Introduction:
In this assignment, I chose to use the same datasets as I did in previous assignments. Both datasets are collections of text snippets. The binary classification dataset is composed of reddit comments – some of them were removed by human moderators while others were not. The multi-class classification dataset is a collection of news headlines and sub-headers classified into one of 20 categories. Both these datasets are interesting because classification today is done manually. By figuring out how to predict classifications for these datasets, we would be able to dramatically reduce the cost of processing large bodies of text. These two datasets are also interesting to compare since both require vectorizing text data, but one is binary while the other is multi-class.

To start off the upcoming experiments, I used the same feature extraction pipeline I used in prior assignments. I converted each of the text snippets into feature vectors using tf-idf. This allowed me to weight commonly encountered words more lightly, while increasing the weight of more uncommon and unique words. Whenever possible, I used sparse matrices to improve performance.

K-means Clustering
I applied k-means clustering to the binary dataset using k=2. I chose k=2 because using my existing domain knowledge, I knew that I wanted to predict of the comments were removed or not. Similarly, for the multi-class dataset, I chose k=20 since I know that there are 20 news topics being used as classification buckets. Euclidean distance is the default distance formula used for k-means clustering in scikit-learn, so I used that first. I ran both k-means++ and random initialization methods.

	Homogeneity	Completeness	V-measure	ARI	Silhouette	Accuracy	Time
Binary {n_clusters: 2}							
Kmean++	0.017	0.017	0.017	0.003	Euclidean: 0.003 Cosine: 0.005	0.459	56s
Random	0.018	0.018	0.018	0.004	Euclidean: 0.002 Cosine: 0.004	0.458	51s
Multi-class {n_clusters: 20}							
Kmean++	0.209	0.297	0.245	0.059	Euclidean: -0.006 Cosine: -0.017	-	774s
Random	0.237	0.317	0.271	0.068	Euclidean: -0.006 Cosine: -0.014	-	567s
{init: k-means++, n_init: 10, max_iter: 300}							

The results measuring the efficacy of clustering were dismal in the binary case and bad in the multi-class case. When comparing the two cases, it's interesting to note that the binary case does a poor job of clustering but is actually more accurate (using v-measure and ARI). An important thing to note is that the accuracy measure for the multi-class case is not label agnostic, since when comparing the clustering results to the expected classifications, the clustering algorithm has no sense of what each cluster means. As a result, I removed the accuracy measurement for the multi-class case. In the multi-class case, homogeneity was better. This is likely because the underlying data of news headlines tends to highlight unique features more, so the data is naturally more clustered. On the other hand, the binary class is trying to separate removed comments from not removed comments, but the content of all comments probably overlaps heavily. The binary class only has better accuracy because ~60% of comments are not removed – this means that it could just be getting lucky. In the multi-class problem, there are 20 possible labels, so it is harder for the model to get it right randomly.

By observing the simpler case of binary classification, my hypothesis was that the underlying data was too noisy for Euclidean distance to handle. Euclidean distance tends to handle outliers poorly – letting any outliers dominate the analysis. Further, it is trying to minimize cluster variation within a cluster, thus weighting big clusters more than smaller clusters. To test whether or not this hypothesis was true, I printed the top words being used to cluster the text bodies into groups in the binary case. The words were

"the, this, to, is, of, and, it." Clearly, the model was using the wrong types of words to differentiate between different clusters and was instead choosing the most common words. The Euclidean distance was letting the most common words dominate the analysis. In order to fix that, I enabled the stop_words function in tf_idf, where the vectorizer removes the most common words in the English language so that only more unique words remain. Indeed, when pulling the top words again, I saw "people like just study message reddit com www post." This overall adjustment actually improved accuracy quite a bit, but the clustering metrics still indicate that the clustering algorithm is not working well in the binary case. Removing common words improved clustering performance in the multi-class case, but overall, the results achieved here are still insufficient to use in practice. Interestingly, the ARI for multi-class indicates a marked improvement in similarity within clusters, and the v-measure indicates a marked improvement in "agreement" with labeling.

