Introduction to Information Retrieval
http://informationretrieval.org

IIR 15-2: Learning to Rank

Hinrich Schütze

Center for Information and Language Processing, University of Munich

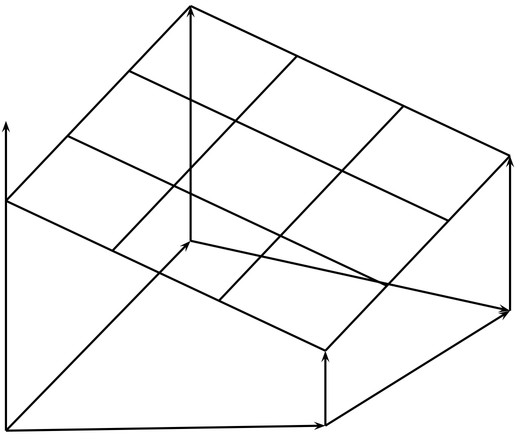
2011-06-05

Overview

Outline

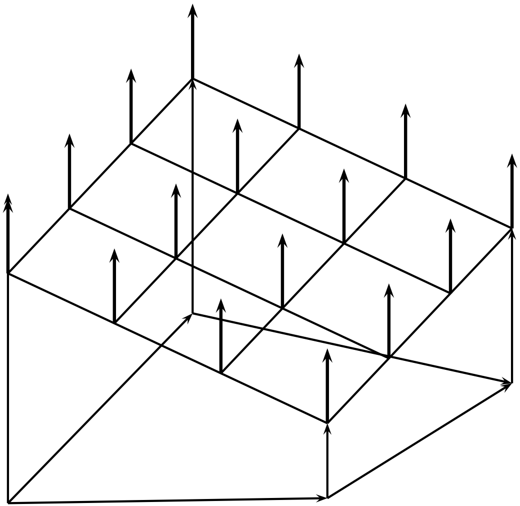
1. Recap
2. Zone scoring
3. Machine-learned scoring
4. Ranking SVMs

A linear classifier in 3D



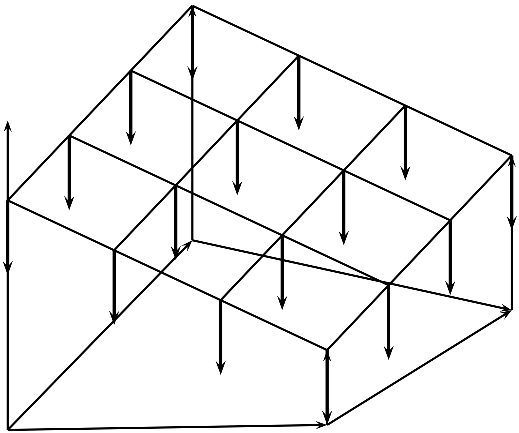
- A linear classifier in 3D is a plane described by the equation $w_1 d_1 + w_2 d_2 + w_3 d_3 = \theta$
- Example for a 3D linear classifier
- Points $(d_1\ d_2\ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 \geq \theta$ are in the class c.
- Points $(d_1\ d_2\ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 < \theta$ are in the complement class $\overline{c}$.

A linear classifier in 3D



- A linear classifier in 3D is a plane described by the equation $w_1 d_1 + w_2 d_2 + w_3 d_3 = \theta$
- Example for a 3D linear classifier
- Points $(d_1 \ d_2 \ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 \geq \theta$ are in the class c.
- Points $(d_1 \ d_2 \ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 < \theta$ are in the complement class $\overline{c}$.

A linear classifier in 3D



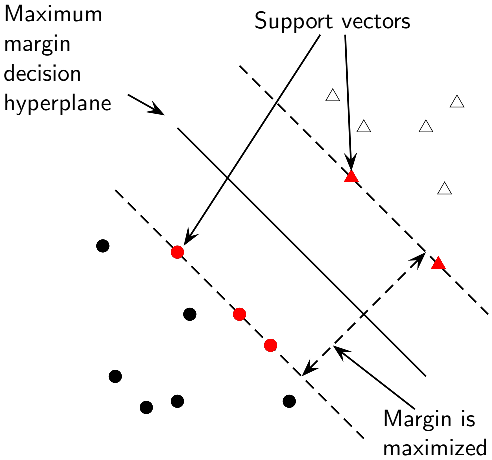
- A linear classifier in 3D is a plane described by the equation $w_1 d_1 + w_2 d_2 + w_3 d_3 = \theta$
- Example for a 3D linear classifier
- Points $(d_1 \ d_2 \ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 \geq \theta$ are in the class c.
- Points $(d_1 \ d_2 \ d_3)$ with $w_1 d_1 + w_2 d_2 + w_3 d_3 < \theta$ are in the complement class $\overline{c}$.

Linear classifiers

- Many common text classifiers are linear classifiers: Naive Bayes, Rocchio, logistic regression, least squares regression, linear support vector machines etc.
- Each method has a different way of selecting the separating hyperplane
  - Huge differences in performance on test documents

Support vector machines

- Binary classification problem
- Simple SVMs are linear classifiers.
- criterion: being maximally far away from any data point $\rightarrow$ determines classifier **margin**
- linear separator position defined by **support vectors**



Optimization problem solved by SVMs

Find $\vec{w}$ and b such that:

- $\frac{1}{2}\vec{w}^{\mathsf{T}}\vec{w}$ is minimized (because $|\vec{w}| = \sqrt{\vec{w}^{\mathrm{T}}\vec{w}}$), and
- for all $\{(\vec{x}_i, y_i)\}$, $y_i(\vec{w}^{\mathsf{T}}\vec{x}_i + b) \geq 1$

Which machine learning method to choose

- Is there a learning method that is optimal for all text classification problems?
- No, because there is a tradeoff between bias and variance.
- Factors to take into account:
  - How much training data is available?
  - How simple/complex is the problem? (linear vs. nonlinear decision boundary)
  - How noisy is the problem?
  - How stable is the problem over time?
    - For an unstable problem, it's better to use a simple and robust classifier.

Take-away today

Take-away today

- Basic idea of learning to rank (LTR): We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.

Take-away today

- Basic idea of learning to rank (LTR): We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Zone scoring: a particularly simple instance of LTR

Take-away today

- Basic idea of learning to rank (LTR): We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Zone scoring: a particularly simple instance of LTR
- Machine-learned scoring as a general approach to ranking

Take-away today

- Basic idea of learning to rank (LTR): We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Zone scoring: a particularly simple instance of LTR
- Machine-learned scoring as a general approach to ranking
- Ranking SVMs

Outline

Main idea

Main idea

- The aim of term weights (e.g., tf-idf) is to measure term salience.

