Joshua Philipose

Dr. Charles Isbell

CS 4641

3/24/19

Unsupervised Learning and Dimensionality Reduction Analysis

Classification Problems

Credit Card Defaults (abbreviated as CC)

The credit card defaults binary classification problem attempts to determine if a credit card client will default on their next credit card payment. The dataset consists of 23 attributes, ranging from credit, gender, age, several previous bill statements, amounts of previous payments, and other relevant data points. With 30,000 entries (6,636 defaulting, 23,364 not defaulting), this dataset requires attention to efficiency and runtime. Predicting credit card defaults was an interesting problem to me as I spent the previous summer working as a technical product manager for Citibank's credit card rewards program, and I am interested in working on credit cards in the future. There was also the imbalance of classifications one can expect in real world problems as the dataset had almost triple the number of non-defaults as defaults, so lessons learned here would apply to future problems I encounter. Furthermore, this classification problem has direct applications in industry, as financial institutions need to manage the collective risk of their clients and allocate monetary reserves accordingly.

King-Rook vs. King-Pawn (abbreviated as KK)

The king-rook vs. king-pawn binary classification problem attempts to determine if white, who has both a king and a rook, can win the game against black, who has a king and a pawn. However, the twist that makes this problem interesting is that black's pawn is always on a7 – one move away from promoting to a queen. As the problem's premise is that it's white's turn to move, the problem essentially boils down to the ability of white in preventing black's pawn from promoting while maneuvering a checkmate or draw. As an avid chess player, I find this problem fascinating because of its confinement to a complex rules-based environment. Furthermore, I've actually been in this position in game before, and I'm curious as to the win prediction for setups different to the one I had. There are 3196 instances, almost evenly split between 1669 instances with a white win and 1527 instances with a black win. For each of these instances, there are 36 attributes describing the board and piece positions. Because of this dataset's use of true/false values and a column of three categorical values, I had to use one-hot encoding to transform the dataset into numerical format to run on algorithms provided by scikit learn.

Methods

The two implemented clustering algorithms (K-Means and Expectation Maximization) was run multiple times, with different hyperparameters, sample sizes, and data conditions. I chose to vary cluster sizes because I had an intuition of how they worked and thought I could therefore derive the most insight by seeing them in action. I played around with number of initializations but didn't obtain any insights besides the obvious increase in "correct" clustering up to a plateau as number of initializations increased. This analysis made heavy use of homogeneity, correctness, and accuracy. We've covered accuracy in some detail, but homogeneity and correctness were new concepts to me, and therefore will be covered in detail below. All algorithms were run in a Jupyter notebook, implemented by scikit learn, and run on data from UCI's machine learning repository. In many experiments, number of clusters was varied. The optimal cluster number was then picked by selecting either the "elbow" of the score vs cluster number curve or the point where the marginal return was insignificant.

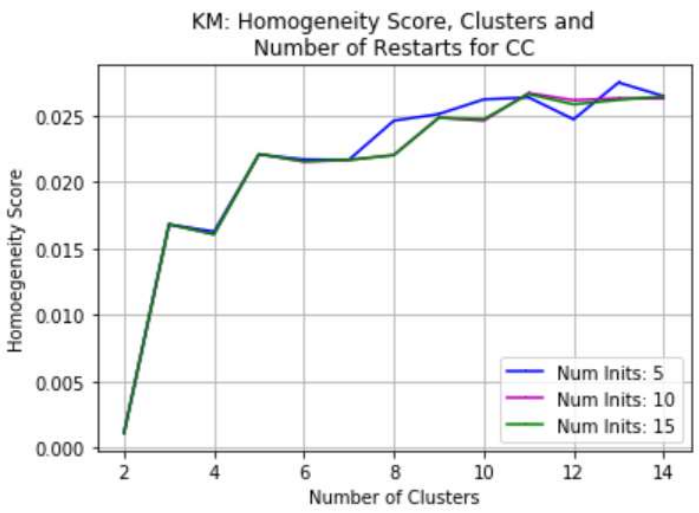
Homogeneity and Completeness:

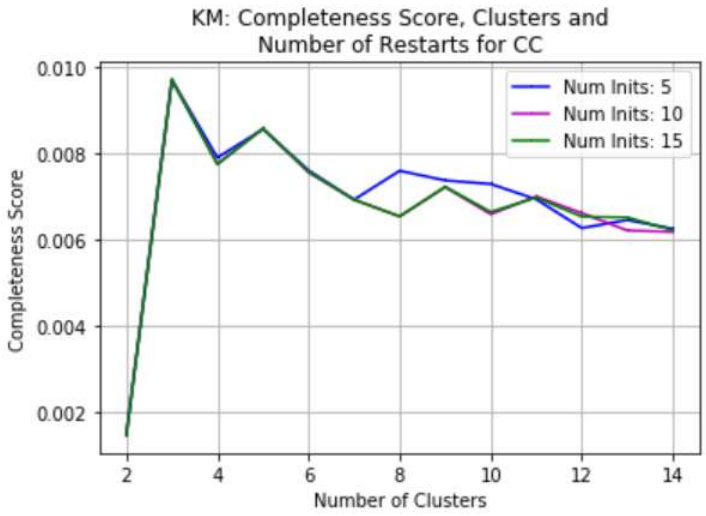
Homogeneity is the attribute of having identically, correctly labeled data points within each cluster, and completeness is the attribute of having all of a labeled class of points within a single cluster. For example, having four clusters, 2 of which are all "apple" and two of which are all "orange" would have a perfect homogeneity score, but a lackluster completeness score, because there are two "apple" clusters instead of just one.

