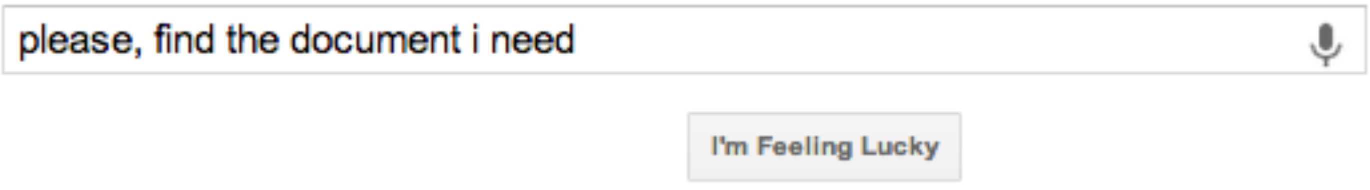
LEARNING TO RANK

Claudio Lucchese
claudio.lucchese@isti.cnr.it

Ranking

- ***Ranking*** is (one of) the most important challenges in Web Search



- We define ***Ranking*** as the problem of sorting a set of documents according to their ***relevance*** to the user query.

Vector space model

- Documents and queries are represented by a ***vector of term weights***
  - weights could be binary, frequency, or something more complex...

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

$$\begin{array}{ccccc} & Term_1 & Term_2 & \ldots & Term_t \\ Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\ Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\ \vdots & \vdots & & & \\ Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt} \end{array}$$

- Document and queries are points in a ***t*-dimensional space**
  - where *t* is the size of the lexicon

Vector space model

- Documents ranked by their *distance/similarity* from the query

$$Cosine(D_i, Q) = \frac{\sum_{j=1}^{t} d_{ij} \cdot q_j}{\sqrt{\sum_{j=1}^{t} {d_{ij}}^2 \cdot \sum_{j=1}^{t} {q_j}^2}}$$

BM25

- BM25 is a *probabilistic model*:
  - using *term independence* assumption to approximate the document probability of being relevant

$$\mathrm{BM25}(d,q) = \sum_t \mathrm{IDF}_t \, \tau(F_t)$$

  - ***$IDF_t = log(N/n_t)$*** is the inverse document frequency
    - N is the number of doc.s in the collection
    - n_t is the number of doc.s containing t
  - frequent terms are not very specific, and their contribution is reduced

BM25

$$\mathrm{BM25}(d,q) = \sum_t \mathrm{IDF}_t \, \tau(F_t)$$

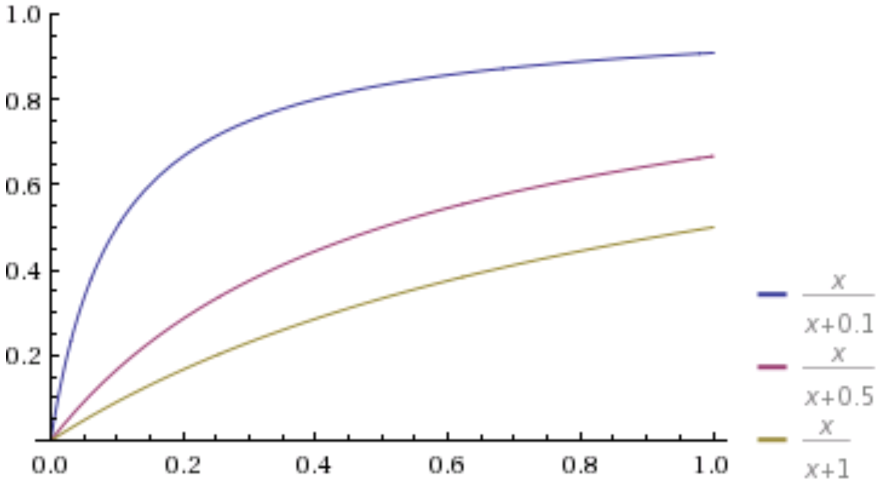
$$F_t = \frac{f_{t,d}}{1 - b + b \cdot l_d / L}$$

- ***$f_{t,d}$*** is the frequency of term *t* in document *d*
- ***l_d*** is the length of document *d*
  - longer documents are less important
- ***L*** is the average document length in the collection
- ***b*** determines the importance of l_d

BM25

$$\mathrm{BM25}(d,q) = \sum_t \mathrm{IDF}_t\, \tau(F_t)$$

$$\tau(F_t) = \frac{F_t}{k + F_t}$$



- τ is a ***smoothing function***, modeling *non-linearity* of terms contribution
  - *Note 1*: let t_1, t_2, t_3 have frequencies 0.1, 0.2, 0.3, the relative importance of t_2 w.r.t. t_1 is greater than t_3 w.r.t. t_2
  - *Note 2*: above a certain threshold the terms' contribution are equally important and not so discriminative

BM25 and BM25F

- BM25 can be extended to handle *structured (multi-field) documents*
  - e.g., title, abstract, summary, author
  - e.g., title, url, body, anchor
- The extended function is named ***BM25F***:

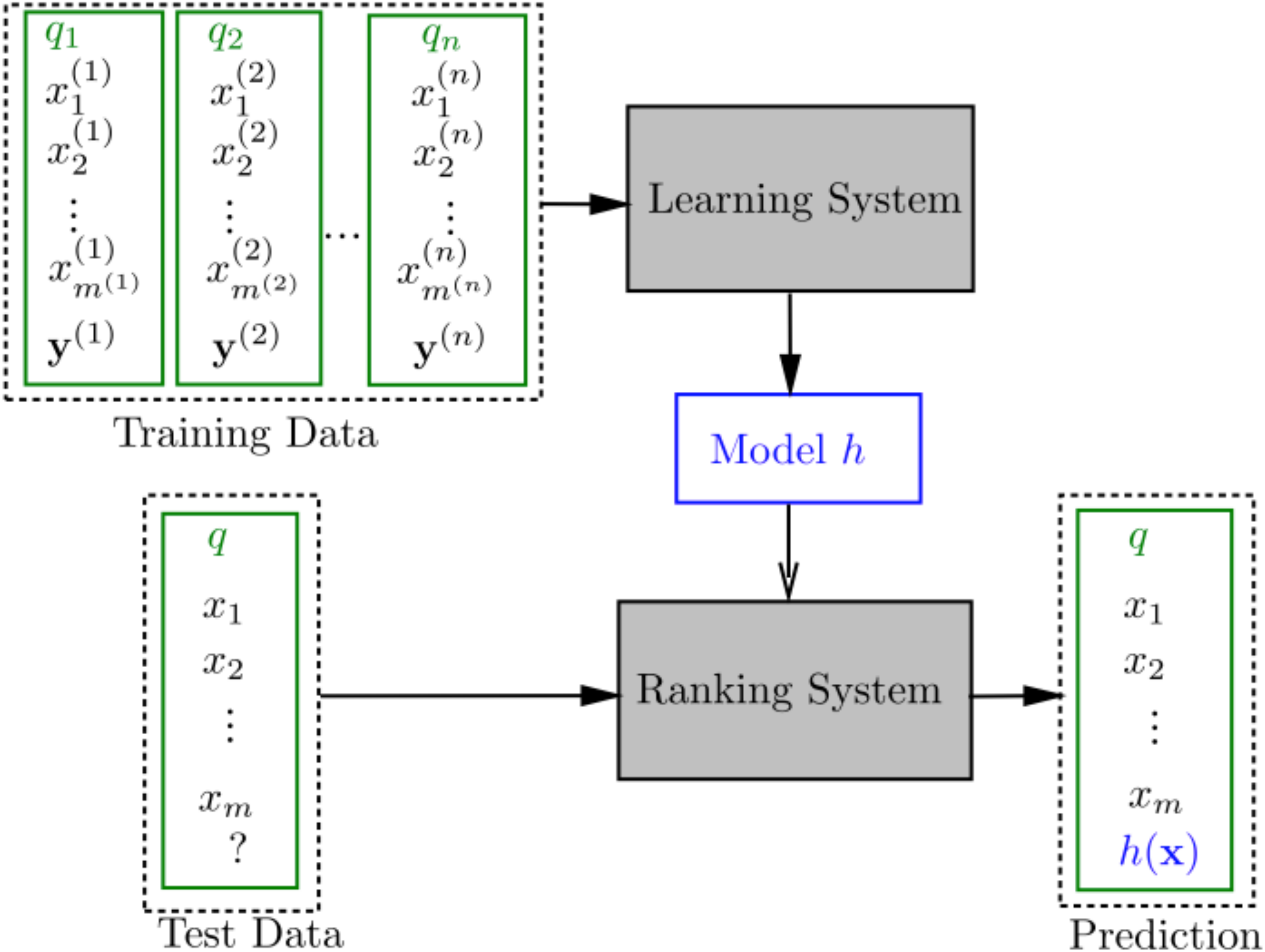
$$\mathrm{BM25F}(d,q) = \sum_t \mathrm{IDF}_t\, \tau(F_t) \qquad F_t = \sum_s \frac{w_s \cdot f_{t,s}}{1 - b_s + b_s \cdot l_s / L_s}$$