Main idea

- The aim of term weights (e.g., tf-idf) is to measure term salience.
  - The sum of term weights is a measure of the relevance of a document to a query and the basis for ranking.

Main idea

- The aim of term weights (e.g., tf-idf) is to measure term salience.
    - The sum of term weights is a measure of the relevance of a document to a query and the basis for ranking.
- Now we view this ranking problem as a machine learning problem – we will learn the weighting or, more generally, the ranking.

Main idea

- The aim of term weights (e.g., tf-idf) is to measure term salience.
  - The sum of term weights is a measure of the relevance of a document to a query and the basis for ranking.
- Now we view this ranking problem as a machine learning problem – we will learn the weighting or, more generally, the ranking.
  - Term weights can be learned using training examples that have been judged.

Main idea

- The aim of term weights (e.g., tf-idf) is to measure term salience.
  - The sum of term weights is a measure of the relevance of a document to a query and the basis for ranking.
- Now we view this ranking problem as a machine learning problem – we will learn the weighting or, more generally, the ranking.
  - Term weights can be learned using training examples that have been judged.
- This methodology falls under a general class of approaches known as machine learned relevance or learning to rank.

Learning weights

Learning weights

Main methodology

- Given a set of training examples, each of which is a tuple of: a query q, a document d, a relevance judgment for d on q

Learning weights

Main methodology

- Given a set of training examples, each of which is a tuple of: a query q, a document d, a relevance judgment for d on q
  - Simplest case: $R(d, q)$ is either relevant (1) or nonrelevant (0)

Learning weights

Main methodology

- Given a set of training examples, each of which is a tuple of: a query q, a document d, a relevance judgment for d on q
  - Simplest case: $R(d, q)$ is either relevant (1) or nonrelevant (0)
  - More sophisticated cases: graded relevance judgments

Learning weights

Main methodology

- Given a set of training examples, each of which is a tuple of: a query q, a document d, a relevance judgment for d on q
  - Simplest case: $R(d, q)$ is either relevant (1) or nonrelevant (0)
  - More sophisticated cases: graded relevance judgments
- Learn weights from these examples, so that the learned scores approximate the relevance judgments in the training examples

Binary independence model (BIM)

Binary independence model (BIM)

- Is what BIM does a form of learning to rank?

Binary independence model (BIM)

- Is what BIM does a form of learning to rank?
- Recap BIM:

Binary independence model (BIM)

- Is what BIM does a form of learning to rank?
- Recap BIM:
  - Estimate classifier of probability of relevance on training set

Binary independence model (BIM)

- Is what BIM does a form of learning to rank?
- Recap BIM:
  - Estimate classifier of probability of relevance on training set
  - Apply to all documents

Binary independence model (BIM)

- Is what BIM does a form of learning to rank?
- Recap BIM:
  - Estimate classifier of probability of relevance on training set
  - Apply to all documents
  - Rank documents according to probability of relevance

Learning to rank vs. Text classification

Learning to rank vs. Text classification

- Both are machine learning approaches

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.
  - We need a query-specific training set to learn the ranker.

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.
  - We need a query-specific training set to learn the ranker.
  - We need to learn a new ranker for each query.

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.
  - We need a query-specific training set to learn the ranker.
  - We need to learn a new ranker for each query.
- Learning to rank usually refers to query-independent ranking.

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.
  - We need a query-specific training set to learn the ranker.
  - We need to learn a new ranker for each query.
- Learning to rank usually refers to query-independent ranking.
- We learn a single classifier.

Learning to rank vs. Text classification

- Both are machine learning approaches
- Text classification, BIM and relevance feedback (if solved by text classification) are query-specific.
  - We need a query-specific training set to learn the ranker.
  - We need to learn a new ranker for each query.
- Learning to rank usually refers to query-independent ranking.
- We learn a single classifier.
- We can then rank documents for a query that we don't have any relevance judgments for.

Learning to rank: Exercise

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.
  - Class 1: the document is relevant to the query.

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.
  - Class 1: the document is relevant to the query.
  - Class 2: the document is not relevant to the query.

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.
  - Class 1: the document is relevant to the query.
  - Class 2: the document is not relevant to the query.
- This is a standard classification problem, except that the data points are query-document pairs (as opposed to documents).

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.
  - Class 1: the document is relevant to the query.
  - Class 2: the document is not relevant to the query.
- This is a standard classification problem, except that the data points are query-document pairs (as opposed to documents).
- Documents are ranked according to probability of relevance of corresponding query-document pairs.

Learning to rank: Exercise

- One approach to learning to rank is to represent each query-document pair as a data point, represented as a vector.
- We have two classes.
  - Class 1: the document is relevant to the query.
  - Class 2: the document is not relevant to the query.
- This is a standard classification problem, except that the data points are query-document pairs (as opposed to documents).
- Documents are ranked according to probability of relevance of corresponding query-document pairs.
- What features/dimensions would you use to represent a query-document pair?

Simple form of learning to rank: Zone scoring

Simple form of learning to rank: Zone scoring

- Given: a collection where documents have three zones (a.k.a. fields): `author`, `title`, `body`

Simple form of learning to rank: Zone scoring

- Given: a collection where documents have three zones (a.k.a. fields): `author, title, body`
- Weighted zone scoring requires a separate weight for each zone, e.g. g_1, g_2, g_3

Simple form of learning to rank: Zone scoring

- Given: a collection where documents have three zones (a.k.a. fields): `author`, `title`, `body`
- Weighted zone scoring requires a separate weight for each zone, e.g. g_1, g_2, g_3
- Not all zones are equally important:
  e.g. `author` $<$ `title` $<$ `body`
  $\rightarrow g_1 = 0.2, g_2 = 0.3, g_3 = 0.5$ (so that they add up to 1)

Simple form of learning to rank: Zone scoring

- Given: a collection where documents have three zones (a.k.a. fields): `author, title, body`
- Weighted zone scoring requires a separate weight for each zone, e.g. g_1, g_2, g_3
- Not all zones are equally important:
  e.g. `author` < `title` < `body`
  $\rightarrow g_1 = 0.2, g_2 = 0.3, g_3 = 0.5$ (so that they add up to 1)
- Score for a zone = 1 if the query term occurs in that zone, 0 otherwise (Boolean)

Simple form of learning to rank: Zone scoring

- Given: a collection where documents have three zones (a.k.a. fields): `author`, `title`, `body`
- Weighted zone scoring requires a separate weight for each zone, e.g. g_1, g_2, g_3
- Not all zones are equally important:
  e.g. `author` < `title` < `body`
  $\rightarrow g_1 = 0.2, g_2 = 0.3, g_3 = 0.5$ (so that they add up to 1)
- Score for a zone = 1 if the query term occurs in that zone, 0 otherwise (Boolean)

Example

Query term appears in `title` and `body` only
Document score: $(0.3 \cdot 1) + (0.5 \cdot 1) = 0.8$.

