**Author: Neil Acharya**

# CS 4641 Project 3: Clustering & Dimensionality Reduction

The purpose of this project is to explore a variety of Clustering algorithms and Dimensionality Reduction methods to see how they transform our datasets and potentially change the performance of certain learners. Namely, for clustering we will be using k-means clustering and EM for k-Gaussian models, and for dimensionality reduction we will be using PCA, ICA, RCA (random projections), and Factor Analysis (FA).

## The Datasets

The datasets used in this project are the same ones I've used in project 1 and project 2: The Pokémon Dataset (PDS) and Handwritten Letters Dataset (HLD).

The first is a compendium of all 800 Pokémon and their alternate forms. One entry includes the name, pokedex number, typing (categorical), secondary typing (categorical), individual stat numbers (hp, attack, defense, sp. attack, sp. defense, speed) and the stat total, the generation they were released in, and a label for whether the Pokémon is considered legendary or not (bool).

The latter is a dataset like MNIST; it contains a bunch of metadata on images of handwritten letters for classification. Instead of having the pixel matrix, however, we are just given 16 different measurements to base the classification on, like the edge count for the rows or columns, the mean x and y count of "on" pixels, etc.

## Data Preprocessing

I took multiple data preprocessing steps on each dataset for optimal formatting for any generic learner. For the PDS, this involved taking the categorical typing features and created a pseudo-one-hot-encoding, where each type was made into a column, and an entry would at most fill two of those columns with 1, else 0. The features of name, generation, pokedex number, and the original type columns were dropped since they weren't necessary for classification.

For HLD, the labels were categorical letters, therefore a label binarizer was used to create 26 additional features in one-hot fashion.

For the purposes of this project, I wanted to make sure the clustering algorithms' concept of distance wasn't skewed by the different scales of the features in either dataset. Therefore, a min-max scaler was used to normalize all the values for a feature to be between 0 and 1. This sort of normalization does disrupt the relationship between certain features (like stat total in PDS, or mean of x,x,y in HLD), but overall for the purposes of clustering I think the tradeoff is worth it.

# Clustering Algorithms on Raw Data

Our first experiment involved running K-Means clustering and Expectation Maximization by k-Gaussians on the data without any feature selection. These algorithms are both parameterized by a positive integer value *k*, which defines the number of clusters these algorithms try to make. In general, when talking about clustering, there isn't a specific goal when one tries to cluster their data but gather some

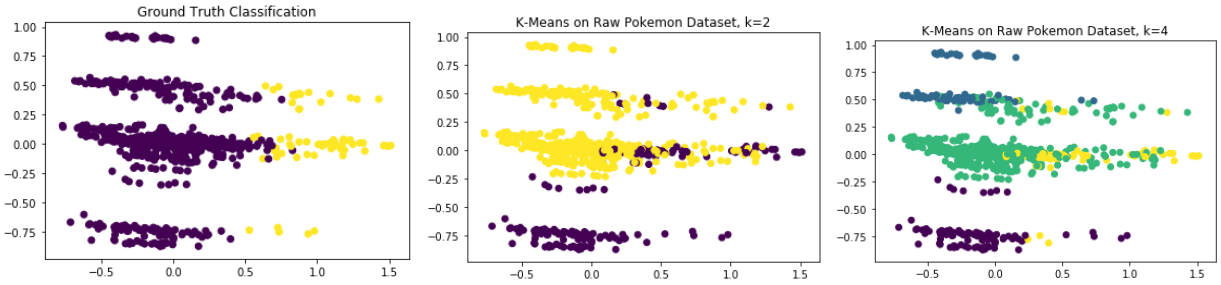
sort of insight on how the data naturally groups together, which may help feature selection/ transformation later, or perhaps create new classifications for the data. Clustering algorithms use a concept of distance to create their groups, which is usually given by domain knowledge. In this case, I will be using Euclidian distance as my distance function in both algorithms. Euclidian distance takes the direct line distance of each point in high dimensional space to say which point are close to each other and is the most intuitive distance metric. It doesn't like it when features are at different scales though since this can counter-intuitively change distance measurements, which is why I decided to min-max scale my features earlier.