  - w_s is a weight of field *s*
  - $f_{t,s}$ is the frequency of term *t* in field *s* (of document *d*)
  - l_s is the length of field *s* (of document *d*)
  - b_s determines the importance of l_s
  - L_s is the average length of field s in the collection

Need for tuning of BM25(F)

- BM25 has **2** free parameters:
  - b,k
- BM25F has **2*S*+1** free parameters (*S* is the no. of fields)
  - w_s, b_s, k
- How to find the best parameter of BM25(F) ?

- *Fit it into a learning-to-rank framework*



q_1
$x_1^{(1)}$
$x_2^{(1)}$
$\vdots$
$x_{m^{(1)}}^{(1)}$
$\mathbf{y}^{(1)}$
q_2
$x_1^{(2)}$
$x_2^{(2)}$
$\vdots$
$x_{m^{(2)}}^{(2)}$
$\mathbf{y}^{(2)}$
$\cdots$
q_n
$x_1^{(n)}$
$x_2^{(n)}$
$\vdots$
$x_{m^{(n)}}^{(n)}$
$\mathbf{y}^{(n)}$
Training Data
Learning System
Model h
q
x_1
x_2
$\vdots$
x_m
?
Test Data
Ranking System
q
x_1
x_2
$\vdots$
x_m
$h(\mathbf{x})$
Prediction

Some features

- http://research.microsoft.com/en-us/projects/mslr/feature.aspx

Baselines of Yahoo! LtR Challenge

	Validation		Test	
	ERR	NDCG	ERR	NDCG
BM25F-SD	0.42598	0.73231	0.42853	0.73214
RankSVM	0.43109	0.75156	0.43680	0.75924
GBDT	0.45625	0.78608	0.46201	0.79013

- BM25F-SD, is a variant of BM25F including proximity
- RankSVM uses a linear kernel
- GBDT (Gradient Boosted Decision Tree) … you will see next

Learning-to-Rank framework

We need:

1. To create a *training set*
   - training queries, training results and their ***relevance annotation***
2. To define an *objective function* to be optimized
   - how to define the quality of a result set?
3. To chose a *machine learning algorithm*
   - Gradient Descent, Neural Networks, Regression trees, Support Vector Machines, …

Learning to Rank approaches

- ***Pointwise***
  - Each query-document pair is associated with a score
  - The objective is to predict such score
    - can be considered a *regression problem*
  - Does not consider the position of a document into the result list
- ***Pairwise***
  - We are given pairwise preferences, d_1 is better than d_2 for query q
  - The objective is to predict a score that preserves such preferences
    - Can be considered a *classification problem*
  - It partially considers the position of a document into the result list
- ***Listwise***
  - We are given the ideal ranking of results for each query
    - NB. It might not be trivial to produce such training set
  - Objective maximize the quality of the resulting ranked list
    - We need some improved approach…

L-t-R applied to BM25F

- We have a training set of ***query/results***
  - Each query has a set of candidate results
  - Each results was *manually annotated with a relevance label (e.g. 1 to 5)*
- A very simple learning algorithm
  - *Gradient Descent*
- We chose a specific quality function
  - ***Normalized Discounted Cumulative Gain @ K***
    - NDCG@K

NDCG @K

- Rationale:
  - Consider only the ***top-K*** ranked documents, and sum up (***cumulate***) their contribution
  - The contribution (***gain***) of a result depends on its ***relevance label***
  - Contribution is diminished (***discounted***) if the result is in the "bottom" positions
  - ***Normalize*** between 0 and 1

$$\mathrm{DCG@k} = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log(i+1)} \quad \mathrm{NDCG@k} = \frac{\mathrm{DCG@k}}{\mathrm{IDCG@k}}$$

- ***rel_i*** is the relevance label of the *i*-th result (e.g., 1..5)
- ***IDCG@k*** is the score of the ideal ranking

L-t-R applied to BM25F

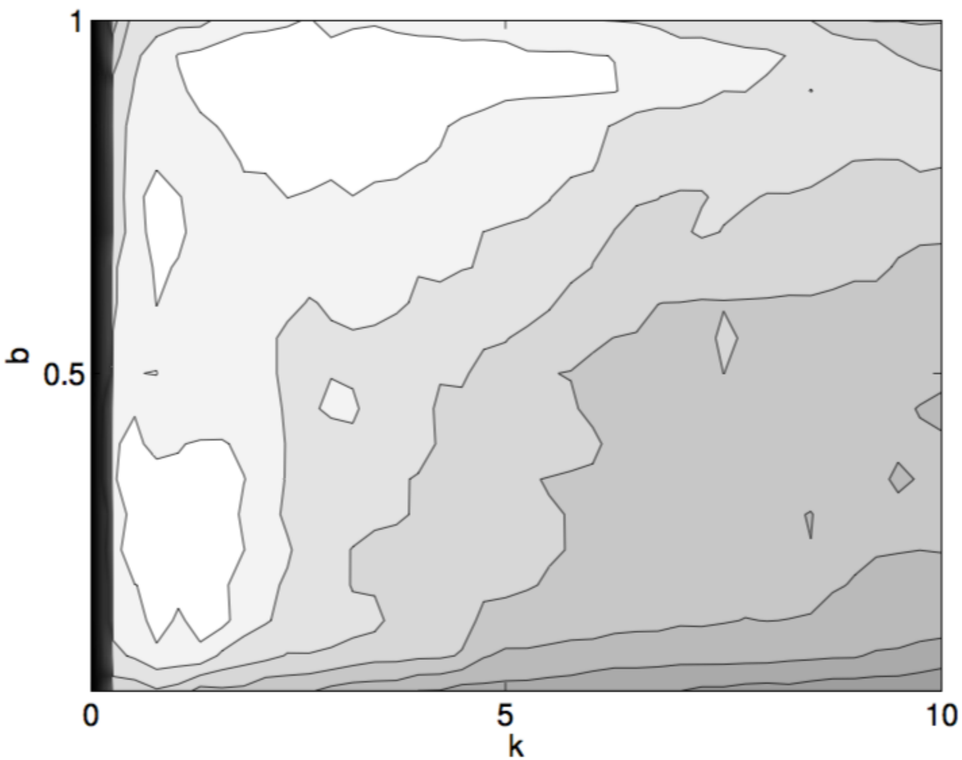
- Given a query ***q*** and a set of documents ***D={d_1,d_2, …}***
  - *Results = retrieve (D | q)*

