

# Problem Set 10: Clustering

**Handed out:** Lecture 20.

**Due:** 11:59pm, Lecture 22.

## Introduction

For this problem set, you will be using k-means clustering to process and analyze data.

You will have to produce only a small amount of code, but you should be sure to understand the code that has been given to you.

## Collaboration

You may work with other students. However, each student should write up and hand in his or her assignment separately. *Be sure to indicate with whom you have worked.*

## Getting Started

Download the files in the zip folder and save them in a single directory: ps10.zip

Submit all your answers to the written questions in a single file called ps10\_writeup.pdf.

## Background Overview

Every decade, the United States Census takes place. The census collects a lot of information from the population around the United States. In this problem set, we are going to use k-means clustering to analyze this information.

**It is essential in this problem set to not try running everything at once. The data set is very large, and some functions will take a long time to run. You should test out each element of the code you are asked to write and/or understand. First, test your code on a sample data set and see if you get the expected results. Once you are comfortable with the outcome, you should produce the graphs for your writeup using the full data set. Running this code may take a while!**

To help you, we've setup the starter code to load two global sets, "points", which contains all the data in counties.txt, and "testPoints", which has only a tenth of the full data (this should be small enough for most of your tests to run fairly quickly).

## Problem 0: Provided classes and functions

In ps10.py, we've provided the following classes that you will use in this problem set:

- **Point Class:** represents the the data points
- **County Class:** represents the counties and contains a method that returns the distance between two clusters. Also extends the Point class.
- **Cluster Class:** represents the clusters

In addition to the above, we've provided the following functions that you will use in this problem set:

- **kmeans():** produces a list of clusters and the maximum distance of any point in a cluster to its centroid
- **readCountyData():** takes a filename and number of features to import. Returns the names, list of data points, and the maximum values
- **buildCountyPoints():** uses readCountyData to create County instances given each county from readCountyData
- **randomPartition():** partitions a list into two lists

Finally, there is one sample function to get you started:

- **test():** takes in a set of data points, uses kmeans to cluster them, and gives a histogram of some average values.

**Read ps10.py carefully before starting, so that you understand the provided code and its capabilities.**

## Problem #1: Graphing Removed Error

In this question, you will need to graph the total error produced by the `kmeans(...)` function.

In order to do this, you must do the following:

1. Iterate over  $k$  in increments of 25 from  $25 \leq k \leq 150$  and for each  $k$  do the following:
  1. Partition your data set into a training and holdout set, where the holdout set should be 20% of all the points.
  2. Using `kmeans(...)`, find the total error of the training set, where the total error is the following equation:  $\sum (\text{distance of point to the centroid of its encapsulating cluster})^2$  where the sum is over all points in the training set. Hint: use the `Point.distance(...)` method.
  3. Given the holdout set, find the error by calculating the squared distance of each point in the holdout set to its nearest cluster.
2. Once you have these error values:
  1. Graph the error in the training set and the error in the holdout set against  $k$  (you can graph these two lines on the same plot).
  2. Also graph the *ratio* of the error of the holdout set over the total error of the training set. It will make sense to do this in a separate plot.
  3. (Hint: It will be much faster if you save all of the error values first and then do your graphing afterwards rather than recomputing the clusters each time.)

```
def graphRemovedErr(points, kval = [25,50,75,100,125,150], cutoff = 0.1):
    """
    Should produce graphs of the error in training and holdout
    point sets for ratio of the error of the points

    kval: int list
    cutoff: float
    """
    # Your Code Here
```

Once you have things working, find the line that says

```
random.seed(123)
```

and comment it out. This line has been making the randomization consistent, which is nice for debugging but doesn't enable us to produce truly randomized results. Next, graph the full dataset, and include in your writeup ([ps10writeup.pdf](#)) your graph, along with a short sentence describing what the results mean. Note any trends or surprising features of the graph. (This need not be a long response).

You will likely observe that the two sets (training and holdout) change error in the same direction, but at different rates. Why do you think that is?