## K-Means Clustering

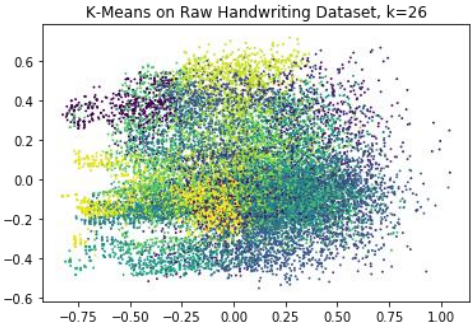
In K-Means clustering, the algorithm picks k random points in the data to be considered as the "center" of each cluster. These centers then poll each sample and gather the closest samples to their cluster. Once all samples are picked, the centers are taken as the mean of the new cluster, and the samples are redistributed. This continues until convergence or negligible change to the mean. This method is not guaranteed to either converge or reach an "optimal" clustering where the samples minimize the square distance to their means (also called inertia). To remedy this, random restart is used to minimize inertia.

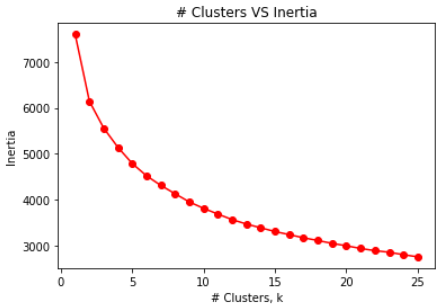
When comparing the ground truth to clusters, we have multiple metrics we can use to measure the quality of a clustering: inertia, homogeneity, completeness, and V-measure. When analyzing all clustering algorithms, I chose to look at inertia and v-measure. Inertia is a non-normalized metric meaning it can't be used to compare one clustering problem to another but can show how much better one clustering is relative to another in the same dataset and cluster number. V-measure is a harmonic average of homogeneity and completeness and is normalized (a value between 0 and 1, 1 meaning you match the ground truth) telling us how far the clustering is from the ground truth clustering.

## Results




When using KMeans on PDS, I visualized the clustering on a PCA-reduced 2D plane but made sure the clustering was run on the raw data. In the charts above, we see the ground truth classification, what K-Means returned as the two clusters, and K-Means for k=4. Though the main comparison for this experiment is between the GT and k=2 cluster, I chose to visualize the k=4 cluster because of the visual the permutation of the color doesn't matter, but the members within a cluster do (i.e. yellow and purple don't specifically represent legendary or non-legendary, just two different clusters). Performing K-means multiple times, the k=2 clustering averaged **1183 for inertia and 0.007 in v-measure**. This becomes apparent when you see how different the clusters are on the projected charts. The ground truth splits the data up vertically, while KMeans split it horizontally. There is also an uneven distribution of labels, heavily favoring normal Pokémon, and since KMeans generally tries to keep clusters the same size, that would explain the low v-measure score.




Compared to my binary classifier above, there are 26 ground truth labels in HLD, meaning I wanted to see k=26 when clustering. This means the visually, the clustering for HLD are not very useful, and instead I will use this dataset to display the relationship between k and the inertia metric.

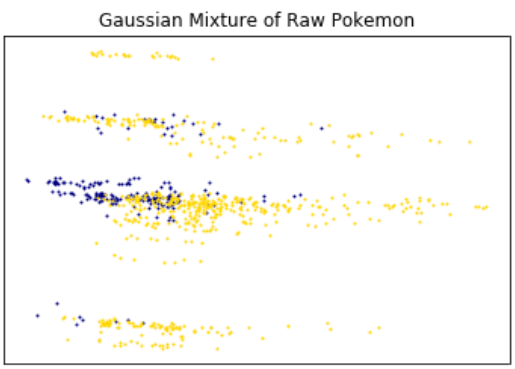
When running KMeans, k=26 multiple times on this data, I got an average **inertia of 2727, and v-measure of 0.349**, much higher than PDS. This meant KMeans more accurately found the clusters that represented the correct classification of letter, but still not very well since 0.349 < 1.0. This data has a much more equal distribution of classes in the samples (around 800 each) so class imbalance doesn't explain this one. Instead, I think it might be due to the scaling of my features, which caused some latent dependencies/distances in the data to be skewed.

