Learning to Rank

Ronan Cummins and Ted Briscoe

Thursday, 14th January

Table of contents

1. Motivation
   - Applications
   - Problem Formulation
2. Approaches
   - Pointwise
   - Pairwise
   - Listwise
3. Challenges/Research Questions
   - Datasets
4. Conclusion

Applications of L2R

- Information Retrieval
- Collaborative Filtering
- Automated Text Scoring (Essay Scoring)
- Machine Translation
- Sentence Parsing

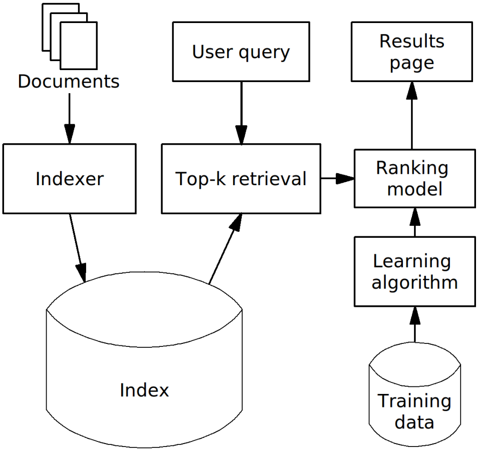
Applications of L2R

- Information Retrieval
- Collaborative Filtering
- Automated Text Scoring (Essay Scoring)
- Machine Translation
- Sentence Parsing
- Applicable to many tasks where you wish to specify an ordering over items in a collection

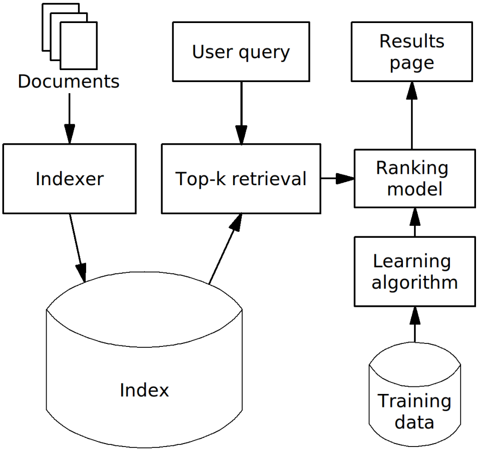
Difference with Other Learning Models

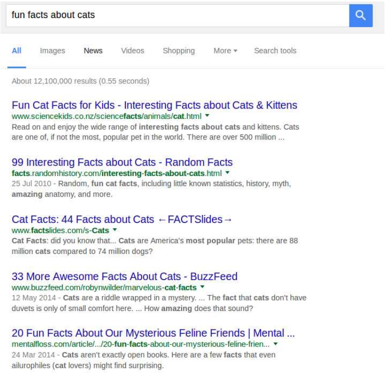
- No need to predict the absolute value of items (unlike regression)
- No need to predict the absolute class of items (unlike classification and ordinal regression)
- The relative ranking of items is all that is important (at least for information retrieval)

Example: Information Retrieval



Example: Information Retrieval





Example: Information Retrieval

- Information retrieval often involves ranking documents in order of *relevance*
- E.g. relevant, partially-relevant, non-relevant
- Assume that we can describe documents (items) using feature vectors $\vec{x}_i^q = \Phi(q, d_i)$ that correspond to features of the query-document pair:

Example: Information Retrieval

- Information retrieval often involves ranking documents in order of *relevance*
- E.g. relevant, partially-relevant, non-relevant
- Assume that we can describe documents (items) using feature vectors $\vec{x}_i^q = \Phi(q, d_i)$ that correspond to features of the query-document pair:

Example Features

- # of query keywords in document
- BM25 score
- document length
- page-rank
- sum of term-frequencies
- ...
- ...

Example of Input Vectors in $\mathcal{R}^N$

y_i	input vectors $\vec{x_i}$		
	$\vec{x}_{i1}$	$\vec{x}_{i2}$	$\vec{x}_{i3}$
3	7.0	9.2	3.2
2	2.0	9.2	4.1
0	2.0	3.5	0.2
2	2.0	9.2	11.2
1	3.0	5.3	2.2
0	0.0	3.2	0.5