	Homogeneity	Completeness	V-measure	ARI	Silhouette	Accuracy	Time
Binary {n_clusters: 2}							
Kmean++	0.000	0.000	0.000	0.003	Euclidean: 0.001 Cosine: 0.001	0.587	36s
Random	0.000	0.005	0.000	-0.001	Euclidean: 0.001 Cosine: 0.000	0.615	31s
Multi-class {n_clusters: 20}							
Kmean++	0.343	0.429	0.381	0.118	Euclidean: 0.006 Cosine: 0.012	-	496s
Random	0.354	0.439	0.392	0.126	Euclidean: 0.007 Cosine: 0.013	-	612s
{n_init: 10, max_iter: 300}							

Another possibility I explored to improve the k-means results was to figure out if the algorithm had fallen to a local minima, since k-means algorithms have a tendency to fall to a local minima. To mitigate this, I tested a couple different values for n_init, since that parameter introduces randomness in how clusters are initialized. In these tests, I used the random init method since we are trying to introduce more randomness, and that method also resulted in a better results in the prior experiment. Adjusting n_init did not make a significant difference in overall results, and also took much longer, which indicates that we were not a local optimum and increasing randomness is unnecessary.

N_init	Homogeneity	Completeness	V-measure	ARI	Silhouette	Accuracy	Time
Binary {n_clusters: 2}							
10	0.000	0.000	0.000	0.003	Euclidean: 0.001 Cosine: 0.001	0.587	36s
20	0.000	0.001	0.000	0.003	Euclidean: 0.001 Cosine: 0.000	0.387	65s
30	0.000	0.005	0.000	-0.001	Euclidean: - 0.001 Cosine: 0.002	0.615	99s
Multi-class {n_clusters: 20}							
10	0.343	0.429	0.381	0.118	Euclidean: 0.006 Cosine: 0.012	0.030	496s
20	0.340	0.413	0.373	0.135	Euclidean: 0.006 Cosine: 0.012	0.080	1007s
30	0.349	0.432	0.386	0.145	Euclidean: 0.007 Cosine: 0.011	0.080	1362s
{init: random, max_iter: 300}							

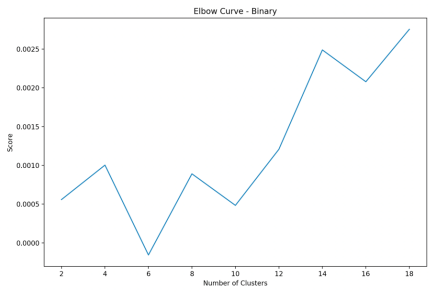
I moved onto considering whether or not high dimensionality was impacting results. Particularly with k-means and Euclidean distances, high dimensionality tends to inflate distance calculations. Because text vectorization essentially creates a dictionary of text with a column per word, there are a lot of dimensions that k-means clustering needs to account for. If the data is also sparse, it could further result in outliers

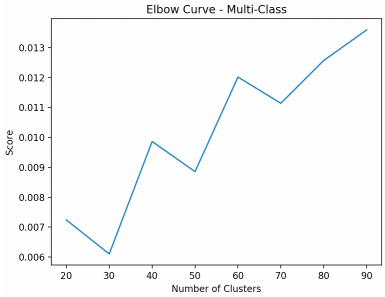
that disproportionately impact k-means clustering. To test this, I tried to force-reduce dimensionality by specifying max_features in the tf-idf vectorizer. The max_features parameter only selects the top words ordered by term-frequency for consideration. For this analysis, I cared most about the silhouette coefficient, since it measures how well clusters are defined by measuring both intra-cluster and inter-cluster distance. As shown in the table below, reducing the number of features significantly reduced processing time. However, as the number of features grew, we start to observe how increasing dimensionality hurts the overall effectiveness of the clustering algorithm in the binary case. In the binary case, accuracy increased slightly, but my suspicion is that the dataset itself already leans towards having the majority of comments not removed, so the model is simply choosing the outcome that occurs most often, but has not learned to differentiate between t. Indeed, when pulling top words, "people just study don think" were used for cluster 1 and "like just people feel don" were used for cluster 2. Notice the significant overlap. In the multi-class case, decreasing the number of features used significantly hurt v-measure compared to the prior experiments before. This is likely because the multi-class case needs more dimensions to determine classification across 20 classes. When pulling top words, the multi-class case had less overlap than the binary class, but still had significant overlap.

Features	Silhouette	Accuracy	Time		Features	Silhouette	V-M	Time
10	Euclidean: 0.430 Cosine: 0.065	0.581	0.8s		10	Euclidean: 0.327 Cosine: 0.406	0.059	14s
20	Euclidean: 0.261 Cosine: 0.041	0.587	1.8s		20	Euclidean: 0.147 Cosine: 0.212	0.06	31s
40	Euclidean: 0.194 Cosine: 0.031	0.589	2.6s		40	Euclidean: 0.104 Cosine: 0.194	0.033	63s
{init: random, n_init: 10, n_clusters: 2, max_iter: 300}					{init: random, n_init: 10, n_clusters: 20, max_iter: 300}			

I also considered using a spherical k-means clustering algorithm, but since cosine similarity does not perform much better in the silhouette measurement, I decided that Euclidean distance was good enough.