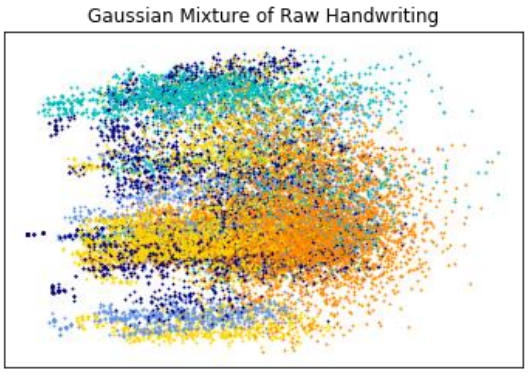
When looking at the k vs inertia graph, it highlights a certain intuition about inertia. The more clusters that are present, the fewer samples per cluster there will be, meaning the within cluster sum of squared error will go down the more clusters you have. This shows why inertia in not effective when comparing KMeans problems of different k.

## Expectation Maximization

Expectation Maximization (EM) isn't directly an algorithm for clustering, but for fitting gaussians to the samples. It's a form of "soft clustering" which allows for a sample to be in multiple clusters. EM iteratively predicts k means and confidences that certain samples belong in that cluster. Using probability, it maximizes the expectation that a sample belongs to one of the k-means by shifting the means and the gaussian covariance matrix. Below, we use a Gaussian mixture model, which utilizes EM to fit k-gaussians across the data.

## Results



Here we again see a very counter-intuitive clustering in PDS, and a visually interesting, but not useful clustering in HLD. Inertia isn't a metric when using GMM's since there isn't a hard cluster that a sample belongs to, but a probability that it belongs to each of the gaussians. Therefore, we will just be using v-measure. On PDS, GMM with k=2 gaussians got a **v-measure of 0.030,** while HLD with GMM k=26 gaussians got **a v-measure of 0.464**. Both cases are markedly higher than K-Means' scores, which implies that GMM does a better job with both unbalanced data and is scale-invariant. This is due to the lack of restriction on GMM for creating "equal" clusters, as well as the probabilistic nature of the method being able to handle scaled distances better.

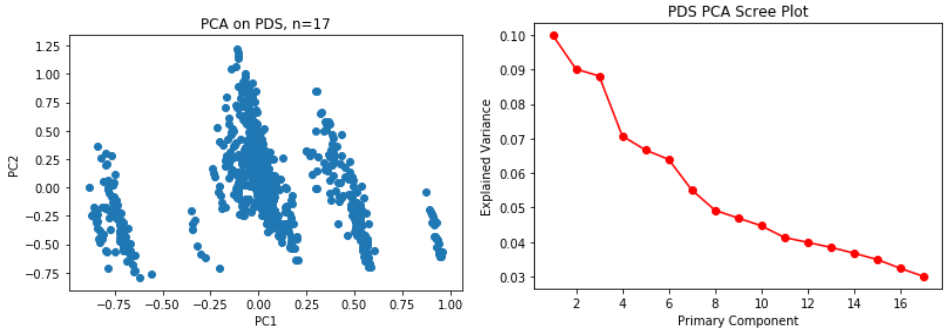
Both clustering algorithms still don't separate the data visually along their principle components well, which makes sense considering the raw data doesn't have the same distances as the PCA-reduced data.

# Dimensionality Reduction

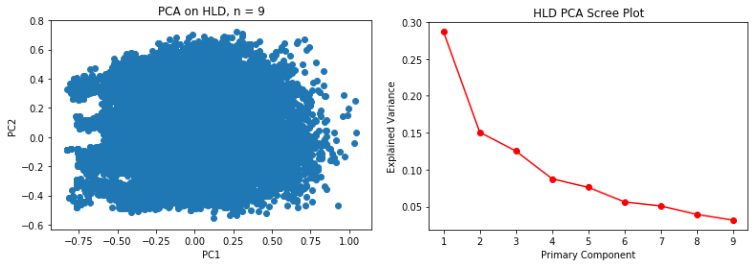
## Principle Component Analysis (PCA)

PCA is a method of Dimensionality Reduction that finds a linear combination for the data that maximize variance in the data. It is the most frequently used decomposition method because it creates new axes for visualizing the data in 2-3 dimensions while still retaining the relative distances between points. A PCA is considered "good" if it can cover most of the variance of the data in fewer features than the original. For the purposes of this assignment, principle components were added until the sum of explained variance for all of them was above 0.90. They were then projected onto the first two components.

