Machine Learning Toolbox for Feature Extraction and Inference

Yo Sup "Joseph" Moon, Dana Modzelewski, William Chambers, Claudia Friedsam

CS51 Final Project April 8th, 2012 Draft
Due 11:59PM, April 8th, 2012.

Handwriting Recognition Toolbox Beta 0.1.0.

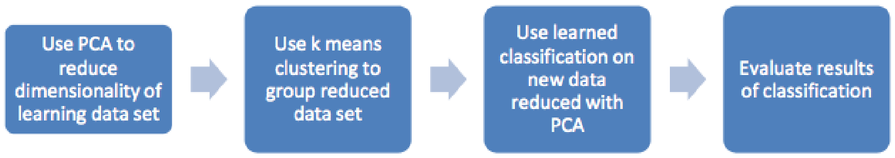
1 Brief Overview

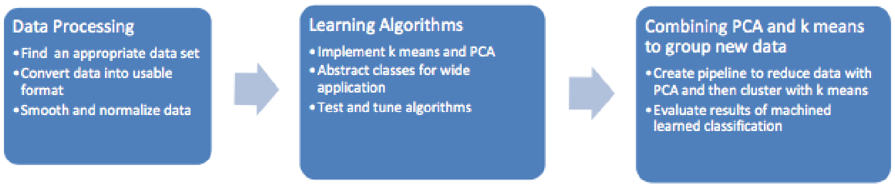
We want to implement two different methods for unsupervised learning with the goal to extract information from high dimensional data. The algorithms we chose are principal component analysis and k-means clustering. We want to implement the basic algorithms, test and compare them on a set of handwriting data and investigate their performance. We plan to address the limitations we identified by extending the algorithms with straightforward solutions like e.g. heuristics or optimization methods.

2 Feature

1. K-means
   Fundamental Features:
   - K-means algorithm
   - Test functions
   - Evaluation methods to examine performance

   Possible Extensions:
   - Heuristic or genetic algorithm for optimization
   - Explore variations of k-means, e.g. density based k- means, soft k-means
2. Principal Component Analysis (PCA)
   Fundamental Features:
   - PCA algorithm
   - Eigenvector Finding: can we find suitable time-efficient algorithm that suits our needs?
   - Test functions: sanity check on small-dimensional data. Test robustness of feature extraction: how much of the "useful" features of the data preserved?
3. Common Features:
   Fundamental Features:
   - Method to load and prepare data to be used
   - GUI to load data, pick the method, adjust input parameters, visualize results and performance for both methods
   - Testing performance of model: k-fold cross validation, logistic regression
   - Testing performance for the k-means and PCA algorithms
   - Testing performance for handwriting data recognition


Use PCA to reduce dimensionality of learning data set
Use k means clustering to group reduced data set
Use learned classification on new data reduced with PCA
Evaluate results of classification


Data Processing
•Find an appropriate data set
•Convert data into usable format
•Smooth and normalize data
Learning Algorithms
•Implement k means and PCA
•Abstract classes for wide application
•Test and tune algorithms
Combining PCA and k means to group new data
•Create pipeline to reduce data with PCA and then cluster with k means
•Evaluate results of machined learned classification

3 Code

3.1 Read Data Module

Functions:

- Read file into array
- Perform data clean up (remove outliers, smoothing, etc)
- Export Data

Exceptions:

- Can't read/find file
- File contains ill-defined data
- Cleanup failed
- Can't write file

```
import string
from numpy import *


#class for reading data file into array

class DataObj:
        def __init__(self,path):
                self.path = path
                |
        def readTextData(self,columns,type = float,separator ='\n'):
                self.data = fromfile(self.path, dtype=type, sep = separator)
                self.data.resize((self.data.shape[0]/columns),columns)
```

3.2 K-means

K-means Functions:

- Initialize Clusters

- Get centroids of partitioned data
- Calculate distances of points to centroids
- Reassign points to closest clusters
- Export data

