CS 686: Data Mining
Project 2: K-Means Clustering

Due in phases, for details see below & Canvas

Phases:
Euclidean distance: Oct. 3
Centroid calculation: Oct. 5
(midterm is Oct. 16 but keep working on the project in this large gap…)
Data normalization: Oct. 17
K-means core algorithm: Oct. 18
Everything finished: Oct. 21

Introduction

In this project, you will implement the K-means clustering algorithm, and apply it to two datasets, the iris data set and the digits data set. I have provided a code framework, in kMeans.py, with the class and method definitions that you should use so that I can more easily grade your programs. This time we will be using the datasets from sklearn instead of the digits from Project 1, and I have provided code to read in the appropriate number of examples. Note that now we have un-named feature vectors instead of the dictionaries you used in the first project (so keys such as '(0,0)' are in a separate part of the data structure provided by sklearn).

What to submit: The main body of your code should stay in kMeans.py that I have provided. If you wish to write auxiliary code in another module (for example to test your code in your own `__main__`, you may do so, but please do not change the main inside kMeans.py so that I may run it as intended.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. Instead, contact the instructor if you are having trouble.

Libraries permitted: In addition to the libraries already provided in the framework code, use only the following imports for this project (you don't have to use all):

```
import math
import numpy as np
import util  # from P1, if desired
```

In particular, do not use `pandas` or `scipy`; you may however add modules from `sklearn`.

1 Q1: Euclidean distance

To compute distance between vectors, we need a metric! We'll use Euclidean distance (in slides from K-nearest neighbors). Fill in the `calc_dist` method to compute this. While there are plenty of libraries to compute this for you, for this question you may only use methods from the `math` Python library; however you may assume that only `numpy` vectors will be passed to this method. You should not make any assumptions about the dimensionality of the input vectors. Review past class demos for reminders about avoiding unnecessary looping.

Question 1 (5 points): Hand in kMeans.py with your fully functional implementation of `calc_dist` filled in. It's okay to have partial implementation for the other questions, I will only grade this part for this due date.

2 Q2: Compute Centroid

Recall the definition of a cluster centroid:

$$\mu(c) = \frac{1}{|c|}\sum_{\mathbf{x}\in c}\mathbf{x}$$

We will need this during our iterative optimization of the k clusters (to implement in question 4). Fill in the `centroid` method to compute this.

Question 2 (10 points): Hand in kMeans.py with your fully functional implementation of `centroid` filled in. It's okay to have partial implementation for the remaining questions, I will only grade this part for this due date.

3 Q3: Normalize the input

Recall the discussion of normalizing your input. The `DM_supervisedSumm` python notebook has an example and `http://scikit-learn.org/stable/modules/preprocessing.html` has more discussion. Use the `StandardScaler` object of the `preprocessing` module of `sklearn` to normalize your training data. Save the object in `self.scaler` for use at testing time, as discussed below in Q5.

Question 3 (5 points): Hand in kMeans.py with your fully functional implementation of `normalize` filled in. It's okay to have partial implementation for the remaining questions, I will only grade this part for this due date.

4 Q4: Iterative K-means clustering

Time to write the core of the clustering algorithm. Fill in the `fit` method of kMeans.py (Note we are using sklearn terminology for creating models instead of the terminology used in P1; this is because you will be comparing your k-means to that of sklearn!). Below is the pseudocode, which I will elaborate on below.

```
Initialize clusters.
Until clustering converges or other stopping criterion:
    For each cluster $\mathbf{c}_j$ //update the centroid of each cluster
        $\mu(\mathbf{c}_j) = \text{centroid}(\mathbf{c}_j)$
    For each instance $x_i$:
        Assign $\mathbf{x}_i$ to the cluster $\mathbf{c}_j$ such that calc_dist($\mathbf{x}_i, \mu(\mathbf{c}_j)$) is min.
```

You may initialize your clusters by randomly assigning each data point to one of the k clusters. **Important:** if this results in an assignment for which one or more clusters are empty, you should move a randomly chosen point to each such cluster.