### Results




When running PCA on PDS, we end up needing the first **17 principle components** to explain **0.9292 of the variances**. The scree plot shows how much each component contributes to the explained variance. Comparing PCA, n=17 to the PCA-reduced graph for KMeans above, we see a similar four-part split in the data visually, although its striated vertically here compared to horizontally above.

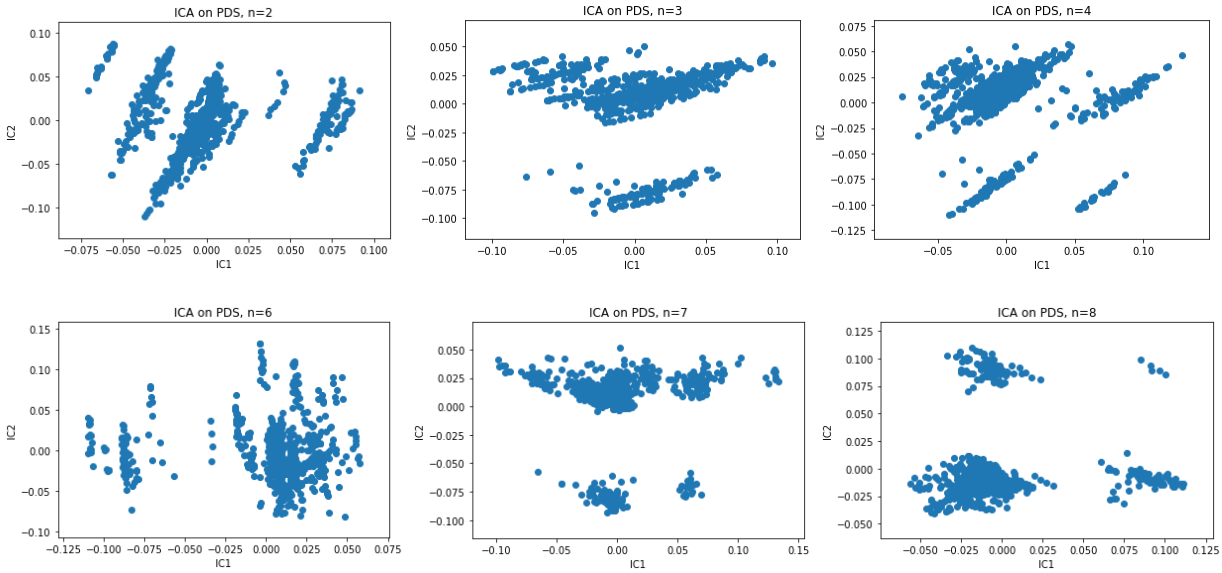


When running PCA on HLD, we need the first **9 principle components** to explain **0.9037 of the variances**. Visually, the graph is still useless, but looking at the scree plot, we can see that each individual component explains more variance here as compared to PDS, which is why fewer components were needed. It also follows that some of the individual features contributed more to the variance in HLD compared to PDS, where some features contributed equally.

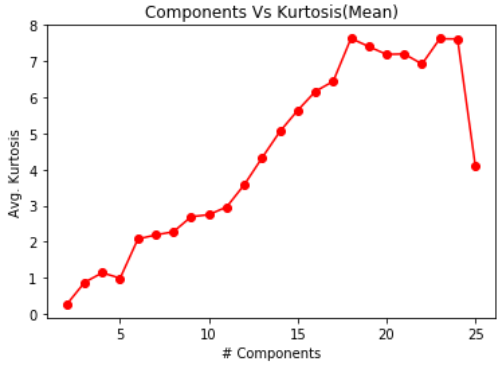
## Independent Component Analysis (ICA)

ICA is like PCA in that it is used to reduce the number of features, but rather than trying to maximize variance, ICA tries to minimize mutual information between features through linear combination. ICA works under the assumption that all the current features are dependent on independent hidden variables in some sort of linear combination and tries to find inverse transformation matrix such that there is no information shared in these new features, while maximizing dependency of the new features with the old features.

## Results




In PDS, I decided to test how changing the number of components that ICA computed would change the shape of the data. There was a constant trend of the data separating into 4 different chunks, some cleaner than others. I also compared the average kurtosis per axis to the number of components to judge whether some splits were better than others.




