Document Analysis –Clustering, Classification and Ranking

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

By

SWARNENDU SENGUPTA (1410110456)

SHIV NADAR UNIVERSITY

Under supervision

of

Dr. Madan Gopal

DEPARTMENT OF ELECTRICAL ENGINEERING

SCHOOL OF ENGINEERING

SHIV NADAR UNIVERSITY

(May, 2018)

Candidate Declaration

I hereby declare that the thesis entitled “Document Analysis – clustering, classification and ranking” submitted for the B Tech Degree program. This thesis has written in my own words. I have adequately cited and referenced the original sources.

(Signature)

(Swarnendu Sengupta)

(1410110456)

Date: __________

CERTIFICATE

It is certified that the work contained in the project report titled “Document analysis – clustering, classification and ranking,” by “Swarnendu Sengupta,” has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Dr. Madan Gopal
Department of Electrical Engineering
School of Engineering
Shiv Nadar University
May, 2018

ACKNOWLEDGEMENTS

The knowledge I gained from this project is unparalleled to any other and the experience I had will continue to guide me throughout my life.

First and most importantly, I express my deep gratitude to my supervisor Dr. Madan Gopal, Department of Electrical Engineering for his assistance, encouragement and guidance throughout the course of this project. I truly appreciated and value his esteemed guidance and knowledge from the beginning to the end without which this project wouldn’t have been possible. I also express my deep gratitude to my supervisor for letting me do the final year project in the field of machine learning with document analysis as a topic.

I express my thankfulness to Shiv Nadar University to have provided me with an opportunity to learn under great minds.

Lastly, I would like to take this opportunity to thank my family and friends with notable emphasis on Ashish Kushwaha for their never ending moral support and motivation.

Abstract

The document analysis is an industrial problem which is comprised of three main parts: clustering, classification and ranking. This report is a brief version of all the algorithms that are widely used in the industry. For clustering K-means was used. For classification K-NN with two distance metrics and Random forest was used. For ranking, rankNet, ranking SVM with kernel approximation, linear regression and lambdaMART was used. Here the algorithms used are very naïve and basic in nature. The algorithms of these need not be understood mathematically but an intuitive understanding would be needed to understanding the basic working principle and how does the parameter tuning work. These are followed by a very basic toy dataset which was used and the ranking algorithms were implemented on them. This would eventually help in understanding the concepts of the ranking algorithms and how each parameter would work. This report would not contain complicated mathematical expression but only its implementation and interpretation details followed by the results.

Keywords: ranking, SVM, RankNet, lambdaMART, random forest

Contents

Table of Figures

Introduction

Document analysis has been an emerging challenge in the industry. With the growing number of publishing papers and other documents online, there has been a growing need of searching through documents which is like searching a page on the internet in many ways but also different in many ways. In the search of a webpage, many features like date and frequency of accessing etc. also come into play. However, not so much in the case of document analysis.

The document analysis comprises of main three fields- clustering, classification and ranking of the documents with respect to the queries which is an integral part of information retrieval.

Clustering of documents is based on the fact that many document have certain features or certain patters common to them. They can be clustered together. This is done in several ways like K-means clustering, Gaussian Mixture Model, etc.

Classification of documents is done to categorize the documents into certain fields of interest of the user. These are predefined in the documents. This is a multi-class classification problem. Many models can be used on this like k-nearest neighbors, support vector and advanced techniques like random forest.

Ranking of documents is an industrial level problem. Here, according to different queries given by users different documents are preferred and ranked accordingly. Many algorithms have been developed which include linear regressing, support vector machines, RankNet, etc.

In this paper, clustering and classification has been done on the Reuters21578 dataset while the ranking has been done on MQ2007 dataset provided by Microsoft as Reuters21578 lacked the queries and the ranking part of the problem.

Literature review

A lot of papers were studied for this topic as document classification is an industrial problem. One of the papers that boosted the start of the project was the work done by M. Arun Kumar in his thesis, FAST AND KNOWLEDGE BASED SVM ALGORITHMS WITH APPLICATIONS. Some of the work has already been described there in great detail including the preprocessing steps in the same document dataset. He has mainly worked with SVM (support vector machines) and its variations in respect to the fields where they are being applied. Further analysis was done on the dataset using the list of stop words that was needed for preprocessing the data. Post that Porter's stemming algorithm was studied which gave the direction of clubbing words like "appear" and "appearing" together. After that different weighing algorithms were used and implemented. Post this in order to gain results, various machine learning algorithms were used.

For the second case that involved ranking of datasets, the papers published by Microsoft scientists were studied. [12] described the use of various algorithms that they had used for the processing. Following their footsteps, many algorithms were implemented. Another paper was read and implemented which suggested the use of an approximate approach of kernelized SVM for ranking purpose. He used Nystrom embedding and Random Fourier Feature extraction for ranking.

Preprocessing

Documents are in itself an unstructured dataset. They contain information in a non-specific model which don't have a predefined data structure. Most of the text consists of words along with punctuation marks, numbers, extra spaces etc. many words are not used in their purest forms. For this purpose, the unstructured part of the text has to undergo a lot of preprocessing and transformations in order to get the text converted to a tabular form with certain features and certain weights.

The preprocessing steps that are usually involved are

1. Conversion of all letters to lowercase

   This is done to gain uniformity throughout the text. In many text articles, there are words which consist of words which consist of words which consists of capital as well as small letters. Thus there is a need to convert all the words from a mixed case into a uniform case. Thus the case chosen in many cases is the lower case. Here also the case chosen was the lower case.

2. Removal of numbers and punctuations
   Many articles not only contain letters and words, they also contain numbers and punctuation marks in order to make the content correct and also grammatically correct. However, these are not needed while considering only the textual part. Thus they are also removed.
3. Removal of stop words
   There are certain highly common words used in texts which are very common. These words are called stop words. Some examples of this are "this","are","is", etc. There are 300 such stop words which are listed in [1].
4. Removal of text articles without content
   There may be occurrences of articles which don't have any content. These consist of noise in the dataset. These documents needed to be removed.
5. Stemming
   Stemming of each word in the document is done. In this case, the root of each word is tried to be figured out. There are several well established algorithms that do this. Out of all, Porter's stemming algorithm [2] is the one that is highly and most commonly used. It involves a process of 7 steps through which the words are trimmed to their root or "stem". However, one drawback in this case is that the words after "stemming" are not the words found in dictionary. They are modified versions of the same.

Weighing algorithms

There have been a huge research on the ways on assigning the weights to the features that were produced. Some methods with their advantages and disadvantages have been portrayed.

1. **Incidence matrix**

This is a primitive technique in which all those places in the matrix are marked one which contains a designated word. Mathematically,

$$n_{i,j} = \begin{cases} 1, & i^{th} \text{ word is present in the } j^{th} \text{document} \\ 0, & \text{otherwise} \end{cases}$$

In this case the matrix only contains values of 0 or 1, i.e. it would be a binary matrix and would not be defined by the use of number of occurrences of the word.

