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

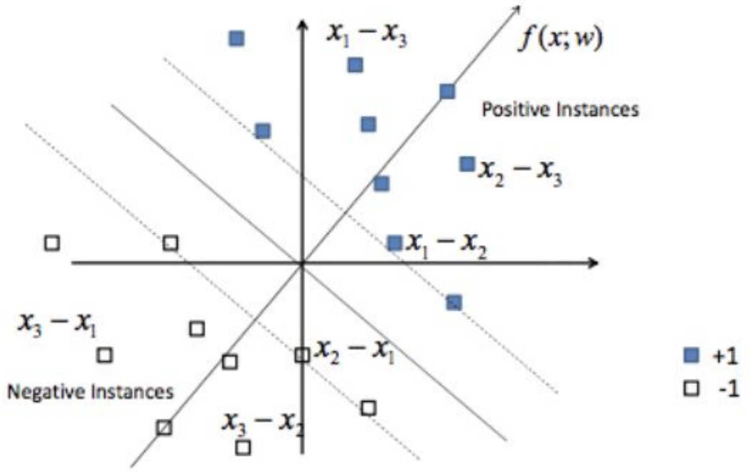
- Learning to rank is a machine learning task for ranking objects
- It can be employed in many areas
  - Information Retrieval (IR)
  - Natural Language Processing (NLP)
  - Data Mining (DM)
- Common applications include document retrieval, search, question answering, document summarization, machine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

$D_{index} \leftarrow \text{CreateIndex}(D)$
$\mathbf{w}_0 \leftarrow \emptyset$
for $i = 1$ to t **do**
    $((\mathbf{a}, y_a, q), (\mathbf{b}, y_b, q)) \leftarrow \text{GetRandomPair}(D_{index})$
    $\mathbf{x} \leftarrow (\mathbf{a} - \mathbf{b})$
    $y \leftarrow \text{sign}(y_a - y_b)$
    $\mathbf{w}_i \leftarrow \text{StochasticGradientStep}(\mathbf{w}_{i-1}, \mathbf{x}, y, i)$
end for
return $\mathbf{w}_t$



Transformation to pairwise SVM

- 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)

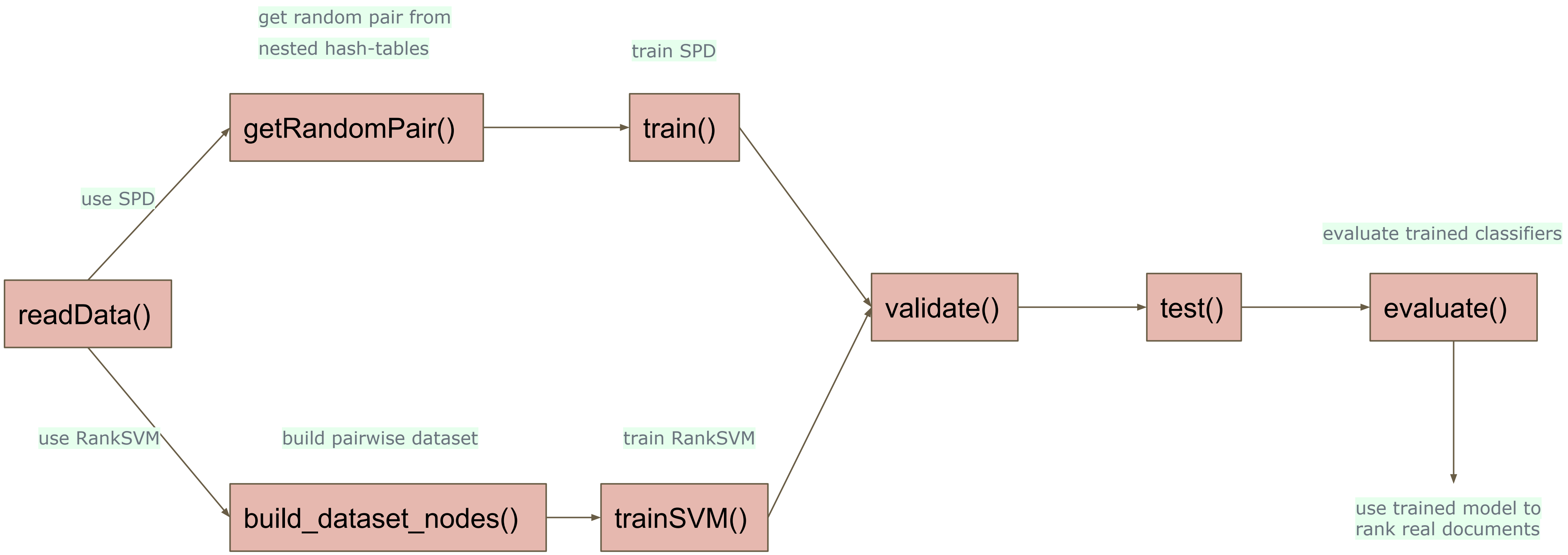
trainSVM()
Train SVM classifier with the pair and compute loss