Table : Sample Dataset

Problem Formulation

- Given a set of input vectors $\{\vec{x_i}\}_{i=1}^n$ and corresponding labels $\{y_i\}_{i=1}^n$ where $\mathcal{Y} = \{1, 2, 3, 4, ..l\}$ specifying a total order on the labels.
- Determine a function f that specifies a ranking over the vectors $\{\vec{x_i}\}_{i=1}^n$ such that f minimises some cost $\mathcal{C}$
- In general you would like to use a cost function $\mathcal{C}$ that is closely correlated to the most suitable measure of performance for the task
- This is not always easy

Three Common Approaches

- Pointwise - Regression, Classification, Ordinal regression (items to be ranked are treated in isolation)
- Pairwise - Rank-preference models (items to be ranked are treated in pairs)
- Listwise - Treat each list as an instance. Usually tries to directly optimise the evaluation measure (e.g. mean average precision)

Three Common Approaches

- Pointwise - Regression, Classification, Ordinal regression (items to be ranked are treated in isolation)
- Pairwise - Rank-preference models (items to be ranked are treated in pairs)
- Listwise - Treat each list as an instance. Usually tries to directly optimise the evaluation measure (e.g. mean average precision)
- We'll just consider linear functions of the form $f(\vec{x}) = < \vec{x}, \vec{w} > + b$

Pointwise Outline

General Criteria

The ranking function f learns to assign an *absolute* score (categories) to each item in isolation.

[1] Adapted from [Hang(2009)Hang]

Pointwise Outline

General Criteria

The ranking function f learns to assign an *absolute* score (categories) to each item in isolation.

	Regression	Classification	Ordinal Regression
Input	input vector $\vec{x}$		
Output	Real Number $y = f(\vec{x})$	Category $y = sign(f(\vec{x}))$	Ordered Category $y = thresh(f(\vec{x}))$
Model	Ranking Function $f(\vec{x})$		
Loss	Regression Loss	Classification Loss	Ordinal Regression Loss

Table : Learning in Pointwise approaches[1]

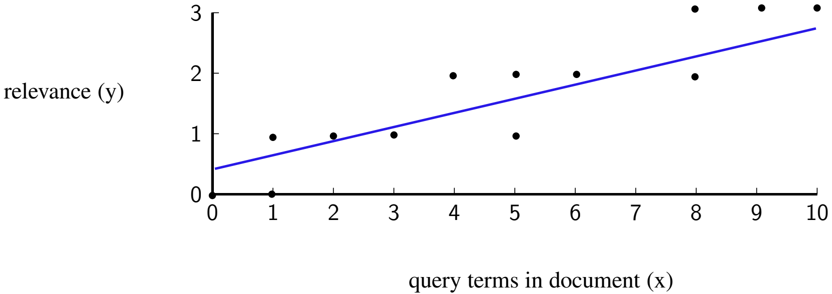
[1] Adapted from [Hang(2009)Hang]

Example of Input Vectors

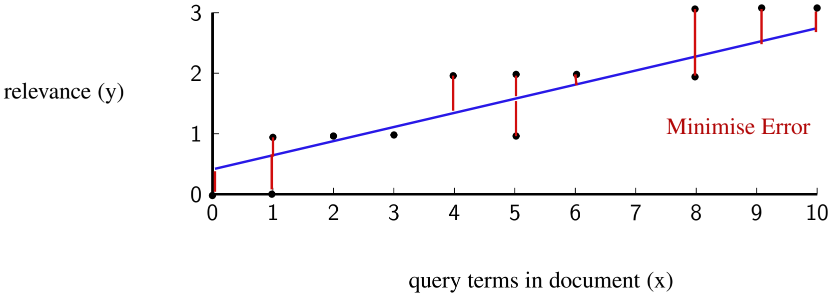
y_i	input vectors $\vec{x_i}$		
	x_{i1}	x_{i2}	x_{i3}
3	7.0	9.2	3.2
2	2.0	9.2	4.1
0	2.0	3.5	0.2
2	2.0	9.2	11.2
1	3.0	5.3	2.2
0	0.0	3.2	0.5

Table : Sample Dataset

A Simple Pointwise Example



A Simple Pointwise Example



Pointwise summary

- Each instance is treated in isolation
- The error from the absolute gold score (or class) is minimised

Pointwise summary

- Each instance is treated in isolation
- The error from the absolute gold score (or class) is minimised
- In general, this is solving a more difficult problem than is necessary

Pairwise Outline I

General Criteria

The ranking function f learns to rank pairs of items (i.e. for $\{\vec{x_i}, \vec{x_j}\}$, is y_i greater than y_j?).

[2] Adapted from [Hang(2009)Hang]

Pairwise Outline I

General Criteria

The ranking function f learns to rank pairs of items (i.e. for $\{\vec{x}_i, \vec{x}_j\}$, is y_i greater than y_j?).