General form of weighted zone scoring

General form of weighted zone scoring

Given query q and document d, weighted zone scoring assigns to the pair (q, d) a score in the interval [0,1] by computing a linear combination of document zone scores, where each zone contributes a value.

- Consider a set of documents, which have l zones

General form of weighted zone scoring

Given query q and document d, weighted zone scoring assigns to the pair (q, d) a score in the interval [0,1] by computing a linear combination of document zone scores, where each zone contributes a value.

- Consider a set of documents, which have l zones
- Let $g_1, ..., g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$

General form of weighted zone scoring

Given query q and document d, weighted zone scoring assigns to the pair (q, d) a score in the interval [0,1] by computing a linear combination of document zone scores, where each zone contributes a value.

- Consider a set of documents, which have l zones
- Let $g_1, ..., g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$
- For $1 \leq i \leq l$, let s_i be the Boolean score denoting a match (or non-match) between q and the i^{th} zone

General form of weighted zone scoring

Given query q and document d, weighted zone scoring assigns to the pair (q, d) a score in the interval [0,1] by computing a linear combination of document zone scores, where each zone contributes a value.

- Consider a set of documents, which have l zones
- Let $g_1, ..., g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$
- For $1 \leq i \leq l$, let s_i be the Boolean score denoting a match (or non-match) between q and the i^{th} zone
  - $s_i = 1$ if a query term occurs in zone i, 0 otherwise

General form of weighted zone scoring

Given query q and document d, weighted zone scoring assigns to the pair (q, d) a score in the interval [0,1] by computing a linear combination of document zone scores, where each zone contributes a value.

- Consider a set of documents, which have l zones
- Let $g_1, ..., g_l \in [0, 1]$, such that $\sum_{i=1}^{l} g_i = 1$
- For $1 \leq i \leq l$, let s_i be the Boolean score denoting a match (or non-match) between q and the i^{th} zone
  - $s_i = 1$ if a query term occurs in zone i, 0 otherwise

Weighted zone score a.k.a ranked Boolean retrieval

Rank documents according to $\sum_{i=1}^{l} g_i s_i$

Learning weights in weighted zone scoring

Learning weights in weighted zone scoring

- Weighted zone scoring may be viewed as learning a linear function of the Boolean match scores contributed by the various zones.

Learning weights in weighted zone scoring

- Weighted zone scoring may be viewed as learning a linear function of the Boolean match scores contributed by the various zones.
- No free lunch: labor-intensive assembly of user-generated relevance judgments from which to learn the weights

Learning weights in weighted zone scoring

- Weighted zone scoring may be viewed as learning a linear function of the Boolean match scores contributed by the various zones.
- No free lunch: labor-intensive assembly of user-generated relevance judgments from which to learn the weights
  - Especially in a dynamic collection (such as the Web)

Learning weights in weighted zone scoring

- Weighted zone scoring may be viewed as learning a linear function of the Boolean match scores contributed by the various zones.
- No free lunch: labor-intensive assembly of user-generated relevance judgments from which to learn the weights
  - Especially in a dynamic collection (such as the Web)
  - Major search engines put considerable resources into creating large training sets for learning to rank.

Learning weights in weighted zone scoring

- Weighted zone scoring may be viewed as learning a linear function of the Boolean match scores contributed by the various zones.
- No free lunch: labor-intensive assembly of user-generated relevance judgments from which to learn the weights
  - Especially in a dynamic collection (such as the Web)
  - Major search engines put considerable resources into creating large training sets for learning to rank.
- Good news: once you have a large enough training set, the problem of learning the weights g_i reduces to a simple optimization problem.

Learning weights in weighted zone scoring: Simple case

Learning weights in weighted zone scoring: Simple case

- Let documents have two zones: `title, body`

Learning weights in weighted zone scoring: Simple case

- Let documents have two zones: `title, body`
- The weighted zone scoring formula we saw before:

$$\sum_{i=1}^{l} g_i s_i$$

Learning weights in weighted zone scoring: Simple case

- Let documents have two zones: `title, body`
- The weighted zone scoring formula we saw before:

$$\sum_{i=1}^{l} g_i s_i$$

- Given q, d, $s_T(d, q) = 1$ if a query term occurs in title, 0 otherwise; $s_B(d, q) = 1$ if a query term occurs in body, 0 otherwise

Learning weights in weighted zone scoring: Simple case

- Let documents have two zones: `title, body`
- The weighted zone scoring formula we saw before:

$$\sum_{i=1}^{l} g_i s_i$$

- Given q, d, $s_T(d, q) = 1$ if a query term occurs in title, 0 otherwise; $s_B(d, q) = 1$ if a query term occurs in body, 0 otherwise
- We compute a score between 0 and 1 for each (d, q) pair using $s_T(d, q)$ and $s_B(d, q)$ by using a constant $g \in [0, 1]$:

$$score(d, q) = g \cdot s_T(d, q) + (1 - g) \cdot s_B(d, q)$$

Learning weights: determine g from training examples

Learning weights: determine g from training examples

Example

Φ_j	d_j	q_j	s_T	s_B	$r(d_j, q_j)$
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

Learning weights: determine g from training examples

Example

Φ_j	d_j	q_j	s_T	s_B	$r(d_j, q_j)$
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- Training examples: triples of the form $\Phi_j = (d_j, q_j, r(d_j, q_j))$

Learning weights: determine g from training examples

Example

Φ_j	d_j	q_j	s_T	s_B	$r(d_j, q_j)$
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- Training examples: triples of the form $\Phi_j = (d_j, q_j, r(d_j, q_j))$
- A given training document d_j and a given training query q_j are assessed by a human who decides $r(d_j, q_j)$ (either relevant or nonrelevant)

Learning weights: determine g from training examples

Example

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

Learning weights: determine g from training examples

Example

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	Relevant
Φ_2	37	penguin	0	1	Nonrelevant
Φ_3	238	system	0	1	Relevant
Φ_4	238	penguin	0	0	Nonrelevant
Φ_5	1741	kernel	1	1	Relevant
Φ_6	2094	driver	0	1	Relevant
Φ_7	3194	driver	1	0	Nonrelevant

- For each training example Φ_j we have Boolean values $s_T(d_j, q_j)$ and $s_B(d_j, q_j)$ that we use to compute a score:

$$score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j)$$

Learning weights

- We compare this score $score(d_j, q_j)$ with the human relevance judgment for the same query-document pair (d_j, q_j).