K-Means Clustering:

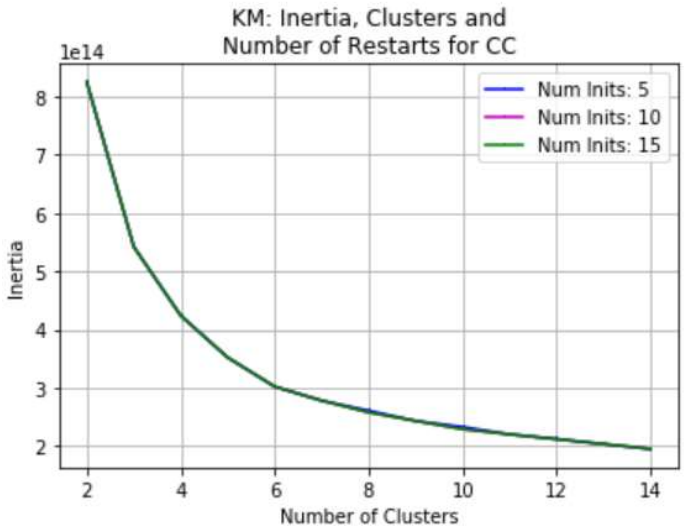
As the name suggests, K-Means clustering is a clustering algorithm that relies on mathematical means (averages). The algorithm attempts to place K center points, often called centroids, in the "center of mass" of an assumed cluster. Every data point belongs to exactly one cluster – the cluster it is closest to. As the centroids reposition themselves by moving to the average value of the data points in its cluster, we eventually gain a good approximation of the cluster, or grouping. This process repeats until convergence, or until a maximum number of iterations is reached. K-Means is therefore similar to standard randomized optimization problems, and appropriately shares the weakness of often getting stuck in local optima. To circumvent this, we utilize either our domain knowledge in selecting the initial locations of centroids, or random restarts. Because we know the nature of random restarts and its effects on our accuracies, I did not include graphs of the phenomenon – we know that as random restarts increase, scores get higher, until the problem reaches a plateau high score by virtue of consistently finding global optima.

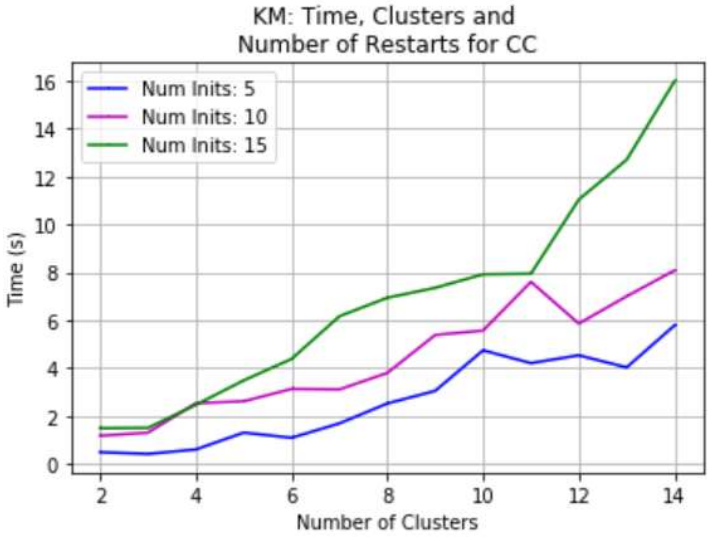
Let us then focus on the homogeneity and completeness scores for our K-Means clustering. Since completeness is inherently linked to the number of clusters, I thought that cluster count would be the logical first place to start experimenting. It's important to remember that the default number of clusters as implemented by scikit learn is eight. As the amount of clusters increases, so too should the homogeneity score, because clumps of data points in odd niches and crevices of the data space can be serviced by a dedicated cluster, rather than shoehorned into a cluster where it may not fit. For completeness, I predicted that as cluster count increased, completeness would decrease, because if we're forced to use, say 10 clusters when 5 would've done the job, we've needlessly increased our cluster count and therefore have decreased our completeness. The graphs below corroborate our predictions – in general, increasing cluster count increases homogeneity, while a downward trend is seen in completeness after the initial spike. This initial spike is not contradictory to our prediction, because of course we need enough clusters to label the data before the different regions can be split by clusters competing for the same label.