2. **Term frequency matrix**

This technique was developed in order to give more weightage to the repetitive words that occur within a specific document. As the phrase "term frequency" is suggestive, it counts the number of a typical word occurring in a particular document.

$$n_{i,j} = \begin{cases} \sum_{k=1}^{n} 1, & i^{th} \text{ word is present in the } j^{th} \text{document n number of times} \\ 0, & \text{otherwise} \end{cases}$$

However, some terms may appear in many documents very often which may not be of that importance. For example, let there be a corpus of mathematical texts. Then the terms like "theorem", "proof", "lemma", etc. may occur very frequently on the documents as well as in many documents. Thus these words should not be given very high weightage. In order to do this, inverse document frequency was introduced.

3. **Term Frequency – Inverse Document Frequency matrix**

This technique was developed to average out the weightage given to the words according to the importance present in them. It helps in decreasing the importance of highly frequent terms occurring in many documents and increases the importance if rare words which are rare globally. Mathematically, IDF is defined as

$$\text{idf}(t, D) = \log \frac{N}{|\{d \epsilon D : t \epsilon d\}|}$$

where N : total number of documents in the corpus $N = |D|$

$|\{d \epsilon D : t \epsilon d\}|$:number of documents where the term t appears.

$$\text{If term t doesn't appear then term frequency is inherently 0}$$

Hence the tf-idf matrix becomes a multiplication of the tf terms and the idf terms element wise. A high weight in tf–idf is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents; the weights hence tend to filter out common terms. Since the ratio inside the idf's log function is always greater than or equal to 1, the value of idf (and tf-idf) is greater than or equal to 0. As a term appears in more documents, the ratio inside the logarithm approaches 1, bringing the idf and tf-idf closer to 0.

Clustering algorithms

This problem is first treated as an unsupervised problem in order to see if some insights can be brought on from the clustering algorithms.

1. K-means centroid clustering

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

$\text{Input}:$ Finite set $S \subset \mathbb{R}^d$; integer k

$\text{Output}: T \subset \mathbb{R}^d$ with $|T| = k$

$\text{Goal}:$ Minimize cost (T) $= \sum_{x\in S} \min_{z\in T}||x - z||^2$

Here the representatives T induce a Voronoi partition of $\mathbb{R}^d$: a decomposition of $\mathbb{R}^d$ into k convex cells, each corresponding to some $z \in T$ and containing the region of space whose nearest representative is z. This partition induces an optimal clustering of the data set, $S = \cup_{z\in T} C_z$, where

$$C_x = \{ x \in S : \text{the closest representative of x is z} \}$$

Thus the $k -$means cost function can be equally written as

$$\text{cost}(T) = \sum_{z\in T}\sum_{x\in C_z} ||x - z||^2$$

In analyzing algorithms, sometimes suboptimal partitions $C_1, C_2, \ldots, C_k$ of S need to be considered. For that case, the cost is defined as

$$\text{cost}(C_{1:k}, z_{1:k}) = \sum_{j=1}^{k} \sum_{x \in C_j} ||x - z_j||^2.$$

The main objective is to minimize the cost. It can be easily be shown that the cost function is monotonically decreasing after each iteration. At each iteration for the kth cluster, the derivative of its cost is tried to minimize by the following formula.

$$\frac{\partial\, \text{cost}_k}{\partial x} = \sum_{x \in C_z} 2(x - z_k) = 0$$

$$\Rightarrow z_k = \frac{1}{|z_k|} \sum_{x \in C_z} x$$

The stopping conditions are following:-

1. The centroid doesn't get updated for two consecutive occurrences.
2. If the process has gone for some successive I iterations and not converged.

Unfortunately there is not guarantee that the centroids achieved will be global minimum centroids. In most of the cases the centroids achieved are local minima.

Algorithm

Input: Training set $S_n = \{x^{(i)}, i = 1, \dots. n\}$, where $x^{(i)} \in \mathbb{R}^d$, integer k

Output: A set of clusters $C_1, C_2, C_3, \dots .., C_k$

1. Initialize centroids$z^{(1)}, \dots .., z^{(k)}$. These centroids are selected on a random basis. They may or may not belong to the dataset taken into consideration.
2. Calculate the distance between each data point and the cluster means. Here the distance can be defined in many ways, like Euclidian, Mahalonobis, etc.
3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
4. Recalculate the new cluster center using

   $$v_i = \frac{1}{C_i} \sum_{j=1}^{C_i} x_j$$

   where C_i represents the number of data points in the ith cluster

5. Recalculate the distance between each data point and the new obtained cluster centers.

6. If any of the stopping condition is fulfilled then stop else repeat step 3.

Disadvantages of k-means clustering which hindered it from being used as a suitable means here are following:-

1. The use of exclusive assignment – If there are two highly overlapping data then k-means will not be able to resolve the two clusters.
2. Inability to handle non-linear cases. It is unable to handle noisy data and outliers.

2. Gaussian Mixture Model

In this type of clustering model, the data is assumed to be a mixture of Gaussian distributions. The Expectation-Maximization (EM) algorithm is a straightforward way to fit mixture models that starts with an initial guess and iteratively improves the fit. Each iteration consists of two steps. The first assigns data points to clusters and the second re-estimates the cluster parameters according to the assignments. In EM, we typically associate each data point to each cluster with some probability rather than using binary assignments.

Since dataset contains multiple attributes the Gaussian distribution here is

$$P(x|\mu,\Sigma) = \frac{1}{(2\pi|\Sigma|)^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}$$

Taking log on both sides, we get

$$\ln\big(P(x|\mu,\Sigma)\big) = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(|\Sigma|) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

Taking the partial derivative of the dependent variables and equating them to zero gives us

$$\mu_{ML} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

$$\Sigma_{ML} = \frac{1}{N}\sum_{n=1}^{N}(x_n-\mu_{ML})(x_n-\mu_{ML})^T$$

In the above equations the intermixing of components in not considered. Let us assume that there is probability of intermixing of each cluster. Then the above equations are bound to change.

$$p(x) = \sum_{k=1}^{K} \pi_k P(x|\mu_k, \Sigma_k) \text{ where } K = \text{number of cluster},$$

$$\pi = \text{mixing coefficient of each cluster}$$

However there needs to be a constraint on the mixing coefficients.

$$0 \leq \pi_k \leq 1 , \sum_{k=1}^{K} \pi_k = 1$$

Considering log likelihood,

$$\ln p(X|\mu, \Sigma, \pi) = \sum_{n=1}^{N} \ln p(x_n) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k P(x_n|\mu_k, \Sigma_k) \right\}$$

The parameters cannot be computed as there is no closed form solution available. They are calculated using Expectation Maximization (EM) technique.

The mixing coefficients could be thought as the prior probabilities for the components. For the given value of ‘x’ the corresponding posterior probabilities, called responsibilities, can be computed using Bayes’ rule.

$$\gamma_k(x) = p(k|x) = \frac{p(k)p(x|k)}{p(x)}$$

$= \frac{\pi_k P(x|\mu_k,\Sigma_k)}{\sum_{j=1}^{K} \pi_j P(x|\mu_j,\Sigma_j)}$ where $\pi_k = \frac{N_k}{N}$ where N_k is the effective number of points assigned to cluster k

Given a Gaussian mixture model, the goal is to maximize the likelihood function with respect to the parameters comprising the mean and covariance of the components and the mixing coefficients.