- We want to learn a **model *h*** that allows to rank the documents in D according to their relevance
  - Results = sort *{h(d_1), h(d_2), …}*
  - where the function ***h* is *BM25F*** with a proper parameter set θ

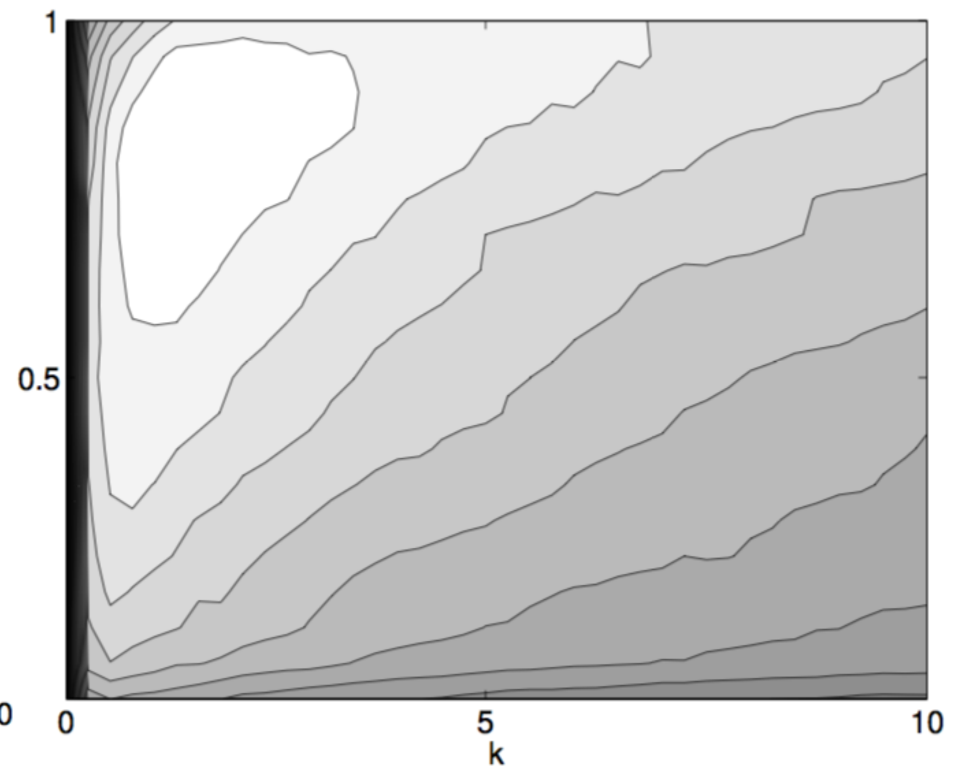
- How to apply Gradient Descent ?
  - we need to compute the ***gradient*** of *sort* w.r.t. θ
  - *but sort is not a continuous and derivable function!*
  - We cannot apply gradient descent

- One of the issues in LtR is how to optimize the sorted results "bypassing" the *sort* operation

NDCG on real-world data



train



test

- NDCG averaged over 512 queries

Pairwise approach

- We are given a collection *R* of document pairs (d_i, d_j), for each pair we now that d_i is better d_j

- Our goal is to find the ***best ranking function r****, such that for every pair $(d_i, d_j) \in R$, $r^*(d_i) > r^*(d_j)$, or such that the *smallest number of such constraints is violated*

- This problem is known to be NP-Hard (rank aggregation), therefore, we need to find some smart approximation

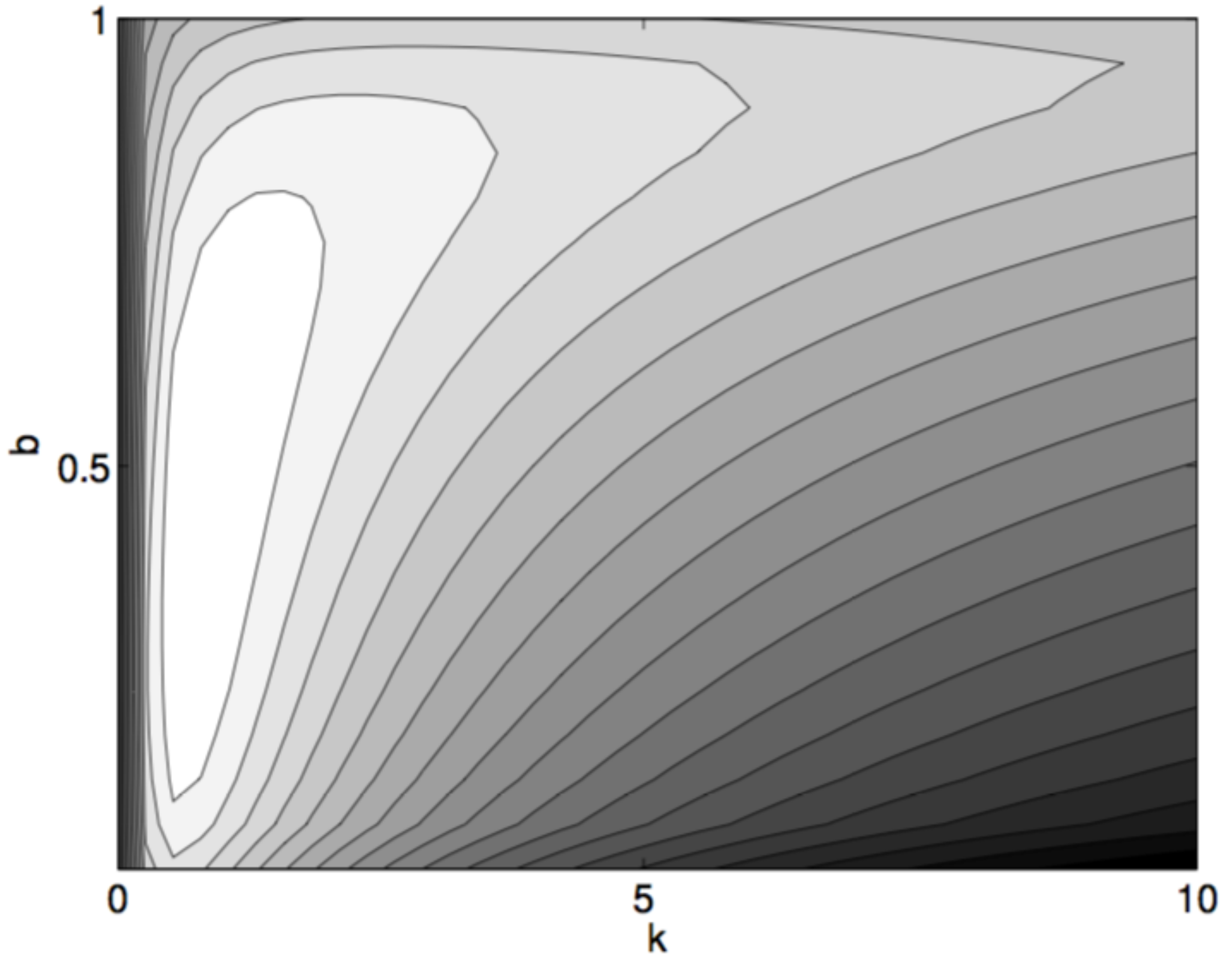
RankNet

- Let the training set be ***result pairs (d_1,d_2)*** where d_1 is *better than* d_2
  - we also say that the ***(true) probability*** that d_1 is better d_2 is $T_{12}=1$
- Let ***h(d)*** be the *score* of document *d*, computed by the learned model
- We define the score difference $Y=h(d_2)-h(d_1)$
  - If $Y<0$ then the documents are ranked correctly
- We map ***Y*** to the ***probability*** P_{12} that d_1 is better d_2 with a logistic function $P_{12} = e^{-Y}/(1+e^{-Y})$
- We measure the ***error*** of the model with ***cross entropy between*** P_{12} ***and*** T_{12}:
  - $C = -T_{12} \log P_{12} - (1-T_{12})\log(1-P_{12})$
    - Cross entropy can be thought as the number of bits needed to encode T_{12} given a coding scheme based on P_{12}
- Since $T_{12}=1$
  - $C = \log(1+e^{Y})$

