality reduction technique.

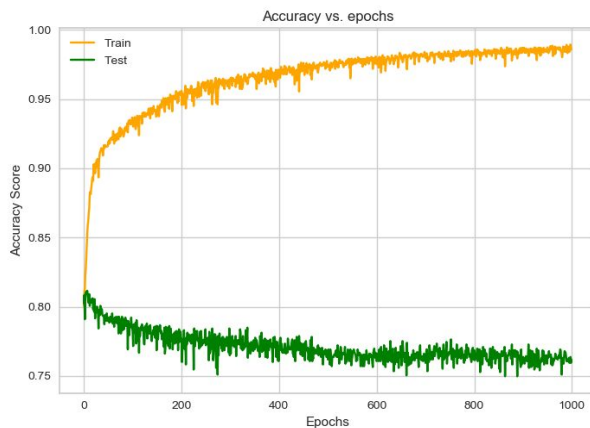
PCA and LDA transformed data had the best results in terms of homogeneity and completeness after being run through the clustering algorithms on the wine and wage datasets with the optimal number of

components and clusters. For both dimensionality reduction algorithms on both datasets the transformed data was clustered the same if not better than the clustering on the original data.

For PCA this is most likely the case because the new components captured enough variance to discern between the small amount of unique classes/labels in each dataset. For LDA, this is understandable because each linear discriminant used essentially creates two infinitely large clusters on each side of the discriminant, which is naturally easier to cluster when data are projected onto the new axes.

## Dimensionality Reduction & Neural Network on Wage Data

### Original

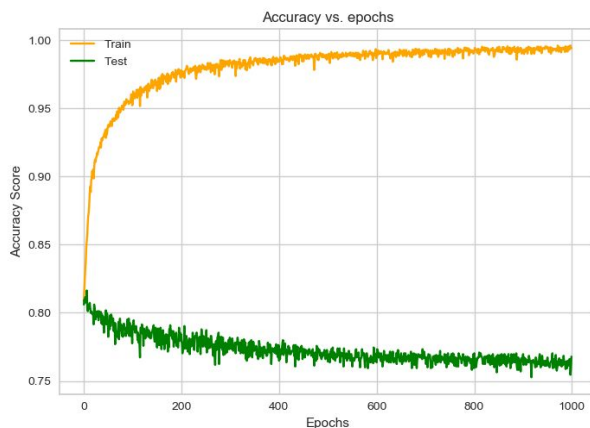


**Test acc: 0.7806060606060606**  
**Train acc: 0.9265671641791045**  
**Train time (s): 41.22605347633362**

The baseline network shows overfitting as the number of training epochs increases.

This is the baseline performance of the original network with the original data. The only difference from the first assignment is that the features are min max normalized.

### PCA



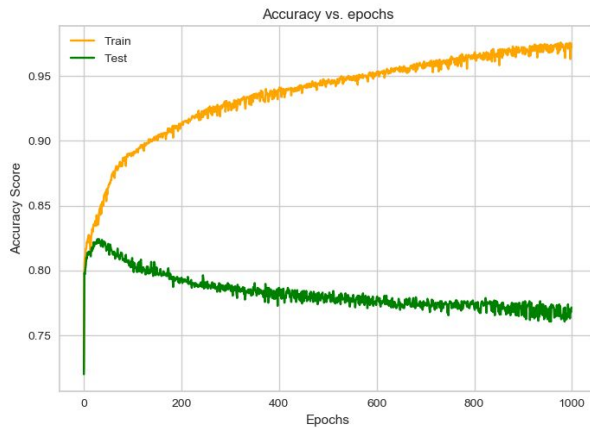
**Test acc: 0.7836363636363637**  
**Train acc: 0.9567164179104478**  
**Train time (s): 30.097635746002197**

The training and testing curves over training epochs are very similar to that of the original data.

Although the training and test accuracy were nearly the same as the original data, there was about a 25% decrease in training of the neural network from the reduction in dimensions.



