

Homework Assignments

Information Retrieval 1 [2015/2016]

Module 3: Learning to Rank (2pts)

Deadline: Monday, January 25th, midnight

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: <your id>-hw3.py

For any questions: (a) Attend the the Thursday/Friday Laptop session, (b) Contact Artem Grotov at a.grotov@uva.nl.

Step 1: Install Theano and Lassagne [0%]

Tip: Use Anaconda to install [Theano](#) and [Lassagne](#).

In the environment you run your code set THEANO_FLAGS to 'floatX=float32'

Step 2: Study the a neural net pointwise algorithm provided [0%]

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) [70%]

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 [0%]

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) [20%]

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 Δ NDCG@Max (LambdaRank) (0.3pts Extra Credit)

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 [10%]

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.