Data Mining
Lecture 13: Learning to Rank

Jo Houghton

ECS Southampton

May 10, 2019

Learning to Rank - Introduction

What is “Learning to Rank”?

Definitions:

- Any machine learning used for ranking problem - broad
- Machine learning for ranking of *objects* given *subject* - narrow

I will use the second definition

Much of this talk is based on tutorials Hang Li ACML 2009 and Tie-Yan Liu WWW 2009

Learning to Rank - Introduction

Why rank?

- Document Search
- Recommender Systems
- Machine Translation
- Essay Scoring

Learning to Rank - Introduction

Why rank?

- Document Search
- Recommender Systems
- Machine Translation
- Essay Scoring

Any task where you need an ordering over items in a collection

Learning to Rank - Introduction

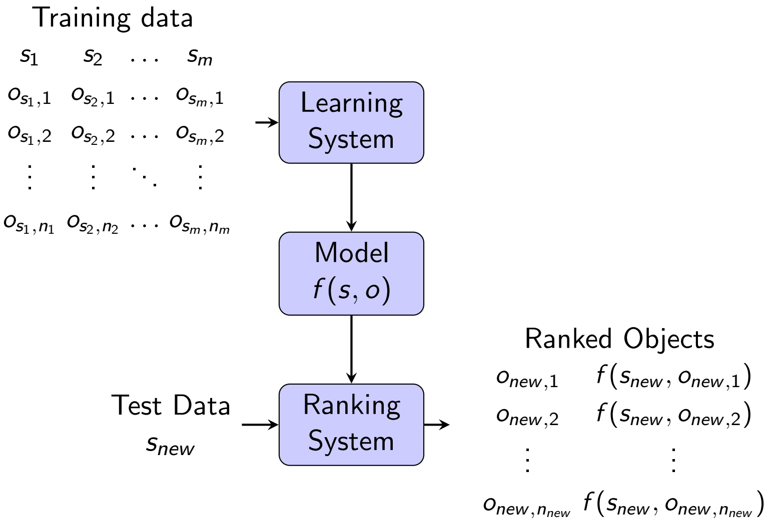
Rank or sort objects given a *feature vector*

Like classification, goal is to assign one of k labels to a new instance. However, *absolute* class is not needed

Like regression, the k labels have order, so you are assigning a value. However this value is not *absolute*

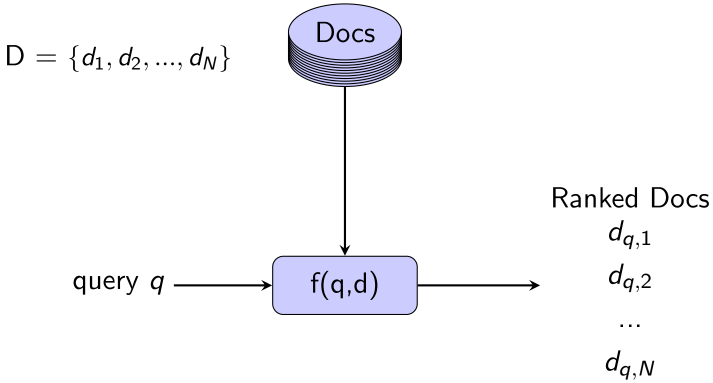
Learning to Rank - Introduction

In general:



Learning to Rank - Introduction

Information Retrieval: ranking documents in order of *relevance*, *importance* and *preference* given a query



Learning to Rank - Introduction

Information Retrieval
In this case, the feature vectors correspond to features of a *query-document* pair, the subject is the query and the object is the document

There are many possible features:

- Number of query terms in document - Relevance
- PageRank - Importance
- BM25 - Relevance
- ...

Features are functions of both the query and the document

Learning to Rank - Introduction

Problem Formulation

Given:

- Set of input vectors $\{x_i\}_{i=1}^{n}$
- Labels $\{y_i\}_{i=1}^{n}$ where $Y = \{1, 2, ..., N\}$ specifying an order

Find function f to give the ranking, minimising some cost C

Learning to Rank - Ranking Metrics

There are a good number of possible cost functions.