Algorithm

1. Initialize the means μ_j, covariance Σ_j and mixing coefficientsπ_j , and evaluate the initial value of the log likelihood.
2. E step. Evaluate the responsibilities using the current parameter values

$$\gamma_k(x) = \frac{\pi_k P(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j P(x|\mu_j, \Sigma_j)}$$

3. M step. Re-estimate the parameters using the current responsibilities.

$$\mu_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n) x_n}{\sum_{n=1}^{N} \gamma_j(x_n)}, \ \Sigma_j = \frac{\sum_{n=1}^{N} \gamma_j(x_n)(x_n - \mu_j)(x_n - \mu_j)^T}{\sum_{n=1}^{N} \gamma_j(x_n)}, \pi_j = \frac{1}{N}\sum_{n=1}^{N} \gamma_j(x_n)$$

4. Evaluate log likelihood

$$\ln p(X|\mu, \Sigma, \pi) = \sum_{n=1}^{N} \ln\left\{\sum_{k=1}^{K} \pi_k P(x_n|\mu_k, \Sigma_k)\right\}$$

5. If there is no convergence, return to step 2.

This model doesn't need to be iterated over a lot of values of k, the number of clusters. The number of clusters that best work for the data can easily be determined by the AIC and BIC metric.

Though AIC and BIC are both Maximum Likelihood estimate driven and penalize free parameters in an effort to combat overfitting, they do so in ways that result in significantly different behavior.

AIC = -2*ln (likelihood) + 2*k,

BIC = -2*ln (likelihood) + ln (N)*k,

Where :

k = model degrees of freedom

N = number of observations

The best model in the group compared is the one that minimizes these scores, in both cases. Clearly, AIC does not depend directly on sample size. Moreover, generally speaking, AIC presents the danger that it might over-fit, whereas BIC presents the danger that it might under-fit, simply in virtue of how they penalize free parameters (2*k in AIC; ln(N)*k in BIC). Diachronically, as data is introduced and the scores are recalculated, at relatively low N (7 and less) BIC is more tolerant of free parameters than AIC, but less tolerant at higher N (as the natural log of N overcomes 2).

Additionally, AIC is aimed at finding the best approximating model to the unknown data generating process. As such, it fails to converge in probability to the true model (assuming one is present in the group evaluated), whereas BIC does converge as N tends to infinity.

Thus that model with that k number of Gaussian mixtures is selected in which the value of AIC and BIC is the lowest simultaneously.

Classification

1. K-nearest neighbors

KNN is a non-parametric test which depends only on the value of k. In classification the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer typically small). If k =1, then the object is simply assigned to the class of that single nearest neighbor.

It is a type of instance-based learning or lazy learning where the function is only approximated locally and all computation is deferred until classification.

Here, the probability is calculated on the average of the predicted class. If all the neighbors are of the same class then the predicted probability is 1, else lower than that depending on the number of nearest neighbors belonging to the specified class.

In this case, two distance metrics were used: - Euclidian and cosine distance metric.

$$\text{consine distance} = \cos(\theta) = \frac{d_1 . d_2}{|d_1||d_2|}$$

$$\text{Euclidian} = \text{dist}(d_1, d_2) = \sqrt{\sum_{i=1}^{n} (d_{i,1} - d_{i,2})^2}$$

Cosine similarity is mainly used to compare the two documents and compare the similarity between them. Typically, $-1 \leq \cos(\theta) \leq 1$. however, in this case all the terms in the matrix are positive. So the modified range of values for $\cos(\theta)$ becomes$0 \leq \cos(\theta) \leq 1$, where 1 means that the two documents are identical and 0 means that there is no match between them.

Algorithm

1. For every observation the distance is calculated in terms of the distance metric provided.
2. Then these distances are arranged in descending order.
3. The top k observations are taken.

4. The classes of these are taken into account. The probability of the maximum occurring classes is taken as the probability of the class with the class as the predicted class. The value of k is increased by 2 each time to avoid ties amongst the predicted class since the problem dealt with here is a binary problem. For an n-ary problem, the value of k is increased by k each time starting from 2.

---

2. Random Forest

Random forest is a special type of bagging algorithm which has the ingenious property to apply random subspace method. Informally, this causes individual learners to not over-focus on features that appear highly predictive/descriptive in the training set, but fail to be as predictive for points outside that set. This causes the same algorithm to function at different levels with different sets of attributes.

Algorithm for random subspace

---

1. Let the number of training points be N and the number of features in the training data be D.
2. Choose L to be the number of individual models in the ensemble.
3. For each individual model l, choose d_l ($d_l < D$) to be the number of input variables for l. It is common to have only one value of d_l for all the individual models.
4. For each individual model l, create a training set by choosing d_l features from D with replacement and train the model.

The algorithm of a decision tree is already known.

Algorithm for random forest

Input: B- number of bootstrapping samples

m- number of variables to sample (usually $\sqrt{p}$ where p is the dimension of x)

n – number of times the bootstrapping needs to be done

Process:

1. Decision tree is made on the bootstrapped samples and selecting randomly m variables out of the p samples available.
2. The decision tree is made in such a way that the tree is allowed to grow till its full depth.

3. The above steps are repeated n times.

While predicting, the input data is fed to all the decision trees created and the output is averaged in regression problem or determined by using maximum voting in case of classification. The prediction of the classified classes can also be obtained.

Introduction to Learning to Rank (LTR)

Learning to Rank is a class of techniques that apply supervised machine learning to solve **ranking problems**. The main difference between LTR and traditional supervised ML is:

- Traditional ML solves a prediction problem (classification or regression) on a single instance at a time. E.g. if you are doing spam detection on email, you will look at all the features associated with that email and classify it as spam or not. The aim of traditional ML is to come up with a class (spam or no-spam) or a single numerical score for that instance.
- LTR solves a ranking problem on a list of items. The aim of LTR is to come up with optimal ordering of those items. As such, LTR doesn’t care much about the exact score that each item gets, but cares more about the relative ordering among all the items.

The training data for a LTR model consists of a list of items and a “ground truth” score for each of those items. The main aim is to train the model such that the query's relevance with the document remains intact. The query's relevance is predefined by human users and the LTR model uses the features of the same to predict the relevance of a query given its ID in most cases.

In this project four different LTR algorithms were studied with their variations. The four algorithms are

1. SVM with kernel approximation
2. Linear regression
3. RankNet
4. LambdaMART

The first choice would essentially be to go for a very basic implementation of ranking which would be fast and accurate. For this purpose SVM was chosen as in literature it has been shown that SVM works well with the document classification. Thus it may be assumed to work also on document ranking. Hence it was given a try.

How is Ranking different from ordinary regression or classification problem?

The main difference is in what is the output of the models in the different problems. In ordinal regression or classification, the task is to either predict a number or predict a label for a given sample, hence the output of a prediction is a label (as is the case for example in multiclass classification). On the other hand, in the problem of learning to rank, the output is an order of a sequence of samples. That is, the output of a ranking model can be seen as a permutation that makes the samples to have labels as ordered as possible. Hence, unlike the ordinal classification model, the ranking algorithm is not able to predict a class label. Because of this, the input of a ranking model does not need to specify class labels, but only a partial order between the samples. In this sense, ranking is an easier problem than ordinal regression: from the numerical labels you can construct an order, but not necessarily the other way round.