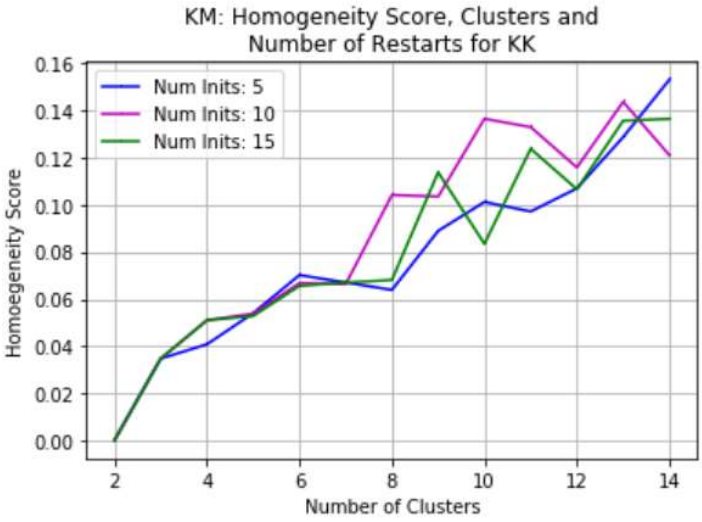


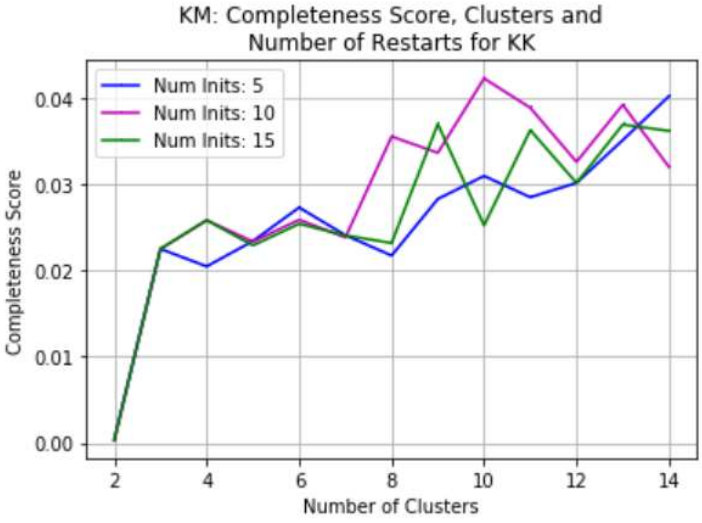
It's quite interesting to note that the number of initializations did not make any significant impact on performance, other than increasing runtime, as depicted below. This gives us an insight into our dataset that was clear in projects 1 and 2: our search space has very few local optima and is generally friendly to randomized optimization algorithms. As the number of initializations is clearly multiplying our runtime without adding any performance boost in terms of homogeneity or completeness, I chose to omit it in future experiments for the sake of time.

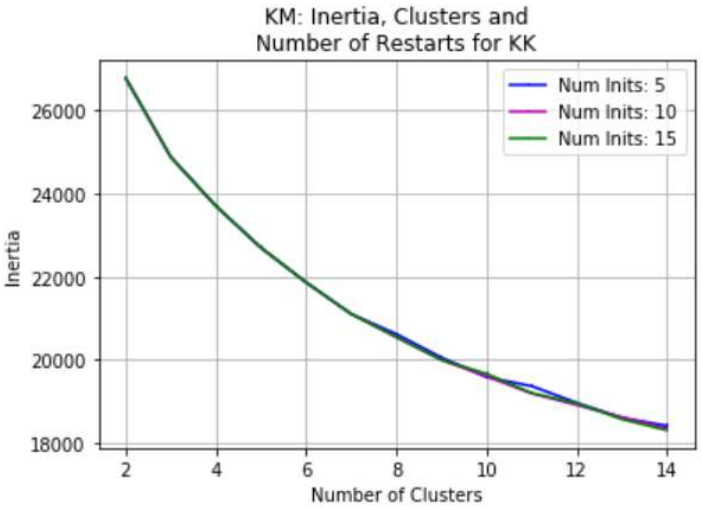


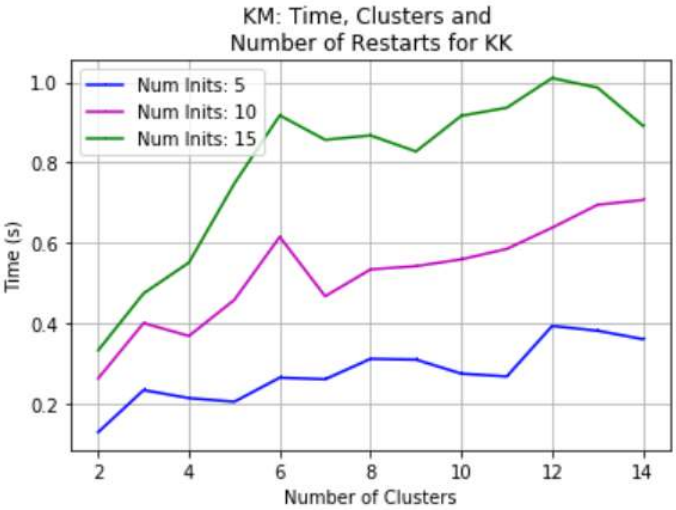


The inertia chart above shows us an interesting, albeit predictable, trend. Inertia is the sum of squared distances from each data point to the center of its cluster. As the number of clusters increases, there will be less data points in each cluster. A fewer amount of distances to sum over in combination with closer proximity to the center as clusters get smaller both contribute to the graph above. The chart shows a exponential decline in inertia because inertia is calculated exponentially, as the sum of squares.