The above analysis, though, assumed that we had some domain knowledge, and this assumption could be impacting overall results. If we stopped worrying about the "accuracy" of the algorithm, and instead focused on being able to cluster text correctly, how would that change our approach? The main difference would be that we would be flexible on the number of components needed to cluster things. This makes particular sense in the binary case, since force fitting the comments into two classes that are potentially largely overlapping does not work well with the k-means algorithm. To test this, I implemented an elbow curve for both the binary and multi-class datasets. Both elbow curves for the binary and multi-class datasets seem to indicate that better clustering can be achieved if we increase the number of clusters.





Expectation Maximization

As discovered above, the k-means clustering algorithm was not performing well, largely because the underlying data is noisy and overlapping. The problem is that k-means cannot measure a sense of probability or uncertainty, and only draws circles on top of data. To address this issue, I tried a different clustering algorithm using Gaussian Mixture Models. Gaussian Mixture Models assign probabilities to

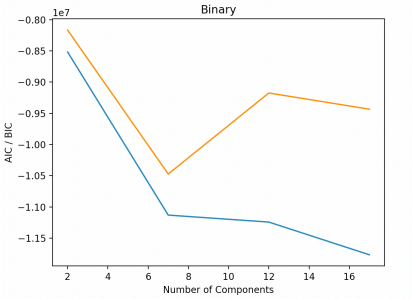
datapoints and attempt to map out different probabilistic models on top of data. To test the performance of this, I ran a number of experiments with different parameters, as shown below. Although GMM is very efficient (note the difference in run time compared to k-means clustering), it is also only implemented for dense matrices, so I had to limit the number of features being used to run the analysis. Since text vectors can grow really big really quickly, it was simply infeasible to run the analysis on everything. I tried using a large machine with 48GB of RAM, and that machine still threw memory errors.

I tested several different types of covariance. "Diag" measures the size of each cluster independently and is useful for representing ellipses. "Spherical" runs faster, but is constrained to equal dimensions and thus is comparable to normal k-means clustering. "Full" allows each cluster to be represented as an ellipse with arbitrary orientation. "Tied" allows each component to share a covariance matrix with the entire dataset, which is useful for overlapping datasets. "Spherical" allows for each component with a single variance. Ultimately, looking across the metrics, kmeans GMM with diag covariance for the binary dataset and spherical covariance for the multi-class dataset worked the best. For the binary dataset, this is likely because the two clusters being selected aren't correlated to each other, so building independent clusters is more effective. For the multi-class dataset, spherical worked the best since each component gets normalized, thus allowing more comments to be considered and reducing the weight of more common components.

	Homo-geneity	Complete-ness	V-measure	ARI	AMI	Silhouette	Accuracy	Time
Binary {n_components: 2}								
k-means, n_init: 1	0.001	0.001	0.001	-0.006	0.001	Euclidean: 0.378 Cosine: 0.023	0.578	0.09s
random, n_init: 1	0.004	0.004	0.004	-0.003	0.004	Euclidean: 0.449 Cosine: 0.084	0.492	0.05s
k-means, n_init: 5	0.006	0.014	0.008	-0.017	0.006	Euclidean: 0.374 Cosine: 0.039	0.573	0.9s
random, n_init: 5, full	0.006	0.009	0.007	-0.016	0.006	Euclidean: 0.398 Cosine: 0.040	0.542	0.18s
k-means, n_init: 5, diag	0.000	0.000	0.000	-0.005	0.000	Euclidean: 0.442 Cosine: 0.071	0.443	0.07s
k-means, n_init: 5, tied	0.006	0.030	0.011	-0.013	0.006	Euclidean: 0.359 Cosine: 0.033	0.405	0.08s
k-means, n_init: 5, spherical	0.000	0.001	0.000	0.003	0.000	Euclidean: 0.372 Cosine: 0.025	0.609	0.14s
k-means, n_init: 5, diag, features:40	0.013	0.013	0.013	-0.008	0.012	Euclidean: 0.156 Cosine: 0.019	0.512	0.28s
Multi-class {n_components: 20}								
k-means, n_init: 5	0.057	0.063	0.060	0.018	0.052	Euclidean: 0.197 Cosine: 0.186	-	9s
random, n_init: 5, full	0.050	0.063	0.056	0.019	0.045	Euclidean: 0.022 Cosine: -0.037	-	7s
k-means, n_init: 5, diag	0.060	0.063	0.062	0.022	0.055	Euclidean: 0.217 Cosine: 0.244	-	1.4s
k-means, n_init: 5, tied	0.062	0.066	0.064	0.023	0.057	Euclidean: 0.255 Cosine: 0.358	-	2.4s
k-means, n_init: 5, spherical	0.056	0.057	0.057	0.017	0.051	Euclidean: 0.336 Cosine: 0.413	-	1.3s
k-means, n_init: 5, diag, features:40	0.073	0.078	0.075	0.019	0.068	Euclidean: 0.077 Cosine: 0.119	-	1.5s
{max_iter: 100, max_features: 10}								

