Assignment #3: Unsupervised Learning and Dimensionality Reduction

Connor Bohan – connor.bohan@gmail.com

Clustering

Datasets

I chose to continue using my Pokemon dataset from previous assignments. This dataset contains 34 unique labels (types of Pokemon) and 25 instances of each label. The dataset is generated at runtime where each Pokemon has a type, a height, and a weight. The type for a given Pokemon is fixed while the height and weight are picked from a gaussian distribution. The means of those distributions are set in code and the standard deviation is always 10% of the mean. What is interesting about this dataset is that, given how similar some Pokemon are to others, no classifier can perfectly classify the data. This could give the clustering algorithms a tough time. I suspect that certain groups of Pokemon will get lumped into the same cluster while some Pokemon will be split among multiple clusters.

The second dataset I chose was a subset of the MNIST digit dataset. This dataset is a series of 28 pixel by 28 pixel labeled images. Each image contains a single digit. I chose to convert the images into the Weka ARFF format. The images are in greyscale which means that each entry in the dataset has 28 times 28 or 784 dimensions. I think this is a good contrast to my Pokemon dataset. The MNIST images should test the clustering algorithms ability to deal with data that has high dimensionality. The subset of the MNIST dataset I chose contains 5 digits (0-4) and 15 examples of each digit. I felt that this should be enough data to allow clustering to work while also keeping things small and easy-to-manage.

K-Means

When using K-means clustering on the Pokémon dataset I chose a k value of 34. I figured that, since there are 34 unique labels, that made the most sense. Below is a list the predominant cluster[s] for each Pokemon.

Bulbasaur 27	Ivysaur 30	Venusaur 25	Charmander 14	Charmeleon 32	Charizard 22	Squirtle 5	Wartortle 5	Blastoise 5
Caterpie 6/8	Metapod 17/23	Butterfree 4/9	Weedle 6/8	Kakuna 7/16/23/24	Beedrill 4/9	Pidgey 31	Pidgeotto 3	Pidgeot 13/2
Rattata 19	Raticate 1/33	Spearow 0/18	Fearow 2/3	Ekans 12	Arbok 28	Pikachu 11	Raichu 11	Sandshrew 21
Sandslash 10/20	Nidoran f. 15	Nidorina 15	Nidoqueen 28	Nidoran m. 15	Nidorino 15	Nidoking 28		

Figure 1: The predominate cluster for each Pokemon when using the K-Means method. Ideally, each Pokemon would be in its own unique cluster. For Pokemon where multiple clusters are listed, that Pokemon didn't have a single cluster that it predominately fell into.

Looking at those clusters, I can see where clustering has issues with this kind of data. K-means did a fairly good job distinguishing between Pokemon for the most part. It stumbled exactly where I expected it to. Pokemon of the same time, with similar weights and heights tended to get grouped together into clusters that were too big. For example: because Nidoran female, Nidorina, Nidoran male, and Nidorino are all medium-sized poison Pokemon they got lumped together. There were a few surprises, however. I didn't expect Squirtle and Blastoise to be grouped together. Although they share a type, Blastoise is 10 times heavier and 3 times taller than Squirtle. I thought that this difference would be enough to separate them. I also didn't foresee the messy groupings for Caterpie, Metapod, Butterfree, Weedle, Kakuna, and Beedrill. These are all small bug type Pokemon, so I expected them to be grouped into a single cluster but instead they were each split among several clusters. On the whole k-means did an adequate job lining up the labels to clusters.

I chose a k value of 5 for the MNIST dataset subset because there are 5 different labels. The clusters for this data weren't as good as I expected. In general, each label of image was placed in a unique cluster. However, cluster 0 contained too many individuals. Labels 2 and 3 were spread over several clusters. I expected that each clusters and labels would line up better than they did. I think that the issue is probably that K-Means is arriving at a stable solution too quickly. I suspect that EM will cluster this dataset much better than K-Means.