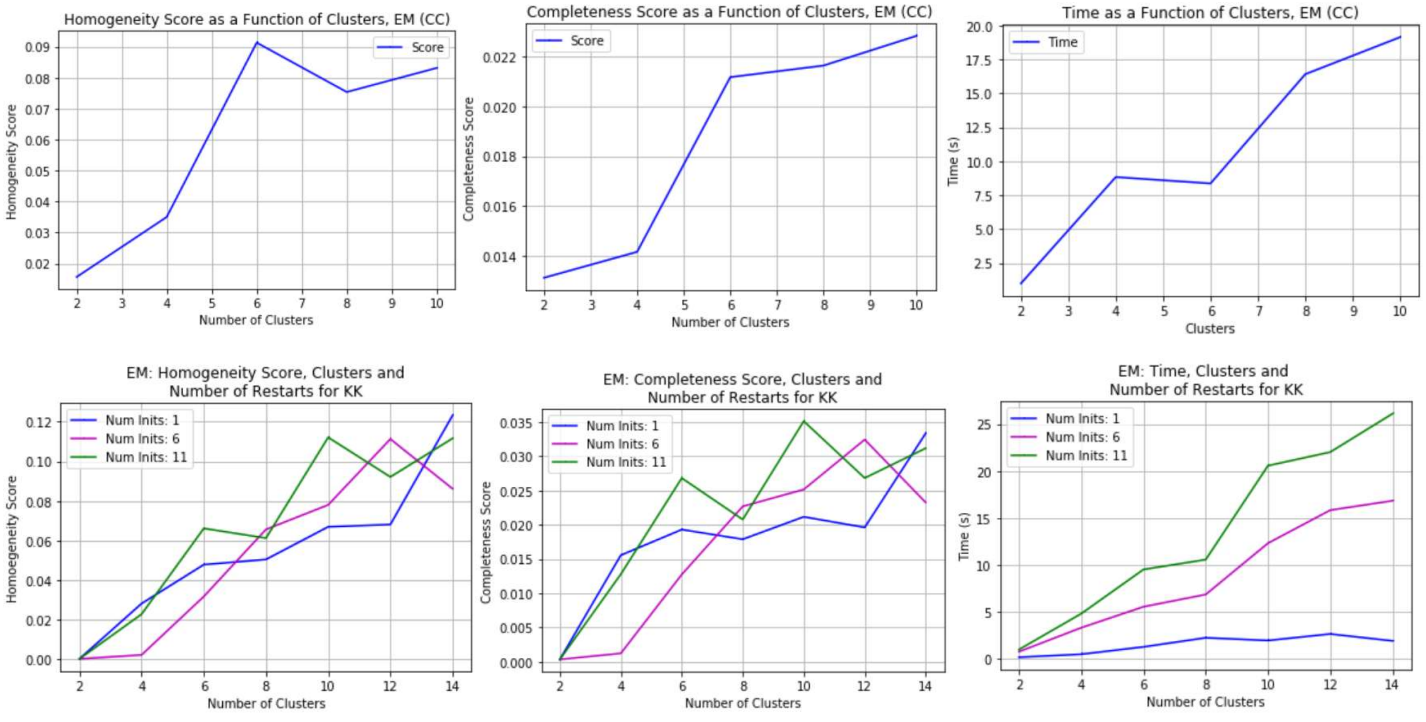


I've included the charts for the chess dataset on the previous page as well to highlight the fact that while they are similar, there is the one notable difference in that the completeness score maintains its upward trend. This is likely due to the fact that the chess dataset has many more features than the credit card dataset, and so there are more valid unique clusters to be formed. However, this observation may not be significant, as the Y axes for homogeneity and completeness give us a significant feeling of disappointment. A perfect score for both measures is a 1.0, but no chart has yet come close to even half of that. However, it is important to note that clustering is typically used for learning more about your data, not pure classification, and so lackluster metrics are not quite as important as actually learning about the nature of your dataset. From the insights we've reached above, we can definitely say that clustering has helped us understand the nature of both our search space and classification problem.

Expectation Maximization:

Expectation maximization is a clustering algorithm that focuses on finding a soft-edged Gaussian cluster. We start with the assumption that the data was generated from a pool of Gaussian distributions that have a uniform variance. We then assign K centers to the data and calculate the probability that the mean of that Gaussian data is the center we've assigned to it. We then adjust the center by weighting all points by the probability of that point being generated by that center and averaging accordingly repeating the process for a set number of iterations. It is important to note that in the end, each data point is not labeled with a cluster, but rather is labeled by the probability if it belonging to each cluster. This gives us some new, more precise information about our dataset. It is also important to note that though I mention Gaussian distributions throughout this analysis, expectation maximization can be configured to use any distribution.

Below, we see that homogeneity and completeness both increase as number of clusters increases. This means that expectation maximization is able to find clusters that both find novel facets of the data and pull in a good amount of the relevant data points to be included in the cluster. Since expectation maximization assumes that the underlying data has a Gaussian distribution in our case, and since the homogeneity score is about four times as high and the completeness score is double that of the same experiment done with K-Means, we can reasonable assume that the underlying data is normally distributed. This makes sense, because the credit card dataset is comprised of individual, independent recordings of the activities of human beings and their outcomes – our intuition would lead us to believe that this kind of environment would follow the central limit theorem. The results for the chess dataset mirror those of the credit card dataset, for similar reasons.