Learning weights

- We compare this score $score(d_j, q_j)$ with the human relevance judgment for the same query-document pair (d_j, q_j).
- We define the error of the scoring function with weight g as

$$\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$$

Learning weights

- We compare this score $score(d_j, q_j)$ with the human relevance judgment for the same query-document pair (d_j, q_j).
- We define the error of the scoring function with weight g as

$$\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$$

- Then, the total error of a set of training examples is given by

$$\sum_j \epsilon(g, \Phi_j)$$

Learning weights

- We compare this score $score(d_j, q_j)$ with the human relevance judgment for the same query-document pair (d_j, q_j).
- We define the error of the scoring function with weight g as

$$\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$$

- Then, the total error of a set of training examples is given by

$$\sum_j \epsilon(g, \Phi_j)$$

- The problem of learning the constant g from the given training examples then reduces to picking the value of g that minimizes the total error.

Minimizing the total error ϵ: Example (1)

Minimizing the total error ϵ: Example (1)

Training examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	1 (relevant)
Φ_2	37	penguin	0	1	0 (nonrelevant)
Φ_3	238	system	0	1	1 (relevant)
Φ_4	238	penguin	0	0	0 (nonrelevant)
Φ_5	1741	kernel	1	1	1 (relevant)
Φ_6	2094	driver	0	1	1 (relevant)
Φ_7	3194	driver	1	0	0 (nonrelevant)

Minimizing the total error ϵ: Example (1)

Training examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	1 (relevant)
Φ_2	37	penguin	0	1	0 (nonrelevant)
Φ_3	238	system	0	1	1 (relevant)
Φ_4	238	penguin	0	0	0 (nonrelevant)
Φ_5	1741	kernel	1	1	1 (relevant)
Φ_6	2094	driver	0	1	1 (relevant)
Φ_7	3194	driver	1	0	0 (nonrelevant)

- Compute score:
  $score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j)$

Minimizing the total error ϵ: Example (1)

Training examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	1 (relevant)
Φ_2	37	penguin	0	1	0 (nonrelevant)
Φ_3	238	system	0	1	1 (relevant)
Φ_4	238	penguin	0	0	0 (nonrelevant)
Φ_5	1741	kernel	1	1	1 (relevant)
Φ_6	2094	driver	0	1	1 (relevant)
Φ_7	3194	driver	1	0	0 (nonrelevant)

- Compute score:
  $score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j)$
- Compute total error: $\sum_j \epsilon(g, \Phi_j)$, where
  $\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$

Minimizing the total error ϵ: Example (1)

Training examples

Example	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	1	1	1 (relevant)
Φ_2	37	penguin	0	1	0 (nonrelevant)
Φ_3	238	system	0	1	1 (relevant)
Φ_4	238	penguin	0	0	0 (nonrelevant)
Φ_5	1741	kernel	1	1	1 (relevant)
Φ_6	2094	driver	0	1	1 (relevant)
Φ_7	3194	driver	1	0	0 (nonrelevant)

- Compute score:
  $score(d_j, q_j) = g \cdot s_T(d_j, q_j) + (1 - g) \cdot s_B(d_j, q_j)$
- Compute total error: $\sum_j \epsilon(g, \Phi_j)$, where
  $\epsilon(g, \Phi_j) = (r(d_j, q_j) - score(d_j, q_j))^2$
- Pick the value of g that minimizes the total error

Minimizing the total error ϵ: Example (2)

Minimizing the total error ϵ: Example (2)

- Compute score $score(d_j, q_j)$

$$score(d_1, q_1) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$$
$$score(d_2, q_2) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_3, q_3) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_4, q_4) = g \cdot 0 + (1-g) \cdot 0 = 0 + 0 = 0$$
$$score(d_5, q_5) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$$
$$score(d_6, q_6) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$$
$$score(d_7, q_7) = g \cdot 1 + (1-g) \cdot 0 = g + 0 = g$$

Minimizing the total error ϵ: Example (2)

- Compute score $score(d_j, q_j)$
  $score(d_1, q_1) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$
  $score(d_2, q_2) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_3, q_3) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_4, q_4) = g \cdot 0 + (1-g) \cdot 0 = 0 + 0 = 0$
  $score(d_5, q_5) = g \cdot 1 + (1-g) \cdot 1 = g + 1 - g = 1$
  $score(d_6, q_6) = g \cdot 0 + (1-g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_7, q_7) = g \cdot 1 + (1-g) \cdot 0 = g + 0 = g$
- Compute total error $\sum_j \epsilon(g, \Phi_j)$
  $(1-1)^2+(0-1+g)^2+(1-1+g)^2+(0-0)^2+(1-1)^2+(1-1+g)^2+(0-g)^2 = 0+(-1+g)^2+g^2+0+0+g^2+g^2 = 1-2g+4g^2$

Minimizing the total error ϵ: Example (2)

- Compute score $score(d_j, q_j)$
  $score(d_1, q_1) = g \cdot 1 + (1 - g) \cdot 1 = g + 1 - g = 1$
  $score(d_2, q_2) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_3, q_3) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_4, q_4) = g \cdot 0 + (1 - g) \cdot 0 = 0 + 0 = 0$
  $score(d_5, q_5) = g \cdot 1 + (1 - g) \cdot 1 = g + 1 - g = 1$
  $score(d_6, q_6) = g \cdot 0 + (1 - g) \cdot 1 = 0 + 1 - g = 1 - g$
  $score(d_7, q_7) = g \cdot 1 + (1 - g) \cdot 0 = g + 0 = g$
- Compute total error $\sum_j \epsilon(g, \Phi_j)$
  $(1-1)^2 + (0-1+g)^2 + (1-1+g)^2 + (0-0)^2 + (1-1)^2 + (1-1+g)^2 + (0-g)^2 = 0 + (-1+g)^2 + g^2 + 0 + 0 + g^2 + g^2 = 1 - 2g + 4g^2$
- Pick the value of g that minimizes the total error
  Setting derivative to 0, gives you a minimum of $g = \frac{1}{4}$.

