# Introduction

The purpose of this lab was to compare unsupervised clustering methods on the Reuters newspaper article data seen in the last lab. After some experimentation with various clustering algorithms, I chose to compare the hierarchical and K-means algorithms with various parameters. This report will discuss the methodologies and give an analysis of the results for each algorithm.

# Preprocessing Changes

During initial testing I came across the need to tweak some of my preprocessed feature vectors to work with these clustering algorithms. I had condensed the vectors into short arrays of indexes of their keywords in a lookup table, but this led to different sized feature vectors. I fixed the problem by resorting back to sparse frequency matrices with the same lookup table. I also made it so duplicate words were summed in the array instead of only having binary values as was designed before. The two feature vectors I decided on were a vector of just body keywords as filtered out in lab 1, and then I created a new vector which appended the topic keyword feature vector to the body keyword vector. I had originally tried to make a matrix of the non-reduced corpus (minus stopwords), but the vectors were over 100,000 entries long and the curse of dimensionality led to terrible performance in every category. I would have compared just the topic keyword and body keyword feature vectors, but the topic keyword vectors were all zeros more often than not, so it made the most sense to concatenate them for one feature vector.

# Clustering

After completion of preprocessing, the two feature vectors were ran through the different clustering algorithms with various metrics. Before starting the clustering however, there was a decision to be made about how to handle clusters entries with multiple topic labels. I decided that it was fair to simply assign a cluster containing multiple topic labels to the most abundant label in that cluster. For example, the cluster [ [‘earn’], [‘earn’], [‘earn’, ‘acq’, ’grain’] ] would flatten to [‘earn’, ‘earn’, earn’]. Similarly, when checking for a successfully predicted label, I simply look in the ground truth label to see if the predicted label is there. So if the ground truth label is [‘earn, ‘acq’, ‘money-fx’] and the predicted label is [‘acq’], I count that as a successfully predicted label. When comparing clustering techniques I use a few statistics which need explanation. These can be seen below.

Average Entropy

To gauge the homogeneity of a cluster, I implemented the normalized entropy function. You can see the code for this in utilities.measureEntropy( ). The *average entropy* measure is the average of all cluster entropies with cluster size greater than 1.

Accuracy

In order to measure the prediction accuracy of the resulting clusters, I had to make a decision on how to label a cluster. I decided the most obvious way was to simply assign the most frequently occurring topic in the cluster as the label. For example, if the cluster contained [‘earn’, ‘earn’, ‘money-fx’, ‘interest’, ‘earn’] it would assign the predicted label of all of the entries inside the cluster to be ‘earn’. In this manner I could compare each member of the cluster to the ground truth label, and get the total number correctly classified divided by the total number of samples to get the overall prediction accuracy.

Skew

The skew represents the standard deviation of cluster lengths returned by the algorithm.

Time

The time represents the number of seconds it took to completely run through the clustering algorithm.

**K-Means clustering**

For the K-means clustering, I went online and found an implementation which allowed me to change the distance metric to one of 20 provided by the scipy.spatial.distance module. In order to properly find the most effective combination of means and metric usage, I first did some initial screening. I tried the Minkowski, dice, Jaccard, Euclidean and cosine distance metrics, and after quick testing I eliminated dice and Jaccard for their inaccuracy compared to the others, and then eliminated Minkowski as well because it was significantly slower than cosine and Euclidean with little to no improvement in classification accuracy. You can see the results of the Euclidean and cosine distance metrics with various amounts of means in figure 1 and figure 2 below. Figure 1 represents just the feature vector with only key body words, while figure 2 used the feature vector with both body and titular words.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metric | Means | Time (seconds) | Average Entropy | Skew | Accuracy |
| Euclidean | 90 | 7.777 | 0.215 | 294.710 | 0.640 |
| Euclidean | 105 | 9.708 | 0.232 | 238.451 | 0.643 |
| Euclidean | 120 | 10.525 | 0224 | 217.886 | 0.655 |
| Euclidean | 135 | 12.447 | 0.2035 | 206.719 | 0.661 |
| Cosine | 90 | 0.920 | 0.236 | 130.343 | 0.724 |
| Cosine | 105 | 0.998 | 0.242 | 111.148 | 0.732 |
| Cosine | 120 | 1.065 | 0.237 | 102.022 | 0.732 |
| Cosine | 135 | 1.226 | 0.231 | 98.855 | 0.737 |