	Learning	Ranking
Input	Order input vector pair $\{\vec{x}_i, \vec{x}_j\}$	Feature vectors $\{x_i\}_{i=1}^n$
Output	Classifier of pairs $y_{ij} = sign(f(\vec{x}_i - \vec{x}_j))$	Permutation over vectors $y = sort(\{f(\vec{x}_i)\}_{i=1}^n)$
Model	Ranking Function $f(\vec{x})$	
Loss	Pairwise misclassification	Ranking evaluation measure

Table : Learning in Pairwise approaches[2]

[2] Adapted from [Hang(2009)Hang]

Pairwise Outline II

- Cost function typically minimises misclassification of pairwise *difference vectors*
- The function learns using paired input vectors $f(\vec{x_i} - \vec{x_j})$
- Any binary classifier can be used for implementation
- Although svm^{rank} is a commonly used implementation[3]

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

Pairwise Transformation

y_i	input vectors $\vec{x_i}$		
	x_{i1}	x_{i2}	x_{i3}
3	7.0	9.2	3.2
2	2.0	9.2	4.1
0	2.0	3.5	0.2
2	2.0	9.2	11.2
1	3.0	5.3	2.2
0	0.0	3.2	0.5

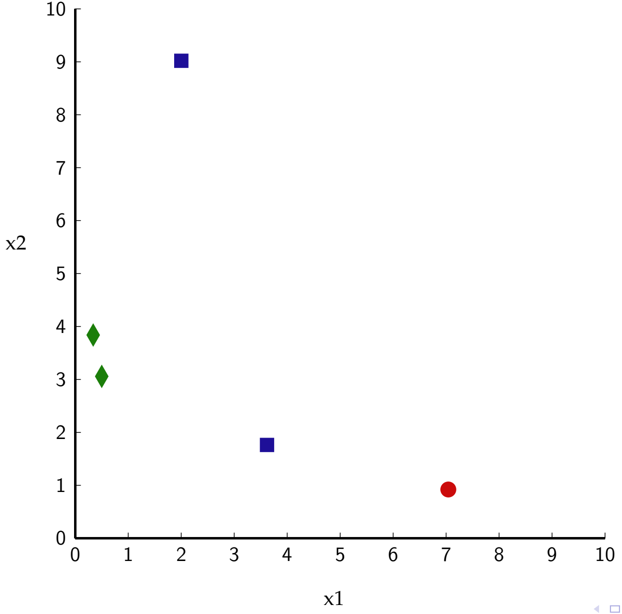
Table : Sample Dataset

Pairwise Transformation

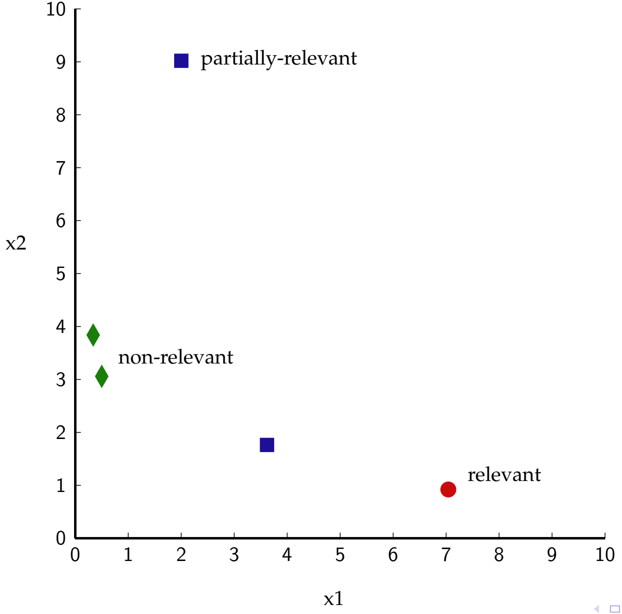
y'_{ij}	input vectors $\vec{x_i} - \vec{x_j}$		
	$x_{i1} - x_{j1}$	$x_{i2} - x_{j2}$	$x_{i3} - x_{j3}$
+(3-2)	5.0	0.0	-0.9
+(3-0)	5.0	5.7	3.0
+(3-2)	5.0	0.0	-8.0
+(3-1)	6.0	3.9	1.0
+(3-0)	7.0	6.0	2.7
+(2-0)	0.0	5.7	3.9
+(2-1)	-1.0	3.9	1.9
+(3-0)	2.0	6.0	3.6
...	...	...	...
-(3-2)	-5.0	0.0	0.9
...	...	...	...

