

# Computational Machine Learning Homework4

Yun-shao Sung\*

---

**Abstract.** This document served as the purpose of answering questions from homework assignment, and also conclude the experiment observations.

**1. Kmeans.** In this section, I used the iris data set and run my regular kmeans. There is interesting thing I observed that as the  $k$  (number of clusters) gets large, there is certain possibility that the cluster cannot get points within that cluster. I think this is mostly due to the random process for picking the initial points. As there are too many clusters, or the data set is very similar spreading in limited dimension space, the cluster centroids will be too close and therefore there are situations cluster have no point. For this particular iris dataset and current random seed, cluster-no-point happened when  $k$  gets to 8. Since now I used random seed, the effect of low  $k$  was not able to be observed. However, regular kmeans is randomly initialize the points, low  $k$  may yield a very different distortion results after running, and, therefore, it cannot guarantee you the best representative of centroid with minimum distortion.

**2. mykmeans-multi.** Function mykmeans-multi is inherent from regular kmeans, but will save the distortion and centroids during each iteration. When finishing running, with return the best centroid with lowest distortion. In this way, this function will make sure return the best minimum to represent the data distribution. It's hard to say whether this is more preferable than regular kmeans, since the data set is relative small in only 4-dimension space. However, I think using mykmeans-multi is always better than regular kmeans, such that we won't get the non-optimal result during the run. As making the plot for distortions and centroids trajectory, we can see from the distortions that it improve very fast during the first few iteration, and improvement become flat between iteration 3 to 5, which imply the it is now within a low slope region. Then distortion got a bit more improve and remained all the same.

**3. mykmeans++.** Function mykmeans++ is doing more delicate considerations when initialize the points. Interesting enough, although at the very first point is still based on random, all the remaining points were picked up based on the possibility proportional to the distance to the closest centroid, therefore we can see the initial distortion is higher than mykmeans-multi, but the final result is much better than kmeans-multi. I think this is due to the nature of kmeans++ that trying to pick the points that are far enough, and we can skip the situation that get stuck in local minimum. Furthermore, the cluster-no-point situation will not happen even the  $k=50$ , and the distortion can get even improved from 80( $k=10$ ) to 34( $k=50$ ).

**4. sklearn.cluster.KMeans tricks.** While reading the code in sklearn.cluster.KMeans, I noticed it save many computations when handling the for-loop, mostly by Cython. Therefore, I tried to improve my code when running for loop by either lambda or put it in numpy.array to compute.

---

\*yss265@nyu.edu

**5. Bag-of-words tutorial.** I think the tutorial of Bag-of-words is very key to understand to last part of assignment. In the Bag-of-words tutorial, we imported the movie reviews, and use CountVectorizer to vectorize the counts of every token. Then feed into TfidfTransformer to have the idea of important key words or not that important words. Therefore, this will be the information when train the classifier. Furthermore, I noticed the choice of classifier will also yield noticeable difference. I got the prediction of 77.4% from naive bayes to 82.4% from SVM.

**6. Reading Scikit-Learn.** Since I am using kmeans++ as the base-method finding centroids and its performance and accuracy will very affect out classification result, I have checked the code of MiniBatchKMeans, especially the method finding the initial centroids. In the method called `_init_centroids`, there is trick parameter `init_size`, which is to lower down the number of samples to randomly sample for speeding up the initialization, but this will sometimes at the expense of accuracy. Also this method was also observed in `_k_init` that instead of the whole samples, it used `n_local_trials` (default:  $2+\log(k)$ ) to do sampling. `_k_init` is the method used to get the init centers from kmeans++, and seems there is also a trick that checking whether point space `X` is sparse, and make it to array and get the specific point if `X` is sparse, otherwise just keep as it is to get the point (`X[center_id]`). Also seem there is a faster way called `pairwise.euclidean_distances` that can directly compute the distance matrix between each pair of vectors, and the advantage will be computationally efficient when dealing with sparse data.

**7. Project.** So here I have used the same `learnvocabulary` and `getbof` method that have been used previously and applied to very small set of our music data, so it's easier to my testing purpose and easier to debug. So the small data set is 10 genres, and 8 songs per genre. In notebook `kmeans_music.ipynb`, I used MultinomialNB as my classifier and the fit and training process went well, but got 0 prediction score. Then in `KNeighborsClassifier_music.ipynb` I used KNeighborsClassifier combined with `grid_search` to train the classifier. The best parameter for `n_neighbors`: 6, `weights`: distance, and `algorithm`: auto. Then the prediction score I got is 0.2 which is better than MultinomialNB. The phenomenon is also observed in my notebook `BagWordsTutorial.ipynb` that SVM has better prediction than MultinomialNB. Below are the experiments that I would like to test, but since it is like a back and forth between experiment and code optimization. So although sometime I feel strong interest in some question but then I made the data size become too big, and have to go back to improve code efficiency or play tricks. Therefore, not all the experiment can be done at this point, but here listed my interests that I am trying to do:

0. The effect if I used different classifier. So if I use the same condition, KNeighborsClassifier and Naive Bayes can give me 23% and 7% or accuracy respectively. Please note, here I am only using very few data 2+2 training set and 2 test set, 20 cluster, and only 30 iterations. So I believe as I shift the testing onto hpc running bigger data set, then I should be able to get better results.

1. The effect if I used different window size for the clips. So here I change the window size from 2.48s to 0.44s, and all other condition is just the experiment number0. The the accuracy for KNeighborsClassifier and Naive Bayes dropped to 11.5% and 6% respectively. Although not have done the test whether bigger window will definitely gives a better prediction, but

---

0.44s might just too few and might not be representative enough for the song, and therefore the accuracy dropped.

2. The effect if I reuse the same data for training classifier. This step is using the same condition as experiment number1, and I just reused the same training set that used for finding centroids to train my classifier. Not sure whether it's a occasion thing, but prediction for KNeighborsClassifier actually improved from 11.5% to 18.5%, but Naive Bayes dropped from 6% to 0%. But I guess this might be just occasion thing especially I only using very small window size and low learnvocabulary iteration, so I think we should better not use the data in case over-fitting and prediction drop to 0.

3. The effect transpose the data point. This is actually the experiment that most interested me. I always wondering what should be the point spreading in the space so we can find the centroid to repret the uniqueness of the song. So all the points in the experiment above are each of the time vector over each mfcc, and the length of that vector corresponds to the window size. So the purpose of this experiment is to test if I transpose the point and now the point is the mfcc vector over each of time, and the size of the vector is the number of mfcc. I am crious whether the centroids found by this transformation will be more representative and gives a better prediction. Then the same parameters as experiment number1, the scored were changed from 11.5% to 12.5% for KNeighborsClassifier and significant changed from 7% to 18%. So spreading the point over the dimension of number mfcc might be a better option. However, this transpose will expose the issue that my code is not efficient enough, and maybe I should implement the `n_local_trials` in my code.