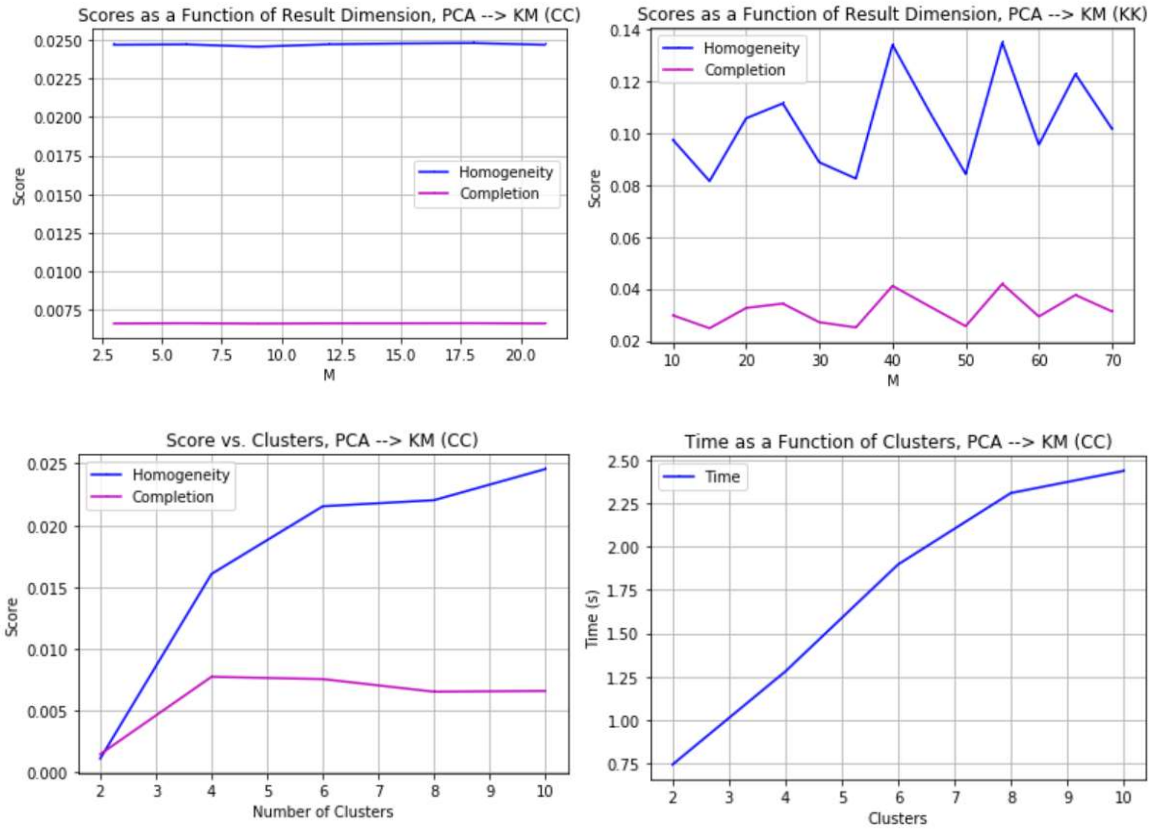
K-Means vs. Expectation Maximization (EM) Comparison:

As expected, EM took far longer than K-Means to run, as emphasized in the charts prior. This is to be expected, because EM has to calculate two probabilities and consider the distributions, whereas K-Means has a relatively simple averaging function. In addition, EM never fully converges and thus has a far longer timeout period, while K-Means can terminate once convergence is achieved. However, EM provides us with better results if we define better to be higher score, most likely due to the independently random nature of the human factors behind both datasets. Again, it has to be emphasized that regardless of relative performance, the absolute performance of both algorithms was abysmal in relation to their homogeneity and completeness scores. This is only passable because the purpose of the algorithms is to teach us more about the datasets, not classify the data itself. Later on in this analysis, we'll use the clusters as features to train a neural net on – hopefully we'll see an improvement in accuracy.

Principal Components Analysis (PCA):

Principle components analysis is a method of dimensionality reduction and data studying that focuses on the level of variance within certain features. The premise behind PCA is that features with highly varied values are expected to give our learners more information about the problem at hand, leading us towards selecting high variance features over low variance features. In practice, the data is projected into a new dimension, and the eigenvalues of the matrix needed to project our data are correlated to the amount of variance in each new feature. In this matter, we can disregard the lower eigenvalue features if we'd so desire. If we have any eigenvalues of 0, it means that the corresponding feature has only 1 value, meaning it gives us no information and is irrelevant, but not necessarily useless.

Because of our experience with prior experiments, I chose 6 clusters as that was previously where we had maxed out our marginal increases in performance. I was quite happy with our performance, as our scores as clusters increased were about double the pre-reduction scores. This boost in performance is explained by the dimensionality reduction, as fewer attributes means less potential noise of only relevant features are kept, as well as an increased focus on high importance features.



Our eigenvalues for the credit card dataset were:

```
PCA Eigenvalues:
[2.77282244e+10 1.34160226e+10 1.38651759e+09 7.68956999e+08
 4.27909012e+08 4.10708298e+08]
```