Table : Transformed Dataset

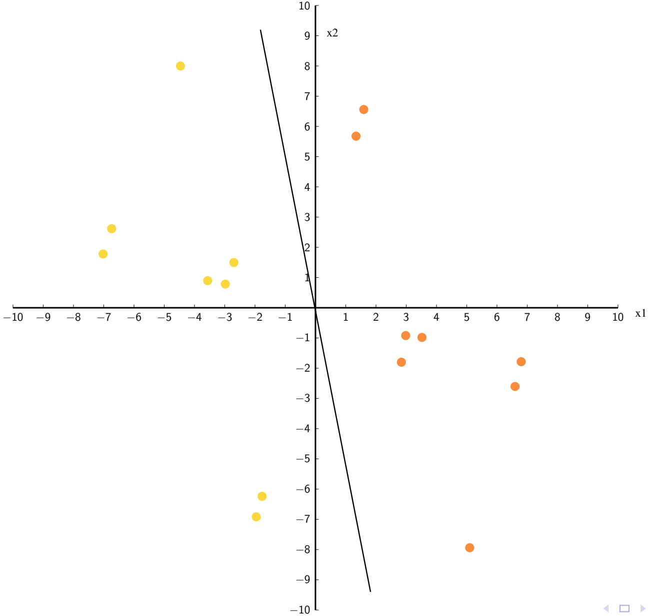
A Graphical Example I



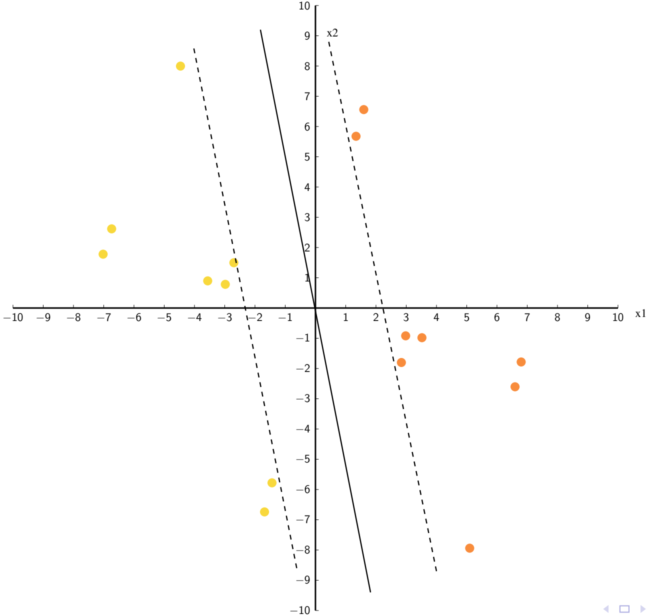
A Graphical Example I



A Graphical Example II



A Graphical Example II



Pairwise Summary

- In general, pairwise approaches outperform pointwise approaches in IR
- Pairwise preference models can be biased towards rankings containing many instances
- However, pairwise approaches often do not optimise the cost function that is usually used for evaluation (e.g. average precision or NDCG)
- For example, correctly ranking items at the top of the list is often more important than correctly ranking items lower down

Example

Pairwise Summary

- In general, pairwise approaches outperform pointwise approaches in IR
- Pairwise preference models can be biased towards rankings containing many instances
- However, pairwise approaches often do not optimise the cost function that is usually used for evaluation (e.g. average precision or NDCG)
- For example, correctly ranking items at the top of the list is often more important than correctly ranking items lower down

Example

- $\{RRRNNN\}$ vs $\{NRRRNN\} \implies ap = 0.638$

Pairwise Summary

- In general, pairwise approaches outperform pointwise approaches in IR
- Pairwise preference models can be biased towards rankings containing many instances
- However, pairwise approaches often do not optimise the cost function that is usually used for evaluation (e.g. average precision or NDCG)
- For example, correctly ranking items at the top of the list is often more important than correctly ranking items lower down

Example

- $\{RRRNNN\}$ vs $\{NRRRNN\} \implies ap = 0.638$
- $\{RRRNNN\}$ vs $\{RRNNNR\} \implies ap = 0.833$

where R and N are relevant and non-relevant respectively.