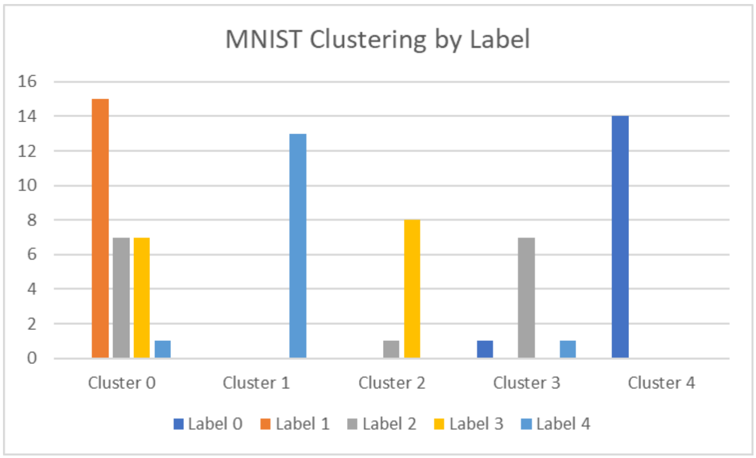

Figure 2: A bar chart indicating which clusters contained which images after running the MNIST data through a K-Means clusterer. E.g. 15 images with the label '0' were put into cluster 0.

Expectation Maximization

For both the Pokemon and MNIST datasets, I used the same k values with Expectation Maximization that with K-Means clustering. In general, EM did much better than K-Means at clustering Pokemon based on their labels. The clusters tended to be messier (i.e. a given label tended to have individuals in several clusters) but there were fewer super clusters that contain many Pokemon.

Bulbasaur 0	Ivysaur 15	Venusaur 33	Charmander 13/31	Charmeleon 1	Charizard 29/30/32	Squirtle 25	Wartortle 3/17	Blastoise 29/32
Caterpie 18	Metapod 2	Butterfree 22	Weedle 16/18	Kakuna 2	Beedrill 22	Pidgey 8	Pidgeotto 11/22	Pidgeot 5/23
Rattata 7	Raticate 20	Spearow 14	Fearow 23	Ekans 12	Arbok 28	Pikachu 6	Raichu 4	Sandshrew 27

Sandslash 26	Nidoran f. 6	Nidorina 24	Nidoqueen 9	Nidoran m. 13	Nidorino 24	Nidoking 9		

Figure 3: The predominate cluster for each Pokemon when using the Expectation Maximization method. Ideally, each Pokemon would be in its own unique cluster. For Pokemon where multiple clusters are listed, that Pokemon didn't have a single cluster that it predominately fell into.

As you can see the Nidoran female, Nidorina, Nidoran male, and Nidorino cluster from K-means was broken into two clusters which makes more sense given the difference in size between Nidoran male/female and Nidorina/Nidorino. Caterpie, Metapod, Butterfree, Weedle, Kakuna, and Beedrill were clustered less messily as well. To improve the quality of clustering for both EM and K-Means I would ensure that the distance metric properly handled the Pokemons' types. Since the type attribute has no noise, any two individuals of different type should never be clumped together.

EM performed no better at clustering the MNIST data than K-Means did. This was surprising to me. I had thought that K-Means was converging too quickly because of the relatively small number of individuals in the data. I believed that EM would be able to overcome this lack of data to converge on a better set of clusters. However, if anything, EM did worse. Both zeros and twos were spread over multiple clusters. I don't have an explanation for why this occurred.

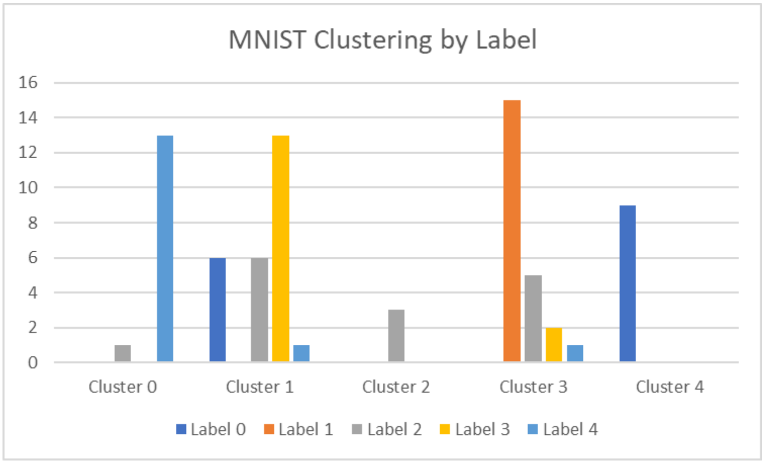

Figure 4: A bar chart indicating which clusters contained which images after running the MNIST data through an EM clusterer. E.g. 13 images with the label '4' were put into cluster 0.