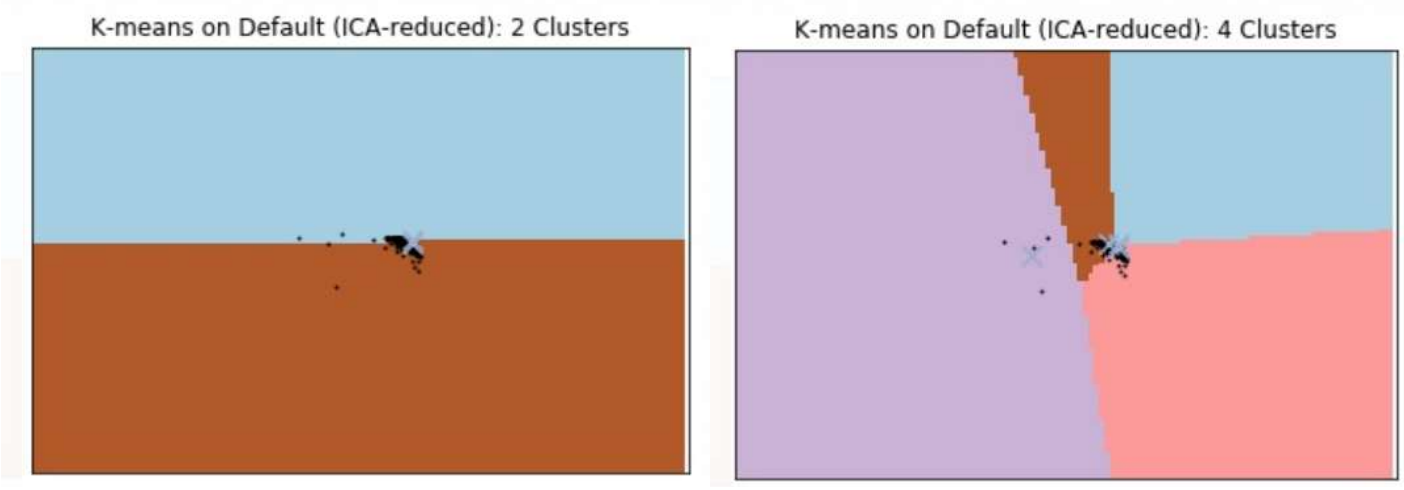
While our eigenvalues for the chess dataset were:

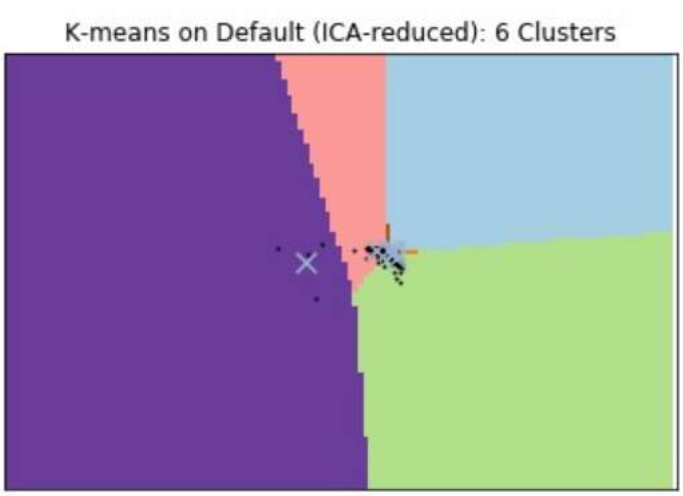
```
PCA Eigenvalues:
[1.32047176 1.07366463 0.904523   0.63580975 0.58447471 0.52234964
 0.44675315 0.40971454 0.35156671 0.32676782 0.27298311 0.24663858
 0.22353205 0.21890684 0.20806253 0.18259356 0.17713984 0.16649057
 0.16380928 0.1358448 ]
```

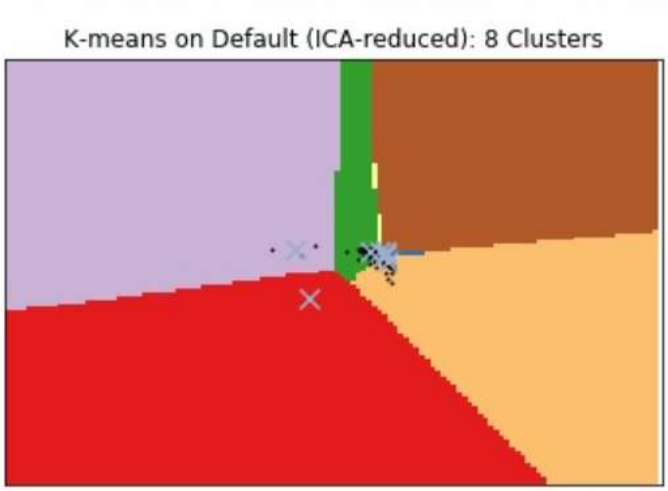
As you can see, the eigenvalues for our credit card dataset were much higher, indicating that there is far more variance in the credit card data than in the chess data. Again, this makes sense because of the realities of the environment: pure recordings of human behavior is going to be a lot more varied than optimal human behavior in a rules based environment like chess. Furthermore, the highest eigenvalue in the credit cards dataset is two orders of magnitude higher than the fifth highest, while in the chess dataset, the fifth eigenvalue is half the first eigenvalue. That's a 50% drop in variance in the chess dataset, compared to a 99% drop in the credit card dataset. This indicates that the first eigenvalue is much more varied (and therefore assumed more important in classification) in the credit card dataset than in the chess dataset. This increased rate of lower marginal returns actually perfectly explains why the optimal number of clusters for the credit card dataset is so much lower than the optimal number for the chess dataset – the point at which the higher variance features "overpower" the lower variance features comes much sooner. Lastly, we confirm an inference made in the first project: the variances are low in the chess dataset because both players play optimally.