- DCG - Discounted Cumulative Gain
- NDCG - Normalised DCG
- MAP - Mean Average Precision
- MRR - Mean Reciprocal Rank
- WTA - Winner Takes All
- Kendall's Tau

The cost function chosen will depend on what you are trying to rank.

Learning to Rank - Ranking Metrics

DCG - Discounted Cumulative Gain

$$DCG_i = \sum_{i=1}^{m} c_i y_i$$

Where c_i is a predefined sequence of non-increasing non-negative discount factors, $c = 1/\log(i+1)$ when $i > k$ and $c = 0$ otherwise [1]

Focusses on quality of ranking at the top of the list.

[1] Jarvelin and Kekalainen, 2002

Learning to Rank - Ranking Metrics

NDCG - Normalised Discounted Cumulative Gain

$$NDCG = \frac{DCG}{IDCG} \quad DCG_j = \sum_{i=1}^{j} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

Where DCG_j is the discounted cumulative gain at rank j, an r_i is the relevance of the result at i, usually from $3, 2, 1$

For example:

i	True Order	True r_i	DCG_j	$NDCG_i$	$\bar{r}_i$	DCG_i	$NDCG_i$
1	d_4	3	7.0	1	2	3.0	0.43

Learning to Rank - Ranking Metrics

NDCG - Normalised Discounted Cumulative Gain

$$NDCG = \frac{DCG}{IDCG} \quad DCG_j = \sum_{i=1}^{j} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

Where DCG_j is the discounted cumulative gain at rank j, an r_i is the relevance of the result at i, usually from $3, 2, 1$

For example:

i	True Order	True r_i	DCG_j	$NDCG_i$	$\bar{r}_i$	DCG_i	$NDCG_i$
1	d_4	3	7.0	1	2	3.0	0.43
2	d_3	3	11.4	1	3	7.4	0.65

Learning to Rank - Ranking Metrics

NDCG - Normalised Discounted Cumulative Gain

$$NDCG = \frac{DCG}{IDCG} \quad DCG_j = \sum_{i=1}^{j} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

Where DCG_j is the discounted cumulative gain at rank j, an r_i is the relevance of the result at i, usually from $3, 2, 1$

For example:

i	True Order	True r_i	DCG_j	$NDCG_i$	$\bar{r}_i$	DCG_i	$NDCG_i$
1	d_4	3	7.0	1	2	3.0	0.43
2	d_3	3	11.4	1	3	7.4	0.65
3	d_2	2	12.9	1	2	8.9	0.69

Learning to Rank - Ranking Metrics

NDCG - Normalised Discounted Cumulative Gain

$$NDCG = \frac{DCG}{IDCG} \quad DCG_j = \sum_{i=1}^{j} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

Where DCG_j is the discounted cumulative gain at rank j, an r_i is the relevance of the result at i, usually from $3, 2, 1$
For example:

i	True Order	True r_i	DCG_j	$NDCG_i$	$\bar{r}_i$	DCG_i	$NDCG_i$
1	d_4	3	7.0	1	2	3.0	0.43
2	d_3	3	11.4	1	3	7.4	0.65
3	d_2	2	12.9	1	2	8.9	0.69
4	d_1	2	14.2	1	3	11.9	0.84

Learning to Rank - Ranking Metrics

MAP - Mean Average Precision. Only looks at top j results.