## Problem #2: k-means and you

In this problem, you are going to look at the specific cluster that your home county ends up in (or a county of your choosing if you are an international student.)

Run k-means three times with  $k = 50$ . For each run, note what cluster your county appears in, and answer the following questions in [ps10writeup.pdf](#):

1. Does your county cluster with the same counties each time? If not, why not?
2. Is your county in an appropriate cluster (i.e. a cluster with similar other counties)? Why or why not?

## Problem #3: Graphing Prediction Error

In this question, you are going to cluster a training set and examine the error if the clusters were used to classify the holdout set.

For this question, you will need to modify `County.weights`, which is an array that affects how important each feature is in the clustering process. Look at how it is defined and used in [ps10.py](#) to make sure you understand how to modify it. The default is for all weights to be 1.0 — this is so that every feature has an equal effect on the clustering. For this problem, all you need to know is if you set a particular weight to 0, that feature will be ignored when clustering the points. (Take a look at the input data file to see which dimension in the weights array corresponds to which feature.)

For this problem, you must do the following:

1. Set the weight of the Poverty feature to 0 and the weight of all other features to 1 in the weights array of the County class (we will examine the effect of changes to this weight vector later.)
2. Partition your data set into a training set and a holdout set, where the holdout set is 20% of all the data and is randomly chosen.
3. Using `kmeans(...)`, cluster the training data into 25-150 clusters as in Problem 1.
4. For each point  $p$  in the holdout set, find the cluster  $c$  that is closest to  $p$ . Find the average poverty of  $c$ . Graph the squared difference of  $p$ 's poverty subtracted from  $c$ 's average poverty as  $k$  changes. **Use the actual poverty value, not the normalized poverty value.**

```
def graphPredictionErr(points, dimension, kval = [25, 50, 75, 100, 125, 150], cutoff = 0.1):  
    """  
    Given input points and a dimension to predict, should cluster on  
    the appropriate values of k and graph the error in the resulting  
    predictions, as described in Problem 3.  
    """  
    # Your Code Here
```

Include the graph in `ps10writeup.pdf`, along with a *short* analysis (no more than 1 paragraph) that explains why the error is high or low and why it varies depending on the value of  $k$ .

#### **Problem #4: Graphing Prediction Error (AGAIN!)**

Now you are going to examine what happens if we change the values of the weights in the County class. In the previous problem, you set the Poverty feature to 0 to make sure it didn't interfere with your predictions (otherwise we would be "predicting" the poverty by choosing values that we already knew were close to the correct answer, which is cheating). Now you will leave the Poverty weight 0, but try changing some of the other weights to see if we can still get better predictions.

1. Find a weights vector (while keeping the Poverty weight set to 0) that produces a graph with significantly lower error than Problem 3. Use weights of either 0 or 1. Put this graph in `ps10writeup.pdf` and be sure to include the weight vector you found. Also include a sentence or two as to why you believe your weight vector produces this result.
2. Find a weights vector (while keeping the Poverty weight set to 0) that produces a graph with lower error than the above part. This time you should use values between 0 and 1. Put this graph in `ps10writeup.pdf` and be sure to include the weight vector you found. Also provide a sentence or two as to why you believe your weight vector produces this result.

# Hand-In Procedure

## 1. Save

Save your code in two files, named `ps10.py` and `ps10writeup.pdf`.

## 2. Test

Run your file to make sure it has no syntax errors or typos! Make sure the code you submit can generate all the graphs we've asked you to make

## 3. Time and Collaboration Info

At the start of your file, in a comment, write down the number of hours (roughly) you spent on the problems, and the names of the people you collaborated with. For example:

```
# Problem Set 10
# Name: Jane Lee
# Collaborators (Worked with): John Doe
# Collaborators (Exact solution): Ben Bitdiddle
# Time: 3:30
#
... your code goes here ...
```

MIT OpenCourseWare  
<http://ocw.mit.edu>

6.00SC Introduction to Computer Science and Programming  
Spring 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.