Dimensionality Reduction

When I ran PCA on the Pokemon dataset, I noticed that there was very little room for dimensionality reduction. This makes sense. Since 9 out of 11 dimensions in the problem relate to the type of the Pokemon, all those dimensions are required. Combining any two of those dimensions in any way would result in a loss of information. However, weight and height are highly correlated. These two dimensions could be rolled together in one. I believe this is what I observed in the eigenvalues. There was one high eigenvalue of 2.1, one low eigenvalue of ~0, and the rest were between .2 and 1.25. The MNIST dataset is a little harder to understand. Looking at the eigenvalues, it seems like most of the information is captured in just the first 5 eigenvectors. Its also important to note that there are only 74 eigenvectors that capture any information at all (there are 75 digits in the dataset I'm using so this makes sense). I suspect that I'll be able to get good clustering results using 5-25 eigenvectors. The other dimensionality reduction techniques I used were ICA, RCA, and random subset (it just removes dimensions at random). I had a hard time deducing what the effects of ICA and RCA were and decided to see how the clustering algorithms responded to them before making any judgements. Random subset simply removes features without trying to preserve information.

Clustering After Dimensionality Reduction

After PCA, both K-Means and EM were unable to cluster the MNIST digit dataset. There tended to be one or two large clusters that contained most of the individuals. It seems that the resulting data after being transformed by PCA is clustered in some way that doesn't correspond to the original labels. I suspect that qualities like brightness were being selected for. This won't work for the images because the brightness of an image is more dependent on the pen stroke width than the digit being written. The Pokemon dataset didn't have this problem. PCA seemed to preserve most of the useful data. Doing a cursory inspection, K-Means and EM both did a little worse on the Pokemon dataset after PCA. Qualitatively, the clusters were more likely to contain multiple Pokemon and it was more likely that a single Pokemon label was split between multiple clusters.

ICA worked much better for the MNIST dataset. K-Means with ICA was a significant improvement. There were no longer problems with digits being split equally between multiple clusters. This was even more so the case for Expectation Maximization. I reran ICA -> EM several times and there tended to fewer than 6 individuals placed in the 'wrong' cluster (e.g. if most images with the label '0' were in cluster 1 then an image with label '0' placed in cluster 2 would be in the wrong cluster). Interestingly, reducing the data to 3 dimensions (from the original 784) seemed to produce the best results. That would tend to indicate that there are only a handful of vital features in the dataset. The Pokemon dataset suffered when exposed to ICA. Clusters tended to include more Pokemon. Pokemon labels often had half their individuals in one group and half in another. In general, it doesn't seem like that dataset can be decomposed into separate components. This was true no matter how many dimensions I reduced (or even increased) the data into.

RCA was unpredictable when used in combination with K-Means on the MNIST dataset. I was to get reasonable clustering with as few as 50 features, but the clustering wasn't guaranteed to be good. I can certainly see the value in reducing the dimensionality of a problem to 6% the original amount. But in order to get good, reliable clusters multiple runs of the clustering algorithm would need to be combined in some way. EM performed similarly to K-Means in combination with RCA. Interestingly, the Pokemon dataset was still clusterable after being put through RCA. I had originally thought that the one hot encoding of the Pokemons' types would get mangled by random reprojection. While both EM and K-Means didn't do as well with the data after it had been put through RCA (reducing the dimensions from 11 to 7), RCA did just about as well as PCA. I suppose that it makes sense given that the randomized projection would preserve the orthogonality of the one hot encoding.

The final dimensionality reduction technique I employed was a random subset filter. This is the most basic dimensionality reduction algorithm. It simply removes features until it hits some target number. I didn't expect this to work particularly well and it didn't. On the MNIST dataset there was some room to remove features since there were 784 of them. I could get to around 500 features before I started to see noticeable degradation in the quality of clustering. The Pokemon set was more harshly impacted. Removing just 2 features caused a noticeable decrease in the quality of clustering. I can't see any reason to employ a random subset filter over randomized projections.

Neural Network Learning After Dimensionality Reduction

After performing my dimensionality reduction to clustering experiments, I combined dimensionality reduction and neural network learning. For PCA, I mapped the percent of Pokemon correctly classified compared to the variance coverage of the PCA.

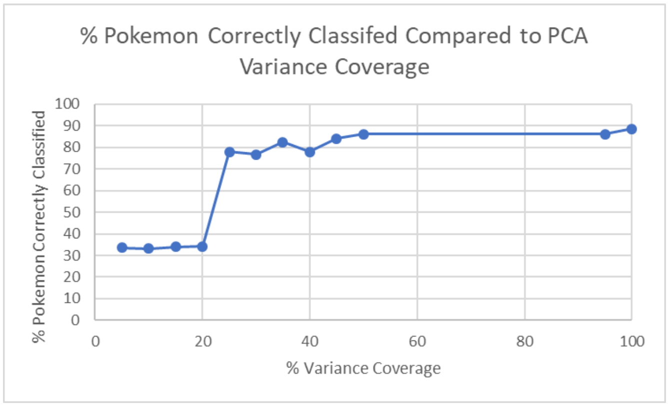

Figure 5: Pokemon correctly classified by the neural network learner after the Pokemon dataset was put through PCA dimensionality reduction. The x-axis corresponds to the variance coverage of the PCA.