This can be better explained with an example. Suppose that the following pairs of (sample, label): {(x_1, 1), (x_2, 2), (x_3, 2)}. Given this input, a ranking model will predict an order of this sequence of samples. For example, for a ranking algorithms, the permutations (1,2,3)→(1,2,3) and (1,2,3)→(1,3,2) are predictions with perfect score since the labels of both sequences {(x_1,1),(x_2,2),(x_3,2)} and {(x_1,1),(x_3,2),(x_2,2)} are ordered. On the other hand, an ordinal regression would predict a label for each of the samples, and in this case the prediction (1, 2, 2) would give a perfect score, but not (1, 2, 3) or (1, 3, 2).

Different types of approaches:-

Pointwise approaches

Pointwise approaches look at a single document at a time in the loss function. They essentially take a single document and train a classifier / regressor on it to predict how relevant it is for the current query. The final ranking is achieved by simply sorting the result list by these document scores. For pointwise approaches, the score for each document is independent of the other documents that are in the result list for the query.

All the standard regression and classification algorithms can be directly used for pointwise learning to rank.

Pairwise approaches

Pairwise approaches look at a pair of documents at a time in the loss function. Given a pair of documents, they try and come up with the optimal ordering for that pair and compare it to the ground truth. The goal for the ranker is to minimize the number of inversions in ranking i.e. cases where the pair of results are in the wrong order relative to the ground truth.

Pairwise approaches work better in practice than pointwise approaches because predicting relative order is closer to the nature of ranking than predicting class label or relevance score. Some of the most popular Learning to Rank algorithms like RankNet, LambdaRank and LambdaMART are pairwise approaches.

Listwise approaches

Listwise approaches directly look at the entire list of documents and try to come up with the optimal ordering for it. There are 2 main sub-techniques for doing listwise Learning to Rank:

1. Direct optimization of IR measures such as NDCG. E.g. SoftRank, AdaRank
2. Minimize a loss function that is defined based on understanding the unique properties of the kind of ranking you are trying to achieve. E.g. ListNet, ListMLE

In short, in pointwise approaches, the input space consists of two items: one query and one document. The question asked is, is the document relevant or not. The output is either a yes or no with or without a probability supporting that decision. In pairwise approach, the input space has a query and 2 documents. The question now changes to gain preference of the given document pair with respect to the query. The output is a preference order may or may not be followed by the probability of it. In listwise approach, a set of documents is inserted and a query is given to ask for the ranking of the documents. Here, usually an information quality measure such as MAP, NDCG, etc. are considered for optimization of the algorithm.

The pairwise approach is derived from the pointwise ranking. In this two vectors are simultaneously computed and a score of +1 is given if the ranking is in order and -1 if not in order. Then using permutations and combinations of them, the "perfect" ranking is said to be achieved. However, between -1 and +1 there is a huge space. Here, IR SVM comes in handy. It is derived from pairwise classification. It converts the {-1, +1} space to a different stringent loss function. It sets different losses for document pairs across different grades and from different queries. To emphasize the importance of correct ranking on the top, the loss function heavily penalizes errors related to the top. To increase the influence of queries with less documents, the loss function heavily penalizes errors from the queries.

Pointwise ranking is rarely used as it has a large variety of problems.

1. Some queries have large number of documents making the hypothesis biased towards it.
2. They do not optimize for any IR measure like NDCG etc.
3. Position of documents in the ranked list is not modelled by the loss function.
4. Approaches might emphasize on unimportant documents if their number is very much more than relevant documents.

Pairwise approach also has many cons in them.

1. Distribution of document pairs is more skewed than distribution of document w.r.t queries
2. Some queries have more query pairs than others
3. Still does not optimize for IR measures
4. Rank ignorant — (d1 > d2) does not encode which ranks are being compared. Rank 1 vs Rank 2 or Rank 99 vs Rank 1000

In listwise approaches, there are direct optimization of the information retrieval measures. They try to optimize information retrieval measures or at least perform something highly correlated to the measures. It minimizes a loss function defined on permutations, which is defined and designed by considering the properties of ranking for IR.

List wise approach is a much more straightforward way. While ranking for query q_i, the ranking model would assign a score to each feature vector x_{ij}. The feature vectors are then sorted based on their scores and a permutation is obtained. Then the permutation which has the highest score amongst all the permutations is the ranking of the documents based on that query. Mathematically, let the query be q_i, the ranking model f (x_{ij}), weights associated with each feature vector x_{ij} be w. Then,

$$f(x_{ij}) = < w, x_{ij} >$$

Let the scoring function S (x_i, πi) be defined as

$$S(x_i, \pi_i) = < w, \sigma(x_i, \pi_i) >$$

Where $\sigma(x_i, \pi_i) = \frac{2}{n_i(n_i-1)} \sum_{k,l;k<l}[z_{kl}(x_{ik} - x_{il})]$ where $z_{kl} = +1$ if $\pi_i(k) < \pi_i(l)$ and $z_{kl} = -1$otherwise.

For any query q_i, S (x_i, πi) is calculated for all the possible permutations of π_i. The permutation with the maximum score is selected.

To give a small example,

Objects: A, B, C

$f_A = \langle w, x_A \rangle$ $f_B = \langle w, x_B \rangle$ $f_C = \langle w, x_C \rangle$
Suppose $f_A > f_B > f_C$

Permutation 1: ABC

Permutation 2: ACB

Permutation 3: CAB

$S_{ABC} = \frac{1}{6} \langle w, ((x_A - x_B) + (x_B - x_C) + (x_A - x_C)) \rangle$

$S_{ACB} = \frac{1}{6} \langle w, ((x_A - x_C) + (x_C - x_B) + (x_A - x_B)) \rangle$

$S_{CAB} = \frac{1}{6} \langle w, ((x_C - x_A) + (x_A - x_B) + (x_C - x_B)) \rangle$

$S_{ABC} > S_{ACB} > S_{CAB}$

However, much of it will depend in the vectors and the weights.
In learning, we would ideally create a ranking model that can maximize the accuracy in terms of a list wise evaluation measure on training data, or equivalently, minimizes the loss function defined below,
$L(f) = \sum_{i=1}^{m} (E(\pi_i^*, y_i) - E(\pi_i, y_i))$

Where πi the permutation on feature is vector xi by ranking model f and yi is the corresponding list of grades. $E(\pi_i, y_i)$ denotes the evaluation result of πi in terms of an evaluation measure (e.g., NDCG). Usually $E(\pi_i^*, y_i) = 1$.

Information Retrieval measures

The evaluation on the performance of a ranking model is carried out by comparison between the ranking lists output by the model and the ranking lists given as the ground truth. Several evaluation measures are widely used in IR and other fields. These include NDCG (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain), MAP (Mean Average Precision), and Kendall's Tau.
Given query q_i and associated documents D_i, suppose that $\pi_i$is the ranking list (permutation) on D_i and y_i is the set of labels (grades) of D_i. DCG measures the goodness of the ranking list with the labels.
Specifically, DCG at position k is defined as:

$$DCG(k) = \sum_{j:\pi_i(j) \leq k} G(j) D(\pi_i(j))$$

where $G(.)$ is a gain function and $D_i(.)$ is a position discount function, and $\pi_i(j)$ is the position of $d_{i,j}$ in π_i. The summation is taken over the top K positions in the ranking list π_i. DCG represents the cumulative gain of accessing the information from position one to position K with discounts on the positions. NDCG is normalized DCG and NDCG at position K is defined as:

$$NDCG = G_{max,i}^{-1}(k) \sum_{j:\pi_i(j)\leq k} G(j)D\big(\pi_i(j)\big)$$

where $G_{max,i}$ is the normalizing factor and is chosen such that a perfect ranking π_i^*'s NDCG score at position K is 1. In a perfect ranking, the documents with higher grades are always ranked higher. Note that there can be multiple perfect rankings for a query and associated documents.