Precision at position j for query q:

$$P_j = \frac{\text{number relevant docs in top } j \text{ results}}{j}$$

Average precision for query q

$$AP_q = \frac{\sum_j P_j . rel(j)}{\text{number relevant docs}}$$

E.g. for a query giving the top 5 results 1, 0, 1, 0, 1

$$AP = \frac{(\frac{1}{1} + \frac{2}{3} + \frac{3}{5})}{3} \approx 0.76$$

MAP - average precision for each query averaged over all Q queries

$$MAP = \frac{\sum_q^Q AP_q}{Q}$$

Learning to Rank - Ranking Metrics

MRR - Mean Reciprocal Rank

Considers only rank position of first relevant document. Reciprocal rank for query q RR_q:

$$RR_q = \frac{1}{K}$$

Mean Reciprocal Rank:

$$MRR_q = \frac{\sum_q^Q}{Q}$$

Learning to Rank - Types of Ranking

Machine learning ranking algorithms are categorised by how they are judged

- Pointwise - treats each object in isolation
  Can use Regression, Classification

Learning to Rank - Types of Ranking

Machine learning ranking algorithms are categorised by how they are judged

- Pointwise - treats each object in isolation
  Can use Regression, Classification
- Pairwise - treats objects in pairs
  RankNet, Frank, RankBoost, Ranking SVM

Learning to Rank - Types of Ranking

Machine learning ranking algorithms are categorised by how they are judged

- Pointwise - treats each object in isolation
  Can use Regression, Classification
- Pairwise - treats objects in pairs
  RankNet, Frank, RankBoost, Ranking SVM
- Listwise - assesses the ordering of the whole list at once
  Tries to directly optimise the ranking metric

Learning to Rank - Pointwise

Task: Learn a ranking function to give an *absolute* score

	Regression	Classification	Ordinal Regression
Input	$\boldsymbol{x}$	$\boldsymbol{x}$	$\boldsymbol{x}$

Learning to Rank - Pointwise

Task: Learn a ranking function to give an *absolute* score

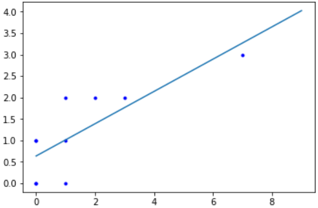
	Regression	Classification	Ordinal Regression
Input	$\boldsymbol{x}$	$\boldsymbol{x}$	$\boldsymbol{x}$
Model	$f(\boldsymbol{x})$	$f(\boldsymbol{x})$	$f(\boldsymbol{x})$

Learning to Rank - Pointwise

Task: Learn a ranking function to give an *absolute* score

	Regression	Classification	Ordinal Regression
Input	$\boldsymbol{x}$	$\boldsymbol{x}$	$\boldsymbol{x}$
Model	$f(\boldsymbol{x})$	$f(\boldsymbol{x})$	$f(\boldsymbol{x})$
Output	Real Number $y = f(\boldsymbol{x})$	Category $y = sign(f(\boldsymbol{x}))$	Ordered Category $y = thresh(f(\boldsymbol{x}))$

Learning to Rank - Pointwise



A simple example in 1D, minimising error.
Cossock and Zhang, COLT 2006: use least squares difference between relevance degree and estimated relevance degree to learn ranking function

Learning to Rank - Pointwise

Also:

- McRank, Li *et al* NIPS2007 Multi class classification to learn ranking, combine outputs of classifiers
- Pranking, Krammer and Singer, NIPS 2002 Perceptron ranking, ordinal regression
- Ranking with Large Margin Principles Shashua and Levin NIPS 2002, SVM

Learning to Rank - Pointwise

Perceptron Ranking

Algorithm 1: Perceptron Algorithm

Data: X, $\boldsymbol{y}$, $runs$, η, N

$w = rand$ vector dependent of X feature vector length;

$i = 0$;

for i =*1 to runs* **do**

    $\hat{\boldsymbol{y}} = X.w$;

    $err = \text{count}(\hat{\boldsymbol{y}}.\boldsymbol{y} > 0)$;

    if $err = 0$ **then**

        return w;

    end

    $r_{idx} = \text{randint}(0, \text{N})$;

    if $\hat{\boldsymbol{y}}[r_{idx}] \times \boldsymbol{y}[r_{idx}] < 0$ **then**

        $w = w + \eta.y[r_{idx}].X[r_{idx}]$;