Since GMM is also using expectation maximization under the hood, it does have a tendency to converge to a local optimum. To address this issue, I tried running different values for n_init just in the binary case. Since overall homogeneity and completeness improved without a significant drag on other metrics, I decided to default to n_init of 5 for other experiments. An important parameter that goes into GMM effectiveness is also the number of components to use. AIC and BIC are good measures for this – they help determine overfitting, and the goal is to minimize those metrics. In the below experiment, I limited

the number of features to 200 (so as to give us adequate room to determine the appropriate n_component), and then plotted AIC and BIC for both the binary dataset. Here, it looks like there is a point at around 7 components where GMM starts to overfit.



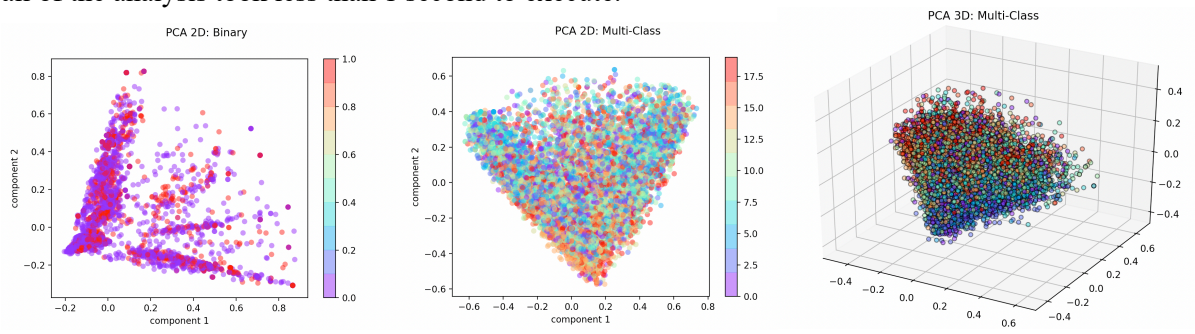
Summary of initial clustering algorithms

The main take-away from the above results are that the underlying data – which is highly overlapping with high dimensionality – is too complex and too noisy for clustering algorithms to effectively do unsupervised learning. In the upcoming section, I focus on trying to reduce the complexity and noise inherent in the underlying data.

PCA

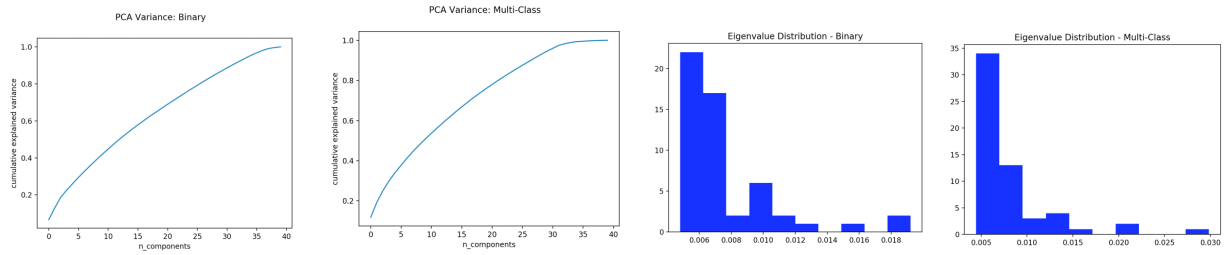
PCA is effective in reducing the dimensionality of a dataset and can also help with reducing noise in the dataset. Given that our current dataset of text vectors does contain a lot of noise in the form of overlapping terms that don't have a significant impact on clustering, PCA should help a lot in normalizing the dataset. Essentially, the information along the least important axes are removed, leaving only the components with the highest variance.

