CSCI 5702/7702– Fall 2019 – Assignment 2

K-Means & Latent Factors

Due: 10/25/2019 – 8:59 pm

Please note that this assignment must be done individually. Your code will be checked against other submissions and other existing resources (such as websites and books) using automatic tools.

Warning: This problem requires substantial computing time (it can be a few hours on some systems). Also it is time consuming to accomplish the goals of the assignment. So, start working on the assignment IMMEDIATELY! ☺

Part 1: K-Means (50 points)

This problem will help you understand the nitty-gritty details of implementing clustering algorithms on Spark. In addition, this problem will also help you understand the impact of using various distance metrics and initialization strategies in practice. You must **not** use the Spark MLlib clustering library (or similar libraries) for this problem.

Let us say we have a set X of n data points in the d-dimensional space R^d. Given the number of clusters k and the set of k centroids C, we now proceed to define various distance metrics and the corresponding cost functions that they minimize.

Euclidean distance Given two points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Euclidean distance between A and B is defined as:

$$||a - b|| = \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2} \quad (1)$$

The corresponding cost function ϕ that is minimized when we assign points to clusters using the Euclidean distance metric is given by:

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} ||x - c||^2 \quad (2)$$

Manhattan distance Given two random points A and B in d dimensional space such that $A = [a_1, a_2 \cdots a_d]$ and $B = [b_1, b_2 \cdots b_d]$, the Manhattan distance between A and B is defined as:

$$|a - b| = \sum_{i=1}^{d} |a_i - b_i| \quad (3)$$

The corresponding cost function ψ that is minimized when we assign points to clusters using the Manhattan distance metric is given by:

$$\psi = \sum_{x \in \mathcal{X}} \min_{c \in \mathcal{C}} |x - c| \quad (4)$$

Iterative *k*-Means Algorithm: We learned the basic *k*-Means algorithm in class which is as follows: *k* centroids are initialized, each point is assigned to the nearest centroid and the centroids are recomputed based on the assignments of points to clusters. In practice, the above steps are run for several iterations. We present the resulting iterative version of *k*-Means in Algorithm 1.

Algorithm 1 Iterative *k*-Means Algorithm

```
1: procedure Iterative k-Means
2:     Select k points as initial centroids of the k clusters.
3:     for iterations: = 1 to MAX ITER do
4:         for each point p in the dataset do
5:             Assign point p to the cluster with the closest centroid
6:         end for
7:         Calculate the cost for this iteration.
8:         for each cluster c do
9:             Recompute the centroid of c as the mean of all the data points assigned to c
10:        end for
11:    end for
12: end procedure
```

Iterative *k*-Means clustering on Spark: Implement iterative *k*-means using Spark. Note that we have provided that centroids for you (see below), so you do not need to select the initial centroids (skip Line 2 of Algorithm 1).

P1 has 3 files:

1. *data.txt* contains the dataset which has 4601 rows and 58 columns. Each row is a document represented as a 58-dimensional vector of features. Each component in the vector represents the importance of a word in the document.

2. *c1.txt* contains *k* initial cluster centroids. These centroids were chosen by selecting *k* = 10 random points from the input data.

3. *c2.txt* contains initial cluster centroids which are as far apart as possible. (You can do this by choosing 1st centroid c1 randomly, and then finding the point c2 that is farthest from c1, then selecting c3 which is farthest from c1 and c2, and so on).

Set the number of iterations to 20 and the number of clusters *k* to 10 for all the experiments carried out in this question. Your driver program should ensure that the correct amount of iterations are run.

a. Exploring initialization strategies with Euclidean distance

1. Using the Euclidean distance (refer to Equation 1) as the distance measure, compute the cost function $\varphi(i)$ (refer to Equation 2) for every iteration *i*. This means that, for your first iteration, you will be computing the cost function using the initial centroids located in one of the two text files. Run the *k*-means on data.txt using *c1.txt* and *c2.txt*. Generate a graph (line plot) where you plot the cost function $\varphi(i)$ as a function of the number of iterations *i*=1..20 for *c1.txt* and also for *c2.txt*.

