Introduction to Information Retrieval

Session 16: **Ranking and Ranx**

Instructor: Behrooz Mansouri
Fall 2022, University of Southern Maine

Previous Session

In previous session we learned about:

✓ Machine Learning for Information Retrieval

✓ Point-wise Learning to Rank

✓ Pair-wise Learning to Rank

✓ List-wise Learning to Rank

Pairwise

Pairwise Approach

Pairwise approaches look at a **pair of documents** at a time in the loss function

- $f \rightarrow$ **partial order** → order → metric

Given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth

The goal for the ranker is to minimize the number of **inversions** in ranking

- Minimize the number of cases where the pair of results are in the wrong order relative to the ground truth

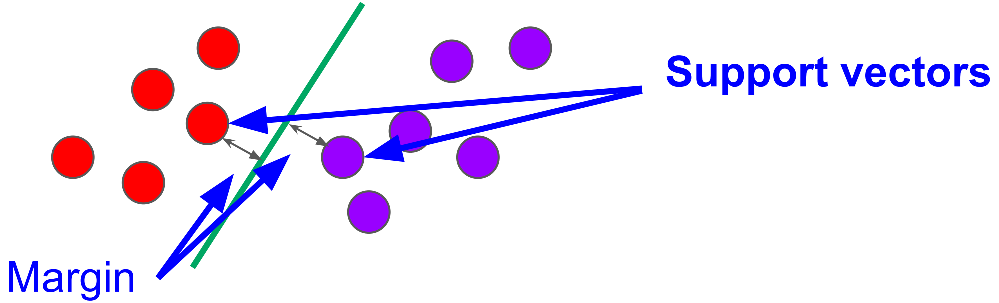
SVMRank - Ranking SVM (Thorsten Joachims, 2002)

Ranking SVM is a variant of the support vector machine algorithm, which is used to solve certain ranking problems

SVM or Support Vector Machine is a linear model for classification that creates a line or a hyperplane which separates the data into classes

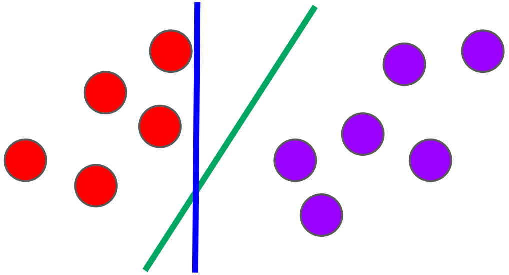
Support vectors are the data points nearest to the hyperplane (critical elements of a data set)

Margin: The distance between the hyperplane and the nearest data point from either set



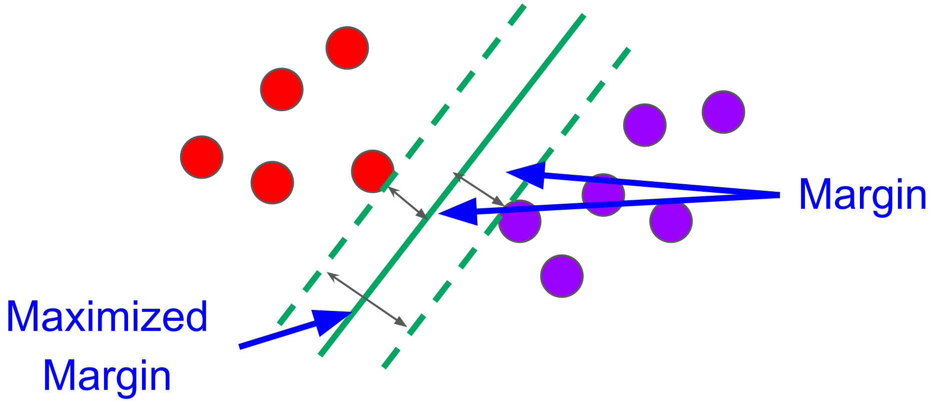
https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html

SVMRank



Which line is a better hyperplane?