Independent Components Analysis (ICA):

Independent components analysis, as its name suggests, attempts to maximize independent components within a dataset. This means that ICA looks to create a linear transformation that results in a set of features that are statistically independent (meaning no mutual information) by taking advantage of the central limit theorem but seeks to remain as consistent with the original data as possible. ICA is used to tease out the independent factors that lead to a combined signal. I chose not to include the graphs for homogeneity score and completeness score for both the credit card and chess datasets because of their remarkable similarity to the same graphs for PCA. After all, though the methods of reduction may vary, dimensionality reduction is overall a boon for classification algorithms, and therefore will boost performance until data loss outweighs the curse of dimensionality. One insight I found during these experiments is that the kurtosis varied way past the kurtosis of three expected of a normally distributed dataset. This indicates that there is an interdependence between the various features within the dataset, preventing a normal distribution.







Above you can see the results of visualizing the clustering of the ICA reduced dataset. Unfortunately, we can see that ICA did not always pick up on the correct features. In the two clusters visualization, we can see that K-Means has difficulty separating the tightly packed clump of data points. If ICA had worked better, that clump would've been more loosely spread around, and we would've seen more clear distinctions. This is because the underlying data is partially interdependent, and ICA works under the premise that the generators of the data are ultimately fully independent. If we think about the attributes found in the credit card dataset, this makes sense. Some attributes are things like age, income, and missed bank statements. Attributes like missed credit card statements in previous months end up snowballing as the amount of debt accumulates. Age comes with experience, which causes higher pay, which in turn can be used to pay off your card. Since some of the features are not fully statistically independent, ICA may not be performing to its fullest potential.

Random Components Analysis (Randomized Projection, RCA):

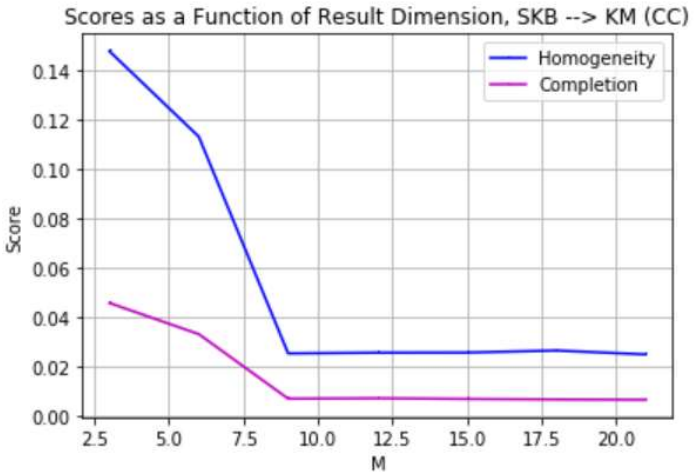
Random components analysis is essentially a random selection of features in an effort to lower the dimensionality of a classification problem. RCA is immensely useful when PCA or ICA are too computationally expensive, and though some runs of RCA may prove fruitless, you'll eventually chance upon a good enough reduction in dimensions to proceed with your intended methods of classification. Care must be taken to avoid becoming overzealous in stripping dimensions, as RCA does this at random, and we want to lower the probability of eliminating strong indicators by limiting the number of dimensions stripped using RCA.

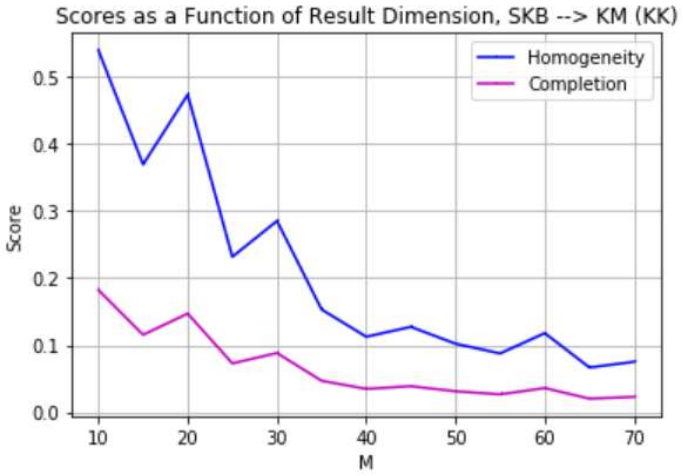
```
Accuracy:  0.7052333333333334  Time:  1.9488835334777832
Accuracy:  0.4785  Time:  0.9194731712341309
Accuracy:  0.7761  Time:  1.5651013851165771
Accuracy:  0.7707666666666667  Time:  1.9178998470306396
Accuracy:  0.7770666666666667  Time:  1.8879179954528809
Accuracy:  0.7673666666666666  Time:  3.6489062309265137
Accuracy:  0.7726  Time:  1.9488823413848877
Accuracy:  0.3857  Time:  2.7074644565582275
Accuracy:  0.7499333333333333  Time:  1.8509411811828613
Accuracy:  0.5208  Time:  1.3422257900238037
Accuracy:  0.7042333333333334  Time:  1.4101910591125488
Accuracy:  0.7596333333333334  Time:  0.7195873260498047
Accuracy:  0.7681333333333333  Time:  1.042405128479004
Accuracy:  0.7715  Time:  1.3152413368225098
Accuracy:  0.7633  Time:  2.156764268875122
Accuracy:  0.7738333333333334  Time:  1.255279302597046
Accuracy:  0.6579  Time:  1.6080653667449951
Accuracy:  0.5164  Time:  0.7305805683135986
Accuracy:  0.7615  Time:  0.8405177593231201
Accuracy:  0.5192666666666667  Time:  1.5161323547363281
0.014961769919444447
```

Above is the code output for 20 successive iterations of random component analysis dimensionality reduction. The "accuracy" of each run is the percentage of correct labeling a neural network was able to achieve after the reduced dataset was trained on it. The final number, .015, is the variance between all the accuracy scores. We can clearly see that though randomized component analysis occasionally has a wild swing or hiccup, it is typically quite consistent with a lower variance than we would expect from a completely random elimination of features. To provide some context, the baseline accuracy of the neural network on un-reduced data is .776, which is on par with most of the accuracy scores generated on randomly reduced data. This experiment proves the raw power of even random dimensionality reduction – we achieved a performance boost in time and complexity saved with no consistent loss in accuracy.