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

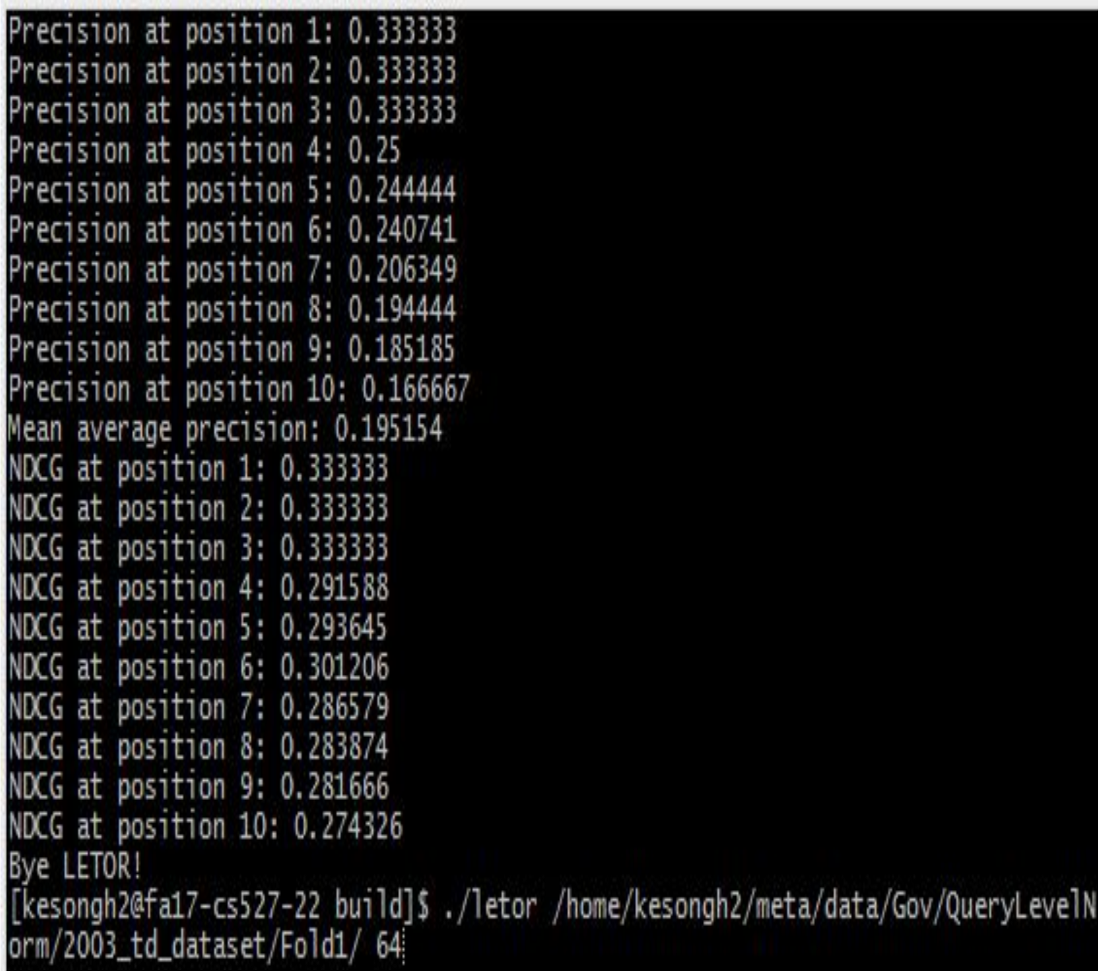

get random pair from
nested hash-tables
train SPD
getRandomPair()
train()
use SPD
evaluate trained classifiers
readData()
validate()
test()
evaluate()
use RankSVM
build pairwise dataset
train RankSVM
build_dataset_nodes()
trainSVM()
use trained model to
rank real documents

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

Algorithm	**NDCG@1**	**NDCG@3**	**NDCG@5**	**NDCG@10**
Ranking SVM	0.32	0.3441	0.3621	0.346
RankBoost	0.28	0.3246	0.3149	0.3122
FRank	0.3	0.2671	0.2468	0.269
SPD	***0.347***	***0.3224***	***0.3179***	***0.3167***

Algorithm	**Prec@1**	**Prec@3**	**Prec@5**	**Prec@10**	**MAP**
Ranking SVM	0.32	0.2933	0.276	0.188	0.2628
RankBoost	0.28	0.28	0.232	0.1700	0.2274
FRank	0.3	0.2333	0.172	0.152	0.2031
SPD	***0.3467***	***0.2933***	***0.2369***	***0.177***	***0.2374***

Results: NP2003 dataset