SVMRank

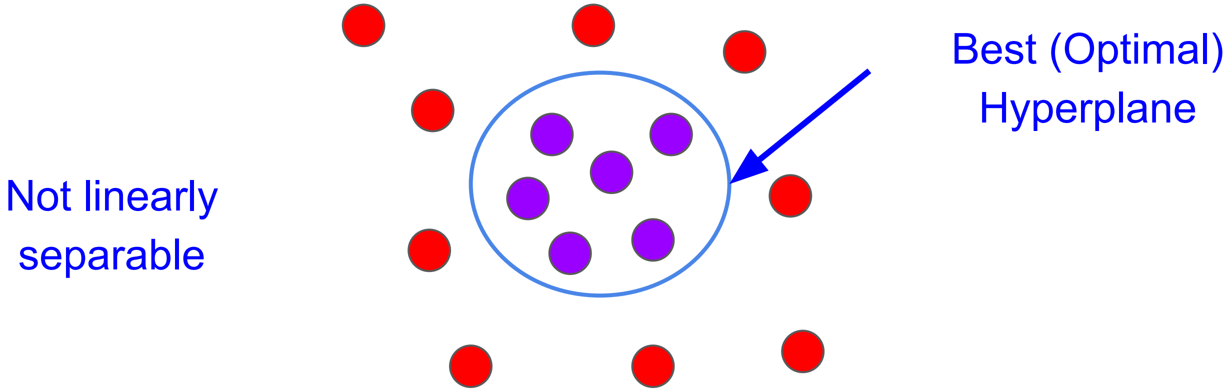


Which line is a better hyperplane? Our goal is to maximize the margin.

The hyperplane for which the margin is maximum is the optimal hyperplane;
A hyperplane in an n-dimensional Euclidean space is a flat, n-1 dimensional subset of that space that divides the space into two disconnected parts

SVM tries to make a decision boundary in such a way that the separation between the two classes (that street) is as wide as possible

SVMRank



Adding more dimension to decide the best hyperplane

Adding a new dimension can be computationally expensive (there can be many new dimensions)

- SVM doesn’t need the actual vectors to work its magic, it actually can get by only with the dot products between them (Kernel function)

SVMRank

Learning a linear ranking function $\vec{w}.\vec{d}_a$

- *w* is a weight vector adjusted by machine learning
- d_a is vector representation for query/document features
- Non-linear functions possible (using ‘kernel trick’)

Weights represent importance of features

- Learned from training data (i.e., preference pairs)
- Document score is dot product of weights + features

SVMRank

Learn w satisfying as many of these conditions as possible:

$$\begin{array}{lclr} \forall(d_i, d_j) \in r_1 & : & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} & \text{Query 1} \\ & & \ldots & \vdots \\ \forall(d_i, d_j) \in r_n & : & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} & \text{Query n} \end{array}$$

For each binary preference, the document that should rank higher has a higher score (first document d_i in each preference pair)

Idea: use a binary classifier based on the difference between rank scores to identify correctly ordered preference pairs ('first doc more relevant')

- Each training sample becomes (d_i – d_j), with label 'd_i more relevant'

SVMRank: Optimization Model

We construct a vector of features $\psi_j = \psi(d_j, q)$ for each document/query pair

For two documents d_i and d_j, we then form the vector of feature differences

$$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$$

$$\vec{w}^{\mathrm{T}}\Phi(d_i, d_j, q) > 0 \quad \text{iff} \quad d_i > d_j$$

Build a classifier that:

Find $\vec{w}$, and $\xi_{i,j} \geq 0$ such that:

- $\frac{1}{2}\vec{w}^{\mathrm{T}}\vec{w} + C\sum_{i,j}\xi_{i,j}$ is minimized
- and for all $\{\Phi(d_i, d_j, q) : d_i > d_j\}$, $\vec{w}^{\mathrm{T}}\Phi(d_i, d_j, q) \geq 1 - \xi_{i,j}$

SVMRank: Optimization Model

$$
\begin{aligned}
minimize: \quad & \frac{1}{2}\vec{w}.\vec{w} + C\sum \xi_{i,j,k} \\
subject\ to: \quad & \\
\forall(d_i, d_j) \in r_1 \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + \underline{1 - \xi_{i,j,1}} \\
& \ldots \\
\forall(d_i, d_j) \in r_n \quad : \quad & \vec{w}.\vec{d_i} > \vec{w}.\vec{d_j} + \underline{1 - \xi_{i,j,n}} \\
& \forall i \forall j \forall k : \xi_i, j, k \geq 0
\end{aligned}
$$

Hinge loss for every preference pair

- ξ (Greek letter xi), are ‘slack variables,’ which are non-negative hinge loss values for weight vector w (ξ_i denotes how misclassified instance i)
- C is a scaling for error weights, used to prevent overfitting (often 1.0 – can be fit using grid search)