Weight g that minimizes error in the general case

Weight g that minimizes error in the general case

- $$g = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}$$

Weight g that minimizes error in the general case

- $$g = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}$$
- $n_{...}$ are the counts of rows of the training set table with the corresponding properties:

n_{10r}	$s_T = 1$	$s_B = 0$	document relevant
n_{10n}	$s_T = 1$	$s_B = 0$	document nonrelevant
n_{01r}	$s_T = 0$	$s_B = 1$	document relevant
n_{01n}	$s_T = 0$	$s_B = 1$	document nonrelevant

Weight g that minimizes error in the general case

- $$g = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}$$
- $n_{...}$ are the counts of rows of the training set table with the corresponding properties:

n_{10r}	$s_T = 1$	$s_B = 0$	document relevant
n_{10n}	$s_T = 1$	$s_B = 0$	document nonrelevant
n_{01r}	$s_T = 0$	$s_B = 1$	document relevant
n_{01n}	$s_T = 0$	$s_B = 1$	document nonrelevant

- Derivation: see book

Weight g that minimizes error in the general case

- $$g = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}}$$
- $n_{...}$ are the counts of rows of the training set table with the corresponding properties:

n_{10r}	$s_T = 1$	$s_B = 0$	document relevant
n_{10n}	$s_T = 1$	$s_B = 0$	document nonrelevant
n_{01r}	$s_T = 0$	$s_B = 1$	document relevant
n_{01n}	$s_T = 0$	$s_B = 1$	document nonrelevant

- Derivation: see book
- Note that we ignore documents that have 0 match scores for both zones or 1 match scores for both zones – the value of g does not change their final score.

Exercise: Compute g that minimizes the error

Exercise: Compute g that minimizes the error

	DocID	Query	s_T	s_B	Judgment
Φ_1	37	linux	0	0	Relevant
Φ_2	37	penguin	1	1	Nonrelevant
Φ_3	238	system	1	0	Relevant
Φ_4	238	penguin	1	1	Nonrelevant
Φ_5	238	redmond	0	1	Nonrelevant
Φ_6	1741	kernel	0	0	Relevant
Φ_7	2094	driver	1	0	Relevant
Φ_8	3194	driver	0	1	Nonrelevant
Φ_9	3194	redmond	0	0	Nonrelevant

Solution

2	n_{10r}	$s_T = 1$	$s_B = 0$	document relevant
0	n_{10n}	$s_T = 1$	$s_B = 0$	document nonrelevant
0	n_{01r}	$s_T = 0$	$s_B = 1$	document relevant
2	n_{01n}	$s_T = 0$	$s_B = 1$	document nonrelevant

$$g = \frac{n_{10r} + n_{01n}}{n_{10r} + n_{10n} + n_{01r} + n_{01n}} = \frac{2+2}{2+0+2+0} = 1$$

Outline

More general setup of machine learned scoring

More general setup of machine learned scoring

- So far, we have considered a case where we combined match scores (Boolean indicators of relevance).

More general setup of machine learned scoring

- So far, we have considered a case where we combined match scores (Boolean indicators of relevance).
- Now consider more general factors that go beyond Boolean functions of query term presence in document zones.

Two examples of typical features

Two examples of typical features

- The vector space cosine similarity between query and document (denoted α)

Two examples of typical features

- The vector space cosine similarity between query and document (denoted α)
- The minimum window width within which the query terms lie (denoted ω)

Two examples of typical features

- The vector space cosine similarity between query and document (denoted α)
- The minimum window width within which the query terms lie (denoted ω)
    - Query term proximity is often indicative of topical relevance.

Two examples of typical features

- The vector space cosine similarity between query and document (denoted α)
- The minimum window width within which the query terms lie (denoted ω)
  - Query term proximity is often indicative of topical relevance.
- Thus, we have one feature that captures overall query-document similarity and one features that captures proximity of query terms in the document.

Learning to rank setup for these two features

Learning to rank setup for these two features

Example

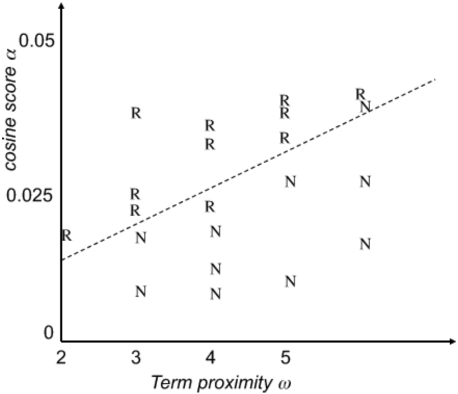
Example	DocID	Query	α	ω	Judgment
Φ_1	37	linux	0.032	3	relevant
Φ_2	37	penguin	0.02	4	nonrelevant
Φ_3	238	operating system	0.043	2	relevant
Φ_4	238	runtime	0.004	2	nonrelevant
Φ_5	1741	kernel layer	0.022	3	relevant
Φ_6	2094	device driver	0.03	2	relevant
Φ_7	3191	device driver	0.027	5	nonrelevant

α is the cosine score. ω is the window width.

This is exactly the same setup as for zone scoring except we now have more complex features that capture whether a document is relevant to a query.

Graphic representation of the training set

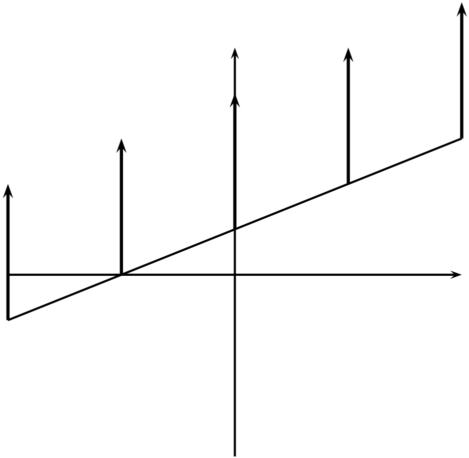
Graphic representation of the training set



This should look familiar.

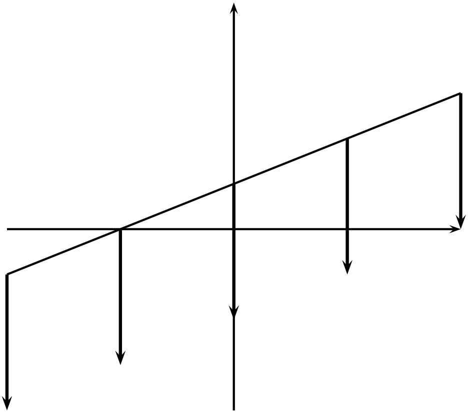
In this case: LTR approach learns a linear classifier in 2D

In this case: LTR approach learns a linear classifier in 2D



- A linear classifier in 2D is a line described by the equation $w_1 d_1 + w_2 d_2 = \theta$
- Example for a 2D linear classifier
- Points $(d_1 \; d_2)$ with $w_1 d_1 + w_2 d_2 \geq \theta$ are in the class c.
- Points $(d_1 \; d_2)$ with $w_1 d_1 + w_2 d_2 < \theta$ are in the complement class $\overline{c}$.