The gain function is normally defined as an exponential function of grade. That is to say, the satisfaction of accessing information exponentially increases when the grade of relevance of information increases.

$G(j) = 2^{y_{i,j}} - 1$ where $y_{i,j}$ is the label (grade) of document $d_{i,j}$in ranking list π_i. The discount function is normally defined as a logarithmic function of position. That is to say, the satisfaction of accessing information logarithmically decreases when the position of access increases.

$$D\big(\pi_i(j)\big) = \frac{1}{\log_2\big(1+\pi_i(j)\big)}$$

where $\pi_i(j)$ is the position of document $d_{i,j}$in ranking list π_i.

Hence, DCG and NDCG at position K becomes

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2\big(1+\pi_i(j)\big)}$$

$$NDCG(k) = G_{max,i}^{-1}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2\big(1+\pi_i(j)\big)}$$

In evaluation, DCG and NDCG values are further averaged over queries.

NDCG (DCG) has the effect of giving high scores to the ranking lists in which relevant documents are ranked high. For perfect rankings, the NDCG value at each position is always one, while for imperfect rankings, the NDCG values are usually less than one. Let's illustrate it with an example.

Perfect ranking	Imperfect ranking	Formula	Explanation
(3,3,2,2,1,1,1)	(2,3,2,3,1,1,1)		Grades : 3,2,1

(7,7,3,3,1,1,1)	(3,7,3,7,1,1,1)	$G(j) = 2^{y_{i,j}} - 1$	Gains due to their grades
(1,0.63,0.5,….)	(1,0.63,0.5,….)	$D(\pi_i(j)) = \frac{1}{\log_2(1+\pi_i(j))}$	Discounts
(7, 11.41, 12.91,…)	(3, 7.41, 8.91,…)	DCG	Discounted Cumulative Gain
(1/7, 1/11.41, 1/12.91, ….)	(1/7, 1/11.41, 1/12.91,...)		Normalizers
(1,1,1….)	(0.43, 0.65, 0.69,…)	NDCG	Normalized Discounted Cumulative Gain

Ranking Models

SVM

SVM or support vector machines typically are a classifier. They help in classification based on marginal separation between the classes. They tend to maximize the margin or separation between the classes. The main reason for maximum margin is that even during some perturbations to the data, the classification should be accurate. Margin are of two types: hard margin and soft margin. Hard margin is defined as the hyperplane in multidimensional data which would exactly classify or categorize the data given to it. This is a very crude way of classification since many applications don't support such crude hyperplanes. When a hard margin hyperplane is being used, the hyperplane tends to overfit the data supplied to it. As a result it is often seen that in case of hard margin classifiers, the training accuracy is very good (nearly 100%), but the accuracy is very poor in case of testing or validation data since the data may not fall into the partitions made by the hard margin. In order to gain higher accuracy in the testing or validation data with very less compromise on the training data, soft margins are introduced.

Difference between hard and soft margin classifiers:

The main difference lies in the tolerance of inaccuracy for the given classifier. Hard margin classifiers are intolerant towards any inaccuracy. For the same, the hyperplane that is generated by the model highly overfits the data. For the hyperplane to be more tolerant to the inaccuracy and to gain a bigger margin of separation barring those mistakes, a slack variable is introduced which is the measure of tolerance in the said margin. The optimized function would have the least amount contributed by the inclusion of the slack variable.

The above is a description on how SVM behaves with hard and soft margins. Though SVM is essentially a classifier, but it can be used for regression and ranking. In regression the SVM estimates a function which would map the input features to an output real number under some precision. In case of SVM regression (SVR), the hard margin optimizer function optimizes the weight given to features such that the error in calculating the regression value comes within ϵ precision. However, in order to maximize the margin like in the case of classification, the weight vector needs to be minimized. Here, in hard margin SVR the error constraint is non-existent as the optimizer function doesn't allow for it. However in the case of soft margin there is relaxation on the error margin in the optimizer. The slack variables in this case are there to deal with the infeasible constraints of the optimization problem by imposing the penalty to the excess deviations from the output.

In ranking, the training set is an ordering of data. Here the initial step is converting the query-document pairs to a uniform base which is invariant in the terms of words or phrases in the document or query but is built on the interaction between them. After this mapping or transformation is done, the output labels are either 2, 1, 0 meaning highly-relevant, relevant or not relevant. So it can be easily seen that the ranking problem was first transformed into a regression problem where the score of the input or features were found and then on the basis of those classification was done.

For SVM ranking, there is an additional step. The good thing about SVM is that it, unlike linear regression, doesn't do pointwise ranking. It goes for pairwise approach of ranking. In this also there are two ways. One is to take documents pertaining to the same query having different rankings and then use classification tool to classify which is greater in ranking. However, if the number of queries are huge then computationally as well as storage-wise this method would be very expensive and might fail drastically. Another approach as mentioned earlier would be to transform the query-document pair in such a feature vector that in the feature vector the output would only be classes "highly relevant", "relevant", "not relevant", etc. the output further can be given numeric form like 2,1,0. This approach would be universal as the feature extraction is done with respect to the query and the document it is queried upon. It can be very well seen that the in the feature vector the query-document pair doesn't exist for all queries with all the documents, i.e. the pairing is incomplete. To give an example, query 1 might be used to rank documents 1,2,5 whereas query 2 might rank documents 1,3,4. Thus the pairing of query 1 with document 3 and 4 doesn't exist.

After the feature extraction has been done, it is then paired in such a way that the first query-document pair is ranked higher than the second. Then this pair is given to the SVM algorithm to either learn or classify whether the first query-document pair is ranked higher than the other. Learning to determine whether the first query-document is larger or better ranked than the second is done through classification technique. In the training part where it is known that the first pair has a higher score or is ranked better than the second one, weights are computed such that the ordering is maintained. However, many such weights would exist. In this case, the weight which gives the maximum margin between the classified works best. Now the question arises that in ranking how does one determine which is the best weight vector.

Determination of the best weight vector

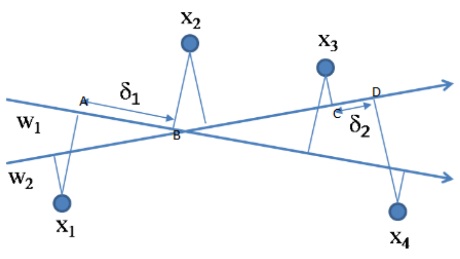

Figure 1: Determination of the weight vector

Let us consider that the input vectors {x_1, x_2, x_3, x_4} need to be ranked. Let us assume that there are two weight vectors w_1 and w_2 in 2D space. It can be easily seen that both of them rank them in the same order i.e. $x_1 > x_2 > x_3 > x_4$. But the question that needs to be addressed is which of these weights are to be chosen in order to maximize the margin. Here the concept of margin is expressed in terms of support vectors. Support vectors are defined as those vectors that define the boundary lines of the hyperplane on either side of the hyperplane.