   Hint: Note that you do not need to write a separate Spark job to compute ϕ(i). You should be able to calculate costs while partitioning points into clusters.

2. Is random initialization of *k*-means using *c1.txt* better than initialization using *c2.txt* in terms of cost $\varphi(i)$? Explain your reasoning.

b. Exploring initialization strategies with Manhattan distance

1. Using the Manhattan distance metric (refer to Equation 3) as the distance measure, compute the cost function $\psi(i)$ (refer to Equation 4) for every iteration *i*. This means that, for your first iteration, you'll be computing the cost function using the initial centroids located in one of the two text files. Run the *k*-means on *data.txt* using *c1.txt* and *c2.txt*. Generate a graph where you plot the cost function $\psi(i)$ as a function of the number of iterations *i*=1..20 for *c1.txt* and also for *c2.txt*.
2. Is random initialization of *k*-means using *c1.txt* better than initialization using *c2.txt* in terms of cost $\psi(i)$? Explain your reasoning.

Code Structure:

Your code must have a kmeans function with the following signature (you can define as many functions as you like, but the kmeans function is the driver function meaning that it should be the only function that is directly called by the user):

kmeans(data, centroids, iterations, euclidean_distance)

- data is an RDD. Each value of data must be a numpy array of type float, containing all the values for one line on the input file. For example, if the input line is 0 3.245 3.2 your array must contain [0, 3.245. 3.2]
- centroids is an RDD. Each value of centroids must be a numpy array of type float, containing all the values for one line on the centroid file. For example, if the input line is 0 3.245 3.2 your array must contain [0, 3.245. 3.2]
- euclidean_distance is a Boolean variable. If it is True (False), your k-means function must use the Euclidean (Manhattan) distance as the distance measure.

You must declare four variables in the very beginning of your code:

- *Iterations = 20 # determines the number of iterations*
- *dataset_path = '' # this is the absolute path to the dataset file*
- *centroids_path = '' # this contains the absolute path the centroid file that being use (either c1.txt or c2.txt)*
- *euclidean_distance = True # or False*

These variables must be immediately followed by calling the kmeans function, which outputs the cost for each of the 20 iterations in the following format (note that the values in the below example are synthetic):

kmeans(data, centroids, max_iterations, euclidean)

output:

Max iterations: 20, distance measure: Euclidean
iteration 1: 5.360
iteration 2: 8.3762
…
iteration 20: 6.61

What to submit for Part 1

a. The code as a Jupyter Notebook (.ipynb) or a .txt file (30 points)
b. A pdf file containing:
    - A plot of cost vs. iteration for two initialization strategies for a.1 (5 points)
    - A plot of cost vs. iteration for two initialization strategies for b.1 (5 points)
    - Your answer for questions a.2 and b.2 (10 points)

Part 2: Latent Factors for Recommendations (50 points)

The goal of this problem is to implement the *Stochastic Gradient Descent* algorithm to build a Latent Factor Recommendation system. We can use it to recommend movies to users. We encourage you to read the slides of the lecture "Recommender Systems 2" again before attempting the problem.

Suppose we are given a matrix R of recommendations. The element R_{iu} of this matrix corresponds to the rating given by user u to item i. The size of R is $m \times n$, where m is the number of movies and n the number of users.

Most of the elements of the matrix are unknown because each user can only rate a few movies.

Our goal is to find two matrices P and Q, such that $R = QP^T$. The dimensions of Q are $m \times k$, and the dimensions of P are $n \times k$. k is a parameter of the algorithm. We define the error as

$$E = \left(\sum_{(i,u) \in \text{ratings}} (R_{iu} - q_i \cdot p_u^T)^2 \right) + \lambda \left[\sum_u \|p_u\|_2^2 + \sum_i \|q_i\|_2^2 \right]. \quad (5)$$

The $\sum_{(i,u)\in \text{ratings}}$ means that we sum only on the pairs (user,item) for which the user has rated the item, i.e., the (i,u) entry of the matrix R is known. q_i denotes the i^{th} row of the matrix Q (corresponding to an item), and p_u the u^{th} row of the matrix P (corresponding to a user u). λ is the regularization parameter. $||.||_2$ is the L2 norm and $\|p_u\|_2^2$ is square of the L2 norm, i.e., it is the sum of squares of elements of p_u.

a. Equations

Let ε_{iu} denote the derivative of the error E with respect to R_{iu}. What is the expression for ε_{iu}? What are the update equations for q_i and p_u in the Stochastic Gradient Descent algorithm?

b. Implementation

Implement the algorithm. Read each entry of the matrix R from disk and update ε_{iu}, q_i and p_u for each entry.

