Assignment No2. Spark

Atif Farah, Telegram: @FarahAtif
Konyshev Yan, Telegram: @DorakAlba
Marina Ivanova, Telegram: @marina_ii

GitHub Repository
https://github.com/farahFif/BigDataAssignment2

Trello Board
https://trello.com/b/amh4n0r1/bd-assignment-2

Introduction

Academic studies on big data have been increasing since the 2012 year. In the information era, enormous data have become available on hand to decision-makers. Due to such rapid growth of data, solutions had to be studied in a fast and effective way. Such values can be effectively analyzed with big data analytics. Big Data is actively used in Data Mining, Decision Trees to help users to understand combinations of attributes that lead to the desired outcomes.
Big data is also used by credit card companies, as they rely on accuracy and speed of database's to identify possible frauds and is used in recommendation systems.

Recommendation systems are defined as algorithms to provide users with relevant items [3]. They have become in the last few years very popular for their advantage of affording to users a rich experience when surfing on the web. They also knew a remarkable success due to the disponibility of data and big data technologies. Epinions.com, which is an an online marketplace, was concerned with representing the relationship of trust between users when posting reviews for goods. That's why one of the possible solution to illustrate this problem is implementing a recommendation system based on a social graph, where each node represents a user and an oriented edge represents "trust". In this report we will describe the implementation of a recommendation system using the algorithm Node2vec and its evaluation using the MAP metric.

Solution

The goal of our project is to implement a recommendation system for Epinions.com [1]. The graph represents the "trustability" of users to customers, where each node represents a user and an oriented edge represents "trust".
During this work, we had to train and test node2vec [2] model using deep learning and train in the usage of Scala and Spark.
The goal is to give 10 recommended edges to each node and measure the quality of recommendations using MAP [3] on the testing data.
Training the model has been done using gradient descent optimizer and the evaluation using MAP to measure accuracy of the model.
The training was written in Scala and MAP module in Python to quickly gather and visualize all results. The following sections provide details of how it was implemented.
It is worth to note that given data has missing nodes. It means there are no recommendations (destination nodes) for such a node.

Training

Training the neural network

- **Read data:** This function does the reading of train and test datasets that are stored in CSV files and located in hdfs. This function takes as parameters the path to the file and SparkSession object and outputs an array of type RDD containing data.
- **Create batches:** This function allows to split the dataset into batches in order to calculate gradient descent for each batch and not the entire dataset. To do so, this function takes an array of type RDD containing data and zip each element in RDD to assign an ID for each edge to simplify indexing. Then, each batch will contain (Size of data / number of batches) edges.
- **Create embedding matrix:** this function creates embedding matrices of source and destination nodes with size of embedding dimension X total number of nodes.
- **Gradient descent:** The goal of gradient descent is to optimize the loss function, it's used here to update parameters of the model that reside in embedding matrices.
  Gradient descent as mentioned before aims to optimize a function, in node2vec algorithm the formula of the function is the following:

$$J(Out, In) = \sum_{(u,n)\in E} \left[-\log \sigma(out_n^T in_u) - \sum_{k\sim G(u)} \log(1 - \sigma(out_k^T in_u)) \right]$$