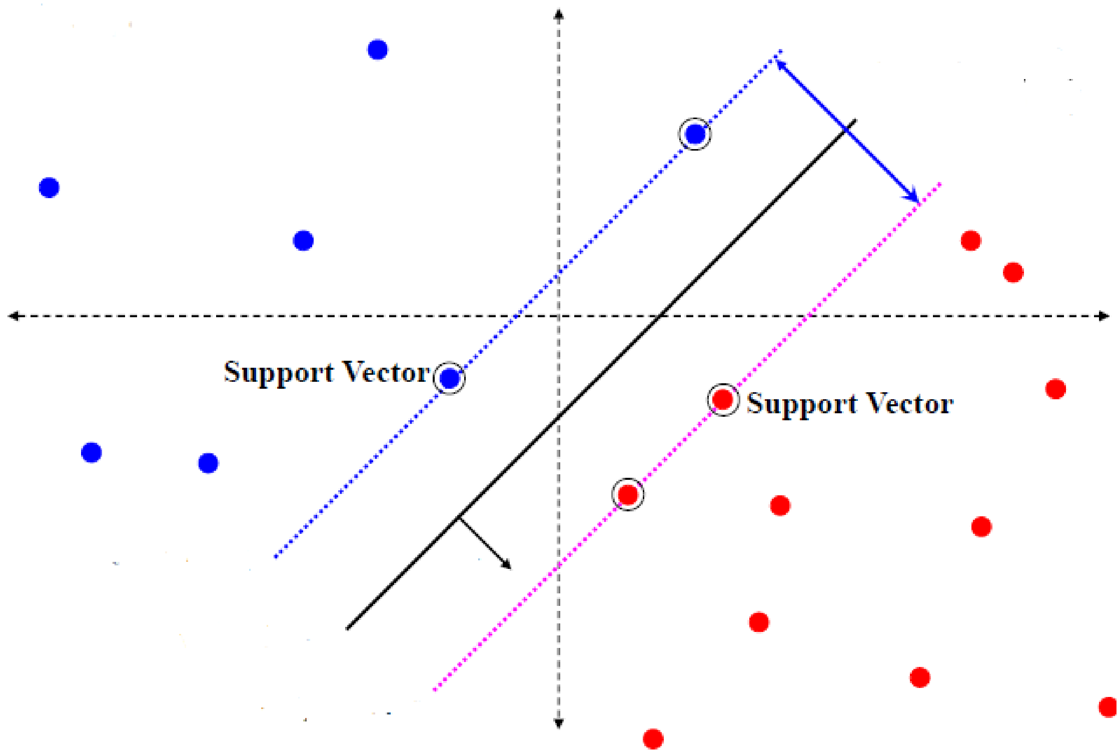

Figure 2: Showing the support vectors

The above picture is the representation of the support vectors pictorially. Here the lines on either side of the margin represent the boundaries and points which define these are the support vectors. Coming back to the ranking problem, the ranking difference between two vectors is defined as the geometric distance between them when projected on the weight vectors. The main motivation is to increase the ranking difference between the closest two vectors. The reason is because these closest vectors would form the support vectors for the hyperplane. Now it is known that to maximize the margin to reduce maximum error due to perturbation of the input the margin width has to be maximum. Since the margin width is controlled by the support vectors, they need to be place the farthest. Thus the distance between the closest ranked documents need to be the farthest. Thus in fig 1 it can be seen that if w_1 is considered then the distance between the closest vectors is δ_1 and that by w_2 is δ_2. But it can be seen that $\delta_1 > \delta_2$. Thus the weight vector to be considered is w_1.

Mathematical explaination

Let "A is preferred to B" be specified as "$A \succ B$". A training set for ranking SVM is denoted as $R = \{(x_1, y_i), (x_m, y_m)\}$ where y_i is the ranking of x_i, that is, $y_i < y_j$ if $x_i \succ x_j$. SVM tries to estimate a ranking function which would estimate the rank of x. A ranking function outputs a score for each data object, from which a global ordering of data is constructed. That is, the target function $F(x_i)$ outputs a score such that $F(x_i) > F(x_j)$ for any $x_i \succ x_j$.

Let R^* be the optimal ranking of data in which the data is ordered perfectly according to user's preference. A ranking function F is typically evaluated by how closely it's ordering R approximates R^*. Using the techniques of SVM, a global ranking function F can be learned from an ordering R.

The goal is to learn F which is concordant with the ordering R and also generalize well beyond R. That is to find the weight vector w such that $w \cdot x_i > w \cdot x_j$ for most data pairs $\{(x_i, xj): y_i < y_j \in R\}$.

The solution can be approximated using SVM techniques by introducing (non-negative) slack variables $\xi_{i,j}$ and minimizing the upper bound $\sum \xi_{ij}$ as follows:

$$\text{minimize: } L_1(w, \xi_{ij}) = \frac{1}{2} w.w + C \sum \xi_{ij}$$

$$\text{subject to: } \forall\{(x_i, x_j): y_i < y_j \in \mathbb{R}\} : w.x_i \geq w.x_j + 1 - \xi_{ij}$$

$$\forall(i, j): \xi_{ij} \geq 0$$

By rearranging the terms, it is found that

$$w(x_i - x_j) \geq 1 - \xi_{ij}$$

Thus the optimization problem is equivalent to classifying SVM on pairwise difference vectors. To understand it more vividly, let there be some points which satisfy the equation $w(x_i - x_j) = 1 - \xi_{ij}$. These will be the support vectors. Unbounded support vectors are ones which are on the margin i.e. the slack variable ξ_{ij} is 0 and bounded vectors are those whose slack vector satisfies $1 > \xi_{ij} > 0$. Some of the vectors might be mis-ranked with $\xi_{ij} > 1$. When this minimization problem is transformed using Lagrange multipliers kernel functions are introduced in ranking.

$$\text{minimize: } L_2(\alpha) = \sum_{ij} \alpha_{ij} - \sum_{ij} \sum_{uv} \alpha_{ij} \alpha_{uv} K(x_i - x_j, x_u - x_v)$$

$$\text{subject to:} C \geq \alpha \geq 0$$

Here $K(.)$ is the kernel function and α_{ij} is the coefficient of the pairwise difference vector $(x_i - x_j)$. Once α is calculated w can be calculated by

$$w = \sum_{ij} \alpha_{ij} (x_i - x_j)$$

And the ranking function F on a new vector z is calculated by

$$F(z) = w.z = \sum_{ij} \alpha_{ij}(x_i - x_j).z = \sum_{ij} \alpha_{ij} K(x_i - x_j, z)$$

The support vectors are the data pairs $(x_i^s - x_j^s)$ such that $w * (x_i^s - x_j^s) = 1$ which is the smallest possible value for all data pairs $\forall (x_i, x_j) \in \mathbb{R}$. Thus, its ranking difference according to $F_w \left(= \frac{w*(x_i^s - x_j^s)}{||w||} \right)$ is also the smallest among them.