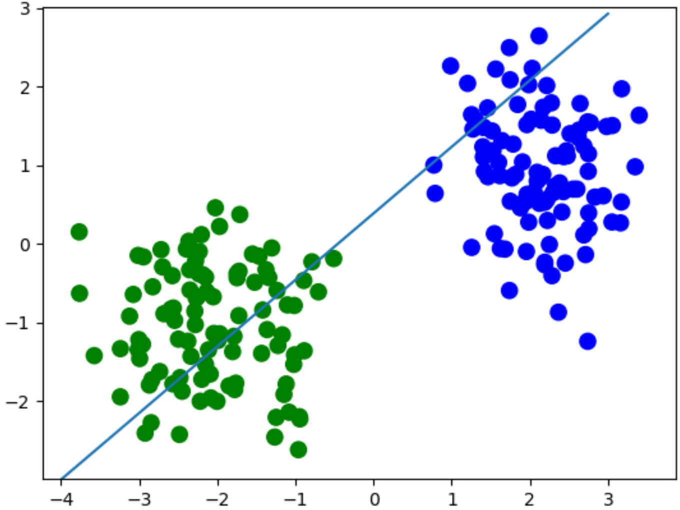
    end

end

Failed to converge ;

Learning to Rank - Pointwise

Movie showing mechanics of perceptron algorithm



Learning to Rank - Pointwise

The Model:

- Input: Feature vectors $\boldsymbol{X}$, ranks $\boldsymbol{y}$ where $\boldsymbol{y} \in \{1, 2, ..., k\}$
- Output: $f(\boldsymbol{X}) = f(\boldsymbol{w.X}, \boldsymbol{b}) \sim \mathcal{N} \in \{1, 2, ..., k\}$
  Where:
  - $\boldsymbol{w}$ is a weights vector
  - $\boldsymbol{b}$ is ranking thresholds, $b_1 \leq b_2 \leq ... \leq b_k = \infty$
  - $f(\boldsymbol{w.X}, \boldsymbol{b})$ takes the form $\min_{r \in \{1,2,\ldots,k\}}\{r : \boldsymbol{w.X} - b_r < 0\}$
- Loss: $\sum_{t=1}^{T} |\hat{\boldsymbol{y}}^t - \boldsymbol{y}|$ for run t of T runs

Learning to Rank - Pointwise

Update Rule:

- Given X_i and y_i input, $f(\boldsymbol{w}.X_i, \boldsymbol{b}) = y_i$ if:
  - $\forall r \in \{1, ..., y_i - 1\}, \boldsymbol{w}.X_i > b_r$
  - $\forall r \in \{y_i, ..., k - 1\}, \boldsymbol{w}.X_i < b_r$
- So *True* ranking vector is $+1$ if $r < y_i$ otherwise -1, i.e. $\{y_1, ..., y_i, y_{i+1}, ..., y_k\}$ gives $\{+1, ..., +1, -1, ..., -1\}$
- If $\exists r : y_r.(\boldsymbol{w}X_i - b_r) \leq 0$ then move values of $\boldsymbol{w}X_i$ and b_r towards each other:
  - $b_r = b_r - y_r$
  - $\boldsymbol{w} = \boldsymbol{w} + \left(\sum_{r:\ y_r.\hat{y}_r \leq 0} y_r\right).X_i$, i.e. only sum over the ranks where there was an error

Learning to Rank - Pointwise

Perceptron Ranking

Algorithm 2: Perceptron Ranking Algorithm

Data: $\boldsymbol{X}$, $\boldsymbol{y}$, T, N, k

$\boldsymbol{w}$ = *rand* vector dependent of $\boldsymbol{X}$ feature vector length;

$b_1, ..., b_{k-1} = 0$, $b_k = \infty$;

for t =*1 to* T **do**

  i = randint(0, N);