To help visualize how this might help, I reduced the dimension of all feature vectors to just 2D, so that they could be plotted. You can see how the binary case actually reduces nicely – showing what looks like four distinct clusters. However, on closer inspection, you actually notice that the four clusters do not reflect the clusters we want to measure – hence why our homogeneity measures are so poor in prior experiments. On the other hand, the two-dimensional representation of the multi-class dataset is less useful. Even by using our eye sight, it looks like all of the different classes get clumped together. To help visualize this further, I tried plotting a 3D representation of the multi-class dataset. Even then, too much is lost, and you still cannot identify clusters. The thing to note is that PCA actually performs very quickly – all of the analysis took less than 1 second to execute.



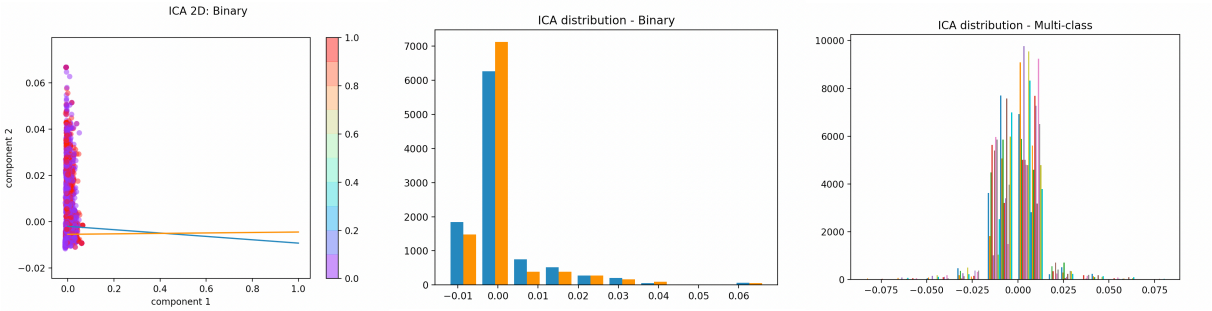
Next, I wanted to come up with a more mathematical measure of what an appropriate value would be for the number of components / dimensions to use to preserve the most variance. To do this, I plotted the cumulative explained variance across a range of components. Interestingly, the binary classification dataset actually converges more slowly – this seems to match with our earlier k-means clustering results, since performance was horrible with n_components=2. Further, when we observe the graphs above with smaller dimensions, no matter how we group things, homogeneity is poor. For the case of multi-class,

n_components that are too low to not capture the necessary variance in the dataset, so larger n_components are required. In fact, the PCA function with scikit learn actually calculates the recommended n_components value given the amount of noise you want in your data and the sample size. If we specify that we only want to keep values that account for variance higher than 50%, then the recommended n_components for multi-class is 53 while the recommended n_components for binary classification is 58. I also wanted to get a sense for what the eigenvalue distribution was after PCA was run. This essentially gives a sense of how each component might impact the overall clustering results. Interestingly, you can see that PCA has found a couple outliers both for binary and multi-class datasets.

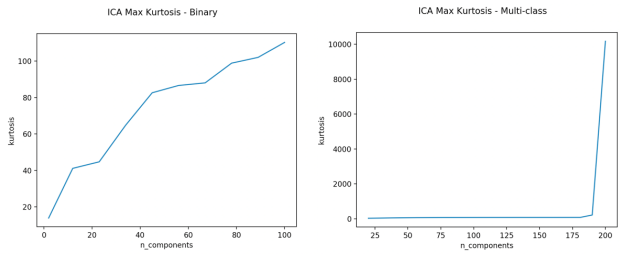


ICA

ICA tries to find features that map to non-Gaussian distributions. Essentially, it attempts to eliminate random noise. Unlike PCA, it is not used for dimension reduction. The ICA implementation is actually very fast, taking less than 10ms for each run. First, I ran a binary implementation with ICA with n_components = 2. The results are displayed below. I plotted both the resulting vector from ICA, as well as a flattened two dimensional representation of the data set using ICA. The vectors do not clearly represent the dataset underneath. However, the ICA distribution has a kurtosis level of 10.8 for the first vector, and 13.9 for the second, indicating that it indeed is non-Gaussian. This can be further seen when observing the distribution curve. For the multi-class problem, kurtosis ranges from 0 to 22 depending on the component. In later analyses – I actually try removing any datapoints that represent CLT.



A likely weakness of ICA is that it assumes statistical independence. Since we already know that the underlying dataset is heavily overlapping, it's possible that ICA will now perform well. Already, you can see that the ICA distribution of two component vectors look really similar to each other.