Algorithm	**NDCG@1**	**NDCG@3**	**NDCG@5**	**NDCG@10**
Ranking SVM	0.58	0.7654	0.7823	0.800
RankBoost	0.6	0.7636	0.7818	0.8068
FRank	0.54	0.7261	0.7595	0.776
SPD	***0.56***	***0.6975***	***0.7176***	***0.7396***

Algorithm	**Prec@1**	**Prec@3**	**Prec@5**	**Prec@10**	**MAP**
Ranking SVM	0.58	0.2711	0.1707	0.092	0.6957
RankBoost	0.6	0.2689	0.1693	0.0940	0.7074
FRank	0.54	0.2533	0.168	0.090	0.6640
SPD	***0.56***	***0.2678***	***0.1702***	***0.0925***	***0.6918***

Results: HP2003 dataset

Algorithm	NDCG@1	NDCG@3	NDCG@5	NDCG@10
Ranking SVM	0.6933	0.7749	0.7954	0.807
RankBoost	0.6667	0.792	0.8034	0.8171
FRank	0.6533	0.7432	0.778	0.797
SPD	***0.6790***	***0.7443***	***0.7606***	***0.7857***

Algorithm	Prec@1	Prec@3	Prec@5	Prec@10	MAP
Ranking SVM	0.6933	0.3089	0.1987	0.104	0.7408
RankBoost	0.6667	0.3111	0.1987	0.1053	0.7330
FRank	0.6533	0.2889	0.1987	0.106	0.7095
SPD	***0.6790***	***0.3101***	***0.1983***	***0.110***	***0.7373754***