RankNet

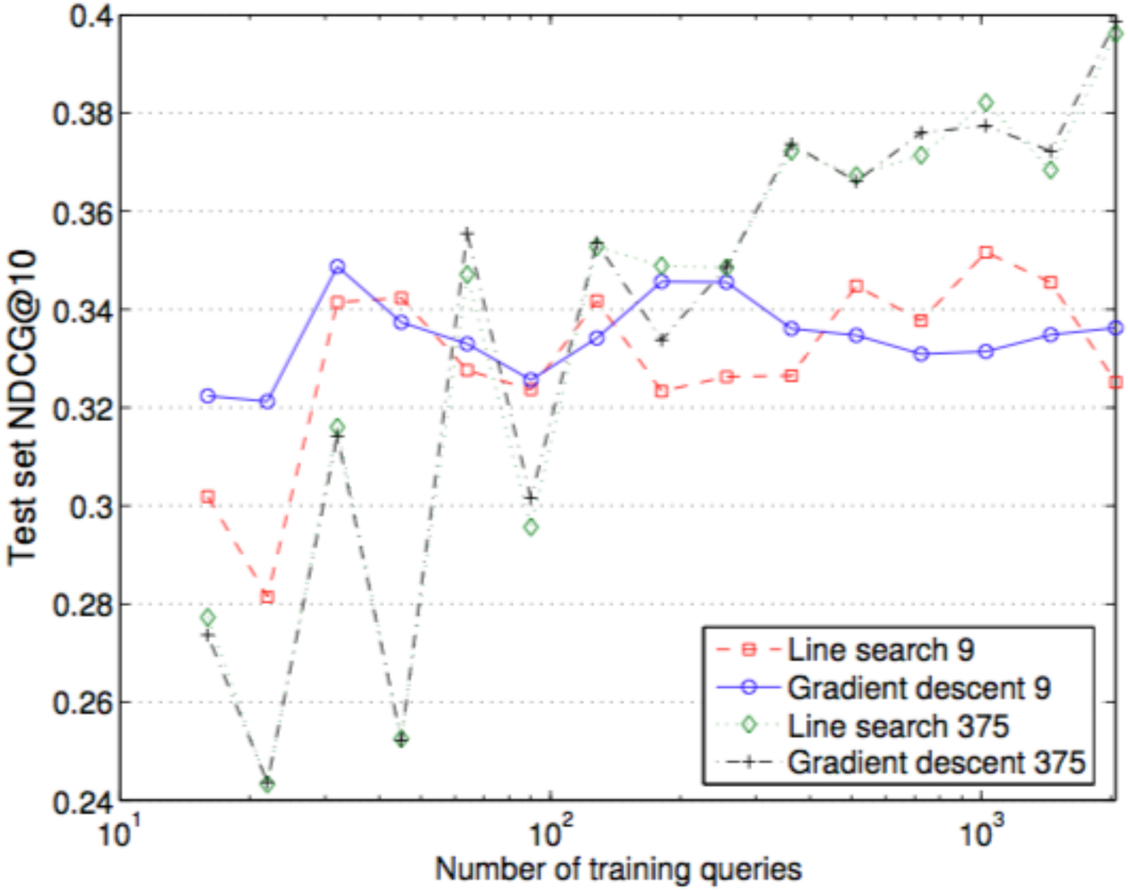
- $C = \log(1+e^{Y})$
- What did we get ?
  - C is minimum if all pairs are ranked in the proper order, therefore ***by minimizing C we improve NDCG***
    - this does <u>*not*</u> imply that the *optimal solution for C is the optimal solution for NDCG or other quality measures*
  - we can compute the ***gradient of C***
    - If *h* is differentiable then also *Y* and C are
- We can directly apply steepest descent
  - Just need derivatives of BM25F

L-t-R applied to BM25F



train

Evaluation



- Our alternative formulation is a good proxy for *NDCG* optimization

L-t-R applied to BM25F (I)

- ***Line Search***
  - It is a *general-purpose optimization algorithm*
  - It ***computes NDCG directly*** by varying "smartly" the parameters of BM25F
- Start from an initial guess $\theta=\{\theta_1, \theta_2, \ldots\}$
  - For each θ_i,
    - consider ***n* sample points z_i** within the interval $[\theta_i - w, \theta_i + w]$
      - keep *fixed all other parameters*
    - ***compute NDCG for each sample point*** and store the best z_i
  - Take the ***line connecting θ to Z***$=\{z_1, z_2, \ldots\}$
    - consider ***n* sample points** along this line
    - compute NDCG for each sample point and take the best result θ'
  - Repeat starting from θ'
    - reduce *w* at each iteration
    - stop until convergence, or until the maximum number of iterations is reached

L-t-R applied to BM25F (II)

- Rationale:
  - apply Gradient Descent bypassing the sorting of results
- Transform the training set into ***result pairs*** $(\mathbf{d_1},\mathbf{d_2})$ where d_1 has a larger label than d_2
  - we also say that the ***(true) probability*** that d_1 is better d_2 is $T_{12}=1$
- We define $Y=h(d_2)-h(d_1)$, and we model the ***probability*** P_{12} that d_1 is better d_2 with a logistic function $P_{12} = e^{-Y}/(1+e^{-Y})$
- We measure the ***error*** of the model with ***cross entropy***:
  - $C = -T_{12} \log P_{12} - (1-T_{12})\log(1-P_{12})$
    - Cross entropy can be thought as the number of bits needed to encode T_{12} given a coding scheme based on P_{12}
- Since $T_{12}=1$
  - $C = log(1+e^{Y})$

