1 Design Document

1.1 cluster_class

First a cluster_class.py was declared, in that file three functions are been defined: Euclidean distance, cosine similarity and dissimilarity.

Next up is the declaration of generic Example class, with attributes such as name, features and labels; with methods like getters and setters, and a distance method to compute how far apart two example is.

Cluster class defines a cluster's behavior, with attributes like centroid and examples in it; methods such as update cluster, compute centroid and the variability among the inner cluster.

1.2 main.py

1.2.1 Doc

Doc is the implementation class of class Example defined in the cluster_class.py, no other member variable was declared.

1.2.2 Doc_of_words

Doc_of_words takes the raw data file as input and outputs a list of lists with every entry mapped with the document accordingly.

1.2.3 buildSVD

buildSVD takes the output from method Doc_of_words and outputs a list of examples(Doc).

1.2.4 kmeans

Defines the basic Kmeans method.

1.2.5 bisecting_kmeans

Defines the bisecting Kmeans.

1.2.6 write_file

Writes result of csv into the subfolder named "output".

Your approach (pseudocode for Bisecting k-Means).

1.3 k-means

```
INPUT = a list of data points
OUTPUT = a list of clusters

Procedure kmeans(examples, k=2, verbose = False, maxIter = 100):
    initialCentroids = random-choose-k-from-examples
    clusters = []
    for e in initialCentroids:
        clusters append cluster[e]
    converged = False
    numIterations = 0
    while not converged and numIterations < maxIter:
        numIterations += 1
        newClusters = []
        for i from 0 to k:
            newClusters append empty cluster
        for e in examples:
            smallestDistance = distance of e to centroid of first cluster
            index = 0
            for i from 1 to k:
                distance = distance e to centroid of clusters[i]
                if distance < smallestDistance:
                    smallestDistance = distance
                    index = i
            newClusters[index].append(e)
        if clusters don't vary:
            end loop
    return clusters
```

1.4 bisectingb k-means

```
INPUT = a list of clusters
OUTPUT = a list of clusters

Procedure bisecting_kmeans(clusters)
    clusterCount = 1
    while clusterCount < 7:
        find the cluster with maximum variability
        divide the cluster with maximum variability into two subClusters
        subClusters = kmeans(cluster with maximum variability)
        clusters += subClusters
        clusterCount += 1
    return clusters
```

Describe, any feature selection/reduction or custom proximity measure you used in this study.

1.5 Word preprocessing

I first turn the sparse matirx of the train.dat into a list of list of words, the feed it into the TfidfVectorizer from sklearn to quantify word feature, then I throw it into the TruncatedSVD to reduce the dimension of each examples.

Implement/Use your choice of internal evaluation metric

1.6 Internal evaluation

For internal evaluation I choose to use cosine similarity between to words, which is defined as the dot product over to vectors' length. Compared with euclidean distance, cosine similarity is not affected by the length of each word vector thus preserves more information.

1.7 Updating centroid

To calculate the centroid of a cluster, I first used euclidean distance and found that it is not very accurate, therefore I used cosine similarity just like how distance is computed. Every feature vector is added up and then divided by the length of it, making it unit length.

1.8 Handling empty clusters

At first it is very easy to cause empty cluster without factoring the input data, however after dimension reduction, it is very hard to yield empty clusters. Nonetheless, I still come up with the code to deal with it in case it happens: loop over the current array of examples and find out with point has the most variability(lowest cosine similarity) and pop if into the empty cluster.