In general, the more components ICA decomposed, the more kurtotic it got, except for 26 components, which happened to be the original number of features in the dataset. This implies that the individual components there are that are independent, the more kurtotic the data becomes, which makes sense, since having more peaks means the distribution favors those values based on single independent random variables.

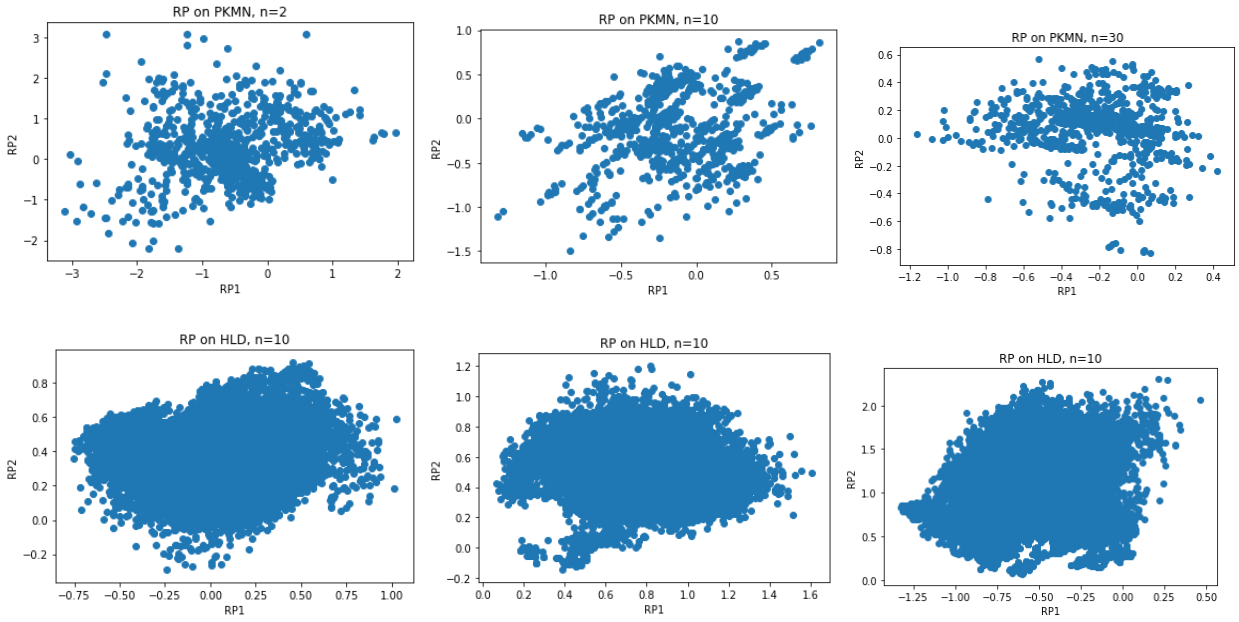
Running ICA on HLD didn't produce useful results since the data didn't visually separate as cleanly (too many data points), but it was found that 13 independent components were the most kurtotic.

## Random Projections (RCA)

Random Projections, also called Random Component Analysis is another form of projecting our features, but rather trying to maximize variance or minimize mutual information, it picks its linear combinations at random. Although in theory this seems worse than either PCA or ICA, it performs well in practice since the execution cost is low, allowing for many random restarts to find a decent combination.

### Results

After running RCA multiple times with different numbers of components, I found that the number of components did not seem to make a difference in how randomly the data was projected. Even running the RCA with the same number of components multiple times resulted in no noticeable increase of decrease in the amount of separation in the data. As the name implied, all my results were random.

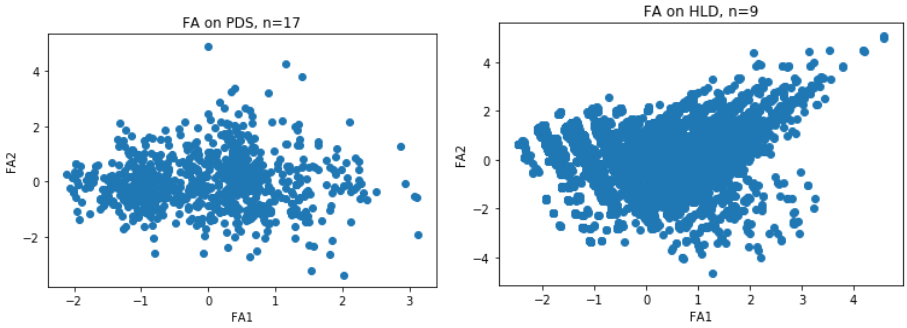



Because of its random nature, out of all the algorithms for dimensionality reduction, I believe that this one is the most scale-invariant since it picks randomly anyway. That means that one way to possibly improve the performance of this algorithm could be to inverse scale the data back to its original form. In general, however, I did not find this algorithm particularly useful in visually clustering my data.