In this case: LTR approach learns a linear classifier in 2D



- A linear classifier in 2D is a line described by the equation $w_1 d_1 + w_2 d_2 = \theta$
- Example for a 2D linear classifier
- Points $(d_1 \ d_2)$ with $w_1 d_1 + w_2 d_2 \geq \theta$ are in the class c.
- Points $(d_1 \ d_2)$ with $w_1 d_1 + w_2 d_2 < \theta$ are in the complement class $\overline{c}$.

Learning to rank setup for two features

Learning to rank setup for two features

- Again, two classes: relevant $= 1$ and nonrelevant $= 0$

Learning to rank setup for two features

- Again, two classes: relevant $= 1$ and nonrelevant $= 0$
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.

Learning to rank setup for two features

- Again, two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.

Learning to rank setup for two features

- Again, two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- A linear classifier is defined by an equation of the form:

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

where we learn the coefficients a, b, c from training data.

Learning to rank setup for two features

- Again, two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- A linear classifier is defined by an equation of the form:

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

  where we learn the coefficients a, b, c from training data.
- Regression vs. classification

Learning to rank setup for two features

- Again, two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- A linear classifier is defined by an equation of the form:

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

  where we learn the coefficients a, b, c from training data.
- Regression vs. classification
  - We have only covered binary classification so far.

Learning to rank setup for two features

- Again, two classes: relevant $= 1$ and nonrelevant $= 0$
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- A linear classifier is defined by an equation of the form:

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

  where we learn the coefficients a, b, c from training data.
- Regression vs. classification
  - We have only covered binary classification so far.
  - We can also cast the problem as a regression problem.

Learning to rank setup for two features

- Again, two classes: relevant = 1 and nonrelevant = 0
- We now seek a scoring function that combines the values of the features to generate a value that is (close to) 0 or 1.
- We wish this function to be in agreement with our set of training examples as much as possible.
- A linear classifier is defined by an equation of the form:

$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c,$$

  where we learn the coefficients a, b, c from training data.
- Regression vs. classification
  - We have only covered binary classification so far.
  - We can also cast the problem as a regression problem.
  - This is what we did for zone scoring just now.

Different geometric interpretation of what's happening

Different geometric interpretation of what's happening

Different geometric interpretation of what's happening

Different geometric interpretation of what's happening

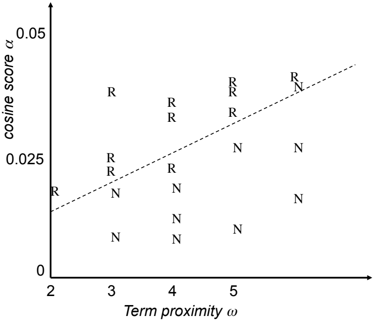
- The function *Score*(α, ω) represents a plane "hanging above" the figure.

Different geometric interpretation of what's happening

- The function $Score(\alpha, \omega)$ represents a plane "hanging above" the figure.
- Ideally this plane assumes values close to 1 above the points marked R, and values close to 0 above the points marked N.

Different geometric interpretation of what's happening

- The function *Score*(α, ω) represents a plane "hanging above" the figure.
- Ideally this plane assumes values close to 1 above the points marked R, and values close to 0 above the points marked N.



Linear classification in this case

Linear classification in this case

Linear classification in this case

Linear classification in this case

- We pick a threshold θ.

Linear classification in this case

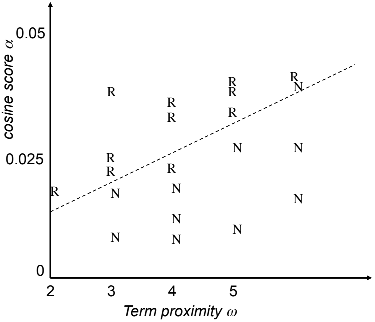
- We pick a threshold θ.
- If $\mathit{Score}(\alpha, \omega) > \theta$, we declare the document relevant, otherwise we declare it nonrelevant.

Linear classification in this case

- We pick a threshold θ.
- If $Score(\alpha, \omega) > \theta$, we declare the document relevant, otherwise we declare it nonrelevant.
- As before, all points that satisfy $Score(\alpha, \omega) = \theta$ form a line (dashed here) $\rightarrow$ linear classifier that separates relevant from nonrelevant instances.

Linear classification in this case

- We pick a threshold θ.
- If $\mathit{Score}(\alpha, \omega) > \theta$, we declare the document relevant, otherwise we declare it nonrelevant.
- As before, all points that satisfy $\mathit{Score}(\alpha, \omega) = \theta$ form a line (dashed here) $\rightarrow$ linear classifier that separates relevant from nonrelevant instances.



Summary

Summary

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.

Summary

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In the example: The classifier corresponds to a line $\mathit{Score}(\alpha, \omega) = \theta$ in the α-ω plane.

Summary

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In the example: The classifier corresponds to a line $\mathit{Score}(\alpha, \omega) = \theta$ in the α-ω plane.
- In principle, any method learning a linear classifier (including least squares regression) can be used to find this line.

Summary

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In the example: The classifier corresponds to a line $\mathit{Score}(\alpha, \omega) = \theta$ in the α-ω plane.
- In principle, any method learning a linear classifier (including least squares regression) can be used to find this line.
- Big advantage of learning to rank: we can avoid hand-tuning scoring functions and simply learn them from training data.

Summary

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In the example: The classifier corresponds to a line $\mathit{Score}(\alpha, \omega) = \theta$ in the α-ω plane.
- In principle, any method learning a linear classifier (including least squares regression) can be used to find this line.
- Big advantage of learning to rank: we can avoid hand-tuning scoring functions and simply learn them from training data.
- Bottleneck of learning to rank: maintaining a representative set of training examples whose relevance assessments must be made by humans.

Learning to rank for more than two features

Learning to rank for more than two features

- The approach can be readily generalized to a large number of features.

Learning to rank for more than two features

- The approach can be readily generalized to a large number of features.
- In addition to cosine similarity and query term window, there are lots of other indicators of relevance: PageRank-style measures, document age, zone contributions, document length, etc.

Learning to rank for more than two features

- The approach can be readily generalized to a large number of features.
- In addition to cosine similarity and query term window, there are lots of other indicators of relevance: PageRank-style measures, document age, zone contributions, document length, etc.
- If these measures can be calculated for a training document collection with relevance judgments, any number of such measures can be used to machine-learn a classifier.