Listwise outline

- Many listwise approaches aim to directly optimise the most appropriate task-specific metric (e.g. for IR it may be average precision or NDCG)
- However, for rank-based approaches these metrics are often non-continuous w.r.t the scores
- E.g. the score of documents could change without any change in ranking
- Two-broad approaches to handling this:
  - Modify the cost function to a continuous (smooth) version
  - Use (or modify) an algorithm that can navigate discrete spaces

Listwise example I

- We'll use SVM^{map} [Yue *et al.*(2007)Yue, Finley, Radlinski, and Joachims] as a brief example
- Each permutation (list) of items is treated as an instance
- Aim to find weight vector $\vec{w}$ that ranks these permutations according to a loss function
- $h(\vec{q};\vec{w}) = arg\_max_{\vec{y}\in\mathcal{Y}} F(\vec{q},\vec{y};\vec{w})$
- And $F(\vec{q},\vec{y};\vec{w}) = \vec{w}\Psi(\vec{q},\vec{y})$

- Essentially each permutation is encoded as summation of ranked pairwise difference vectors (while negating incorrectly ranked pairs before summation)
- As a result, each list instance is mapped to a feature vector in $\mathcal{R}^N$
- Vectors with high feature values are good rankings
- As each input vector is a list, a list-based metric can be used as a smooth loss function (hinge-loss)

Listwise example II

- Essentially each permutation is encoded as summation of ranked pairwise difference vectors (while negating incorrectly ranked pairs before summation)
- As a result, each list instance is mapped to a feature vector in $\mathcal{R}^N$
- Vectors with high feature values are good rankings
- As each input vector is a list, a list-based metric can be used as a smooth loss function (hinge-loss)
- Number of permutations (rankings) is extremely large and so *all* lists are not used for training (see [Yue *et al.*(2007)Yue, Finley, Radlinski, and Joachims] for details)

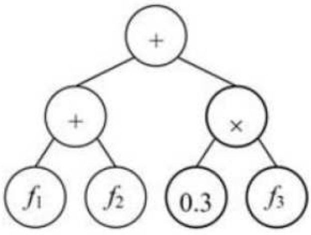
Navigating Non-continuous Spaces

- RankGP [Yeh *et al.*(2007)Yeh, Lin, Ke, and Yang] - Uses genetic programming to evolve ranking functions from a set of features and operators (e.g. $+, -, /, \times$)

[4] http://research.microsoft.com/en-us/people/tyliu/learning_to_rank_tutor

Navigating Non-continuous Spaces

- RankGP [Yeh *et al.*(2007)Yeh, Lin, Ke, and Yang] - Uses genetic programming to evolve ranking functions from a set of features and operators (e.g. $+$, $-$, $/$, $\times$)



4

[4] http://research.microsoft.com/en-us/people/tyliu/learning_to_rank_tutor

Other Listwise Approaches

- In general when you can represent a list as a vector in $\mathcal{R}^N$, you can optimize w such that it can rank these lists
- lambdaRANK [Burges *et al.*(2006)Burges, Ragno, and Le]
- softRANK [Taylor *et al.*(2008)Taylor, Guiver, Robertson, and Minka]

Open Questions

- Learning to rank for other tasks/domains (e.g. essay scoring)
- Optimising the "True loss" for ranking. What might that be?
- Ranking using deep learning
- Ranking natural language texts using distributed representations

Datasets

- LETOR Datasets [5]
- Yahoo! Learning to Rank Challenge
  https://webscope.sandbox.yahoo.com/#datasets

[5] http://research.microsoft.com/en-us/um/beijing/projects/letor/

Take-away Messages

- Applications of learning to rank abound
- Three main categories of approaches:
  - pointwise
  - pairwise
  - listwise
- Challenges in L2R
- Many open research questions

Burges, C. J. C., Ragno, R., and Le, Q. V. (2006).
Learning to Rank with Nonsmooth Cost Functions.
In B. Schölkopf, J. C. Platt, T. Hoffman, B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *NIPS*, pages 193–200. MIT Press.

Hang, L. (2009).
Learning to rank.
ACL-IJCNLP 2009.

Taylor, M., Guiver, J., Robertson, S., and Minka, T. (2008).
Softrank: Optimizing non-smooth rank metrics.
In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 77–86, New York, NY, USA. ACM.

Yeh, J.-Y., Lin, J.-Y., Ke, H.-R., and Yang, W.-P. (2007).
Learning to rank for information retrieval using genetic programming.
In T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, editors, *SIGIR 2007 workshop: Learning to Rank for Information Retrieval*.

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007).

A support vector method for optimizing average precision.
In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278. ACM.