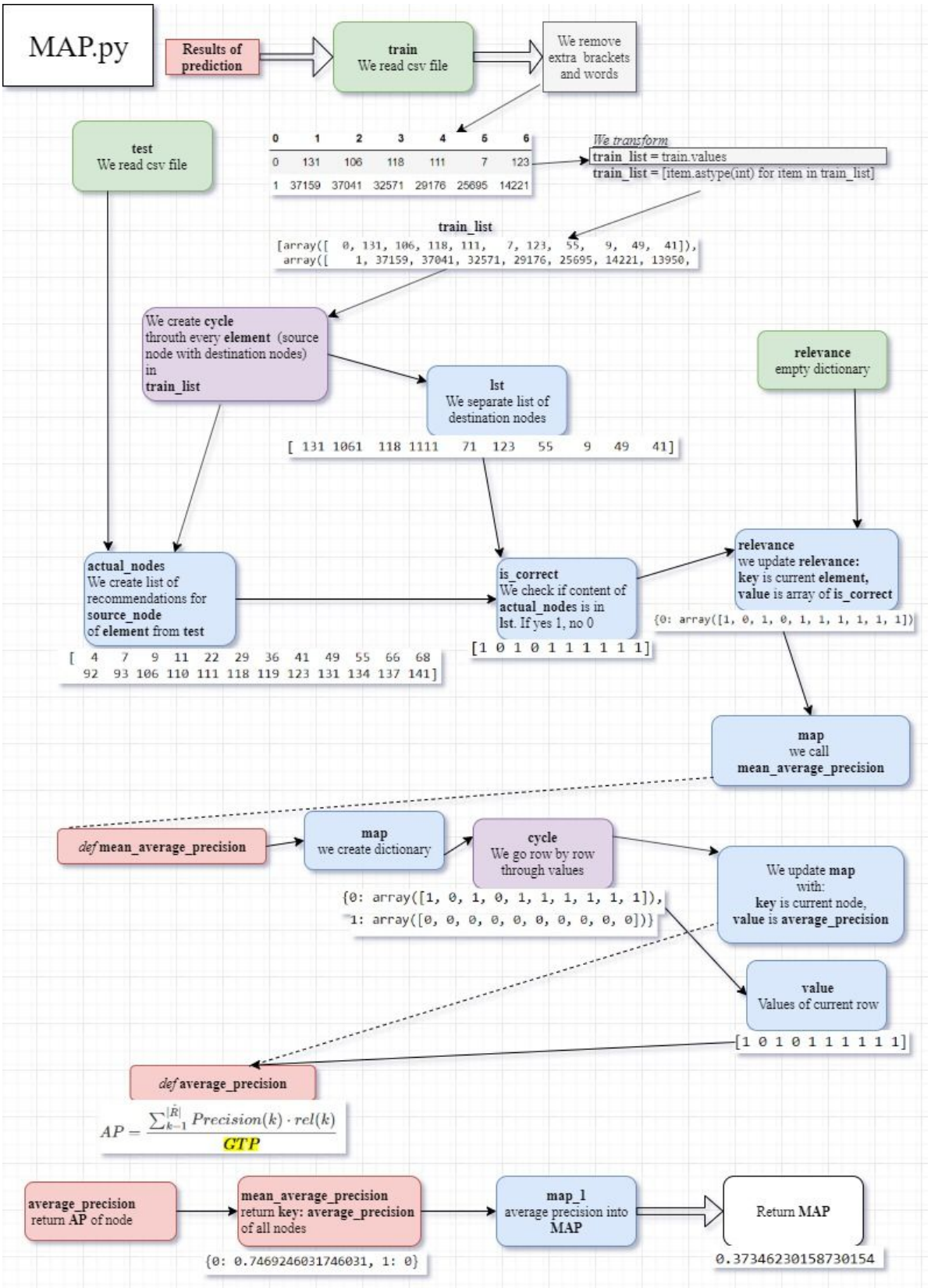
  For this solution, gradient descent will be implemented in batch. To do so we created the following functions, but before a step should be done which is the broadcasting of embedding matrices to the nodes of the cluster.
  - **Estimate gradients for edges:** this function calculates the derivative of the function above for true samples.
  - **Estimate gradients for negative samples:** this function calculates the derivative of the function above for negative samples.
  - **Update gradients:** this function allows to update embedding matrices containing calculated gradients using the following formula.

$$In = In - \alpha \frac{dJ}{dIn}$$

$$Out = Out - \alpha \frac{dJ}{dOut}$$

Evaluation

- For every node in the graph, we rate possible neighbors and sort candidates by predicted rating in the descending order, then filter out nodes with existing edges from the recommendation. To do so we implemented the function **estimate top ten neighbors.** This function estimates the top ten neighbors for a given node in the graph.
- **Map** : In order to assess the quality of the model, we used the Mean Average Precision (IDA) metric. This metric is a good choice for quality assessment if the algorithm produces ranked ordering of items by their priority, where every item is either relevant or irrelevant. The following chart explains the implementation of recommendation system evaluation with MAP. Python was chosen as the language of implementations because the code is going to be more compact than in Scala. It's possible to find better to find a better model for estimating the quality of model (figure 1) .


MAP.py
Results of prediction
train
We read csv file
We remove extra brackets and words
test
We read csv file
0 1 2 3 4 5 6
0 131 106 118 111 7 123
1 37159 37041 32571 29176 25695 14221
We transform
train_list = train.values
train_list = [item.astype(int) for item in train_list]
train_list
[array([0, 131, 106, 118, 111, 7, 123, 55, 9, 49, 41]),
array([1, 37159, 37041, 32571, 29176, 25695, 14221, 13950,
We create cycle throuth every element (source node with destination nodes) in train_list
lst
We separate list of destination nodes
[131 1061 118 1111 71 123 55 9 49 41]
relevance
empty dictionary
actual_nodes
We create list of recommendations for source_node of element from test
[4 7 9 11 22 29 36 41 49 55 66 68
92 93 106 110 111 118 119 123 131 134 137 141]
is_correct
We check if content of actual_nodes is in lst. If yes 1, no 0
[1 0 1 0 1 1 1 1 1 1]
relevance
we update relevance:
key is current element,
value is array of is_correct
{0: array([1, 0, 1, 0, 1, 1, 1, 1, 1, 1])
map
we call mean_average_precision
def mean_average_precision
map
we create dictionary
cycle
We go row by row through values
{0: array([1, 0, 1, 0, 1, 1, 1, 1, 1, 1]),
1: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])}
We update map with:
key is current node,
value is average_precision
value
Values of current row
[1 0 1 0 1 1 1 1 1 1]
def average_precision
$AP = \frac{\sum_{k=1}^{|\hat{R}|} Precision(k) \cdot rel(k)}{GTP}$
average_precision
return AP of node
mean_average_precision
return key: average_precision of all nodes
{0: 0.7469246031746031, 1: 0}
map_1
average precision into MAP
Return MAP
0.37346230158730154

