Pairwise Learning to Rank for MeTA

(Track: Software)

Rachneet Kaur, Mihika Dave, Anthony Huang
Outline

- Motivation & Goal: Learning to Rank
- Stochastic Pairwise Descent Algorithm
- Implementation Details
- Software Usage & Demo
- Results
- Challenges & Future Work
Motivation

- MeTA is a C++ data science toolkit
- MeTA is used by students of UIUC, Coursera as well as the TIMAN Research Group
- MeTA doesn’t currently support Learning To Rank
- Modern search platform like Solr have implementation for RankSVM, LambdaMART for ranking
Motivation

- Learning to rank is a machine learning task for ranking objects
- It can be employed in many areas
  - Information Retrieval (IR)
  - Natural Language Processing (NLP)
  - Data Mining (DM)
- Common applications include document retrieval, search, question answering, document summarization, machine translation, etc
Goal

- We extended MeTA toolkit with Learning to Rank
- We implemented a pairwise algorithm:
  - Stochastic Pairwise Descent
- Pairwise learning to rank problem can be reduced to learning a binary classifier
- MeTA supports SVM classifier which can be thus utilized to implement the above algorithm
Algorithm: Stochastic Pairwise Descent

1: $D_{index} \leftarrow \text{CreateIndex}(D)$
2: $w_0 \leftarrow \emptyset$
3: for $i = 1$ to $t$ do
4:     $((a, y_a, q), (b, y_b, q)) \leftarrow \text{GetRandomPair}(D_{index})$
5:     $x \leftarrow (a - b)$
6:     $y \leftarrow \text{sign}(y_a - y_b)$
7:     $w_t \leftarrow \text{StochasticGradientStep}(w_{i-1}, x, y, i)$
8: end for
9: return $w_t$

- Link to research paper: http://www.eecs.tufts.edu/~dsculley/papers/large-scale-rank.pdf

*Sculley, D. "Large scale learning to rank." NIPS Workshop on Advances in Ranking, 2009.*
Indexed Sampling

- Pairs are sampled by the following method:
  - Indexing the data D into nested hash table P
    - Let Q - unique values in D
    - For q ∈ Q, map Y[q] to set of unique y values such that (x, y, q′) ∈ D with q = q′
    - For q ∈ Q and y ∈ Y[q], map P[q][y] to (x, y′, q′) ∈ D with q = q′ and y = y′
  - Sampling from P in O(1) time
    - Uniformly sample a query q from Q
    - For the sampled query, we sample 2 data samples with different output labels
  - Such randomly sampled pairs are used for training the classifier

Faster when D fits in the memory.
Our Implementation in Meta

- **Dataset**
  - LETOR 3.0 dataset:
    - TD2003, NP2003, HP2003, OHSUMED

- **Training iterations**
  - 100,000 (as suggested in the paper)
  - fixed for all datasets

- **Optimization**
  - SGD

- **Sampling**
  - Indexed Sampling

- **Evaluation**
  - computed the following over each dataset
    - Precision
    - MAP
    - NDCG
  - compared results with those published in LETOR 3.0 paper
Our Implementation in Meta: (letor.cpp)

- **readData()**: Read data from dataset and store it as nested hash-tables.
- **getRandomPair()**: Return a random pair of tuple for training the svm classifier. Tuple is of type (feature_vec, label, qid).
- **trainSVM()**: Train SVM classifier with the pair and compute loss.
- **validate()**: Validate the learnt model.
- **test()**: Test the model on test data.
- **evaluate()**: Evaluate the ranking for various measures using average precision, NDCG, IDCG.

- Link to our code: [https://github.com/mihikadave/meta/tree/spd](https://github.com/mihikadave/meta/tree/spd)
readData()

getRandomPair() -> train()

generate random pair from nested hash-tables

train SPD

build_dataset_nodes() -> trainSVM()

use SPD

build pairwise dataset

train RankSVM

validate() -> test() -> evaluate()