F-Value ANOVA (Select K Best, SBK):

Using the statistical power of F-Value ANOVA, I was able to find the k best features. In this case, "best" is defined as correlation to the true labeling of the instance. This method of feature selection is very similar to information gain, as it chooses features that will directly help with classification. It should therefore come as no surprise that this feature selection method performed the best when data was dimension-reduced using this and then fed into a neural network. Selecting the proper value of k is the most difficult part of using this feature reduction technique, but this is a matter of domain knowledge. It's advisable to try out a few values of k in an experiment if one is unsure of the best value of k to use. This is precisely what I've done below, and it was how I was able to determine that 3 was the best final dimension size for the credit card data set (CC) when filtering through it using ANOVA to select the k best features.





Applications to Neural Networks:

Let's pause a minute to reflect on what we've done so far. Throughout our experiments with clustering and dimensionality reduction, we've made it pretty clear that the main thing we're doing is learning more about our data, labeling it, or otherwise transforming it into a more useable state. Now that we've done all that, why don't we put the fruits of our labor to good use and see how our data modifications fare in a neural network? Below is a table of results after reducing credit card data with each of the four dimensionality reduction techniques and then using it to train a neural network.

	Unreduced Data	PCA	ICA	RCA	SKB
Accuracy	.776	.749	.803	.716	.821
Time (s)	3.14	1.43	12.22	1.38	2.66

As you can see, randomized projection performed the best on time, understandably, while our select k best algorithm in conjunction with ANOVA provided us with the best results. Randomized projection (RCA) should always provide us with the best time out of the options given because it involves no complicated computations, just a random selection of

features to utilize. On the other hand, it makes sense that ICA is quite computationally expensive and takes significantly longer to run, as solving for a linear transformation is much more extensive than performing regression calculations or randomly picking values. The probability work behind the scenes and the kurtosis calculations add up as well, emphasizing that while ICA is quite useful for identifying important features, it comes with the heavy computing requirement to balance it out. We can further see that both PCA and randomized projection, while twice as fast to train on than pre-reduction data, does come with a noticeable loss in performance as useful features are eliminated.

But let's take it a step further. We've used our clustering algorithms to label our data and gain some insight on the topography of our search space. Now that we've gone through the trouble of labeling our data, can our neural net classifier use those labels as attributes to increase our overall accuracy? In order to find out, I isolated the labels that K-Means created for each instance and used these labels alone and with the rest of the attributes to train a neural network. The results of this experiment are below, with the unmodified instances provided as a baseline.

	Cluster Labeling	Baseline	Baseline + Cluster Labeling
Accuracy	.7788	.7743	.7743
Time (s)	1.10	1.91	1.81

As we can see above, the cluster labeling alone actually marginally outperformed the entire dataset. We can therefore conclude that clustering is an immensely powerful data labeling tool – it does a majority of the legwork in finding relevant features and organizing the data so that it is easily digestible by classifiers like neural networks. In addition, the time taken to train on the clusters alone was significantly shorter than the time taken to train on the entire dataset, understandably, and so clustering can act as a pre-processing tool to cut down on training times for computationally expensive learners.

Conclusion

Overall, we got to explore the clustering and data exploration power of K-Means and Expectation Maximization (EM) quite thoroughly. I would like to touch upon the attributes of richness, scale invariance, and consistency as a way of wrapping up our clustering conversation. K-Means and EM are both scale-invariant, as scaling a system up or down wouldn't affect an averaging function nor the system's distribution. They're both also consistent, as they work in local zones around a center, unlike single linkage clustering. Shrinking intracluster distances and expanding intercluster distances would only strengthen the cluster for both K-Means and EM. By process of elimination, neither clustering algorithm can possess the attribute of richness, as proved by Kleinberg's impossibility theorem for clustering. Aside from this, we found that EM gives us a more accurate probabilistic representation of which cluster each data point belongs to, at the cost of increased computational power required.

We then shifted gears to focus on feature selection and feature transformation algorithms for the purpose of reducing the number of dimensions in our classification problem with minimal information loss, so as to alleviate the curse of dimensionality. The algorithms we looked at (principal component analysis (PCA), independent component analysis (ICA), random component analysis (RCA), and select k best with ANOVA (SBK)) were all examples of filtering algorithms, where all the data passed through a filter before ever reaching the target learner algorithm. This method is quicker than wrapper methods which required feedback from the learner, but of course has the tradeoff of not being as tuned to the bias of the learner, necessitating our domain knowledge to function optimally. In our experiments with these algorithms, the usefulness of ICA in painting a picture of our data stood out, as did the raw speed of RCA and the high accuracy performance of SBK. Putting clustering and dimensionality reduction together in our experiments with neural nets highlighted the combined strengths of both, as our neural network trained quickly and performed just as well. All in all, this was quite an illustrative assignment, and I'm looking forward to exploring these concepts further in the future.