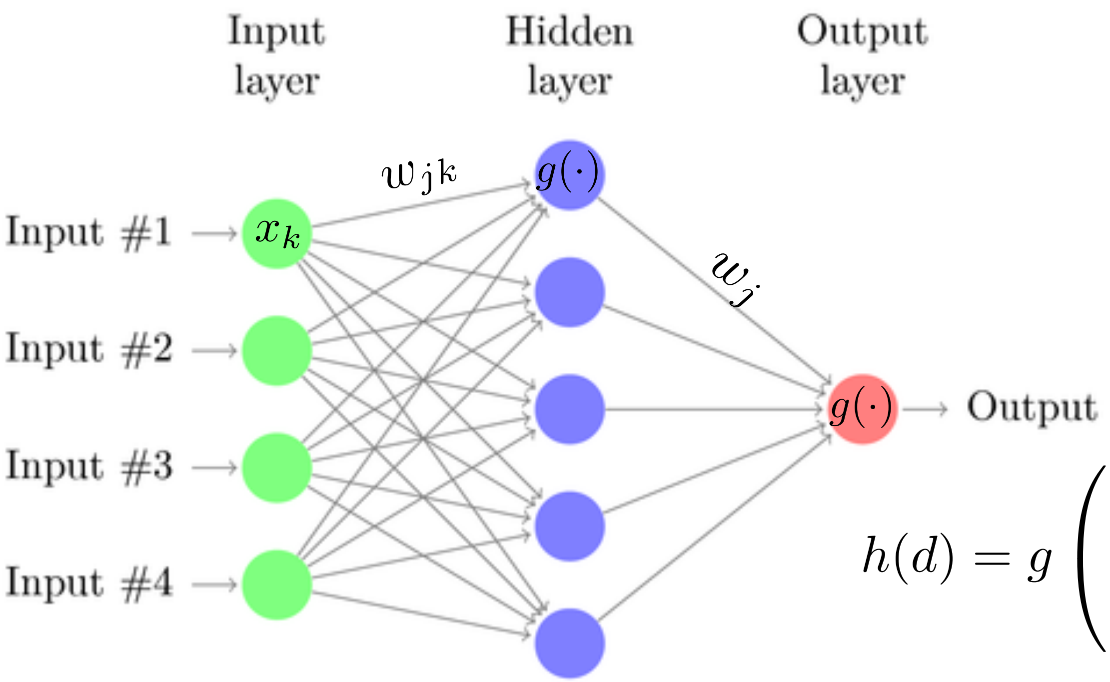
L-t-R applied to BM25F (II)

- ***$C = \log(1+e^Y)$***
- What did we get ?
  - we can compute the ***gradient of C***
    - C is a function of *Y*, and *Y* is a function of *BM25F*, and ***all are derivable***
  - working with the ***gradient is much cheaper*** than re-ranking all results to compute the *NDCG*
  - C is minimum if all pairs are ranked in the proper order, therefore, ***by minimizing C we improve NDCG***
    - this does <u>*not*</u> imply that the *optimal solution for C is the optimal solution for NDCG*

Comparison

- Line search:
  - *Pros*: it optimizes NDCG
  - *Cons*: It is expensive and therefore it may not scale to large features/training sets
- Alternative optimization solution:
  - *Pros*: it can be fast by using a gradient descent method
    - It scales with the number of features
    - It might require some subsampling of the training pairs
  - *Cons*: it optimizes a different cost function

RankNet



$$h(d) = g\left(\sum_j w_j g\left(\sum_k w_{jk} x_k + b_j\right) + b\right)$$

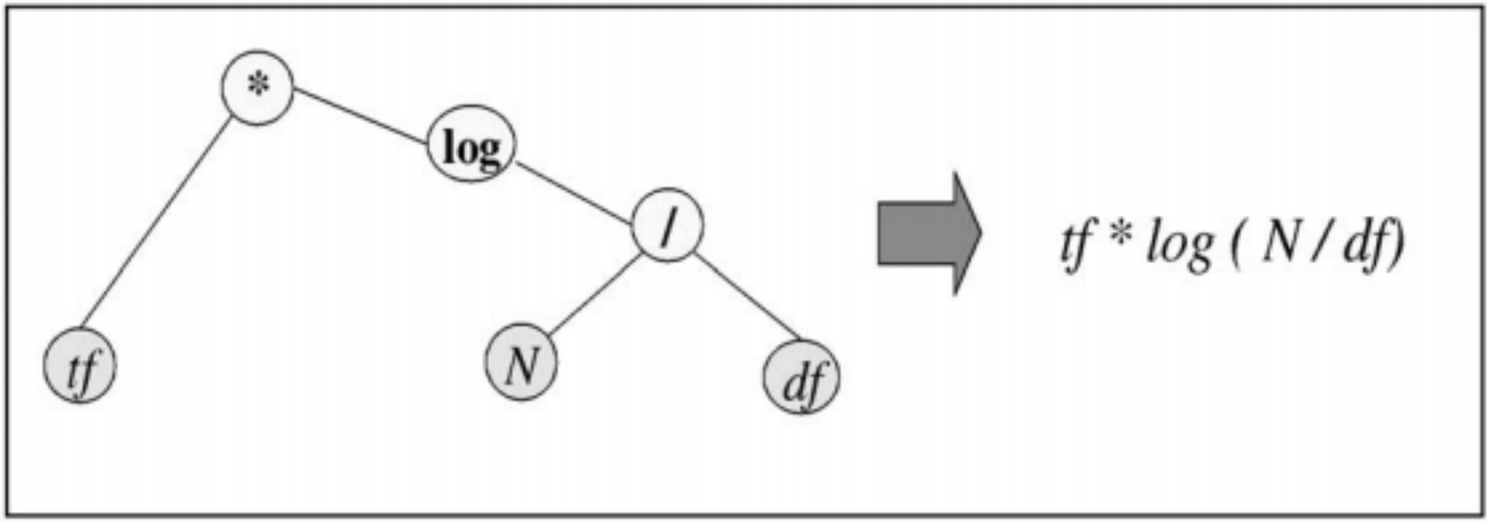
- x_k, is the *k*-th feature of document *d*
- *w* and *b* are the weights and offset
- *g* is a *non–linear activation function*, usually sigmoid
- gradient descent is used to find *w* and *b*

Genetic Algorithms

- Overview of a genetic algorithm:
  1. Generate a ***random population*** of solutions
  2. ***Score each individual*** in the population
  3. (*reproduction*) Select some of the ***best individuals***
  4. (*crossover*) Select pairs at random and ***"mix"*** their representation
     - Repeat to get a sufficient number of individuals
  5. Repeat from step 2 with the new population
     - until a maximum number of iterations

Genetic Algorithms

- The trick is in the representation



- Trees can represent complex functions, where nodes are operations and leaves are features
- Crossover is performed by exchanging subtrees at random

Genetic Algorithms

- Operations:
  - +, *, /, log
- Features

Terminals	*Statistical Meaning*
tf	Same as TF: how many times the term appeared in a document
tf_max	The maximum tf for a document
tf_avg	The average tf for a document
tf_doc_max	The maximum tf in the entire document collection
df	Same as DF: the number of unique documents the term appeared
df_max	The maximum df for the entire collection
N	The total number of documents in the entire text collection
length	The length of a document
length_avg	The average length of a documents in the entire collection
R	The real constant number randomly generated by the GP system
n	The number of unique terms in a document

Some Results

Query	GA	BM25	NN	GA vs. BM25	GA vs. NN
Short	**0.25**	0.23	0.11	+10.71%	+130%
Long	**0.36**	0.31	0.23	+17.01%	+56%

- Mean Average Precision
  - Precision is the number of relevant documents divided by the number of returned documents
  - Precision is computed whenever a new relevant document is found in the result list
  - Precision values are eventually averaged

Genetic Algorithm

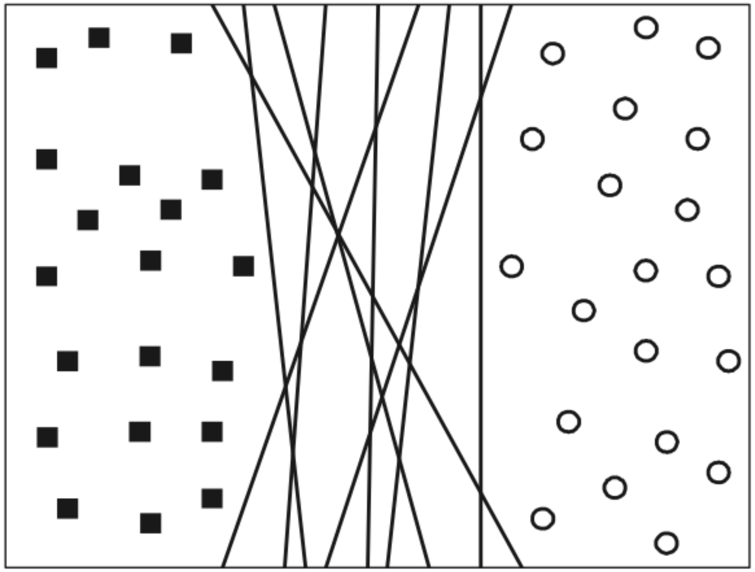
- The formula:

$$\frac{\log\left(tf \times \left(tf\_avg + \frac{tf}{\log(tf^2 \times tf\_avg)} + \frac{tf \times N}{df} \times \frac{tf\_avg \times (tf\_doc\_max + n)}{df}\right)\right)}{n + 2 \times tf\_doc\_max + 0.373}$$