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

RankNet and LambdaRank

RankNet was originally developed using neural nets, but the underlying model can be different and is not constrained to just neural nets. The cost function for RankNet aims to minimize the number of inversions in ranking

- Use Neural Network as model, and gradient descent as algorithm, to optimize the cross-entropy loss
- Evaluate on single documents: output a relevance score for each document w.r.t. a new query

Burgess et. al. found that during RankNet training procedure, you don't need the costs, only need the gradients (λ) of the cost with respect to the model score. You can think of these gradients as little arrows attached to each document in the ranked list, indicating the direction we'd like those documents to move (LambdaRank)

Burges, Christopher JC, Robert Ragno, and Quoc Viet Le. "Learning to rank with nonsmooth cost functions." NIPS. Vol. 6. 2006
Burges, Christopher JC. From ranknet to lambdarank to lambdamart: An overview. Learning, 11(23-581), 2010

Pairwise Approaches

Pros

- One step closer to ranking documents compared to predicting class labels

Cons

- Some queries have more query pairs than others
- Still does not optimize for IR measures
- Rank ignorant — (d1 > d2) does not encode which ranks are being compared: Rank 1 vs Rank 2 or Rank 1 vs Rank 1000

Listwise

Listwise Approach

Directly optimize what is used to evaluate the ranking results

- Alternatively Minimize loss for the permutation of the list

AdaRank: Use IR measure to iteratively update the distribution
LambdaMART: Combines LambdaRank and MART (Multiple Additive Regression Trees)

While MART uses gradient boosted decision trees for prediction tasks, LambdaMART uses gradient boosted decision trees using a cost function derived from LambdaRank for solving a ranking task

On experimental datasets, LambdaMART has shown better results than LambdaRank and the original RankNet

Listwise Approach

Advantages

- Uses all documents associated with a query as the learning instance
- Rank position is visible to the loss function

Problems

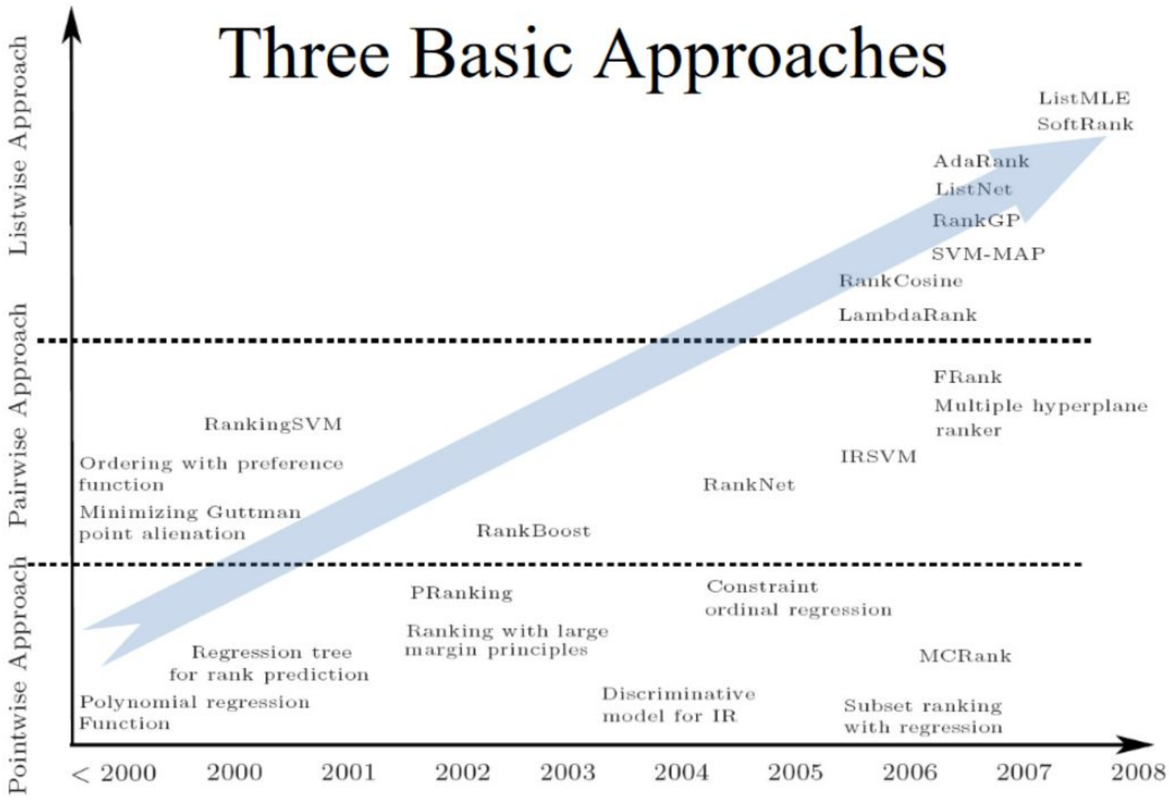
- Complexity demands more training data

What is the Best Approach?