Stopping criterion: You should continue to iterate until the assignment of points to cluster no longer changes, or until `max_iter` iterations, whichever comes first.

Question 4 (30 points): Hand in kMeans.py with your fully functional implementation of `fit` filled in. It's okay to have partial implementation for the remaining questions, I will only grade this part for this due date.

5 Q5: Predict

Of course, you will want to be able to assign (seen or unseen) data points to your learned clusters.

Question 5 (10 points): Fill in the `predict` method of kMeans.py for this question. Hints: Remember that you have normalized your data during training. Since you don't have your test data at training time, you can't normalize it until testing time. This is where you will use the `self.scaler` that you saved at training time in the `normalize` method. The rest is pretty easy since you already needed to assign points to clusters for the `fit` method! All you are doing now is handling possibly new points and returning the list of cluster assignments.

Now you should be able to run the `main` method I provided to you. Note I have now put just the main into kMeans_frame.py, which fixes some problems with the main in the original kMeans.py. Here is an example of running the program:

```
$ python kMeans_frame.py
Training...
Testing...
Score for your partitioning with 150 training examples and 150 test examples: 0.592
```

6 Q6: Compare to sklearn

You now have a fully functioning kMeans.py. But how well does it compare to the sklearn version? There are some clever tricks that are provided in sklearn.cluster.KMeans, I'm not asking you to implement them, but let's see what happens if we put the two systems on more of a level playing ground. But how will we compare them?

We're playing with clustering in an artificial environment, in which we know the "true" cluster labels according to the Y (or target) values. Notice that I've provided an 'eval_clustering' method. You can read about the sklearn method that it calls, but in short it checks if your cluster's labels align well with the Y labels.

Your final task is to implement the `compare_sklearn` method of kMeans.py. This method should take a training set, and a test set, and learn a k-means clustering with the sklearn version of KMeans (check kMeansDemo.py on Canvas for hints), using the training set. **NOTE:** you should also scale the data given to sklearn, otherwise, obviously, your algorithm would do better. If you first run this without scaling, I'm guessing your version would win, otherwise, it might depend on the luck of the draw, in terms of initial assignment of points to clusters.

Your `compare_sklearn` should also create the appropriate number of clusters, based on the k value passed in. You should then apply 'eval_clustering' to the test set. Return the accuracy. If you want a more fair comparison to your kMeans, make the 'init' option of sklearn.cluster.KMeans random.

Question 6 (8 points): Fill in the `compare_sklearn` method of kMeans.py for this question, as described above.

Here is an example of running the program to compare the algorithms:

```
$ python kMeans_frame.py -s
Training...
Testing...
Score for your partitioning with 150 training examples and 150 test examples: 0.620
Score for sklearn's partitioning with 150 training examples and 150 test examples: 0.730
You lose!

$ python kMeans_frame.py -s -t 100
Training...
Testing...
Score for your partitioning with 100 training examples and 50 test examples: 0.447
Score for sklearn's partitioning with 100 training examples and 50 test examples: 0.619
You lose!
```

7 Extra Credit: $W(C)$

For this optional question, you will implement the within-cluster variation metric, $\sum_{i=1}^{k} W(C_i)$, defined in class. Put this in a new method `W(clustering)`, where the input is your chosen cluster representation, and the output is the total $W(C)$ over all clusters input. Your fit method should call this and print out the result. You can choose to only print it out every 5 or 10 iterations.

Extra credit Question (Up to 6 points): Implement `W(clustering)` as described above. Now explore the behavior of your k-means for one of the two datasets (tell me which one you chose).

Turn in a plot of the values of $W(C)$ for your chosen dataset.

For full extra credit, answer the questions below in a plain text file that you submit with your assignment.

1. Did your algorithm converge to a local minima, or did you have to increase 'max_iter' in order for it to converge?
2. How many iterations does your clustering take to converge?
3. Did the rate of change of $W(C)$ seem to change as the algorithm got closer to converging, based on examination of your plot?