The soft margin SVM allows bounded support vectors whose $\xi_{ij} > 0$ as well as unbounded support vectors whose $\xi_{ij} = 0$, in order to deal with noise and allow small error for the $\mathbb{R}$ that is not completely linearly rankable. Thus, maximizing the margin generates the effect of maximizing the differences of close data pairs in ranking. It can be observed that ranking SVM improves the generalization performance by maximizing the minimal ranking difference. SVM computes the weight vector w that maximizes the differences of close data pairs in ranking.

The reason for using Kernels

The above cases were used when the data was linearly separable or could be done by introducing a slack variable which would not lead to too much of error. But what if the data is non-linearly separable? Here, there comes a need to figure out a decision boundary which would behave like a non-linear decision boundary. For this purpose kernels are used. This is often done using a mapping function which maps the data to a higher dimension where the separation would be done using a linear separation. For example a polynomial feature map would map the features x to $k(x, y) = (x^T y + c)^d$, where k is the kernel function and the features x, y are given as input to the function with c as bias and d as the polynomial degree of the polynomial.
In many cases, only an inner product k(x, y) is specified directly and under mild condition (if k is a Mercer-kernel), there exists a space H for which k is the scalar product. It is possible to construct a mapping from the original $\mathbb{R}_n \Rightarrow$ H. One characteristics is that the kernalised SVMs are independent of the number of features but heavily dependent on the number of data points given the kernel. It can easily be shown that the storage requirement of the kernel values of each pair of data points would be of the order of the square of the number of data points.
Let the above fact be demonstrated with an example. A very well-known kernel used rapidly is the RBF or Radial Basis Function which is of the form $k(x, y) = e^{-(||x-y||^2)}$. For every pair of (x, y) the kernel would output an output which would be same as the value for the pair (y,x). In other words the value stored in the left lower triangular part of the matrix is the same as the

top right triangle of the matrix with the diagonal value as 1. Thus the even if half a matrix is stored. Computationally also it is the same. It is of the order of n^2.

Kernel approximation

Using kernels might prove to be computationally and storage wise heavy in storing the kernel matrix even if parallel computations are done. To handle this problem, stochastic gradient descent (SGD) has been used to learn linear classifiers. These are fast algorithms - a single iteration is linear in the dimensionality (or rather in the number of non-zero features per sample, which is very nice for sparse data with very few non-zero features per sample) and convergence is sublinear in the number of samples. But classifiers learned in this way are usually linear in the data.

On one hand kernelized SVMs work well on complicated data but is heavy in terms of space and computational complexity and on the other hand SGD is an efficient way of computation but only for linear classifiers. One way to achieve better, faster and more accurate results is to combine the two concepts and produce results. This is called Kernel approximation. In kernel approximation, the mapping to infinite-dimensional space is reduced to a finite subspace. The finite subspace is spanned by the images of the training data. This is given by representer theorem.

If all the training examples are taken to map into an N-dimensional space and it would have the same problem as the kernelized SVM. The kernelized approximation involves using a subset of all the training dataset. This will only yield an approximate embedding but if the number of samples used is kept same, the resulting embedding is independent of dataset size. This algorithm is called Nyström method (or Nyström embedding).

Theorem for Nystrom embedding

K is a kernel matrix for a kernel method. Consider the first $q < n$ points in the training set. Then there exists a matrix K of rank q:

$\overline{K} = \widehat{K_{n,q}}\widehat{K_q}^{-1}\widehat{K_{n,q}^{T}}$, where

$$\left(\widehat{K_q}\right)_{i,j} = K(x_i, x_j), \quad i, j = 1, \dots\dots, q$$

$\widehat{K_q}$ is an invertible matrix and

$$\left(\widehat{K_{n,q}}\right)_{i,j} = K(x_i, x_j), \quad i,j=1,\dots\dots\dots,q$$

Thus the matrix of rank N is reduced to rank of q.

Another way to understand it is illustrated below.

The key assumption in Nyström is that the kernel function is of rank m. That means that any kernel matrix is going to have rank at most m, and in particular

$$K = \begin{bmatrix} k(x_1, x_1) & \dots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \dots & k(x_n, x_n) \end{bmatrix},$$

is rank m. Therefore there are m nonzero eigenvalues, and we can write the Eigen decomposition of K as $K = U\Lambda U^T$ with eigenvectors stored in U, of shape $n \times m$, and eigenvalues arranged in Λ, a $m \times m$ diagonal matrix. M elements were picked, usually uniformly at random but possibly according to other schemes – all that matters in this simplified version is that K_{11} be of full rank. Once it's done, the points are relabeled so that the kernel matrix in blocks is:

$$K = \begin{bmatrix} K_{11} & K_{21}^T \\ K_{21} & K_{22} \end{bmatrix},$$

where evaluation of each entry in K_{11} (which is $n \times m$) and K_{21} $\big((n - m) \times m\big)$ is done.

Now, the Eigen decomposition can be split according to this block structure too:

$$K = U\Lambda U^T$$

$$= \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \Lambda \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^T$$

$$= \begin{bmatrix} U_1\Lambda U_1^T & U_1\Lambda U_2^T \\ U_2\Lambda U_1^T & U_2\Lambda U_2^T \end{bmatrix},$$

where U_1 is $m \times m$ and U_2 is $(n - m) \times m$. But now it can be seen that $K_{11} = U_1\Lambda U_1^T$. So U_1 and Λ can be found by Eigen decomposing the known matrix K_{11}.

It is also known that $K_{12} = U_2\Lambda U_1^T$. Here, everything in this equation is known except U_2, so it can be solved for what eigenvalues that implies: right-multiply both sides by $(\Lambda U_1^T)^{-1} = U_1\Lambda^{-1}$ to get $U_2 = K_{21}U_1\Lambda^{-1}$.

From the above values K_{22} can be calculated.

$$\begin{aligned} K_{22} &= U_2\Lambda U_2^T \\ &= (K_{21}U_1\Lambda^{-1})\Lambda(K_{21}U_1\Lambda^{-1})^T \\ &= K_{21}U_1(\Lambda^{-1}\Lambda)\Lambda^{-1}U_1^T K_{21}^T \\ &= K_{21}U_1\Lambda^{-1}U_1^T K_{21}^T \\ &= K_{21}K_{11}^{-1}K_{21}^T \end{aligned}$$

$$= (K_{21}K_{11}^{-\frac{1}{2}})(K_{21}K_{11}^{-\frac{1}{2}})^{T}$$

Here, the feature matrix $(K_{21}K_{11}^{-\frac{1}{2}})$, which is shape $(n - m) \times m$, corresponds to these imputed kernel values. If $K_{11}^{-\frac{1}{2}}$ is used for the m points, the set of m-dimensional features would be:

$$\Phi = \begin{bmatrix} K_{11}^{\frac{1}{2}} \\ K_{21}K_{11}^{-\frac{1}{2}} \end{bmatrix}$$

The only problem that may be pointed out is, is the kernel correct in representation of the data? It could be found out if the product of the kernel and its transpose give the original value of the data.

$$\Phi\Phi^{T} = \begin{bmatrix} K_{11}^{\frac{1}{2}} \\ K_{21}K_{11}^{-\frac{1}{2}} \end{bmatrix}\begin{bmatrix} K_{11}^{\frac{1}{2}} \\ K_{21}K_{11}^{-\frac{1}{2}} \end{bmatrix}^{T}$$

$$= \begin{bmatrix} K_{11}^{\frac{1}{2}}K_{11}^{\frac{1}{2}} & K_{11}^{\frac{1}{2}}K_{11}^{-\frac{1}{2}}K_{21}^{T} \\ K_{21}K_{11}^{\frac{1}{2}}K_{11}^{-\frac{1}{2}} & K_{21}K_{11}^{-\frac{1}{2}}K_{11}^{-\frac{1}{2}}K_{21}^{T} \end{bmatrix}$$

$$= \begin{bmatrix} K_{11} & K_{21}^{T} \\ K_{21} & K_{21}K_{11}^{-1}K_{21}^{T} \end{bmatrix}$$

$$= \begin{bmatrix} K_{11} & K_{21}^{T} \\ K_{21} & K_{22} \end{bmatrix}$$

$$= K$$

So, all that needs to be done is train our regular learning model with the m-dimensional features Φ. This will be exactly the same as the kernelized version of the learning problem with K.

