Final Project Description

**Preprocessing Data**

After downloading the abstracts to a folder, I used glob to get a list of files that ended in txt. For each file in the list, I read the lines after “abstract:”. Since we did not need to read any other features besides the abstract, I ignore them. After I created a list of all the abstracts, then remove some of the abstracts since they did not have an abstract. Each abstract is then stripped of all newline and tab characters. This allows the abstract to be represented as one long string. Next, I removed all character at are not letters or numbers and set the string to lower case. Then, I removed the stop words from the abstracts. Stop words are frequency used words therefore they should improve the clustering if they were removed. Next, I used NLTK’s PorterStemmer to stem each word in the abstract. Stemming helps removes morphological affixes from words, leaving only the word stem. After, I also used NLTK’s WordNetLemmatizer. Lemmatizing gets the base for each word in the abstract. After cleaning all the abstracts, I utility Gensim’s Doc2Vec class to convert each abstract into a fix vector length. I converted each abstract to a length of 50 vector. Also, I used Gensim’s Word2Vec to compare to Doc2Vec. For each word in an abstract, I added the vector value and divide by the total number of words in the abstract to get a fixed vector size for an abstract.

**K-Means**

My K-Means implementation follow the normal formula from the slides. First the algorithm creates k random vectors to become a centroid. Then for each abstract we get the cosine similarity of all the centroids. I assign the abstract to the centroid that has the highest similarity. After assigning each abstract to a vector, we move the centroid to the center of the cluster. In order to do this, I sum all the vectors in the cluster then divide by the number of abstracts with in a cluster. I then repeat these steps until the centroids do not change. The only thing different I include is that I include a max iteration, which stop the k-means after a max amount of iterations.

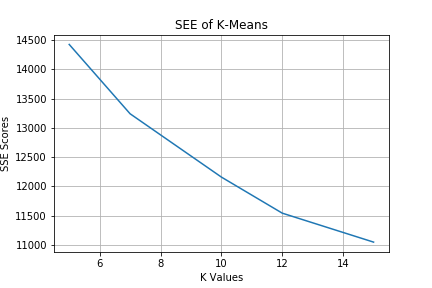
**DBSCAN**

DBSCAN is a cluster based on density with parameters of min points and eps. The way I implemented DBSCAN was at the start I create a label list with length of the number of abstract and set all values to zero. This list will hold the assign of each point in the list. For each abstract, I first check if an abstract is not been claim by a cluster but checking the index of label list. If not, then we find all neighboring points of to that abstract. In order to get the neighboring points, we calculate absolute value of the cosine distance between the current abstract and all other abstracts. If the number of neighborhood points is less than the number of neighborhood points, then that point is the noise point. If the point as more the minimum number of points, then I mark the point for that cluster, and I expand the cluster. When I expand the cluster, I take the neighborhood abstract and check if the neighborhood points are enough for a cluster, and then add the points to the neighborhood abstract. We mark all the neighborhood point with the cluster number. I repeat this until all abstract are list as noise point or in a cluster.

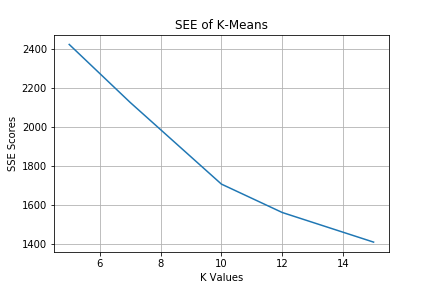
**Results**

**K-Means**

Doc2Vec



Word2Vec



For K-Means, Word2Vec seem to outperform Doc2Vec in SSE score. From K equals 5 to 10 the SSE score were decreasing sharper than from 10 to 12. The biggest drop in score is when K is 10. Doc2Vec’s SSE scores are significantly higher than Word2Vec, but I think this is because I only used vector size of 50 and trained on 100 epochs.

**DBSCAN**

For DBSCAN, I found that there was a heuristic for choosing the minimum number of points, which was to take the natural log of the number of points. Therefore, I took ln(49046), which is about 10.8. I set the min point to 10. My code took a while to run, so I used scikit to find the best eps and run it on my code once to verify the SSE score. With 10 as the Min Point and .13 as the eps, the sse was 14909.88.

**Comparison**

The SSE scores is lower on K-Means and DBSCAN clustered most of the points on one cluster. Around 40,000 abstracts were clustered in one cluster, which will cause the SEE to be higher.