## Factor Analysis (FA)

Factor Analysis is a very similar technique to PCA in that it tries to decompose features into its "latent factors" or hidden variables. It is a generative model that utilizes a gaussian noise parameter to account for noise in the data. Its in the variance of this noise that PCA and FA differ, as PCA makes the simplifying about the gaussian noise being isotropic, where as FA does not. This means FA is a more expensive, slower operation than PCA. The resulting factors can look like PCA but aren't bound to be orthogonal.

### Results




I wanted to compare FA to PCA more directly, so I chose the same number of components for each respective dataset (17 for PDS and 9 for HLD). Since FA isn't constrained to having purely orthogonal components, nor having a single component completely explain a certain percentage of variance, it wasn't possible to make a scree plot for this comparison. Instead, I wanted to take a more qualitative approach and see how the data separates visually. In PDS, we can see that FA does not as clearly separate the data as well as PCA. In fact, it looks almost as bad as RP. I imagine this is due to the increased cost of FA meaning more iterations may have been necessary to improve its performance in this dataset. Comparatively, in HLD, we see much clearer separation in FA as compared to PCA, with striation apparent here where no other algorithm can even compare. I believe this is since the features in HLD are heavily dependent on one another, so the variance can't be explained by single features alone. FA considers multiple features' contribution to the variance (covariance in a sense) and therefore separated the data more cleanly.

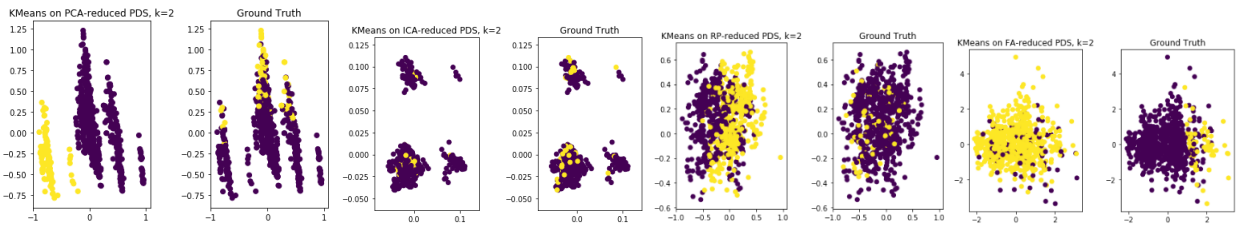
## Clustering on Reduced Data

After running each of these dimensionality reduction algorithms on our data, it makes sense to see if any new clusters could be created from it. Since some of the algorithms separated the data very cleanly, it would make sense that these algorithm's clusters would score better.

### The Pokémon Dataset

| Column1 | time | inertia | v-meas |
|---|---|---|---|
| KMeans on PCA-reduced PDS, k=2 | 0.14s | 1053 | 0.006 |
| KMeans on ICA-reduced PDS, k=2 | 0.06s | 7 | 0.034 |
| KMeans on RP-reduced PDS, k=2 | 0.06s | 1026 | 0.003 |
| KMeans on FA-reduced PDS, k=2 | 0.09s | 11957 | 0.034 |

For PDS, I ran each dimensionality reduction algorithm with the parameters that I deemed "best". For PCA that meant where explained variance >0.90 which is n=17, for ICA it was the greatest average kurtosis per axis, for RP, I arbitrarily picked n=20, and for FA, I wanted to compare it to PCA, so it was also n=17.