  $\hat{y}_i = \min_{r \in \{1,2,...,k\}} \{r : \boldsymbol{w}.X_i - b_r < 0\}$;

  if $\hat{y}_i \neq y_i$ **then**

    for $r = 1$ *to* $k - 1$ **do** **if** $y_i \leq r$ **then** $trv_r = -1$;

    else $trv_r = +1$;

    for $r = 1$ *to* k **do** **if** $(\boldsymbol{w}.X_i - b_r) trv_r \leq 0$ **then**

     $\tau_r = trv_r$ **else** $\tau_r = 0$;

    $\boldsymbol{w} = \boldsymbol{w} + (\sum_r \tau_r) X_i$;

    for $r = 1$ *to* $k - 1$ **do** $b_r = b_r - \tau_r$;

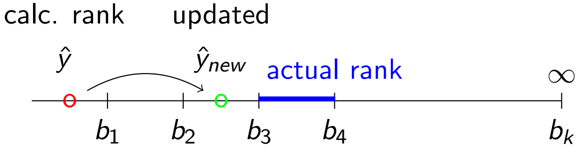
  end

end

Learning to Rank - Pointwise

To update $\boldsymbol{w}$: shift towards actual rank

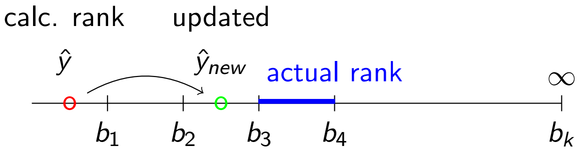
$$\boldsymbol{w} = \boldsymbol{w} + (\sum_r \tau_r) X_i$$



Learning to Rank - Pointwise

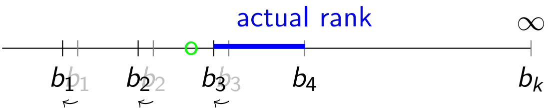
To update $\boldsymbol{w}$: shift towards actual rank

$$\boldsymbol{w} = \boldsymbol{w} + (\sum_r \tau_r) X_i$$



To update $\boldsymbol{b}$: move those intervals to make the actual rank closer

$$b_r = b_r - tr v_r \quad \text{where rank is incorrect}$$

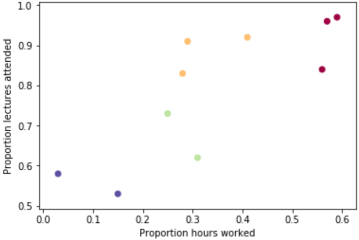


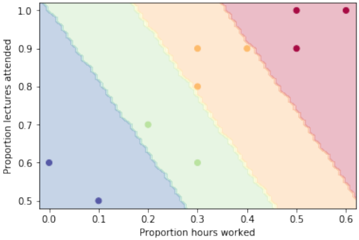
Learning to Rank - Pointwise

Example Perceptron Ranking:

Sample data:

Rank	$\boldsymbol{x_1}$	$\boldsymbol{x_2}$
2	0.4	0.9
2	0.3	0.8
3	0.2	0.7
3	0.3	0.6
1	0.5	1.0
2	0.3	0.9
1	0.6	1.0
4	0.1	0.5
4	0.0	0.6
1	0.5	0.9





Learning to Rank - Pointwise

Problems?

- Error from all the actual ranks is minimised
  This is not necessary, we only need relative order
- In IR, some queries have more matches than those with less
  This means the loss function can be dominated by queries with many matches
- Position of document in list is not visible to loss functions here
  There may be too much emphasis on irrelevant documents

Learning to Rank - Pairwise

	Learning	Ranking
Input	vector pair $\{\boldsymbol{x_i}, \boldsymbol{x_j}\}$	vector $\boldsymbol{x_k}$
Model	$f(\boldsymbol{x})$	$f(\boldsymbol{x})$
Output	$y_{ij} = sign(f(\boldsymbol{x}_i, \boldsymbol{x}_j))$	$\boldsymbol{y} = sort(\{f(\boldsymbol{x}_k)\}_{i=1}^{n})$