From the data we can infer that there is a critical dimension added at the 20%-25% variance coverage mark. Interestingly, at the 50% variance coverage mark there is no difference between using the data from the PCA dimensionality reduction algorithm and the original data. It seems like PCA is actually able to retain enough important information for the neural network learner to work. The dimensionality reduction did not help much with performance though. At 50% variance coverage the neural network learner required 5.97 seconds on average to finish training while at 100% variance coverage the neural network learner required 6.72 seconds on average to finish training. While this difference is noticeable, its not worth the effort. I suspect that the improvement was so small because the data only has 11 dimensions to begin with. Data with more dimensions, such as the MNIST dataset, would likely benefit more.

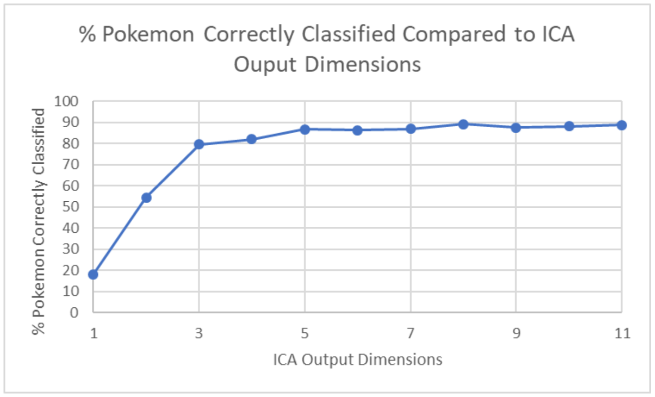

Figure 6: Pokemon correctly classified by the neural network learner after the Pokemon dataset was put through ICA dimensionality reduction. The x-axis corresponds to the number of dimensions after ICA dimensionality reduction.

With clustering, I noted that ICA did best with only 3 dimensions. Looking at how the neural network learner performs, there is a similar trend visible here as well. Each dimension added dramatically increases the neural network learner's ability to classify data up to 3 dimensions. From 3-5 dimensions there is steady, but slow, improvement. And from 6 dimensions on, there is no real improvement in the neural network learner's ability to classify. There was no observable time benefit to doing the dimensionality reduction. For 1 and 2 dimensions, the neural network learner took ~5 seconds to learn, while for 3-11 it took between 5.5 and 6.3 seconds learn. This isn't enough to justify the dimensionality reduction.

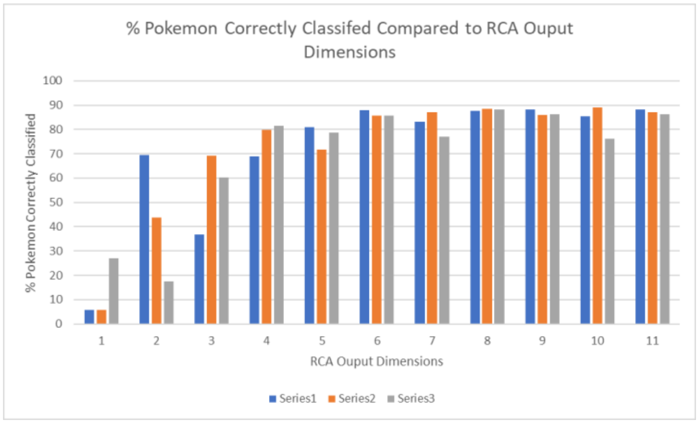

Figure 7: Pokemon correctly classified by the neural network learner after the Pokemon dataset was put through RCA dimensionality reduction. The x-axis corresponds to the number of dimensions after ICA dimensionality reduction. Each number of output dimensions was tested 3 times due to the unpredictability of RCA.

RCA, unsurprisingly, proved unpredictable. With as few as 2 dimensions, I was able to get decent results with the neural network learner. The variance was large though. The best accuracy achieved by a neural network learner with 2-dimensional input was 70% while the worst was 17%. This was expected. Since the axes that RCA re-projects the data onto are random, they can be either very useful for classification or completely useless. An interesting experiment would be to find the axes that are the most useful using repeated runs of PCA and compare them to PCA and ICA to see which they most resemble.