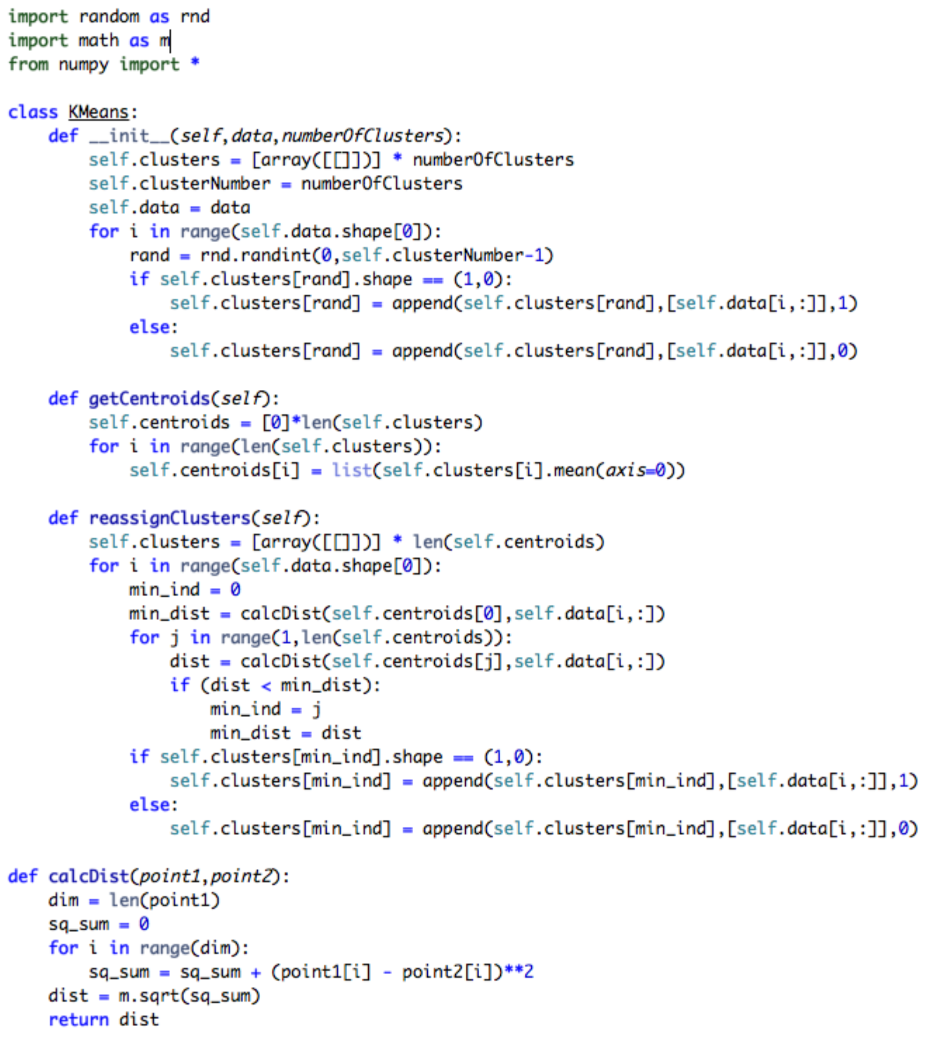
Exceptions:

- Number of clusters too big
- Empty clusters
- Can't write file

```
import os
from numpy import *
from readData import *
from kmeans import *
from plotData import *
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

def main():
        #read data file into array
        dataFile = DataObj('testData.txt')
        dataFile.readTextData(3, float, '\n')
        #plot inital data
        plotInitData = PlotData()
        plotInitData.scatter3D(dataFile.data,'b')
        numberOfClusters = 3
        #pick initial clusters
        clusters = KMeans(dataFile.data,numberOfClusters)
        #plot initial clusters
        plotInitClust = PlotData()
        plotInitClust.scatter3D(clusters.clusters[0],'b')
        plotInitClust.scatter3D(clusters.clusters[1],'g')
        plotInitClust.scatter3D(clusters.clusters[2],'r')
        #calculate initial centroids
        clusters.getCentroids()
        oldCentroids = []
        # beginning of loop, while new_centroids not equal old_centroids
        while (oldCentroids != clusters.centroids):
                oldCentroids = clusters.centroids
                # calculate new clusters
                clusters.reassignClusters()
                # recalculate new centroids
                clusters.getCentroids()
        plotFinalClust = PlotData()
        plotFinalClust.scatter3D(clusters.clusters[0],'b')
        plotFinalClust.scatter3D(clusters.clusters[1],'g')
        plotFinalClust.scatter3D(clusters.clusters[2],'r')
        plt.show()


if __name__ == "__main__":
        main()
```



```
import random as rnd
import math as m
from numpy import *

class KMeans:
    def __init__(self,data,numberOfClusters):
        self.clusters = [array([[]])] * numberOfClusters
        self.clusterNumber = numberOfClusters
        self.data = data
        for i in range(self.data.shape[0]):
            rand = rnd.randint(0,self.clusterNumber-1)
            if self.clusters[rand].shape == (1,0):
                self.clusters[rand] = append(self.clusters[rand],[self.data[i,:]],1)
            else:
                self.clusters[rand] = append(self.clusters[rand],[self.data[i,:]],0)

    def getCentroids(self):
        self.centroids = [0]*len(self.clusters)
        for i in range(len(self.clusters)):
            self.centroids[i] = list(self.clusters[i].mean(axis=0))

    def reassignClusters(self):
        self.clusters = [array([[]])] * len(self.centroids)
        for i in range(self.data.shape[0]):
            min_ind = 0
            min_dist = calcDist(self.centroids[0],self.data[i,:])
            for j in range(1,len(self.centroids)):
                dist = calcDist(self.centroids[j],self.data[i,:])
                if (dist < min_dist):
                    min_ind = j
                    min_dist = dist
            if self.clusters[min_ind].shape == (1,0):
                self.clusters[min_ind] = append(self.clusters[min_ind],[self.data[i,:]],1)
            else:
                self.clusters[min_ind] = append(self.clusters[min_ind],[self.data[i,:]],0)

def calcDist(point1,point2):
    dim = len(point1)
    sq_sum = 0
    for i in range(dim):
        sq_sum = sq_sum + (point1[i] - point2[i])**2
    dist = m.sqrt(sq_sum)
    return dist
```

3.3 K-means Evaluation Module

As a starting point we look at two criteria to investigate how well the clustering works. The first step is to look at the density of the clusters to see how compacted the clusters are. As a second criteria we will determine the silhouette, which is describing how distant the points in a specific cluster are from the other clusters to investigate the separation of the clusters.

Functions:

- Calculate Density
- Silhouette

3.4 PCA

Our PCA function takes a matrix and a desired number n of principal components as arguments. The function generates the covariance matrix for the input matrix and finds the eigenvalues and associated eigenvectors of the covariance matrix. Lastly, the function generates and returns a projection matrix consisting of the data projected onto the first n principal components.