I also tried seeing how kurtosis was correlated with the number of components. Based on the results show in the graphs, it does look like increasing the number of components can improve results.

RP

The random projection attempts to efficiently reduce the dimensionality of data without sacrificing too much variance. In this case, I set the n_components to the number of categories I wanted to classify. I then ran it multiple times, and found the mean of the resultant components generated to get a sense for how much variation occurred. In the binary case, the means I got were [-0.0488 0.045 -0.1396 -0.0260 -0.0138]. In the multi-class case, the means I got were [0.0062 0.0310 -0.0142 0.0059 0.0166]. As you can see, there is quite a bit of randomness in the results of random projection.

K-means clustering with Dimensionality and Feature Selection

The next step I wanted to try was to use dimensionality to try running classification again. I also included LDA, which is a common feature selection method for text analysis. For both the binary and multi-class dataset, I used randomized initialization of k-means, since that had worked better in earlier tests. PCA with 2 components actually achieved good accuracy, although the clustering metrics continued to be poor. The interesting thing to note here is that 58 components was suggested by the PCA algorithm. In this case, this is an example of where domain knowledge actually trumps the actual algorithm itself. I think the main issue is that again, the text data overlaps a lot, so homogeneity and completeness are hard to achieve when we have not yet identified the characteristics that are most important to cluster membership. Surprisingly, LDA performed worse than chance – likely because LDA suffers from unbalanced datasets – a key characteristics of our underlying binary dataset. ICA does not perform well for the binary dataset, and this is not surprising given that ICA kurtosis for the binary dataset was very small, with no clear components that were more important than the others.

Moving onto the multi-class dataset, AMI increased, but V-measure and other clustering metrics decreased. This ultimately showed that the results were in-conclusive. This is not surprising since PCA, ICA, and LDA all involve either removing dimensions, reducing noise, or overlaying a probability estimate, all of which remove important data that impacts clustering. Interestingly, LDA performed the worst, despite being commonly used in text analysis. One possibility is that tf_idf with stop_words already helped improve metrics, so including LDA no longer made as big of an impact. Final note, for both datasets, performance time is markedly improved firstly because dimension reduction algorithms require dense matrices so I limited max_features to 200, and also because dimension reduction reduces the amount of data clustering algorithms need to process.

	Homogeneity	Completeness	V-measure	ARI	Silhouette	Accuracy / AMI	Time
Binary {n_clusters: 2}							
PCA (58 components)	0.000	0.000	0.000	-0.000	Euclidean: 0.137 Cosine: 0.096	0.407	0.7s
PCA (2 components)	0.000	0.005	0.001	-0.001	Euclidean: 0.908 Cosine: 0.219	0.615	0.15s
ICA (2 components)	0.000	0.000	0.000	0.004	Euclidean: 0.657 Cosine: 0.726	0.426	0.07s
ICA – removed kurtosis	0.000	0.000	-0.000	-0.000	Euclidean: 0.672 Cosine: 0.531	0.406	0.09s
LDA (2 topics)	0.008	0.014	0.010	0.030	Euclidean: 0.805 Cosine: 0.925	0.380	0.09s
Multi-class {n_clusters: 20}							
PCA (53 components)	0.169	0.183	0.176	0.057	Euclidean: 0.091 Cosine: 0.162	0.165	2.1s
ICA (20 components)	0.190	0.201	0.196	0.076	Euclidean: 0.168 Cosine: 0.254	0.186	1.7s
ICA – removed kurtosis	0.157	0.164	0.161	0.062	Euclidean: 0.170 Cosine: 0.255	0.152	1.4s

LDA (20 topics)	0.057	0.062	0.060	0.023	Euclidean: 0.266 Cosine: 0.234	0.052	0.8s
{init: random, max_iter: 300, n_init: 10}							