To emphasize, **you are not allowed to store the matrix *R* in memory**. You have to read each element R_{iu} one at a time from disk and apply your update equations (to each element). If you keep more than one element in memory at a time, you will receive no point for Part b. For example, File.ReadAllText() and File.ReadAllLines() in C# are examples of functions that read in the entire file into memory. On the other hand, a StreamReader object in C# reads one line at a time. Each iteration of the algorithm will read the whole file.

Choose $k = 20$, $\lambda = 0.1$ and number of iterations = 40. Find a good value for the learning rate η. Start with $\eta = 0.1$. The error E on the training set ratings.train.txt discussed below should be less than 65000 after 40 iterations.

Based on the values of η, you may encounter the following cases:

- If η is too big, the error function can converge to a high value or may not monotonically decrease. It can even diverge and make the components of vectors *p* and *q* equal to infinity.
- If η is too small, the error function will not have time to significantly decrease and reach convergence. So, it can monotonically decrease but not converge, *i.e.,* it could have a high value after 40 iterations because it has not converged yet.

Programming Languages

You can use either **Python or Java** to implement this part, but Python is strongly recommended.

Dataset

ratings.csv: this is the matrix *R*. Each entry is made of a movie id, user id, and a rating that is an integer between 1 and 5.

Code Structure

You need to define the following five parameters in the beginning of your code:

- iterations = 40
- k = 20
- regularization_factor = 0.1 #λ
- learning_rate = 0.1 # η
- data_path = ''

Your code must have a driver function with the below signature, which is the only function that is directly called by the TAs for grading. However, you can use as many functions as you like.

latent_factor_recommnder(data_path, regularization_factor, learning_rate, iterations, k)

The driver function must output the error for each iteration in the following format (the provided values are synthetic):

Initialization step: 652,434

Iteration 1: 23,4567

Iteration 2: 81,548

…

iteration: 40, error: 64,000

Hints:

- **P and Q**: we would like q_i and p_u for all users u and items i such that $q_i \cdot p_u^T \in [0, 5]$. A good way to achieve that is to initialize all elements of P and Q to random values in $[0, \sqrt{5/k}]$.

- **Updating the equations**: in each update, we update q_i using p_u and p_u using q_i. Compute the new values for q_i and p_u using the old values, and then update the vectors q_i and p_u.
- You should compute E at the end of a full iteration of training. Computing E in pieces during the iteration is incorrect since P and Q are still being updated.

What to submit for Part 2

- The equations for ε_{iu}. Update equations in the Stochastic Gradient Descent algorithm (2.a) **(5 points)**
- What was the lowest error you got? What was the value of η? **(5 points)**
- For the best η, plot of E vs. number of iterations. Make sure your graph has a y-axis so that we can read the value of E. **(5 points)**
- The code as a Jupyter Notebook (.ipynb) or a .txt file. **(35 points)**

Submission

You need to submit a .zip file on Canvas, named as your-lastname_your-first-name.zip containing:

- One folder named P1 containing all the requirements for Part 1.
- One folder named P2 containing all the requirements for Part2.

Please download your assignment after submission and make sure it is not corrupted. We will not be responsible for corrupted submissions and will not be able to take a resubmission after the deadline.

Need Help?

In case you need help with the details of this assignment, please email Shahab at shahab.helmi@ucdenver.edu or go to his office hours (Tuesday/Thursday 9-10 am).

Evan will be able to help only with the general Spark/Python questions for this assignment.

You are highly encouraged to ask your question on the designated channel for Assignment 2 on Microsoft Teams (not monitored by the TA's). Feel free to help other students with general questions. DO NOT share your solution.