Results

Results for random nodes

```
(281,ArrayBuffer(37720, 1478, 17171, 5355, 450, 36127, 28723, 25725, 36563, 16217))
(278,ArrayBuffer(32863, 26003, 25517, 29448, 17398, 7463, 31452, 17171, 2972, 23606))
(224,ArrayBuffer(15135, 33638, 31879, 13304, 36127, 5639, 25265, 19266, 7463, 25725))
(125,ArrayBuffer(17957, 15135, 22401, 17171, 30940, 2972, 28027, 36779, 25265, 7463))
(125,ArrayBuffer(17957, 15135, 22401, 17171, 30940, 2972, 28027, 36779, 25265, 7463))
(218,ArrayBuffer(14401, 30198, 32175, 2972, 36098, 7463, 28027, 25265, 14948, 22566))
(31,ArrayBuffer(12938, 25265, 10202, 21070, 31879, 22598, 21448, 37720, 20234, 7463))
(246,ArrayBuffer(37820, 2972, 13268, 35649, 12324, 39546, 22566, 17171, 30940, 25265))
(216,ArrayBuffer(14302, 7463, 17965, 22566, 1478, 35649, 17171, 32876, 2972, 32279))
(166,ArrayBuffer(30940, 39032, 30198, 25265, 27146, 29440, 14401, 24259, 2972, 7463))
(102,ArrayBuffer(30337, 32730, 8124, 17965, 35649, 17171, 2972, 17398, 1008, 7463))
(205,ArrayBuffer(7463, 19398, 37720, 17965, 10202, 5355, 31196, 25265, 35649, 8767))
(173,ArrayBuffer(18535, 35649, 25493, 9478, 31879, 17171, 2972, 17965, 28027, 37720))
(39,ArrayBuffer(35151, 15135, 10202, 31879, 14948, 7463, 25517, 25265, 37820, 37720))
(286,ArrayBuffer(21448, 25265, 15135, 24420, 20617, 12467, 10202, 28027, 37451, 7463))
(159,ArrayBuffer(21490, 2972, 3671, 37720, 17675, 34114, 5355, 22566, 7463, 38497))
(19,ArrayBuffer(13950, 645, 25695, 8607, 37041, 2163, 14221, 29176, 37159, 32571))
(198,ArrayBuffer(30198, 8767, 3671, 17171, 34114, 7463, 18468, 37720, 17965, 25265))
(136,ArrayBuffer(38262, 3711, 34114, 7635, 32279, 34159, 33462, 14302, 5726, 25265))
```

Results analysis

```
model precision 1.1494252873563218 %
```

Low accuracy reasoned with the low quality of the model because of its simplicity. As we used simplified node2vec in this task. The usage of the proper algorithm would allow us to achieve higher accuracy.
Accuracy has a connection to the test data set, as he has many missing nodes in it.
We could further increase accuracy if we are going to find time for toying with hyperparameters.

Discussion

For this task one of the variants is the usage of the pipeline, this could streamline the process, but the embedding process is going to take more time.
Another possible way of improvement is trying different parameter values to improve the model quality.

Task splitting

Member	Task
Farah ATIF	Project creation, training and report
Yan Konyshev	MAP, report
Marina Ivanova	Trello, report

Conclusion

This project gave as a concrete idea and a deep understanding of how a recommendation system can be implemented using the algorithm node2vec.
We created a basic node2vec model using Scala and we implemented MAP function in Python. We found out that the simple model was not effective for a such task comparing with the original node2vec model and a further testing should be done but for organization and schedule constraint we were not able to go further. However, we believe that the model can be optimized more and extended to many other domains.

References

[1] - Epinions.com
[2] - node2vec: Scalable Feature Learning for Networks - Aditya Grover, Jure Leskovec
[3] - Breaking Down Mean Average Precision (mAP) - Ren Jie Tan
[4] - Baptiste Rocca. Introduction to recommender systems overview of some major recommendation algorithms.