## ICA

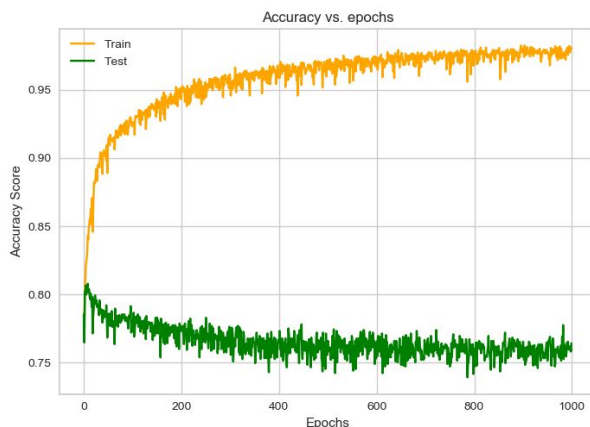


**Test acc: 0.8036363636363636**  
**Train acc: 0.8955223880597015**  
**Train time (s): 31.651535034179688**

Again the training and testing curves are very similar to PCA and the original data, however there is less variance here than the previous methods as well as a clear optimal epoch for training.

ICA dimension reduction was able to achieve a slightly higher accuracy than the previous methods, at a slightly higher time cost which is still about 25% faster than using the original data.

## RP



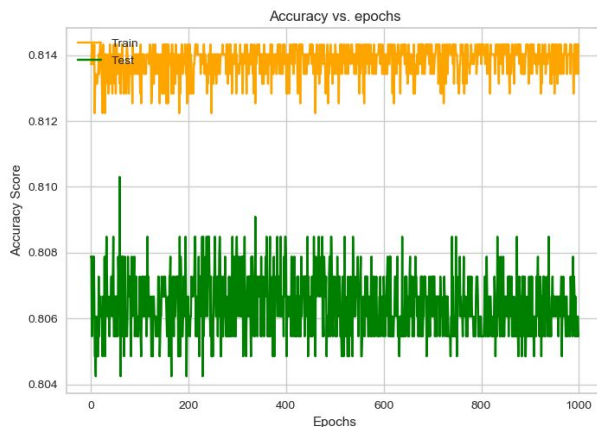
**Test acc: 0.7866666666666666**  
**Train acc: 0.9214925373134328**  
**Train time (s): 34.1926326751709**

This is very similar to the PCA and Original data curves however there is more variance in the accuracy as the epochs increase.

Although random projection can be fit much faster on the dataset, it did not help as much as PCA and ICA in terms of training time because of the number of components needed to avoid useful information loss.



## LDA



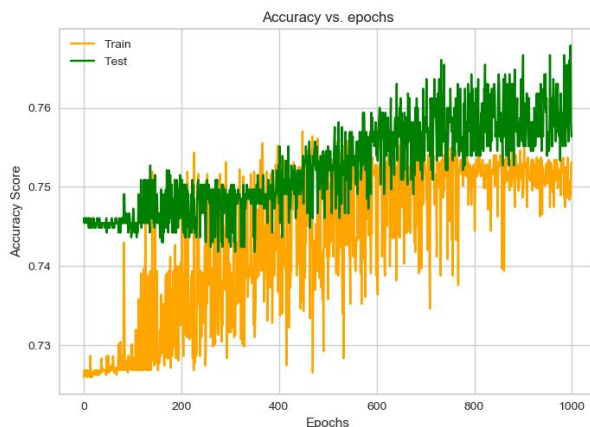
**Test acc: 0.8066666666666666**  
**Train acc: 0.813134328358209**  
**Train time (s): 6.023937940597534**

The training and test accuracy were consistently the same through training epochs on the transformed data. This is because as the data points are transformed they contain a single component that decides their class due to their being only 2 labels as opposed to the 50+ components that the other methods have.

Because of the very low number of components produced by LDA which was influenced by the low number of classes in this dataset (2). The training time was extremely low. And there ended up being high accuracy, this is probably because the data are linearly separable as I found out in the first assignment.