- The authors claim this is somehow similar to BM25
  - ??
- Interestingly
  - Term frequency and inverse document frequency play an important role
  - The denominator is related to the the number of unique terms in the document (~length) and the max term frequency (~length ~specificity)

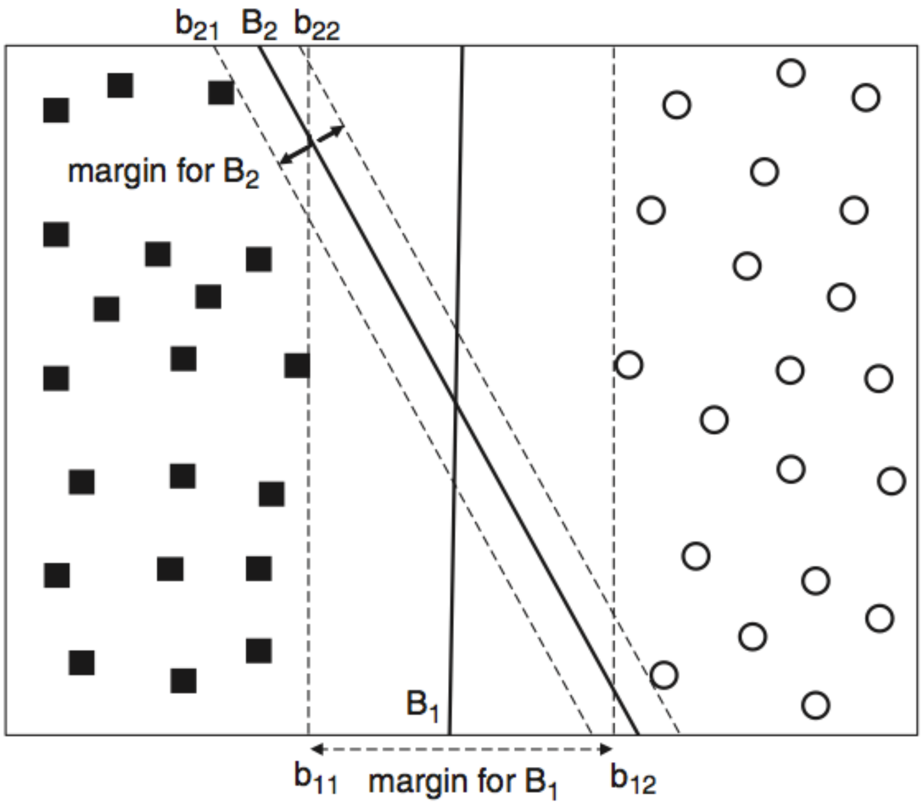
Support Vector Machines

- Classification technique, aiming at maximizing the generalization power of its classification model



  - Given the above points in a 2D space, what is the line that best "separates" the squares from the circle?

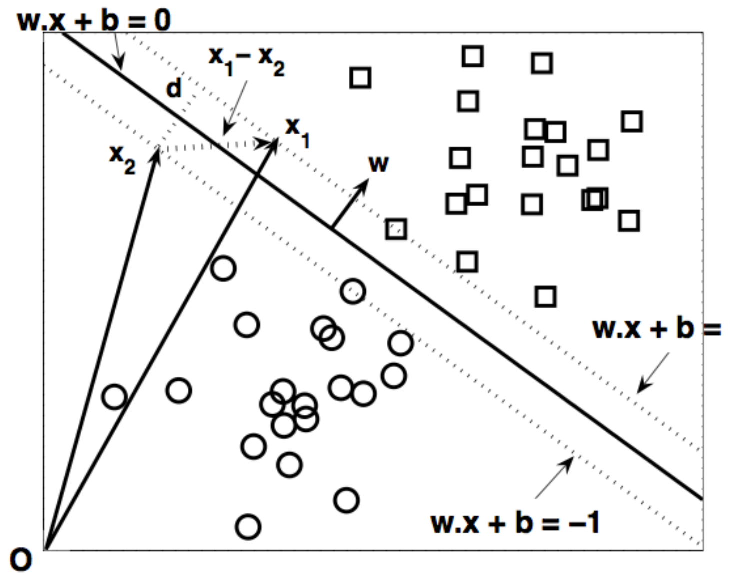
Support Vector Machines



- We call ***margin*** the distance between closest instances of opposite classes along the *perpendicular* direction to the selected decision boundary
  - The smaller the margin, the larger the *misclassification risk*
  - The instances determining the margin are named ***support vectors***

Linear decision boundary

- A linear decision boundary is ***B***:
  - $\boldsymbol{w^T x + b = 0}$
  - where w weighs the features of x
- For objects "*above*" B:
  - $w^T x + b = k'$, with $k'>0$
- For objects "*below*" B:
  - $w^T x + b = k''$, with $k''<0$
  - k' and k'' are *proportional* to the distances from the decision boundary



- Let x_S and x_C be the closest objects of the two classes, we can rescale w and b such that
  - $w^T x_S + b = 1$ and $w^T x_C + b = -1$
  - by definition, the distances d_S and d_C of x_S and x_C from $w^T x + b$ are:
    - $d_S = |w^T x_S + b| / |w|_2 = 1/|w|_2$ *and* $d_C = |w^T x_C + b| / |w|_2 = 1/|w|_2$
  - Therefore the margin $\boldsymbol{d = d_S + d_C = 2/|w|_2}$
- *To maximize the margin d, we should minimize* $|w|_2$

Linear SVM formulation

- Let $y_i \in \{+1,-1\}$ be the class of the *i*-th instance, the (linear) SVM (binary) classification problem is:

  - *Minimize* $\frac{1}{2}\ |w|^2$
  - *Subject to:* $y_i\ (w^T x_i + b) \geq 1$

    or: $y_i\ (w^T x_i + b) - 1 \geq 0$

  - Since the objective function is quadratic, and the constrains are linear in *w* and *b*, this is know to be a *convex optimization problem*.