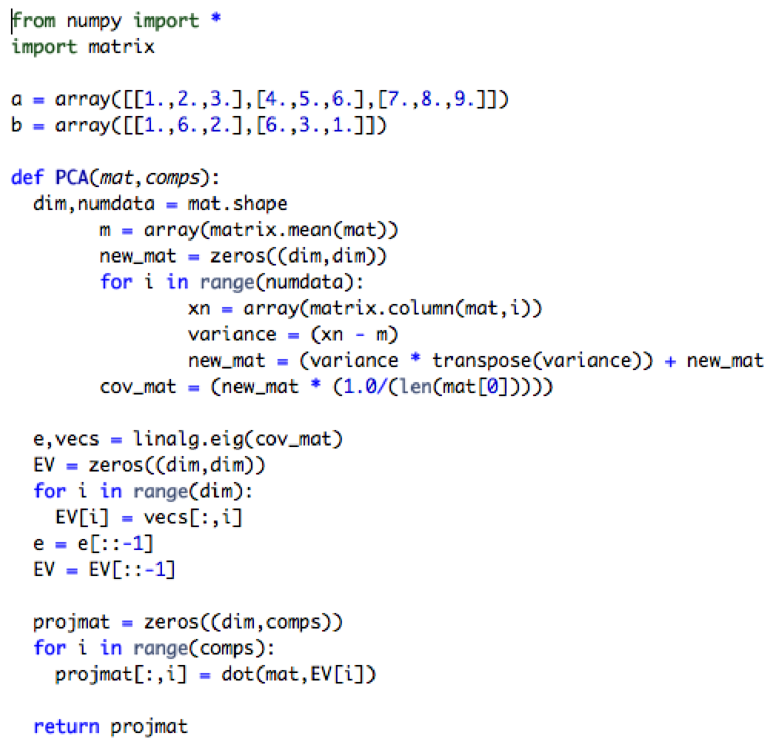
3.5 Description of PCA Module

PCA Functions:

- Calculate covariance matrix, S
- Find M largest eigenvalues of S
- Find eigenvectors corresponding to eigenvalues
- Project data onto eigenvectors
- Export Data

Exceptions:

- Can't compute covariance matrix
- Can't find eigenvalues for covariance matrix
- Can't write file



```
from numpy import *
import matrix

a = array([[1.,2.,3.],[4.,5.,6.],[7.,8.,9.]])
b = array([[1.,6.,2.],[6.,3.,1.]])

def PCA(mat,comps):
 dim,numdata = mat.shape
    m = array(matrix.mean(mat))
    new_mat = zeros((dim,dim))
    for i in range(numdata):
        xn = array(matrix.column(mat,i))
        variance = (xn - m)
        new_mat = (variance * transpose(variance)) + new_mat
    cov_mat = (new_mat * (1.0/(len(mat[0]))))

 e,vecs = linalg.eig(cov_mat)
 EV = zeros((dim,dim))
 for i in range(dim):
  EV[i] = vecs[:,i]
 e = e[::-1]
 EV = EV[::-1]

 projmat = zeros((dim,comps))
 for i in range(comps):
  projmat[:,i] = dot(mat,EV[i])

 return projmat
```

3.6 PCA Evaluation Module

As a preliminary criterion for determining the success of our PCA algorithm, we will look at the variances from projecting the data onto the principal components. *Functions*

- Calculate variances

4 Timeline

Today is Sunday, April 8th, 2012. There are 3 weeks left.

- First Week:
  - Completely finish rudimentary (but working) versions of k-means clustering and PCA.
  - Parse data to format that we want. Currently the datasets are in python pickle format.
  - Run our algorithms on actual data.

  - Important: identify BEST way to tackle character recognition! Look at different papers to get an idea of what people have found useful in the machine learning community.
- Second Week:
  - Actually implement character recognition
  - We have 10,000 training data points. We will train our model, run validation step on 1000 randomly chosen data points (possibly k-fold cross check validation)
  - After training our model, we will run the algorithm on 1000 data points of the test set.
  - Look at the results. Plot histogram, implement some other visual approaches to make sure our results are robust.
- Third Week:
  - Make sure implementation is free from bugs
  - Make sure implementation solves the problem for character recognition with responsible probability of success
  - Implement GUI and other "nice" features on top of the python framework.
  - Possibly make it a standalone?
  - Look at PyQt
  - Write up final specs as a README

5 Progress Report

We have written up rudimentary versions of the k-means and PCA algorithms. We have also written up extensive helper functions to read data and plot very basic results. We have integrated our code to be used with the following python modules: Python Image Library and Numpy. We have set up a git repository and have organized the code in the following manner:

- README.txt - the main README file of the Handwriting Recognition Toolbox.
- doc - the draft documents outlining the progress of our work. In particular it contains the latex documents for our drafts.
- data - contains the handwriting data retrieved from online data base. Also contains some randomly generated test data to test the accuracy/robustness of our algorithms. Matrix files parsed as python pickle files for reusability.
- src - contains source code for PCA and k-means. Also has upper-level scripts that handle general implementations such as parsing.
- output - contains output files, in text and pickle format. Will also contain plots and other graphical elements in the future.

Our git repository can be found at

`https://github.com/friedsam/cs51-final/`

or by cloning at the following address

`git://github.com/friedsam/cs51-final.git`