## Clustering & Neural Network on Wage Data

### KMeans

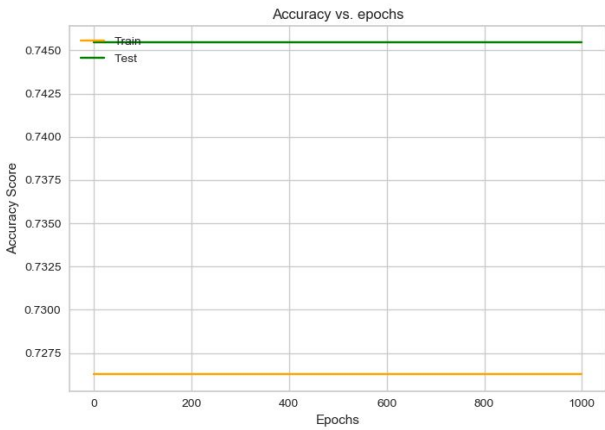


**Test acc: 0.7454545454545455**  
**Train acc: 0.7265671641791045**  
**Train time (s): 6.7745935916900635**

Using the clusters from KMeans as features for the neural network, was not as effective as the dimensionality reduction algorithms. However there is lower variance in the accuracy of the neural network trained on these features.

Similar to LDA, KMeans has very few clusters so the training time is very low. It is over 5x faster than training the neural network with the original data with the trade off of about 5-8% accuracy.

## Expectation Maximization

 <p>The graph shows two horizontal lines representing accuracy over 1000 epochs. The 'Train' line (orange) is at approximately 0.726, and the 'Test' line (green) is at approximately 0.745. The y-axis is labeled 'Accuracy Score' and ranges from 0.7275 to 0.7450. The x-axis is labeled 'Epochs' and ranges from 0 to 1000.</p>	<p><b>Test acc: 0.7454545454545455</b> <b>Train acc: 0.7262686567164179</b> <b>Train time (s): 2.600907325744629</b></p>
<p>The neural network trained on EM transformed data shows no change in accuracy for test or train as epochs increase. I transformed the EM data by using the posterior probability of each datapoint belonging to a cluster as the features.</p>	<p>EM had the same test accuracy and nearly the same train accuracy as KMeans, however the neural network trained much faster with the EM transformed data.</p>

## General Conclusions

KMeans found slightly better clusters for the larger wage dataset, while Expectations Maximization found better clusters for the smaller wine dataset. Both of the clustering algorithms generally found clusters that aligned with the number of unique classes in the Y label vector. PCA reduced to fewer components and was clustered by both clustering algorithms better than ICA for these data. RP's performance varied however it generally was not able to reduce the number of components without losing useful information, that being said, it was clustered better than ICA but not as well as PCA. LDA is a different beast because it requires labels to be trained and is therefore a supervised version of dimensionality reduction, it created the fewest components because the number of components here were based on the amount of unique classes in the labels which were separated by linear discriminants. The neural network performed the best with LDA reduced data because most of the heavy lifting of the classification was done by LDA, which drastically decreased the neural networks training time. LDA is very similar to clustering because the discriminated regions could be thought of as infinitely large clusters.

## Sources for Models and Analysis

KMEANS: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>  
EM: <https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>  
LDA: [https://scikit-learn.org/stable/modules/generated/sklearn.discriminant\\_analysis.LinearDiscriminantAnalysis.html](https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html)  
RP: [https://scikit-learn.org/stable/modules/generated/sklearn.random\\_projection.SparseRandomProjection.html#sklearn.random\\_projection.SparseRandomProjection](https://scikit-learn.org/stable/modules/generated/sklearn.random_projection.SparseRandomProjection.html#sklearn.random_projection.SparseRandomProjection)  
ICA: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html>  
PCA: <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>  
NN: [https://scikit-learn.org/stable/modules/generated/sklearn.neural\\_network.MLPClassifier.html](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)  
Homogeneity: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity\\_score.html](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html)  
Completeness: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness\\_score.html#sklearn.metrics.completeness\\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html#sklearn.metrics.completeness_score)  
V Measure: [https://scikit-learn.org/stable/modules/generated/sklearn.metrics.v\\_measure\\_score.html#sklearn.metrics.v\\_measure\\_score](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html#sklearn.metrics.v_measure_score)  
Reconstruction error: <https://stackoverflow.com/questions/36566844/pca-projection-and-reconstruction-in-scikit-learn>  
Choose K: <https://www.scikit-yb.org/en/latest/api/cluster/elbow.html>  
Calinski Harabaz: [https://scikit-learn.org/0.19/modules/generated/sklearn.metrics.calinski\\_harabaz\\_score.html](https://scikit-learn.org/0.19/modules/generated/sklearn.metrics.calinski_harabaz_score.html)