CSEP 546 Data Mining/Machine Learning, Winter 2014: Homework 4

Due: Monday, March 3^{rd}, beginning of class

1 Recommender systems for fun and profit [40 points]

For this problem, you will explore Matrix completion to predict movie ratings.

1.1 Matrix factorization

Given a set of "ratings" by the user u for movie v, $\mathcal{D} = \{(u, v, X_{uv})\}$, the task at hand is to find a rank-k approximation of X,

$$\hat{X} = \sum_{i=0}^{k-1} L_{:,i} R_{i,:} = LR, \quad , L \in \mathbb{R}^{n_u \times k}, R \in \mathbb{R}^{k \times n_m}.$$

This is to be done in such a way as to minimized the following cost,[1]

$$\begin{aligned} g(L, R) &= \frac{1}{2} \sum_{(u,v)\in\mathcal{D}} \left((X_{uv} - \hat{X}_{uv})^2 + \lambda(\|L_{u,:}\|^2 + \|R_{v,:}\|^2) \right), \\ &= \frac{1}{2} \sum_{(u,v)\in\mathcal{D}} \left((X_{uv} - \langle L_{u,:}, R_{:,v} \rangle)^2 + \lambda(\|L_{u,:}\|^2 + \|R_{v,:}\|^2) \right). \end{aligned}$$

The values of λ, k are to be picked, as always, *via* cross validation.

One of the ways of obtaining a Matrix Factorization (MF), is by running Stochastic gradient descent (SGD) over the dataset. You're provided with a Python implementation of SGD for the MF problem. The basic algorithm initializes L and R with small random values, before taking small steps "stochastically" over all the data points.

$\mathcal{D} \leftarrow \{(u, v, X_{uv})\}$
$L \leftarrow \epsilon \times \texttt{random}(n_u, k)$
$R \leftarrow \epsilon \times \texttt{random}(k, n_m)$,
for $(u, v, X_{uv}) \in \mathcal{D}$ **do**
  $r \leftarrow X_{uv} - \langle L_{u,:}, R_{:,v} \rangle$
  $L_{u,:} \leftarrow L_{u,:} - \eta(-r \times R_{:,v} + \lambda L_{u,:})$
  $R_{:,v} \leftarrow R_{:,v} - \eta(-r \times L_{u,:} + \lambda R_{:,v})$
end for

1.2 Movie Lens

You will now use the provided solver to make predictions using the Movielens 1M database. The dataset contains the following files: `movies.dat ratings.dat README users.dat`. The description of these files is given in the `README` file. The basic usage is summarized in `matrixcomp_starter.py`.

[1] The symbology $\langle x, y \rangle$ is just fancier notation for the dot product $x^T y$.

1. (*10 points*) Using the value of $\lambda = 1 \times 10^{-2}$ [2], train the given implementation of Matrix Factorization on 60% of the dataset for the values of $k \in \{10, 20, 50, 100\}$. For each value of k, use half of the remaining data points, to compute the validation MSE,

$$\frac{1}{|D_v|} \sum_{(u,v) \in D_v} (X_{uv} - \langle L_{u,:}, R_{:,v} \rangle)^2.$$

   Plot the validation MSE for each k in a graph.

2. (*5 points*) Report the value of k^*, which minimizes the validation MSE error.

3. (*5 points*) Using the remaining dataset, compute (and report) the test MSE, for the value of k^* that you found in (2).

4. (*10 points*) In a list, report the top-10 movie predictions for user "id=100" (in Python, this would correspond to the array index 99). Compare these with the top-10 ratings of the user. Do these predictions make sense ? The movie names are listed in the file `movies.dat`.

   Repeat the same for user "id=99".

5. (*10 points*) Modify the function `matrix_factorize`, to use different learning rates (the parameter `ssize` passed onto `stochastic_gd`). Try the following learning rates: {constant, $1/t$, $1/t^{0.75}$}, and comment on the convergence rates (and the training MSE) that you obtain. How do these compare with the default $1/\sqrt{t}$? Can you come up with a better learning rate ? You can use the value of k^* from (2), and the given λ, for all your experiments.

   Show a plot of the training MSE for each step of the outer loop.

2 Programming Question (Clustering with K-means) [50 points]

In class we discussed the K-means clustering algorithm. Your programming assignment this week is to implement the K-means algorithm on digit data.

2.1 The Data

There are two files with the data from handwritten digits dataset MNIST. The first

`mnist2500_X.txt`

contains the 2500 observations of 784 pixels (28×28 image) concerning handwritten digits. The second file

`mnist2500_labels.txt`

contains the true digit label (integer from 0 to 9). You can read both data files in with

```
X = numpy.loadtxt("mnist2500_X.txt")
Y = numpy.loadtxt("mnist2500_labels.txt")
```

Please note that there are no IDs for the digits. Please assume that the first line is ID 0, the second line is ID 1, and so on. The labels correspond to the digit file, so the first line of `mnist2500_labels.txt` is the label for the digit in the first line of `mnist2500_X.txt`.