LETOR makes it possible to perform fair comparisons among different learning to rank methods.

Empirical studies on LETOR have shown that the listwise ranking algorithms seem to have certain advantages over other algorithms, especially for top positions of the ranking result, and the pairwise ranking algorithms seem to outperform the pointwise algorithms

Three Basic Approaches


Three Basic Approaches
Listwise Approach
Pairwise Approach
Pointwise Approach
ListMLE
SoftRank
AdaRank
ListNet
RankGP
SVM-MAP
RankCosine
LambdaRank
FRank
Multiple hyperplane ranker
RankingSVM
IRSVM
Ordering with preference function
RankNet
Minimizing Guttman point alienation
RankBoost
PRanking
Constraint ordinal regression
Ranking with large margin principles
Regression tree for rank prediction
MCRank
Discriminative model for IR
Polynomial regression Function
Subset ranking with regression
< 2000
2000
2001
2002
2003
2004
2005
2006
2007
2008

Unsupervised Fusion

Unsupervised Reranking Methods

So far, we learned about the supervised ranking models
What if there is not enough training data?

Unsupervised re-ranking methods combine results based on score or rank or both

https://github.com/AmenRa/ranx

https://github.com/diegoches/PyRanker

Comb*

Use score of the document on the different lists as the main ranking factor:

- This can be the Retrieval Status Value of the retrieval model

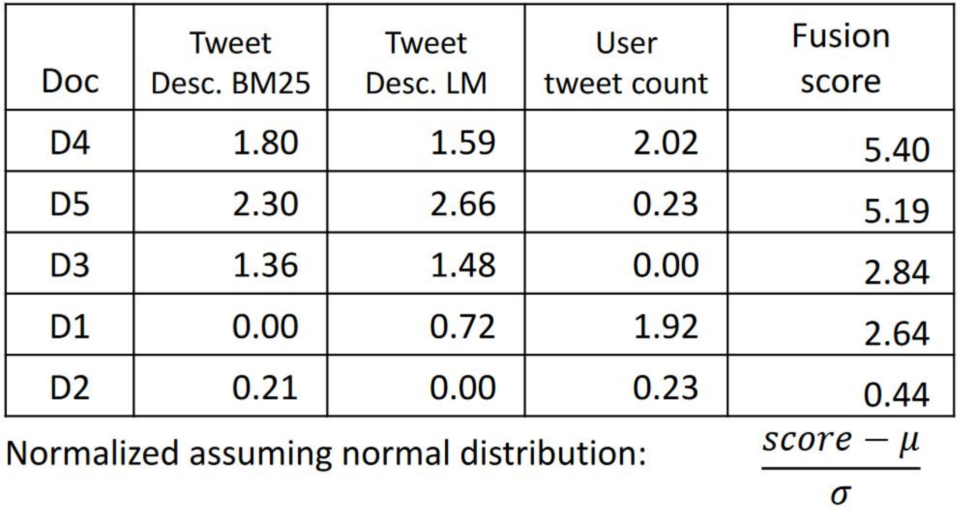
$$CombMAX(d) = \max\{s_0(d), \dots, s_n(d)\}$$

$$CombMIN(d) = \min\{s_0(d), \dots, s_n(d)\}$$

$$CombSUM(d) = \sum_i s_i(d)$$

CombSUM example

CombSUM is used by Lucene to combine results from multi-field queries:



Doc	Tweet Desc. BM25	Tweet Desc. LM	User tweet count	Fusion score
D4	1.80	1.59	2.02	5.40
D5	2.30	2.66	0.23	5.19
D3	1.36	1.48	0.00	2.84
D1	0.00	0.72	1.92	2.64
D2	0.21	0.00	0.23	0.44

Normalized assuming normal distribution: $\frac{score - \mu}{\sigma}$

Lucene already normalizes scores returned by retrieval models

- But scores may not follow normal distribution or be biased on small samples (e.g. 1000 documents retrieved by Lucene)