LTR features used by Microsoft Research (1)

LTR features used by Microsoft Research (1)

- Zones: body, anchor, title, url, whole document

LTR features used by Microsoft Research (1)

- Zones: body, anchor, title, url, whole document
- Features derived from standard IR models: query term number, query term ratio, length, idf, sum of term frequency, min of term frequency, max of term frequency, mean of term frequency, variance of term frequency, sum of length normalized term frequency, min of length normalized term frequency, max of length normalized term frequency, mean of length normalized term frequency, variance of length normalized term frequency, sum of tf-idf, min of tf-idf, max of tf-idf, mean of tf-idf, variance of tf-idf, boolean model, BM25

LTR features used by Microsoft Research (2)

LTR features used by Microsoft Research (2)

- Language model features: LMIR.ABS, LMIR.DIR, LMIR.JM

LTR features used by Microsoft Research (2)

- Language model features: LMIR.ABS, LMIR.DIR, LMIR.JM
- Web-specific features: number of slashes in url, length of url, inlink number, outlink number, PageRank, SiteRank

LTR features used by Microsoft Research (2)

- Language model features: LMIR.ABS, LMIR.DIR, LMIR.JM
- Web-specific features: number of slashes in url, length of url, inlink number, outlink number, PageRank, SiteRank
- Spam features: QualityScore

LTR features used by Microsoft Research (2)

- Language model features: LMIR.ABS, LMIR.DIR, LMIR.JM
- Web-specific features: number of slashes in url, length of url, inlink number, outlink number, PageRank, SiteRank
- Spam features: QualityScore
- Usage-based features: query-url click count, url click count, url dwell time

LTR features used by Microsoft Research (2)

- Language model features: LMIR.ABS, LMIR.DIR, LMIR.JM
- Web-specific features: number of slashes in url, length of url, inlink number, outlink number, PageRank, SiteRank
- Spam features: QualityScore
- Usage-based features: query-url click count, url click count, url dwell time
- See link in resources for more information

Shortcoming of our LTR approach so far

Shortcoming of our LTR approach so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.

Shortcoming of our LTR approach so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.
- Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus regression problems (where a real number is predicted).

Shortcoming of our LTR approach so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.
- Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus regression problems (where a real number is predicted).
- In between is the specialized field of ordinal regression where a ranking is predicted.

Shortcoming of our LTR approach so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.
- Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus regression problems (where a real number is predicted).
- In between is the specialized field of ordinal regression where a ranking is predicted.
- Machine learning for ad hoc retrieval is most properly thought of as an ordinal regression problem, where the goal is to rank a set of documents for a query, given training data of the same sort.

Shortcoming of our LTR approach so far

- Approaching IR ranking like we have done so far is not necessarily the right way to think about the problem.
- Statisticians normally first divide problems into classification problems (where a categorical variable is predicted) versus regression problems (where a real number is predicted).
- In between is the specialized field of ordinal regression where a ranking is predicted.
- Machine learning for ad hoc retrieval is most properly thought of as an ordinal regression problem, where the goal is to rank a set of documents for a query, given training data of the same sort.
- Next up: ranking SVMs, a machine learning method that learns an ordering directly.

Exercise

Exercise

Example

Example	DocID	Query	Cosine	ω	Judgment
Φ_1	37	linux	0.051	3	relevant
Φ_2	37	linux	0.04	5	nonrelevant
Φ_3	238	operating system	0.3	2	relevant
Φ_4	238	operating system	0.12	3	relevant
Φ_5	518	runtime	0.04	2	relevant
Φ_6	518	runtime	0.005	10	nonrelevant

Give parameters a, b, c of a line $a\alpha + b\omega + c$ that separates relevant from nonrelevant.

Outline

Basic setup for ranking SVMs

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.
- But we do not represent them as query-document-judgment triples.

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.
- But we do not represent them as query-document-judgment triples.
- Instead, we ask judges, for each training query q, to order the documents that were returned by the search engine with respect to relevance to the query.

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.
- But we do not represent them as query-document-judgment triples.
- Instead, we ask judges, for each training query q, to order the documents that were returned by the search engine with respect to relevance to the query.
- We again construct a vector of features $\psi_j = \psi(d_j, q)$ for each query-document pair – exactly as we did before.

Basic setup for ranking SVMs

- As before we begin with a set of judged query-document pairs.
- But we do not represent them as query-document-judgment triples.
- Instead, we ask judges, for each training query q, to order the documents that were returned by the search engine with respect to relevance to the query.
- We again construct a vector of features $\psi_j = \psi(d_j, q)$ for each query-document pair – exactly as we did before.
- For two documents d_i and d_j, we then form the vector of feature differences:

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

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.
- Notation: We write $d_i \prec d_j$ for "d_i precedes d_j in the results ordering".

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.
- Notation: We write $d_i \prec d_j$ for "d_i precedes d_j in the results ordering".
- If d_i is judged more relevant than d_j, then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1.

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.
- Notation: We write $d_i \prec d_j$ for "d_i precedes d_j in the results ordering".
- If d_i is judged more relevant than d_j, then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1.
- This gives us a training set of pairs of vectors and "precedence indicators". Each of the vectors is computed as the difference of two query-document vectors.

Training a ranking SVM

- Vector of feature differences: $\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$
- By hypothesis, one of d_i and d_j has been judged more relevant.
- Notation: We write $d_i \prec d_j$ for "d_i precedes d_j in the results ordering".
- If d_i is judged more relevant than d_j, then we will assign the vector $\Phi(d_i, d_j, q)$ the class $y_{ijq} = +1$; otherwise -1.
- This gives us a training set of pairs of vectors and "precedence indicators". Each of the vectors is computed as the difference of two query-document vectors.
- We can then train an SVM on this training set with the goal of obtaining a classifier that returns

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

Advantages of Ranking SVMs vs. Classification/regression

Advantages of Ranking SVMs vs. Classification/regression

- Documents can be evaluated relative to other candidate documents for the same query, rather than having to be mapped to a global scale of goodness.

Advantages of Ranking SVMs vs. Classification/regression

- Documents can be evaluated relative to other candidate documents for the same query, rather than having to be mapped to a global scale of goodness.
- This often is an easier problem to solve since just a ranking is required rather than an absolute measure of relevance.

Advantages of Ranking SVMs vs. Classification/regression

- Documents can be evaluated relative to other candidate documents for the same query, rather than having to be mapped to a global scale of goodness.
- This often is an easier problem to solve since just a ranking is required rather than an absolute measure of relevance.
- Especially germane in web search, where the ranking at the very top of the results list is exceedingly important.