[2] ... which has been given to you by an all-knowing oracle.

2.2 The algorithm

Your algorithm should be implemented as follows:

1. Select k starting centers that are points from your data set. You should be able to select these centers randomly or have them given as a parameter.
2. Assign each data point to the cluster associated with the nearest of the k center points.
3. Re-calculate the centers as the mean vector of each cluster from (2).
4. Repeat steps (2) and (3) until convergence or iteration limit.

Define convergence as no change in label assignment from one step to another **or** you have iterated 40 times (whichever comes first). Please count your iterations appropriately: after 40 iterations, you should have re-calculated the centers 40 times.

2.3 Within group sum of squares

The goal of clustering can be thought of as minimizing the variation within groups and consequently maximizing the variation between groups. A good model has low sum of squares within each group. We define sum of squares in the traditional way. Let C_k be the kth cluster and let μ_k be the empirical mean of the observations x_i in cluster C_k. Then the within group sum of squares for cluster C_k is defined as:

$$SS(k) = \sum_{i \in C_k} |x_i - \mu_{C_k}|^2$$

Please note that the term $|x_i - \mu_{C_k}|$ is the euclidean distance between x_i and μ_{C_k}, and therefore should be calculated as $|x_i - \mu_{C_k}| = \sqrt{\sum_{j=1}^{d} (x_{ij} - \mu_{C_kj})^2}$, where d is the number of dimensions. Please note that that term is squared in $SS(k)$. If there are K clusters total then the "sum of within group sum of squares" is just the sum of all K of these individual $SS(k)$ terms. .

2.4 Mistake Rate

Given that we know the actual assignment labels for each data point we can attempt to analyze how well the clustering recovered this. For cluster C_k let its assignment be whatever the majority vote is for that cluster. If there is a tie, just choose the digit that is smaller numerically as the majority vote. For example if for one cluster we had 270 observations labeled **one**, 50 labeled **three**, 9 labeled **five**, and 0 labeled **seven** then that cluster will be assigned value **one** and had $50 + 9 + 0 = 59$ mistakes. For each cluster, its mistake rate (between 0 and 1) is the number of mistakes divided by the number of samples in the cluster.

If we add up the total number of "mistakes" for each cluster and divide by the total number of observations (2500) we will get our total mistake rate, between 0 and 1.

2.5 Questions

When you have implemented the algorithm please report the following:

1. [15pts] The values of sum of within group sum of squares and mistake rates for $k = 5$, $k = 10$, $k = 15$, and $k = 20$. Please start your centers with the first k points in the dataset. So, if $k = 5$, your initial centroids will be samples with ID from 0 to 4, which correspond to the first five lines in the file.
2. [5pts] The number of iterations that k-means ran for $k = 20$, starting the centers as in the previous item. Make sure you count the iterations correctly. If you start with iteration $i = 0$ and at $i = 3$ the the cluster assignments don't change, the number of iterations was 4, as you had to do steps 2 and 3 to figure this out.
3. [5pts] The mistake rate of each cluster for $k = 20$.

4. [10pts] When $k = 20$, find out the cluster with the smallest mistake rate and the cluster with the largest mistake rates. For each of these 2 clusters, randomly select 3 samples from each cluster and show them as 3 binary images of handwritten digits.

   For each sample (row) in X, its 784 features are the pixel values of a 28×28 image. For example, the first 28 features correspond to the 28 pixels in the first column of the image, for i^{th} row in X, it can be plotted as a binary image by using the following code:

   ```
   figure(1)
   matplotlib.pyplot.imshow(reshape(X[i,:],(28,28)),cmap = cm.binary)
   show()
   ```

5. [5pts] Can you explain the result from the images? For instance, why does k-means work well on the images from the cluster with the smallest mistake rate ?

6. [5pts] A plot of the sum of within group sum of squares versus k for $k = 1, 2, 3, \cdots, 20$. Please start your centers randomly (choose k points from the dataset at random).

7. [5pts] A plot of total mistake rate versus k for $k = 1, 2, 3, \cdots, 20$. Please start your centers randomly (choose k points from the dataset at random).

For the last two items, you should run the experiment for each k about 5 times, and report the average within class sum of squares and average mistake rate for each k on your plots, just in order to make sure you don't submit a plot where k-means got really unlucky centers in the beginning. Only submit one plot for each question though. Also remember to submit your code.

2.6 Some useful functions

There are two python functions you may find useful in building your k-means algorithm. The first one returns a random permutation of integers from 1 to n:

```
numpy.random.permutation(n)
```

The second returns a matrix D storing the pairwise distances between points from X and Y, i.e., $D_{i,j}$ is the (euclidean) distance between the row vectors X_i and Y_j:

```
D = scipy.spatial.distance.cdist(X, Y, 'euclidean')
```