Linear SVM solution

- Standard technique of Lagrange multipliers. The problem is reformulated as:
  - Minimize: $$L_P = \frac{1}{2}\|w\|^2 - \sum_i \lambda_i \left(y_i(w^T x_i + b) - 1\right)$$
    - where $\lambda_i \geq 0$
  - the first term is the old objective function
  - the second term comes from the previous constraints:
    - If an instance is misclassified, the error generates an increment of the objective function

Linear SVM solution

- It is possible to show that:
  - $\lambda_i \neq 0$ only if x_i is a support vector
  - Minimizing L_P is equivalent to maximizing

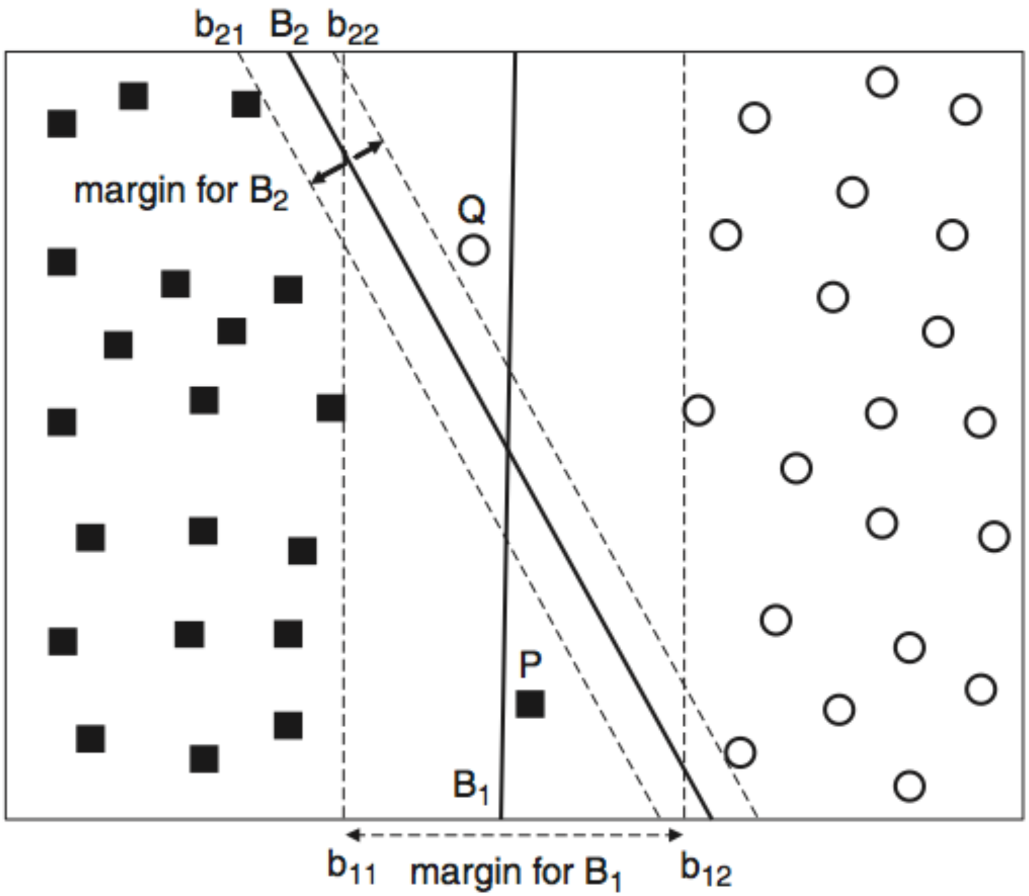
$$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j x_i x_j$$

    - which involves only the data and the Lagrangians
    - L_D is the dual Lagrangian formulation
  - L_D can be solved with numerical methods
  - the decision boundary can be computed as:

$$\left(\sum_i \lambda_i y_i x_i \quad \cdot \quad x \right) + b = 0$$

    - which depends only on the support vectors

Soft margin



- What if a decision boundary has a large margin and a small error rate ?
- What if there is not an error-free decision boundary ?
  - *Non-linearly separable classes*

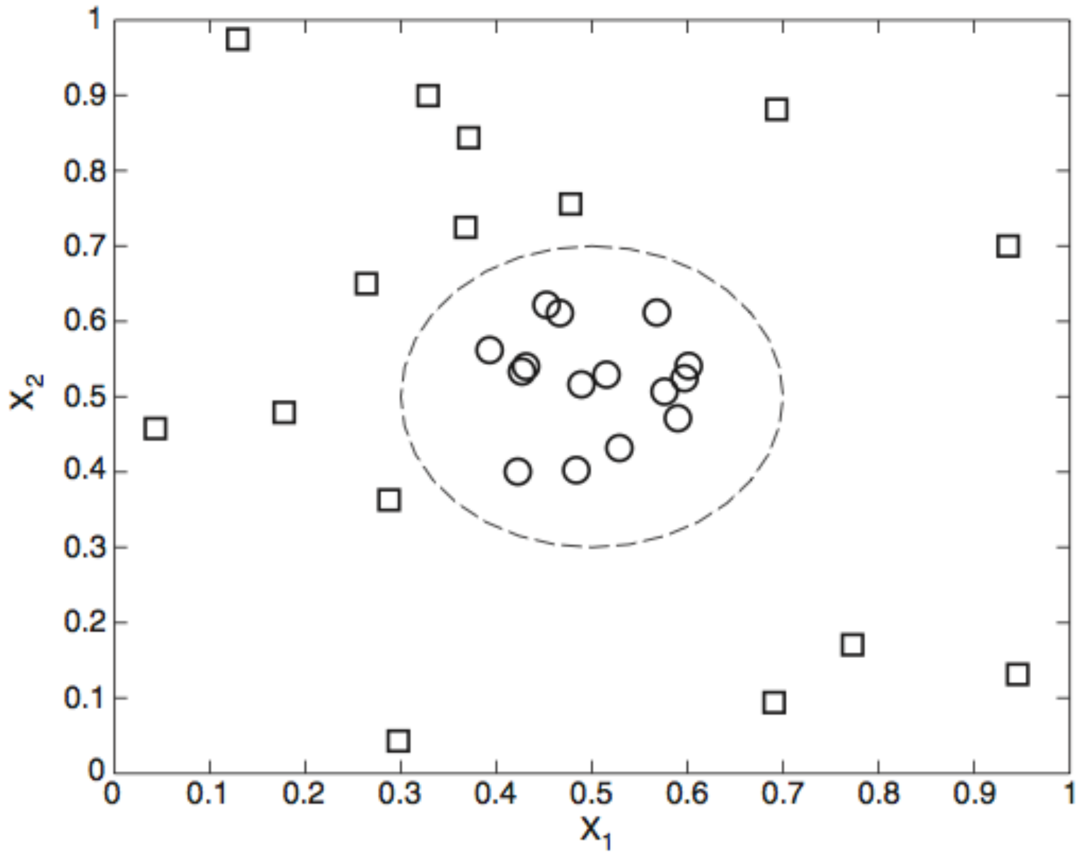
Soft margin

- We need to relax the previous constraints, introducing *slack variables* $\xi_i \geq 0$
  - *Minimize* $\frac{1}{2} \, |w|^2 + \boxed{C \sum \xi_i}$
  - *Subject to:* $y_i \, (w^T x_i + b) \geq 1 \boxed{- \xi_i}$

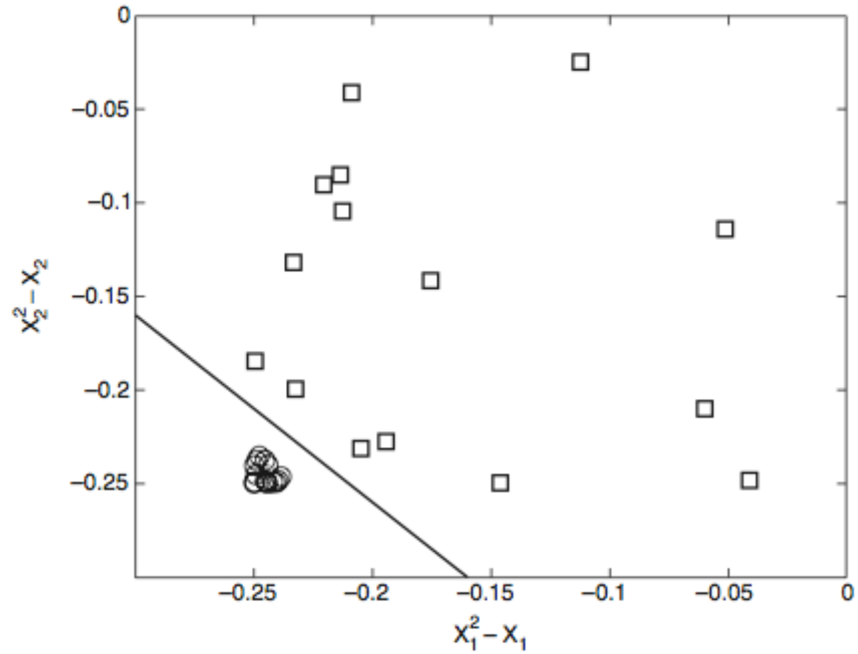
    $\xi_i \geq 0$
  - At the same time, this relaxation must be minimized.
  - C defines the trade-off between training error and large margin
  - The problem has the same dual formulation as before, with addition constraint $0 \leq \lambda_i \leq C$

Nonlinear SVM

- How to deal with a non linear decision boundary ?