evaluate trained classifiers

use trained model to rank real documents
Software usage

- We provide 2 command line arguments:
  - `dataset_path`: path to the dataset
  - `num_features`: number of features in the sample
- From the build folder run the following command:
  ```
  ./letor -dataset_path [PATH] -num_features [N_FEATURES]
  ```
- Running the above command will save the LETOR model and print out the MAP, NDCG values for the test data.
## Results: TD2003 dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.32</td>
<td>0.3441</td>
<td>0.3621</td>
<td>0.346</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.28</td>
<td>0.3246</td>
<td>0.3149</td>
<td>0.3122</td>
</tr>
<tr>
<td>FRank</td>
<td>0.3</td>
<td>0.2671</td>
<td>0.2468</td>
<td>0.269</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.347</strong></td>
<td><strong>0.3224</strong></td>
<td><strong>0.3179</strong></td>
<td><strong>0.3167</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Prec@1</th>
<th>Prec@3</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.32</td>
<td>0.2933</td>
<td>0.276</td>
<td>0.188</td>
<td>0.2628</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.28</td>
<td>0.28</td>
<td>0.232</td>
<td>0.1700</td>
<td>0.2274</td>
</tr>
<tr>
<td>FRank</td>
<td>0.3</td>
<td>0.2333</td>
<td>0.172</td>
<td>0.152</td>
<td>0.2031</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.3467</strong></td>
<td><strong>0.2933</strong></td>
<td><strong>0.2369</strong></td>
<td><strong>0.177</strong></td>
<td><strong>0.2374</strong></td>
</tr>
</tbody>
</table>
## Results: NP2003 dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.58</td>
<td>0.7654</td>
<td>0.7823</td>
<td>0.800</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.6</td>
<td>0.7636</td>
<td>0.7818</td>
<td>0.8068</td>
</tr>
<tr>
<td>FRank</td>
<td>0.54</td>
<td>0.7261</td>
<td>0.7595</td>
<td>0.776</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.56</strong></td>
<td><strong>0.6975</strong></td>
<td><strong>0.7176</strong></td>
<td><strong>0.7396</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Prec@1</th>
<th>Prec@3</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.58</td>
<td>0.2711</td>
<td>0.1707</td>
<td>0.092</td>
<td>0.6957</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.6</td>
<td>0.2689</td>
<td>0.1693</td>
<td>0.0940</td>
<td>0.7074</td>
</tr>
<tr>
<td>FRank</td>
<td>0.54</td>
<td>0.2533</td>
<td>0.168</td>
<td>0.090</td>
<td>0.6640</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.56</strong></td>
<td><strong>0.2678</strong></td>
<td><strong>0.1702</strong></td>
<td><strong>0.0925</strong></td>
<td><strong>0.6918</strong></td>
</tr>
</tbody>
</table>
## Results: HP2003 dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.6933</td>
<td>0.7749</td>
<td>0.7954</td>
<td>0.807</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.6667</td>
<td>0.792</td>
<td>0.8034</td>
<td>0.8171</td>
</tr>
<tr>
<td>FRank</td>
<td>0.6533</td>
<td>0.7432</td>
<td>0.778</td>
<td>0.797</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.6790</strong></td>
<td><strong>0.7443</strong></td>
<td><strong>0.7606</strong></td>
<td><strong>0.7857</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Prec@1</th>
<th>Prec@3</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.6933</td>
<td>0.3089</td>
<td>0.1987</td>
<td>0.104</td>
<td>0.7408</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.6667</td>
<td>0.3111</td>
<td>0.1987</td>
<td>0.1053</td>
<td>0.7330</td>
</tr>
<tr>
<td>FRank</td>
<td>0.6533</td>
<td>0.2889</td>
<td>0.1987</td>
<td>0.106</td>
<td>0.7095</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.6790</strong></td>
<td><strong>0.3101</strong></td>
<td><strong>0.1983</strong></td>
<td><strong>0.110</strong></td>
<td><strong>0.7373754</strong></td>
</tr>
</tbody>
</table>
## Results: OHSUMED dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@5</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.4958</td>
<td>0.4207</td>
<td>0.4164</td>
<td>0.414</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.4632</td>
<td>0.4555</td>
<td>0.4494</td>
<td>0.4302</td>
</tr>
<tr>
<td>FRank</td>
<td>0.53</td>
<td>0.4812</td>
<td>0.4588</td>
<td>0.443</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.4522</strong></td>
<td><strong>0.4594</strong></td>
<td><strong>0.4371</strong></td>
<td><strong>0.4098</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Prec@1</th>
<th>Prec@3</th>
<th>Prec@5</th>
<th>Prec@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ranking SVM</td>
<td>0.5974</td>
<td>0.5427</td>
<td>0.5319</td>
<td>0.486</td>
<td>0.4334</td>
</tr>
<tr>
<td>RankBoost</td>
<td>0.5576</td>
<td>0.5609</td>
<td>0.5447</td>
<td>0.4966</td>
<td>0.4411</td>
</tr>
<tr>
<td>FRank</td>
<td>0.6429</td>
<td>0.5925</td>
<td>0.5638</td>
<td>0.501</td>
<td>0.4439</td>
</tr>
<tr>
<td><strong>SPD</strong></td>
<td><strong>0.5923</strong></td>
<td><strong>0.5623</strong></td>
<td><strong>0.5286</strong></td>
<td><strong>0.4723</strong></td>
<td><strong>0.4279</strong></td>
</tr>
</tbody>
</table>
Results Overview

- Published results are similar to the results obtained by our implementation
Challenges

- Iterations in SGD
  - Currently fixed at 100,000
  - Good for the tested datasets, may not be optimal for other datasets
  - Need to fully utilize validation samples to tune SGD iteration number

- Batch learning v.s. online learning
  - Currently read whole training file and process
  - Need to use dataset view in MeTA
  - Can incorporate with online learning provided in MeTA
Future Work

- Finished implementing and testing the algorithm
- We can compute running time, memory
- Might be possible to further optimize by:
  - Tuning the number of iterations
    - Better use of validation samples
  - Train LETOR model using other classification algorithms in MeTA
    - libSVM, Logistic Regression, etc
  - Implement other optimization methods for SPD
    - Pegasos SVM, Passive-Aggressive Perceptron, ROMMA
- Other possible directions:
  - Feature extraction/ingestion in ranker package
    - Already in MeTA: TF, IDF, BM25, and other language model based features
    - Other document or query features like PageRank
References

- Sculley, D. "Large scale learning to rank." *NIPS Workshop on Advances in Ranking*, 2009.