Expectation maximization with Dimensionality and Feature Selection

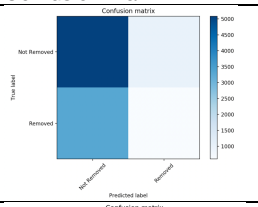
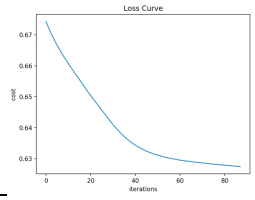
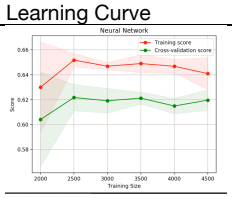
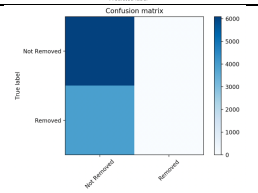
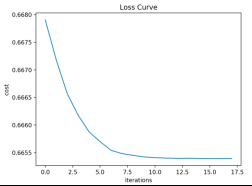
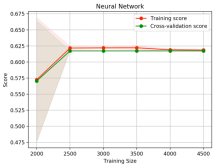
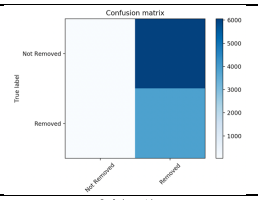
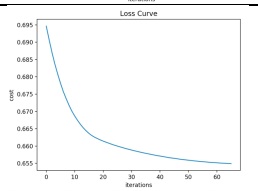
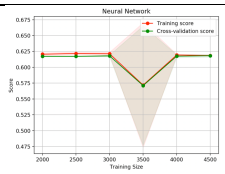
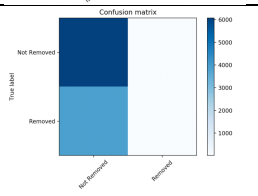
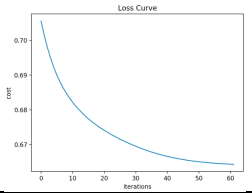
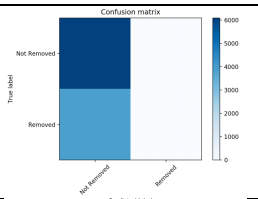
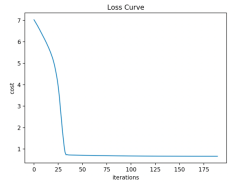
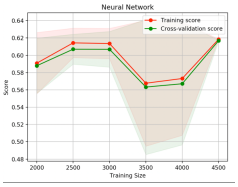
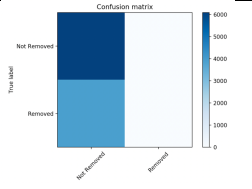
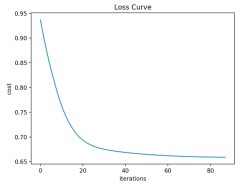
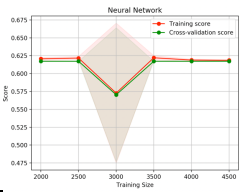
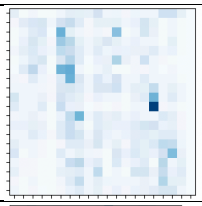
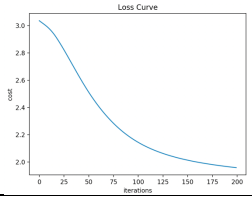
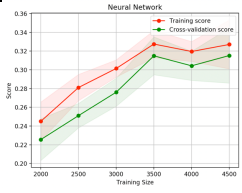
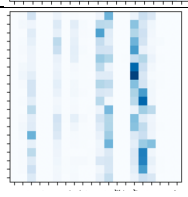
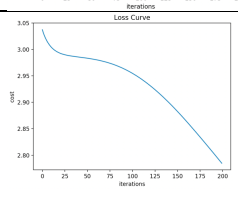
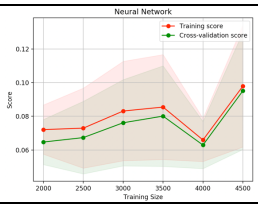
For k-means clustering, I used the settings that worked well before – spherical covariance for multi-class, diag covariance for binary. Interestingly, PCA with 58 components outperformed PCA with 2 components. This is probably because GMM can handle more complex datasets, so including more components actually allows it to excel. Overall, GMM performs slightly better than k-means clustering, but not by much. In the multi-class case, LDA continues to perform the worst. Neither GMM nor k-means clustering perform well, so the results are inconclusive in terms of which one performs better.