Nonlinear SVM



- Idea:
  - First transform the data, potentially mapping to a space with higher dimensionality, then use a linear decision boundary as before.

  - Minimize $\frac{1}{2} \|w\|^2$
  - *Subject to:* $y_i (w^T \boxed{\Phi(x_i)} + b) \geq 1$

  - The dual is: $$L_D = \sum_i \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \Phi(x_i) \Phi(x_j)$$

The Kernel trick

$$K(x_i, x_j) = \Phi(x_i)\Phi(x_j)$$

- Observations:
  - We do not need $\Phi(x_i)$, but the dot product $\Phi(x_i)\,\Phi(x_j)$
  - For some mapping functions Φ, the dot product can be computed directly without explicitly mapping to the new space
  - $K(x_i,x_j)$ can be computed directly from the attributes of x_i,x_j
  - K is called *kernel function*

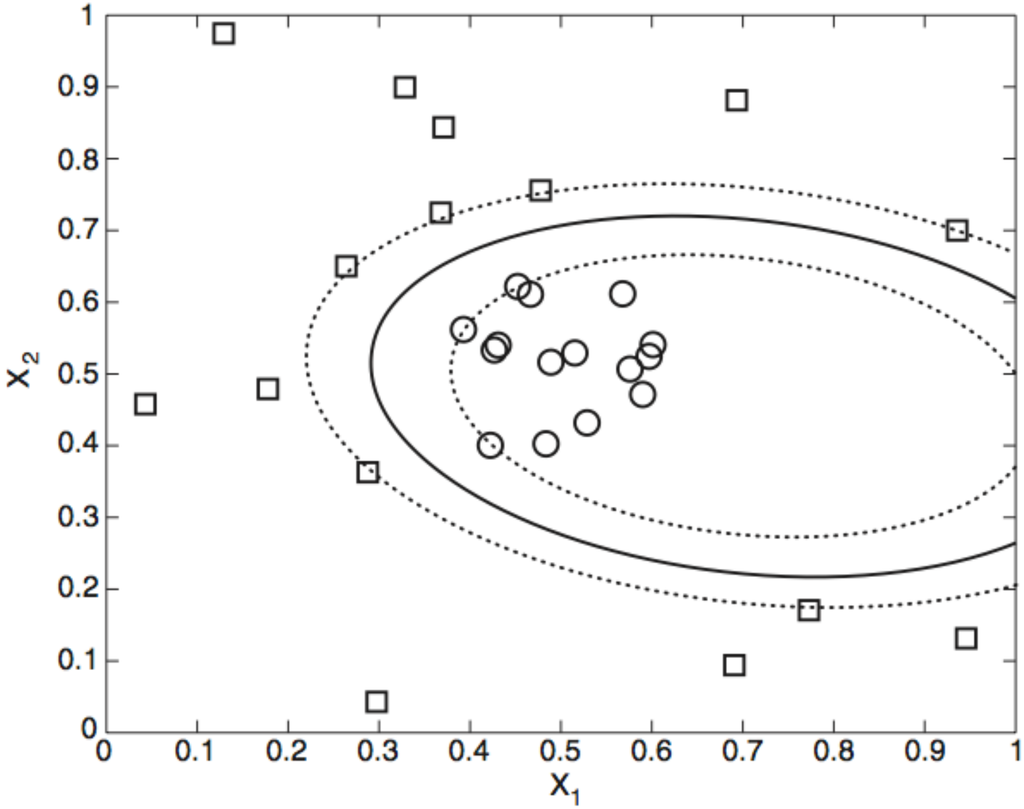
The Kernel trick

- Some kernel functions:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$

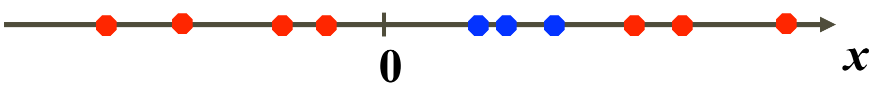
$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)}$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta)$$

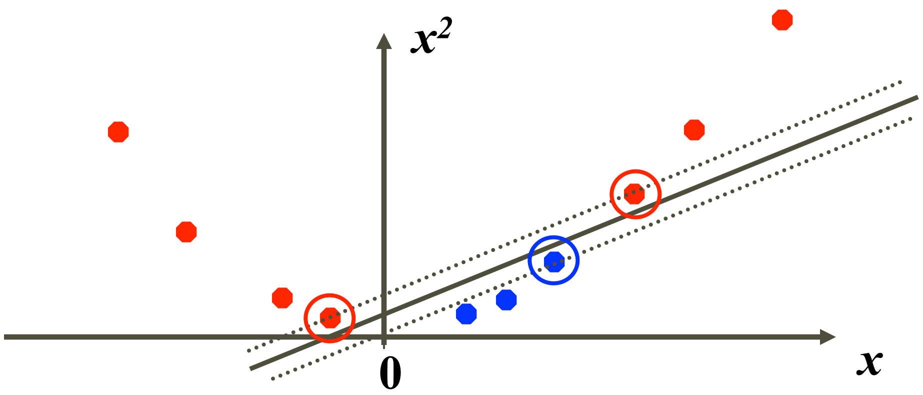


Mapping to many dimensions

- 1D, non linearly separable problem



- After mapping to 2D, it is linearly separable



(Linear) Ranking SVM

- In case of a linear combination of features:
  $h(d) = w^T d$

- Our objective is to find w, such that:
  - $h(d_i) \geq h(d_j)$
  - $w^T d_i \geq w^T d_j$
  - $w^T (d_i - d_j) \geq 0$

- We approximate by adding *slack variables* ξ and minimizing this "relaxation"
  - given the *k*-th document pair, find the weights *w* such that

$$w^T (d_i - d_j) \geq 1 - \xi_k \qquad \text{with} \quad \xi_k \geq 0$$

and ξ_k is minimum

(Linear) Ranking SVM

- The full formulation of the problem is
  - *Minimize* $\frac{1}{2}\,|w|^2 + C \sum_k \xi_k$
  - *Subject to* $w^T(d_i - d_j) \geq 1 - \xi_k$

    $\xi_k \geq 0$
  - where C allows to trade-off error between the margin ($|w|^2$) and the training error
- This is an SVM classification problem !
  - Is convex, with no local optima, it can be generalized to non-linear functions of documents features.

Issues of the pairwise approach

- We might not realized that some queries are really badly ranked
- Top result pairs should be more important than other pairs
- In general, the number of document pairs violations, might not be a good indicator