CombMNZ

CombMNZ multiplies the number of ranks where the document occurs by the sum of the scores obtained across all lists

$$CombMNZ(d) = |\{i|d \in Rank_i\}| \cdot \sum_i s_i(d)$$

Despite normalization issues common in score-based methods, CombMNZ is competitive with rank-based approaches

Reciprocal Rank Fusion (RRF)

The reciprocal rank fusion weights each document with the inverse of its position on the rank

- Favors documents at the “top” of the rank
- Penalizes documents below the “top” of the rank

$$RRFscore(d) = \sum_{i} \frac{1}{k + r_i(d)},$$

where k = 60

Score-based $$RRFscore(d) = \sum_{i} \frac{s_i(d)}{k + r_i(d)},$$

where k = 60

Ranx

Unsupervised Reranking Methods

ranx is a library of fast ranking evaluation metrics implemented in Python, leveraging Numba for high-speed vector operations and automatic parallelization

The link for ranx GitHub:
https://github.com/AmenRa/ranx

The link for the notebooks:
https://github.com/AmenRa/ranx/tree/master/notebooks

Unsupervised Reranking Methods

ranx is a library of fast ranking evaluation metrics implemented in Python, leveraging Numba for high-speed vector operations and automatic parallelization

The link for ranx GitHub:
https://github.com/AmenRa/ranx

The link for the notebooks:
https://github.com/AmenRa/ranx/tree/master/notebooks

Installing ranx command:

```
! pip install ranx
```

Evaluation with ranx

Note: First time you run the code it would take some time as they must be compiled first!

```
from ranx import Qrels, Run, evaluate



qrels = Qrels.from_file("qrel_arqmath", kind="trec")
run = Run.from_file("Runs/prim_DPRL_task1_auto_both_A.tsv", kind="trec")

print(evaluate(qrels, run, "precision@10"))
temp = evaluate(qrels, run, ["map@100", "mrr@10", "ndcg@10"]) # temp is a dictionary
#per query results
evaluate(qrels, run, ["map@100", "mrr@10", "ndcg@10"], return_mean=False)
```

https://github.com/AmenRa/ranx/blob/master/notebooks/3_evaluation.ipynb

Comparison with ranx

When you compare two models, it is essential to provide statistic test!

```
from ranx import compare

qrels = Qrels.from_file("qrel_arqmath", kind="trec")
run_1 = Run.from_file("Runs/prim_DPRL_task1_auto_both_A.tsv", kind="trec")
run_2 = Run.from_file("Runs/prim_MathDowsers-task1-alpha05-auto-both-P.tsv", kind="trec")
print(evaluate(qrels, run_1, "precision@10"))
print(evaluate(qrels, run_2, "precision@10"))
# Compare different runs and perform Two-sided Paired Student's t-Test
report = compare(
   qrels=qrels,
   runs=[run_1, run_2],
   metrics=["precision@10", "ndcg@10"],
   max_p=0.05,  # P-value threshold
   stat_test="student"
)

report
```

superscripts denote statistical significance differences

https://github.com/AmenRa/ranx/blob/master/notebooks/5_fusion.ipynb

Fusion with ranx

```python
from ranx import fuse, evaluate

run_1 = Run.from_file("Runs/prim_DPRL_task1_auto_both_A.tsv", kind="trec")
run_2 = Run.from_file("Runs/prim_MathDowsers-task1-alpha05-auto-both-P.tsv", kind="trec")
qrels = Qrels.from_file("qrel_arqmath", kind="trec")

combined_run = fuse(
        runs=[run_1, run_2],
        norm="min-max",  # Default normalization strategy
        method="mnz",
    )

print(combined_run.name, evaluate(qrels, combined_run, "ndcg@100"))
print(run_2.name, evaluate(qrels, run_2, "ndcg@100"))
print(run_1.name, evaluate(qrels, run_1, "ndcg@100"))

combined_run.save("combMNZ.tsv", kind="trec")
```

https://github.com/AmenRa/ranx/blob/master/notebooks/5_fusion.ipynb

WHAT HAVE
YOU LEARNED?

Summary

Today we learned about:

✓ Learning to rank

✓ PointWise, PairWise, Listwise

✓ SVMRank

✓ Unsupervised Fusion Methods

Next Session

Learning to Rank

Classification and Clustering

To do:

- Reading: Chapters 15 of Manning’s book
- Work on Project Part 2