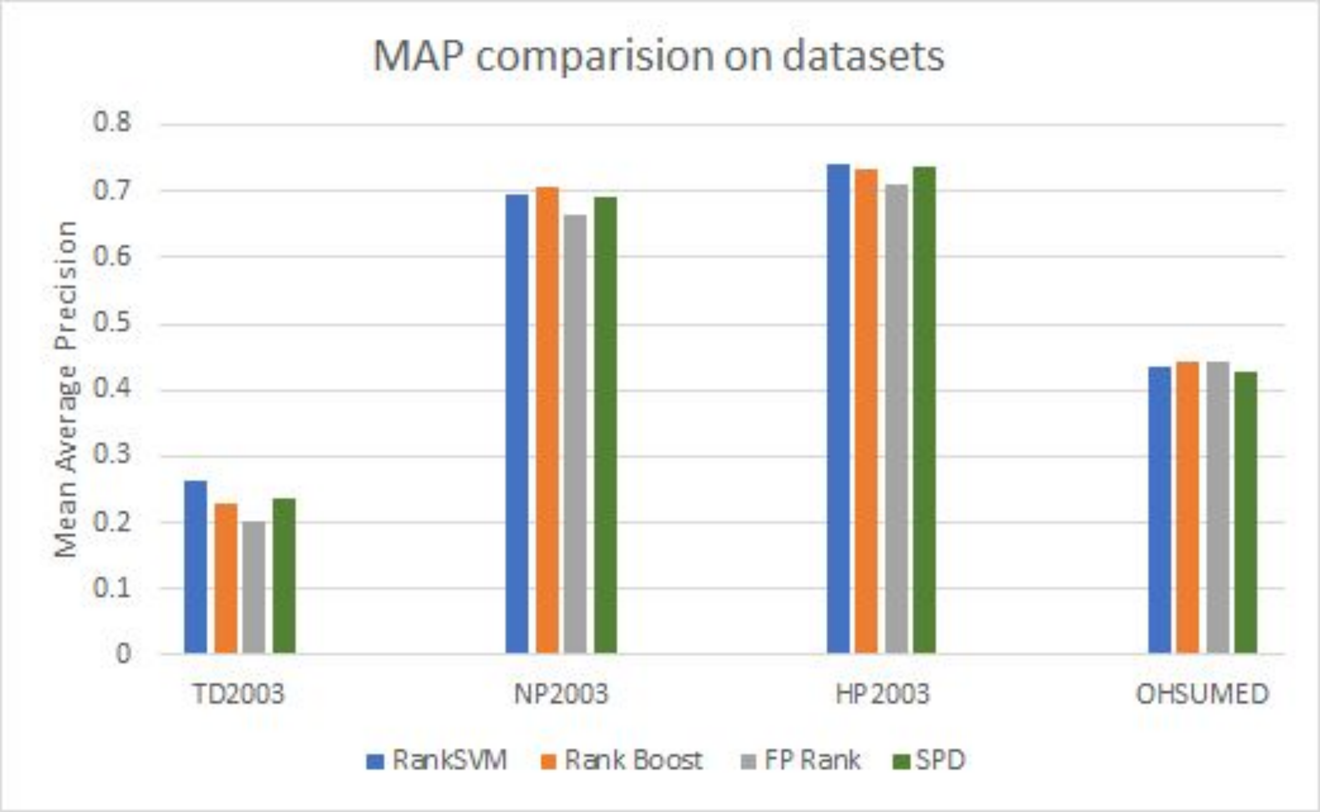
Results: OHSUMED dataset

Algorithm	**NDCG@1**	**NDCG@3**	**NDCG@5**	**NDCG@10**
Ranking SVM	0.4958	0.4207	0.4164	0.414
RankBoost	0.4632	0.4555	0.4494	0.4302
FRank	0.53	0.4812	0.4588	0.443
SPD	***0.4522***	***0.4594***	***0.4371***	***0.4098***

Algorithm	**Prec@1**	**Prec@3**	**Prec@5**	**Prec@10**	**MAP**
Ranking SVM	0.5974	0.5427	0.5319	0.486	0.4334
RankBoost	0.5576	0.5609	0.5447	0.4966	0.4411
FRank	0.6429	0.5925	0.5638	0.501	0.4439
SPD	***0.5923***	***0.5623***	***0.5286***	***0.4723***	***0.4279***

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.
- Liu, Tie-Yan, et al. "Letor: Benchmark dataset for research on learning to rank for information retrieval." *Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval*. Vol. 310. 2007.