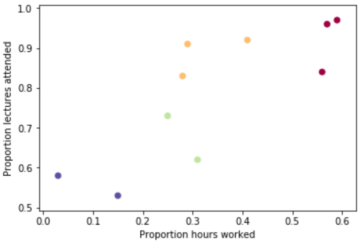
Minimises misranking of pairs of feature vectors.
The model learns to rank pairs of vectors, any binary classifier can be used

Learning to Rank - Pairwise

Transforming data in to vector pairs

Sample data:

#	Rank	x_1	x_2
a	2	0.41	0.92
b	2	0.28	0.83
c	3	0.25	0.73
d	3	0.31	0.62
e	1	0.57	0.96
f	2	0.29	0.91
g	1	0.59	0.97
h	4	0.15	0.53
i	4	0.03	0.58
j	1	0.56	0.84

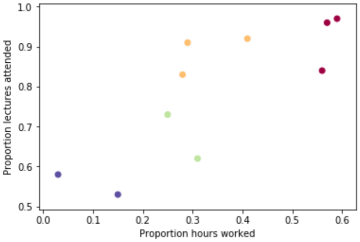


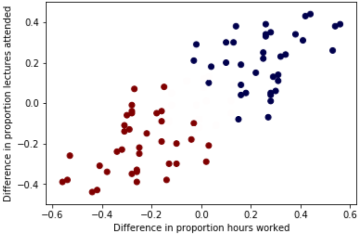
Learning to Rank - Pairwise

Transforming data in to vector pairs

Sample data:

Pair	Comp. Rank	x_1	x_2
a - b	0	0.13	0.09
a - c	1	0.16	0.19
a - d	1	0.10	0.30
a - e	-1	-0.16	-0.04
a - f	0	0.12	0.01
a - g	-1	-0.18	-0.05
a - h	1	0.26	0.39
a - i	1	0.38	0.34
a - j	-1	-0.15	0.08
b - a	0	-0.13	-0.09
⋮	⋮	⋮	⋮





Learning to Rank - Pairwise

Half the comparisons are redundant, as they are the opposite of the other.

We can just take cases where the comparative rank is 1 as half of the pair comparisons are duplicates of the other half.

The aim to to combine a set of ranking features such that the comparison $f(x_i) > f(x_j)$ means x_i is higher rank than x_j.

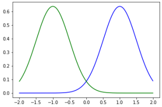
The function $f()$ is a linear function $\boldsymbol{w}.\boldsymbol{X}$

Learning to Rank - Pairwise

Bradley-Terry-Luce model
Assumes that there is an underlying parameter p_i that determines the ranking for item $\boldsymbol{x_i}$

$$P(\boldsymbol{x_i} > \boldsymbol{x_j}) = \frac{p_i}{p_i + p_j}$$

p_i could be skill for game rankings, relevance for information retrieval e.t.c.
Thurstone Model (1929) states that comparisons are made by drawing a variable with the intrinsic value as the mean and a normal distribution,



Learning to Rank - Pairwise

Bradley-Terry-Luce model
Iterative rank aggregation algorithm: finds underlying scores from pairwise comparisons
n items of interest, represented as $[n] = \{1, 2, ..., n\}$ Assume for each item $i \in [n]$ there is an associated score $w_i \in \mathbb{R}_+$. Vector $w \in \mathbb{R}_+$ is associated weight vector of all items.
Given a pair of items i and j $Y_i^l j = 1$ if i is preferred over j, and 0 otherwise during the l^{th} comparison for $1 \leq l \leq k$ where k is the total number of comparisons

$$P(Y_{ij}^l = 1) = \frac{w_i}{w_i + w_j}$$

i is compared to j with probability $\frac{d}{n}$

Learning to Rank - Pairwise

Random Walk approach

a_{ij} is fraction of times i preferred to j

$$a_{ij} = \frac{1}{k}\sum_{l=1}^{k} Y_{ij}^{l}$$

A random walk on a directed graph $G = ([n], E, A)$ where a pair i, j have an edge if they have been compared. Edge weights A are given by $A_{ij} = \frac{a_{ij}}{a_{ij}+a_{ji}}$