Why simple ranking SVMs don't work that well

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- Some queries have many relevant documents, others few.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- Some queries have many relevant documents, others few.
  - Depending on the training regime, too much emphasis may be put on queries with many relevant documents.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- Some queries have many relevant documents, others few.
  - Depending on the training regime, too much emphasis may be put on queries with many relevant documents.
- In most IR settings, getting the order of the top documents right is key.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- Some queries have many relevant documents, others few.
  - Depending on the training regime, too much emphasis may be put on queries with many relevant documents.
- In most IR settings, getting the order of the top documents right is key.
  - In the simple setting we have described, top and bottom ranks will not be treated differently.

Why simple ranking SVMs don't work that well

- Ranking SVMs treat all ranking violations alike.
  - But some violations are minor problems, e.g., getting the order of two relevant documents wrong.
  - Other violations are big problems, e.g., ranking a nonrelevant document ahead of a relevant document.
- Some queries have many relevant documents, others few.
  - Depending on the training regime, too much emphasis may be put on queries with many relevant documents.
- In most IR settings, getting the order of the top documents right is key.
  - In the simple setting we have described, top and bottom ranks will not be treated differently.
- $\rightarrow$ Learning-to-rank frameworks actually used in IR are more complicated than what we have presented here.

Example for superior performance of LTR

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).
Proposed by: Yue, Finley, Radlinski, Joachims, ACM SIGIR 2007.

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).

Proposed by: Yue, Finley, Radlinski, Joachims, ACM SIGIR 2007.

Performance compared to state-of-the-art models: cosine, tf-idf, BM25, language models (Dirichlet and Jelinek-Mercer)

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).
Proposed by: Yue, Finley, Radlinski, Joachims, ACM SIGIR 2007.
Performance compared to state-of-the-art models: cosine, tf-idf, BM25, language models (Dirichlet and Jelinek-Mercer)

	TREC 9		TREC 10	
Model	MAP	W/L	MAP	W/L
SVM^{Δ}_{map}	0.242	–	0.236	–
Best Func.	0.204	39/11 **	0.181	37/13 **
2nd Best	0.199	38/12 **	0.174	43/7 **
3rd Best	0.188	34/16 **	0.174	38/12 **

Example for superior performance of LTR

SVM algorithm that directly optimizes MAP (as opposed to ranking).
Proposed by: Yue, Finley, Radlinski, Joachims, ACM SIGIR 2007.
Performance compared to state-of-the-art models: cosine, tf-idf, BM25, language models (Dirichlet and Jelinek-Mercer)

	TREC 9		TREC 10	
Model	MAP	W/L	MAP	W/L
SVM^{Δ}_{map}	0.242	–	0.236	–
Best Func.	0.204	39/11 **	0.181	37/13 **
2nd Best	0.199	38/12 **	0.174	43/7 **
3rd Best	0.188	34/16 **	0.174	38/12 **

Learning-to-rank clearly better than non-machine-learning approaches

Optimizing scaling/representation of features

Optimizing scaling/representation of features

- Both of the methods that we've seen treat the features as given and do not attempt to modify the basic representation of the query-document pairs.

Optimizing scaling/representation of features

- Both of the methods that we've seen treat the features as given and do not attempt to modify the basic representation of the query-document pairs.
- Much of traditional IR weighting involves nonlinear scaling of basic measurements (such as log-weighting of term frequency, or idf).

Optimizing scaling/representation of features

- Both of the methods that we've seen treat the features as given and do not attempt to modify the basic representation of the query-document pairs.
- Much of traditional IR weighting involves nonlinear scaling of basic measurements (such as log-weighting of term frequency, or idf).
- At the present time, machine learning is very good at producing optimal weights for features in a linear combination, but it is not good at coming up with good nonlinear scalings of basic measurements.

Optimizing scaling/representation of features

- Both of the methods that we've seen treat the features as given and do not attempt to modify the basic representation of the query-document pairs.
- Much of traditional IR weighting involves nonlinear scaling of basic measurements (such as log-weighting of term frequency, or idf).
- At the present time, machine learning is very good at producing optimal weights for features in a linear combination, but it is not good at coming up with good nonlinear scalings of basic measurements.
- This area remains the domain of human feature engineering.

Assessment of learning to rank

Assessment of learning to rank

- The idea of learning to rank is old.

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper
- But it is only very recently that sufficient machine learning knowledge, training document collections, and computational power have come together to make this method practical and exciting.

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper
- But it is only very recently that sufficient machine learning knowledge, training document collections, and computational power have come together to make this method practical and exciting.
- While skilled humans can do a very good job at defining ranking functions by hand, hand tuning is difficult, and it has to be done again for each new document collection and class of users.

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper
- But it is only very recently that sufficient machine learning knowledge, training document collections, and computational power have come together to make this method practical and exciting.
- While skilled humans can do a very good job at defining ranking functions by hand, hand tuning is difficult, and it has to be done again for each new document collection and class of users.
- The more features are used in ranking, the more difficult it is to manually integrate them into one ranking function.

Assessment of learning to rank

- The idea of learning to rank is old.
  - Early work by Norbert Fuhr and William S. Cooper
- But it is only very recently that sufficient machine learning knowledge, training document collections, and computational power have come together to make this method practical and exciting.
- While skilled humans can do a very good job at defining ranking functions by hand, hand tuning is difficult, and it has to be done again for each new document collection and class of users.
- The more features are used in ranking, the more difficult it is to manually integrate them into one ranking function.
- Web search engines use a large number of features $\rightarrow$ web search engines need some form of learning to rank.

Exercise

Exercise

Write down the training set from the last exercise as a training set for a ranking SVM.

Recall: Vector of feature differences:
$\Phi(d_i, d_j, q) = \psi(d_i, q) - \psi(d_j, q)$,
$\vec{w}^{\mathsf{T}}\Phi(d_i, d_j, q) > 0 \quad \text{iff} \quad d_i \prec d_j$

Example

Example	DocID	Query	Cosine	ω	Judgment
Φ_1	37	linux	0.03	3	relevant
Φ_2	37	penguin	0.04	5	nonrelevant
Φ_3	238	operating system	0.04	2	relevant
Φ_4	238	runtime	0.02	3	nonrelevant

Take-away today

- Basic idea of learning to rank (LTR): We use machine learning to learn the relevance score (retrieval status value) of a document with respect to a query.
- Zone scoring: a particularly simple instance of LTR
- Machine-learned scoring as a general approach to ranking
- Ranking SVMs

Resources

- Chapters 6 and 15 of IIR
- Resources at `http://cislmu.org`
  - References to ranking SVM results
  - Microsoft learning to rank datasets