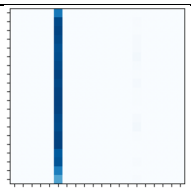
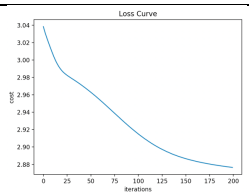
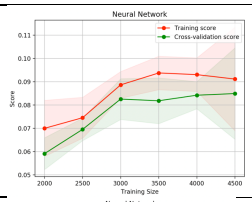
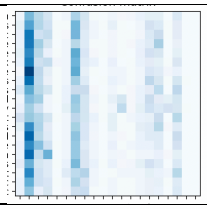
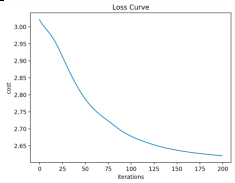
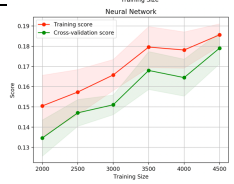
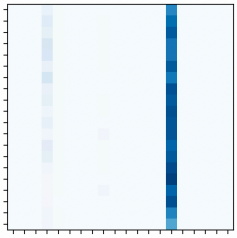
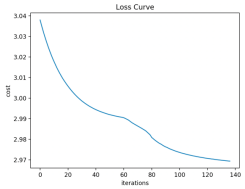
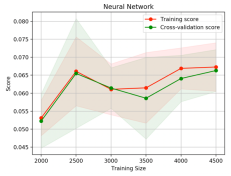
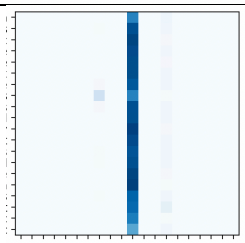
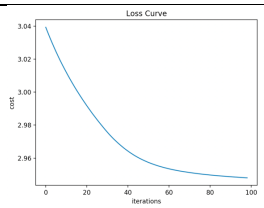
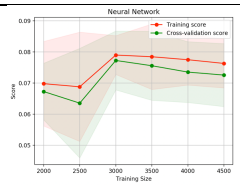
	Homogeneity	Completeness	V-measure	ARI	Silhouette	Accuracy / AMI	Time
Binary {n_clusters: 2}							
PCA (58 components)	0.005	0.009	0.006	0.021	Euclidean: 0.459 Cosine: 0.643	0.616	1.7s
PCA (2 components)	0.003	0.003	0.003	-0.003	Euclidean: 0.908 Cosine: 0.219	0.503	0.2s
ICA (2 components)	0.002	0.002	0.002	-0.005	Euclidean: 0.540 Cosine: 0.753	0.513	0.15s
ICA – removed kurtosis	0.002	0.002	0.002	-0.005	Euclidean: 0.558 Cosine: 0.751	0.513	0.16s
LDA (2 topics)	0.007	0.014	0.009	0.026	Euclidean: 0.780 Cosine: 0.896	0.379	0.07s
Multi-class {n_clusters: 20}							
PCA (53 components)	0.165	0.189	0.176	0.049	Euclidean: 0.098 Cosine: 0.131	0.160	2.0s
ICA (20 components)	0.181	0.196	0.188	0.063	Euclidean: 0.162 Cosine: 0.244	0.177	0.92s
ICA – removed kurtosis	0.180	0.189	0.184	0.069	Euclidean: 0.163 Cosine: 0.241	0.176	1.3s
LDA (20 topics)	0.038	0.044	0.040	0.014	Euclidean: 0.232 Cosine: 0.139	0.032	0.9s
{init: kmeans, max_iter: 100, n_init: 5}							

Neural Network with Dimensionality and Feature Selection

In the below chart, I ran several different analyses comparing performance results for neural networks. I included feature selection, dimension reduction, and clustering algorithms applied to the dataset prior to running the neural network. Most important to note is that by applying these algorithms, it is actually possible to run a neural network. The time to run these algorithms was dramatically reduced. In prior iterations, I had to reduce the dataset size, reduce the number of iterations, and couldn't plot learning curves. I was also able to increase the number of hidden layers to 5.

Overall, applying dimension reduction makes performance worse. None of the confusion matrices were as good as the ones I had achieved using the full dataset. Although the accuracy score across these algorithms is comparable to the ones I achieved before, the confusion matrix is the most telling metric to look at, since it helps us identify false positives and false negatives. Using clustering algorithms to generate dimensions was not effective, and although the accuracy score looks good, the confusion matrix indicates to us that the algorithm only performs well because the underlying dataset is shaped in a way that incorrect guesses result in a surprisingly good accuracy score. This is likely due to unbalanced data.

	Confusion Matrix	Loss Curve	Learning Curve	~ Score	Time
Binary PCA (58 Components)				0.62	0.6s
Binary ICA – removed kurtosis				0.625	0.07s
Binary LDA (2 topics)				0.61	0.2s
Binary RP				0.61	0.4s
Binary k-means				0.58	0.6s
Binary GMM				0.61	0.3s
Multi-class PCA (53 components)				0.32	3.7s
Multi-class ICA				0.1	2.32s

Multi-class LDA (20 topics)				0.09	2.1s
Multi-class RP				0.19	4s
Multi-class k-means				0.065	1.5s
Multi-class GMM				0.79	1.1s