The random walk is represented by the transition matrix P where $P_{ij} = P(X_{t+1} = j | X_t = i)$

Learning to Rank - Pairwise

Rows and columns are normalised, so edge weights are scaled by $1/d_{max}$ where d_{max} is the maximum out-degree of a node.

$$P_{ij} = \begin{cases} \frac{1}{d_{max}} A_{ij} & \text{if } i \neq j \\ 1 - \frac{1}{d_{max}} \sum_{k \neq i} & \text{if } i = j \end{cases}$$

From an arbitrary starting distribution p_o (where $(p_o(i)) \in \mathbb{R}^n_+$) over $[n]$ the transition matrix is repeatedly applied

$$p_{t+1}^T = p_t^T P$$

The rank is then calculated by finding the stationary distribution $\pi = \lim_{t \to \infty} p_t$, which will converge to the top left eigenvector of P
The stationary distribution of the random walk is a fixed point of:

$$\pi(i) = \sum_j \pi(j) \frac{A_{ji}}{\sum_l A_{il}}$$

Item is high rank if preferred to many items, or other high rank items

Learning to Rank - Pairwise

Advantages:

- Better performance than pointwise
- Gives relative rank

Disadvantages:

- Does not optimise cost function normally used
- Ranking items at the top often more important than lower down

Learning to Rank - Pairwise

Other pairwise approaches:

- RankBoost (Freund *et al* JMLR 2003)
- RankingSVM (Herbrich *et al* 2000)
- FRank (Tsai *et al* SIGIR 2007)
- RankNet (Burges *et al* ICML 2005)
- Learning to Order (Cohen *et al* NIPS 1998)

Learning to Rank - Listwise

Two approaches:

- Directly optimise the metric used
- Minimise loss for the permutation of the list

Learning to Rank - Listwise

	Minimise Loss	Direct Optimisation
Input	Docs Set $X = \{x_j\}_{j=1}^m$	Docs Set $X = \{x_j\}_{j=1}^m$
Model	sort $f(\boldsymbol{X})$	$f(\boldsymbol{X})$
Output	Permutation π_y	Ordered categories $\boldsymbol{y} = \{y_j\}_{j=1}^m$

Learning to Rank - Listwise

Direct optimisation:
e.g. for information retrieval could be NDCG, or MAP
Unfortunately these measures are non-continuous, so not differentiable, so no gradient descent.
Approaches used so far:

- make cost function smooth - SoftRank
- optimise only smooth upper bound of metric - SVM-MAP
- use an algorithm that can deal with discontinuous metrics - AdaRank, RankGP

Learning to Rank - Listwise

Listwise loss minimisation:
Non-Trivial!
e.g.

- ListNet (Cao *et al* ICML 2007)
  Minimises KL divergence between permutation probability distributions, using a NN model and gradient descent
- ListMLE (Xia *et al* ICML 2008)
  uses MLE algorithm with NN model and SGD to maximise likelihood of permutation

Learning to Rank - Listwise

Advantages:

- rank position is visible to loss function
- uses all documents

Disadvantages:

- very high complexity, not practical
- position information is sometimes insufficient

Learning to Rank - Summary

Pointwise:

- Relatively simple
- Can give good results for easier problems
- But:
- Does more than necessary
- Gives absolute score when we only need a rank

Learning to Rank - Summary

Pointwise:

- Relatively simple
- Can give good results for easier problems
- But:
- Does more than necessary
- Gives absolute score when we only need a rank

Pairwise:

- Good Performance
- Solves the right problem
- But..
- Can be high complexity
- Sometimes need more than just rank

Learning to Rank - Summary

Pointwise:

- Relatively simple
- Can give good results for easier problems
- But:
- Does more than necessary
- Gives absolute score when we only need a rank

Pairwise:

- Good Performance
- Solves the right problem
- But..
- Can be high complexity
- Sometimes need more than just rank

Listwise:

- Better performance
- But..
- Very high complexity