For PCA, ICA, and RP, the inertia of the clusters for k=2 in KMeans was lower than that of the raw data clusters. FA instead had an inertia 10 times greater than the raw data. The v-measure however was greatest in ICA and FA, while greatly reduced in RP, and staying close to the same for PCA relative to the raw clustering. This shows off that ICA greatly improved the cluster behavior for this dataset, implying that the ICA assumption that these features are the linear combination of latent independent variables is true.

| Column1 | time | v-measure |
|---|---|---|
| GMM-PCA PDS | 0.00s | 0.007 |
| GMM-ICA PDS | 0.00s | 0.007 |
| GMM-RP PDS | 0.00s | 0.006 |
| GMM-FA PDS | 0.00s | 0.017 |

Compared to KMeans, the effects of dimensionality reduction on this clustering method is less significant. The only significant improvement this time is Factor Analysis which has a v-measure of .017. I also found it interesting how this algorithm performs clock-time wise as compared to KMeans, always clocking in under 100 milliseconds. This is probably due to probabilities converging faster than clusters in EM as opposed to KMeans.

## The Handwriting Dataset

| Column1 | time | inertia | v-meas |
|---|---|---|---|
| PCA-reduced HLD | 19.81s | 2103 | 0.353 |
| ICA-reduced HLD | 14.14s | 5 | 0.446 |
| RP-reduced HLD | 24.21s | 2909 | 0.293 |
| FA-reduced HLD | 19.37s | 58337 | 0.402 |

For HLD, I ran PCA with n=9, ICA with n=13, RP was arbitrarily picked to be n=10, and FA was pick with n=9 to compare it with PCA. In this case, PCA and ICA once again both saw a sharp decrease in inertia from 2727, while RP and FA both saw increases, FA being twenty times greater. In terms of v-measure, PCA, ICA, and FA all had better results than the raw data clustering, which had 0.349.

One reason for why FA has a significantly greater inertia in both cases is probably because it isn't constrained to only produce orthogonal projections. Therefore, graphing it assuming orthogonality of the axes probably ruins the scale of the data.

As for why RP does worse in this case is more than likely due to the random nature of the algorithm; if I had run it X more times, it might have produced a set of projections that perform better than ICA.

| Column1 | time | v-meas |
|---|---|---|
| GMM-PCA HLD | 0.00s | 0.469 |
| GMM-ICA HLD | 0.00s | 0.494 |
| GMM-RP HLD | 0.00s | 0.425 |
| GMM-FA HLD | 0.00s | 0.461 |

This time, compared to KMeans, the effects of dimensionality reduction on the clustering was negligible for the most part, as the v-measure did not change too much from the raw score of 0.465. I believe this is due to the data having so many different labels, the scaling of the data skewed the relationship between features too much for the EM algorithm to properly cluster.

## New Learner on Reduced Data

After exploring all these dimensionality reduction algorithms, a new learner was used to see how performance changes after these techniques are applied. I chose to do my Pokémon Legendary Classifier again, since in project 1 it scored miserably in terms of F1 score. As a reminder, my classifier from project 1 had an average f1 score of 0.02531, only correctly classifying one legendary correctly. I chose F1 score because of the class imbalance in the data, meaning identifying true positives was the most important thing, misclassifying a legendary as normal was just as bad as misclassifying a normal as a legendary.

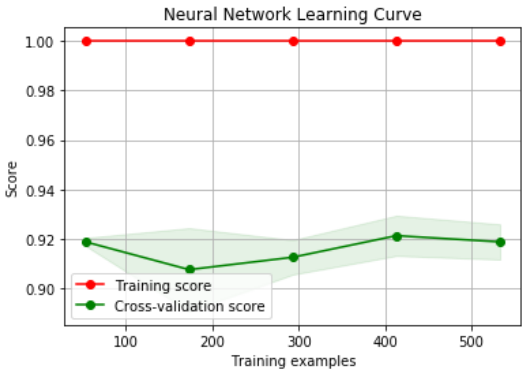
Since that project, I've made many changes to the data, including one-hot-encoding the types, and scaling all the features. In each of these cases, a dimensionality reduction algorithm was applied to the data, reducing the feature count from 25 to a lesser number. I also included a new baseline learner after running all my data preprocessing since the first project for comparison's sake.