**Figure 1**: K-means clustering results for the feature vector with topic and body keywords.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Metric | Means | Time (seconds) | Average Entropy | Skew | Accuracy |
| Euclidean | 90 | 9.543 | 0.228 | 309.274 | 0.661 |
| Euclidean | 105 | 9.958 | 0.203 | 289.970 | 0.662 |
| Euclidean | 120 | 12.185 | 0.218 | 222.571 | 0.673 |
| Euclidean | 135 | 12.381 | 0.213 | 198.373 | 0.676 |
| Cosine | 90 | 0.971 | 0.244 | 102.706 | 0.739 |
| Cosine | 105 | 1.044 | 0.249 | 91.991 | 0.744 |
| Cosine | 120 | 1.222 | 0.245 | 86.400 | 0.746 |
| Cosine | 135 | 1.262 | 0.255 | 74.375 | 0.745 |

**Figure 2**: K-means clustering results for the feature vector with topic and body keywords.

As you can see from the results, adding the title words to the body vector made very little change on the statistics when using the K-means algorithm. I found that the optimal number of clusters for K-means was always right around the number of total unique topics classes (118). Using the cosine similarity metric I was able to achieve the correct clustering prediction of about 75% of the data. The scalability of K-means was very reasonable, especially using cosine similarity, which finished clustering the entire dataset in around 1 second. Cosine similarity gave a much more homogenous clustering than Euclidean, with a standard deviation in cluster length of less than half that of the Euclidean.

**Hierarchical clustering**

I found a flexible hierarchical clustering method in the sklearn.cluster module under the name AgglomerativeClustering. The algorithm works in a ‘bottom up’ fashion, starting with the given number of leaf nodes and eventually merging to one cluster. The method allows for switching distance metrics between Manhattan, cosine and Euclidean, and changing the linkage to average, complete, or ward. After some initial testing I found the Manhatten distance to give poor results compared to Euclidean and cosine distances, and I found the difference between Euclidean and cosine to be negligible, so instead stuck with the default of Euclidean and focused on differing linkage. ‘Complete’ linkage represents minimizing the maximum distance between all members of one cluster and a subsequent one. This resulted in relatively bad clustering results, with very disproportionate cluster sizes and an underwhelming classification accuracy. ‘Ward’ linkage merges based on minimizing the variances between two clusters. This resulted in much more homogenous cluster sizes and significantly higher accuracy. The exact numbers for the two feature vectors can be seen in figures 3 and 4 below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Linkage | Leaf Nodes | Time (seconds) | Average Entropy | Skew | Accuracy |
| Ward | 105 | 284.59 | 0.237 | 259.486 | 0.689 |
| Ward | 120 | 285.91 | 0.233 | 187.932 | 0.693 |
| Ward | 135 | 285.71 | 0.236 | 178.076 | 0.699 |
| Complete | 105 | 288.93 | 0.279 | 946.302 | 0.403 |
| Complete | 120 | 288.84 | 0.28 | 885.365 | 0.417 |
| Complete | 135 | 288.26 | 0.246 | 832.593 | 0.423 |

**Figure 3**: Hierarchical clustering results for the feature vector containing only key body words.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Linkage | Leaf Nodes | Time (seconds) | Average Entropy | Skew | Accuracy |
| Ward | 105 | 292.53 | 0.239 | 259.486 | 0.694 |
| Ward | 120 | 284.60 | 0.234 | 187.932 | 0.693 |
| Ward | 135 | 286.42 | 0.233 | 178.076 | 0.701 |
| Complete | 105 | 296.44 | 0.282 | 946.302 | 0.421 |
| Complete | 120 | 295.55 | 0.272 | 885.365 | 0.423 |
| Complete | 135 | 295.50 | 0.251 | 832.593 | 0.424 |

**Figure 4**: Hierarchical clustering results for the feature vector containing only key body words

The results show that complete linkage was in all aspects, worse than ward (minimizing variance) linkage. The best clustering without what I can only assume would be over fitting was at around 135 leaf nodes and resulted in a classification accuracy of about 70%. Hierarchical clustering took about 5 minutes to run, which was magnitudes higher than in K-means for no increase in classification accuracy. The scalability of hierarchical clustering seems questionable, and I would not expect it to be practical for use in very large datasets.

# Conclusion

The Reuters dataset when converted to feature vectors as designed in lab 1, were best clustered by using the K-means method with the cosine similarity metric and 120 means. The average entropy of clusters for the two methods was similar. In general, K-means was significantly faster than hierarchical clustering, and resulted in slightly better label prediction accuracy.