Final Draft Specification

Group:

Joanne Koong <joannekoong@college.harvard.edu>
Oriana Wang <orianawang@college.harvard.edu>
Joon Yang <joonhyukyang@college.harvard.edu>
Trevor Lutzow <trevorlutzow@college.harvard.edu>

Signature / Interfaces

K-MEANS CLUSTERING

We are using the k-means clustering algorithm to train our data. The k-means algorithm will split our data set into 10 (k = 10) clusters (for digits 0-9).

This algorithm aims to minimize the function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

which is an indicator of the distance of the n data points from their respective cluster centres.

Our steps in implementing the algorithm are as follows:

1. Initialize the 10 clusters; these points will represent initial group centroids
2. Assign each object to the group that has the closest centroid
3. When all objects have been assigned, recalculate the positions of the K centroids
4. Repeat steps 2 and 3 until the centroids no longer move. This helps ensure that objects are separated into the groups of which they are members.

PSEUDOCODE FOR LLOYD'S ALGORITHM

```
Pass in data vectors x_n for each of our data points 1 to N
Pass in the number of clusters K

initialize the responsibilities
for each n in 1 to N do
```

```
  zero the responsibilities
  r_n = [0,0,...0]

  choose a random responsibility to initialize to 1
  k' = RandomInteger(1,K)

  r_nk' = 1
end for
repeat

  loop over clusters
  for each k in 1 to K do

    compute the number assigned to the cluster
    N_k = sum of all r_nk from n=1 to N

    compute the mean of the kth cluster
    mu_k = (1/N_k)* sum of all r_nk x_n from n=1 to N
  end for

  loop over data
  for each n in 1 to N do

    zero the responsibilities
    r_n = [0,0,...0]

    find the closest mean
    k' = arg min_k ||x_n - mu_k||^2

    r_nk' = 1
  end for
until none of the r_n change

Return labels r_n for each data item n=1 to N
Return cluster means mu_k for each k in 1 to K
```

METHODS AND ACTUAL CODE:

```
def dist(x,c):

     return numpy.linalg.norm(x-c)


def classify_points (data_set, centroids, groups):
     appropriate_centroids = []
     for point in data_set:
          ## instantiate smallest_dist
          smallest_dist = dist(point, centroids[0])
```

```
            appropriate_centroids = centroids[0]
            for centroid in centroids:
                  distance = dist(point, centroid)
                  if distance < smallest_dist:
                        smallest_dist = distance
                        appropriate_centroid = centroid
            appropriate_centroids.append(appropriate_centroid)
      for element in appropriate_centroids:
            groups[centroids.index(element)].append(
                                    data_set.index(element))
      return groups

def move_means (centroids, groups):
      for i, set in enumerate(groups):
            total = 0
            for j, set2 in enumerate(set):
                  ind_mean = numpy.mean(set2)
                  total = total + ind_mean
            centroids[i] = total / (j + 1)
      return centroids

groups = [[] for i in range(0,10)]
oldcentroids = []
#centroids = random vectors to begin

while (oldcentroids != centroids):

      oldcentroids = centroids

      groups = classify_points (data_set, centroids, groups)

      centroids = move_means(centroids, groups)
```

ADDITIONAL CLASSIFIER EXTENSIONS AFTER IMPLEMENTING K-MEANS CLUSTERING ON DIGITS DATASET:

CONCEPTUAL IDEA BEHIND K-NEAREST NEIGHBORS ALGORITHM:

K-nearest neighbors classifies new cases based on its distance functions to the nearest data points (its neighbors). A case is classified by a majority vote of its neighbors.

CONCEPTUAL IDEA BEHIND SVM ALGORITHM:

SVMS construct a hyperplane that maximizes the margin between classes. It uses a subset of input (the support vectors) that are the observations **closest** to the separating hyperplane.

RUNNING RANDOM FOREST ON SCI-KIT LEARN TO COMPARE AS BENCHMARK

```
clf_rf = RandomForestClassifier()
clf_rf.fit(X_train, y_train)
y_pred_rf = clf_rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred_rf)

where X_train, y_train :
 (trainX, testX, trainY, testY) = train_test_split(dataset.data
                            / 255.0,dataset.target.astype("int0",
                            test_size = 0.40)
```

RUNNING SVM, K-NEAREST NEIGHBORS, AND OTHER CLASSIFIERS ON SCI-KIT LEARN

1. Load dataset into a python object (download the 55 mb MNIST dataset)

```
from sklearn import datasets, svm, metrics
from sklearn.datasets import fetch_mldata
dataset = datasets.fetch_mldata("MNIST original")
X, Y = mnist.data, mnist.target
```

2. Split data to generate training and testing set (we will use 40% for testing and 60% for training):

```
(trainX, testX, trainY, testY) = train_test_split(dataset.data
                            / 255.0,dataset.target.astype("int0",
                            test_size = 0.40)
```

3. Learn and predict

```
from sklearn import svm
svc = svm.SVC(kernel='linear')
```

Timeline & Roadmap

Finish implementing k-means clustering code in python : April 22
Finish running sci-kit tests using SVM and other such classifiers : April 24
Functionality Report : April 24 at 5 PM
Add additional features beyond digit recognition : April 28

Tidy up code : April 30

DEADLINE: May 1 at 5 PM

Progress Report

We are currently working on implementing k-means clustering in python. We have spent most of the time gathering the necessary resources such as online tutorials, books, videocasts, etc. about Python. We have been reading about python to get a better sense of python syntax and how the language works.

We also have downloaded sci-kit and are getting acquainted with how sci-kit learns. We have also spent considerable amounts of time reading the necessary background on machine learning (such as supervised vs. unsupervised learning, what classifiers exist, etc.). We have shared code through google docs and the next step will be to import it to PyCharm and make an executable.