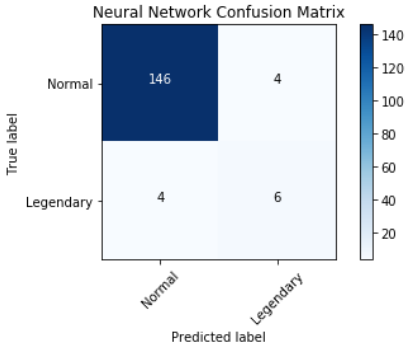
| Column1 | n_components | f1 score | TP |
|---|---|---|---|
| PCA | 17 | 0.32 | 4 |
| ICA | 23 | 0.416 | 5 |
| RP | 20 | 0.733 | 11 |
| FA | 17 | 0.428 | 6 |
| Baseline | 25 | 0.412 | 4 |

All learners do better than the learner on the raw data from project 1, as they all identify more than one legendary correctly. For these new learners based on the dimensionally reduced data, we see that RP does the best, scoring a 0.733 f1 score, which also beats out my f1 score in the HLD learner. This shows the power of RP, and that even though it may not be useful in visually separating data, neural networks still have the power to utilize any linear combination of features more so than the raw data. What I also find interesting is that FA performed the second best and PCA did the worst, which indicates that the simplifying assumption made by PCA may help humans identify clusters but doesn't necessarily help neural networks as much. ICA did do significantly better than the raw data learner, but despite the clusters in ICA being way more visually distinct, these clusters probably had nothing to do with the legendary labeling associated with the Pokémon.

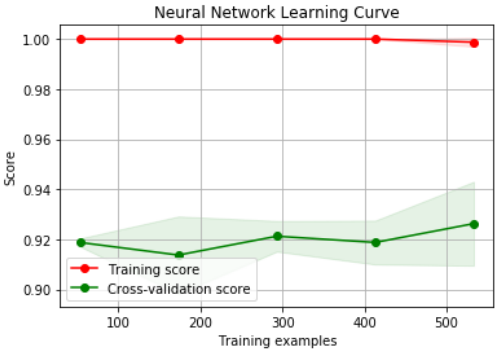
## New Learner with Feature Transformation (Clustering)

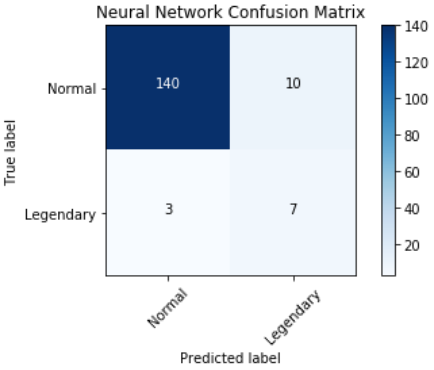
And in this final experiment, we see how the clustering algorithms may give insight to a learner on how to classify the instances. This is done by appending the cluster information to the feature list of the data. In situations where k=n_labels and the clusters accurately represent the ground truth; these columns are mutually dependent on the class labels and therefore have a lot of information regarding them. This should theoretically help a learner a lot if the clusters are good. If the clusters don't represent the ground truth at all, then that may just add another feature that doesn't give information to the learn, hurting it overall.




For K-Means cluster-appended data, we got a f1 score of 0.6, which is almost as good as the RP-based learner from the last experiment. If classified 6 legendries in the test set correctly, which means the cluster information carried by K-Means contributed to it greatly. The clusters made by K-Means captured a lot of what it meant for a Pokémon to be “legendary” or at least better than normal.




For GMM cluster-appended data, we got an f1 score of 0.51 but classified 7 Pokémon correctly in the test set. There were more false positives in this learner, meaning GMM clusters considered too many normal Pokémon as legendary. This probably has to do with the probabilities of those instances being in a probabilistic mid ground where GMM soft clustering couldn’t strictly consider them as one or the other. This is interesting because that means there are quite a few “normal” Pokémon that could be clustered with legendaries, but few legendaries that can be considered “normal”. Overall, these two performed similarly but both provided different insights into the data that a learner could learn from.

## Conclusion

In sum, after exploring KMeans and EM clustering algorithms, and a variety of different dimensionality reduction algorithms, I’ve seen how they work in a variety of different problem spaces. KMeans tends to create equal size clusters with hard limits while EM creates more flexible clustering that sometimes are too lenient. Of all the dimensionality reduction algorithms, each one has its spot given certain pieces of domain knowledge, like using ICA in blind source situations or PCA for condensing data into fewer dimensions. We also learned that RP may not be visually appealing but can work well when paired with a supervised learner. Overall, unsupervised learning and dimensionality reduction are tools to help us understand our data and establish some domain knowledge for supervised learners to utilize.