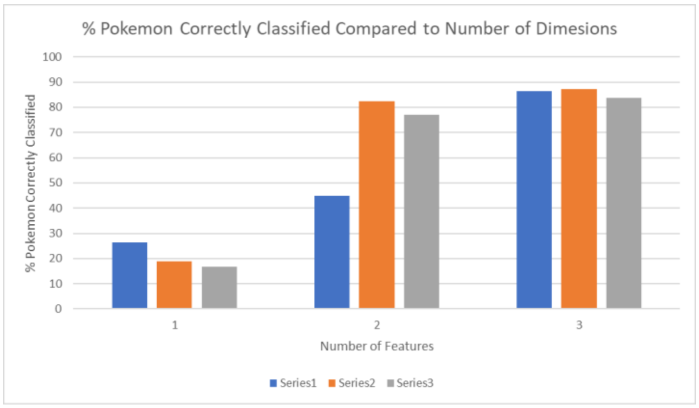

Figure 8: Pokemon correctly classified by the neural network learner after the Pokemon dataset was put through random subset filter dimensionality reduction. The x-axis corresponds to the number of dimensions after the random subset filter dimensionality reduction. Each number of output dimensions was tested 3 times due to the unpredictability of a random subset filter. Note: whereas PCA, ICA, and RCA turn the 'type' of a Pokemon into 9 one-hot encoded dimensions, the random subset filter does not. Therefore, the maximum number of dimensions is 3.

The random subset filter is fairly uninteresting. There wasn't any significant increase in speed after dimensionality reduction and the method proved too unreliable to be useful. One thing to note is that during one of the 2-feature runs the accuracy of the neural network learner was comparable to a 3-feature run. This means that 2 of the dimensions (probably height and weight) are nearly redundant.

Next, I applied PCA dimensionality reduction onto the Pokemon dataset, performed K-means clustering on the dimensionality reduced data, wrote those clusters back into the data, and taught a neural network learner using 10-fold cross validation to see if the dimensionality reduced data with the clusters would be more 'learnable' than the data without clusters. The average accuracy of the neural network learner after 6 runs was 84.51% which was worse than before. There could be several reasons for this. Firstly, the 10-fold cross validation means that less data was being use for both training and testing. Using less data for training means that the neural network learner will be less accurate. Another problem is that the clusters might not have been very useful. For many Pokemon there were one or two Pokemon per type that weren't grouped in the same cluster as the rest. A neural network learner that was using this clustering information might have been thrown off by this. I don't see this method being particularly useful for datasets like the Pokemon dataset, since it doesn't provide any additional accuracy but adds more steps where bugs could occur. Exchanging K-means for expectation maximization did nothing to improve the accuracy of the learner.

I performed the same series of tests using ICA dimensionality reduction as well. Since ICA clustering did the best with 3 features, I also graphed the accuracy of the neural network learner against the number of features after ICA dimensionality reduction (note the actual number of features is the ICA features plus the cluster). With the ICA clusters, there was also no improvement in accuracy. The accuracy of the learner increases quickly up to 3 features, then continues to increase slowly up to around 7 features before plateauing. I suspect the neural network learner is leaning more heavily on the clusters the fewer features there are. As the number of features increase, there is more information for the learner to use, so it gets close to the original accuracy levels.

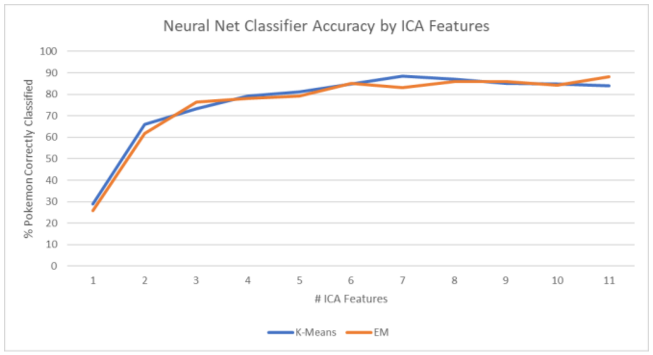

Figure 9: Pokemon correctly classified by the neural network learner after the Pokemon dataset was put through ICA dimensionality reduction, clustering, and having those clusters put back into the data. The end result is that the learner does no better than it did without the clustering information. Using either K-means or EM seems to have no impact on the accuracy of the learner.