Homework Assignments

Information Retrieval 1 [2016/2017]

Module 3: Learning to Rank

Deadline: Thursday, February 2nd, 23:59
Submit: The modified python code (No Ipython Notebook this time) with the necessary (a) **implementation**, (b) **explanations**, (c) **comments**, and (d) **analysis**. Code quality, informative comments, detailed explanations of what each new function does and convincing analysis of the results will be considered when grading.
Filename: <student id1>-<student id2>-hw3.py

Note: The full assignment is 90 points worth. There will be a mini assignment worth the 10 more points some time during next week. This mini assignment will only require 1 hours of manual effort in constructing a training set of query-document pairs collectively, and no technical work. These will be the 10 easiest points you will ever get.

Step 1: Install Theano and Lassagne [0pts]
Tip: Use Anaconda to install Theano and Lassagne.
In the environment you run your code set THEANO_FLAGS to 'floatX=float32'

Note: Code is provided to you for this task in Theano/Lassagne. However if you feel like using a different framework, e.g. TensorFlow, go for it; you will just need to reimplement parts of the code.

Step 2: Study the a neural net pointwise algorithm provided [0pts]
The code provided implements a simple Neural Network in Theano that learns to predict the label of each query-document feature vector. Read and understand the code.

Step 3: Transform the pointwise algorithm into a pairwise (RankNet) [50pts]
A good overview of RankNet, LambdaRank, and LambdaMART can be found in "From RankNet to LambdaRank to LambdaMART: An Overview" [link] The slides presented in the class provide a more clear picture on the algorithmic implementation of LambdaRank

The algorithm that you will need to implement - as described in the lecture slides - is:

1. Initialize the Neural Net with random weights
2. For a number of epochs:
    a. For each query:
        i. Rank documents based on the output of the Neural Network
        ii. Consider all pairs of documents within this ranking
        iii. For each pair:
            1. Compute S_{ij} using the actual labels of the documents

2. Compute λ_{ij}

iv. For each document :

1. Aggregate λ_{ij}
2. Compute the gradients of scores with respect to weights
3. Multiply the gradient by lambdas
4. Update the weight by using the resulting gradients

The highlighted text above is essentially the part of you will need to implement. In the provided code you will find a number of TODO's. These are the points of the code that you will need to change/implement.

Step 4: Load the dataset [0pts]
The provided dataset is already split and ready for a 5-fold cross-validation. Each one of the 5 folders contain three files, the train.txt, vali.txt, and test.txt files, for the purpose of training, validating and testing your code. Each line in any of these files corresponds to a query-document pair. The first value is the label of the document wrt the query. The second the query id, and the last the document id. Everything in between is the feature vector extracted from this query-document pair (64 features in total) in the format feature id:feature value.

Two auxiliary functions are provided to you that allow you to essentially load the queries, query.py and document.py.

Step 5: Train and evaluate the two algorithms by NDCG@10 (5-fold cross-validation) [20pts]
Even though the auxiliary code for loading data and the code for the pointwise Neural Net is provided for you the experimental design (i.e. the functions to actually run the experiment over the data) is missing.

Implement the experiment so that the algorithm is trained over the train.txt, validated over the vali.txt (this is an optional step) and tested over the test.txt. Regarding the validation - optional - you can use the vali.txt dataset to choose the epoch that gave you the optimal model, but considering each model after each epoch and computing NDCG@10 over the vali.txt set.

Advice 1: Build the experimental design using the provided algorithm, before introducing the RankNet changes so you can easier check for bugs.
Advice 2: Use one fold, e.g. Fold1 to run your experiments end-to-end before you run for the entire 5-fold cross validation.

Step 6: Multiply Lambda's with the ΔNDCG@Max (LambdaRank) [10pts]
Implement the LambdaRank variant of RankNet. Note here that when you compute NDCG to be used to extend the definition of lambda's use NDCG@max, i.e. NDCG at the end of the ranked list and not at 10.

Step 7: Analysis [10pts]
Compare the pointwise, pairwise (RankNet), and listwise (LambdaRank - if implemented) algorithms in terms of mean NDCG@10 - first calculate it across all queries in each test set and then average across all five folds. Write your conclusions.

NOTE: The code has been tested after the implementation of LambdaRank. However, it has not been fully tested at it's current state (i.e. after removing all the unnecessary code to build the assignment)
NOTE: The NDCG@10 you should be getting using the pairwise and listwise approaches should be in the ballpark of 0.6.
NOTE: The data come from a homepage finding task, i.e. users posting a query are simply looking for a homepage (e.g. facebook, or iamsterdam, etc.) Hence there is only a single relevant document per query. Given that there is a single relevant document the question that arises is why need pairwise or listwise algorithms here. The answer comes from considering is how these algorithms try to find this relevant document and where do they spend their effort on. A pointwise will be trying to correctly predict all the zeros (and most likely it simply predict everything as a zero and ignore the single 1). This is where a pairwise will do much better, because the pairs that are non ties are the ones really involving the single relevant document. The hw asks for a pairwise algorithm (RankNet). A listwise algorithm, indeed, would not bring much in such a task, but it may happen to learn faster. The reason is that for a pairwise algorithm a ranked list [0 0 1 0 0 0 ... 0] is not that bad, since it go right all but two pairs. However, for LambdaRank this is a very bad ranking, since the ndcg of this ranking is way worse than the ideal.