Random Fourier Features

Another method to do kernel approximation is random Fourier features as explained by Ali Rahimi and Ben Recht in Random features for large-scale kernel machines. Stationary or shift-invariant kernels rely only on a vector difference of the inputs:

$$k_s(z) = k(x, y) \text{ where } z = x - y$$

According to Bochner's theorem there always exists a transformation of positive definite function of form:

$$k_s(z) = k_s(x - y) = \int p(\omega)\, e^{j\omega^T(x-y)}\, d\omega$$

where $p(\omega)$ is a positive finite measure, which means that $k_s(z)$ is a Fourier transform of function $p(\omega)$. In turn, finite function $p(\omega)$, when scaled, is a proper probability distribution. For the radial basis function kernel $k_s(z) = e^{-\gamma||z||^2}$, where $\gamma = \frac{1}{2\sigma^2}$, that has been taken into consideration, such a distribution is defined the following way:

$$\frac{p(\omega)}{k_s(0)} = \frac{1}{2\sqrt{\pi\gamma}} e^{-\frac{||w||}{2}} = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{2\gamma}} e^{-\frac{||w||}{2}} = \frac{1}{\sqrt{2\gamma}} f(\omega|0,1)$$

where $f(\omega|\mu, \sigma^2)$ is a normal distribution.

By transformation, given $\zeta_\omega(x) = e^{j\omega' x}$, an unbiased estimate of $k_s(z)$ can be constructed:
$k_s(z) = k_s(x - y) = E[\zeta_\omega(x)\zeta_\omega(y)^*]$, where is drawn from $p(\omega)$.

Kernel value $k_s(x - y)$ can be approximated with any target variance by taking average result of D such mappings. Since kernel function $k_s(z)$ is real-valued $e^{j\omega^T x}$ can be replaced with $\cos\omega(x - y)$under the integral sign.

$\zeta_\omega(x)$ can be estimated to be $\sqrt{2}\cos(\omega^T x + b)$ where b is drawn uniformly from$[0,2\pi]$. Now, $k_s(z)$ can be approximated by the mean over m Fourier components.

$$\zeta(x) = \sqrt{\frac{2}{m}}[\cos(\omega_1^T x + b_1) + \cdots + \cos(\omega_m^T x + b_m)]^T,$$

where $\omega_i \in \mathbb{R}^d$ is sampled from the distribution $p(\omega)$ and $b_i \in \mathbb{R}$ is uniformly sampled from $[0,2\pi]$.

RankNet

To understand RankNet, the concept of neural network has to be understood to great depth. Neural networks is an interconnection of neurons connected in a specific order in order to give a certain output. Unlike SVM which specializes in classification, neural network specializes in regression problems. Another huge problem with SVM is that the model's error metric which is to be minimised is a non-differentiable function. In order to overcome this problem, a model whose output is differentiable is to be used. According to Tom Mitchel in 1997, a two layer neural networks can approximate any bounded continuous function, the model often used is a neural network. Neural network can be visualized as a sequence of neurons connected in such a fashion so as to mimic the brain functionality. Inside our brain, the neurons are interconnected and each neuron sends only a high or a low signal to the other neuron. Similarly, the neurons in the network also are modeled in the same manner. They accept inputs from other neurons and then with the help of some activation function and weight vector, it generates an output which may or may not be binary. This output is then either sent to other neurons or is treated as the output. The basic neural network consists of an input layer and an output layer. The output layer may be a single neuron or may be many neurons depending on the number of classes present in the system. The basic functioning of a neural network involves two major processes: forward propagation which helps us get the output of the network for any desired input, and back-propagation which is used to re-calibrate the weights in the network in order to get the desired results in a faster and more efficient way.

The activation function used to gain the non-linear nature of the network used mostly is the sigmoid function. However, many other activation functions can be used like tan-hyperbolic, RELU, etc. all of which are differentiable. Thus the model which the neural network makes is also differentiable. Thus its usage in the field of classification and regression can be easily known. However, in the field of ranking there are certain changes that are made in order to implement regression of a different kind.

It uses the pairwise approach to ranking the documents. In this type of ranking, given that item A appears higher than item B in the output list, the user concludes that the system ranks A higher than, or equal to, B; no mapping to particular rank values, and no rank boundaries, are needed; to cast this as an ordinal regression problem is to solve an unnecessarily hard problem, and RankNet avoids this additional step. This model is not specific only to neural networks. This model can be easily applied to any other algorithm provided the output cost function is a differentiable one. For the above mentioned conditions, neural network is being used.
Visualizing this ranking algorithm is not that difficult. Assume that there are two documents A and B being queried for the same query Q such that the rank of A is higher than that of B. What RankNet does is through forward propagation it calculates the output of both the document-query pairs, stores all the parameters associated with the network and then computes the cost function which is made up of the difference of the outputs. The adjustments to the weights are done using the back-propagation algorithm. Neural network is being used as a supervised algorithm. Thus the output is to be known. Since it mostly gives an output in [0,1], it is often considered that this is the probability of a certain class. However, in the case of ranking, the output is often the difference in the rank amongst the documents subject to the query. In most cases of document ranking, the ranks are either 0 or 1. This makes it easy for ranking. However, if the situation arises where the ranks are more than binary, then the same technique is used.

The output is just scaled in order to fit the difference. Finally, cost functions are used that are functions of the difference of the system's outputs for each member of a pair of examples, which encapsulates the observation that for any given pair, an arbitrary offset can be added to the outputs without changing the final ranking; again, the goal is to avoid unnecessary learning. This way to define the output not in terms of classes but in terms of probability such that the models' optimizer function can be differentiable and the cost can be computed in terms of the previously known probability and the computed probability, is often known as the cross-entropy cost. It can be easily seen that the document pairing for each query need not be complete. Even on different query this algorithm would work.

One might suggest that instead of using the same network twice for a given query-document pair and then subtracting the two networks, one can just feed in the difference and be done with it. Intuitively this would work fine as this is what is being tries to achieve through various feature extraction techniques. However, it can be shown mathematically that the first one would provide better results as the constraint on the feature and its corresponding weights are different than the next case. The first one is differentiable at each step and it comes out as a difference between the two query-document pairs' weight and activation function outputs.

Mathematical expression

RankNet can thus be seen as a learning algorithm in which a pair of documents is given to the model and a probability is set as an output which states the probability of the first document being rated higher than the other. It has been shown that neural networks have very high accuracy and performance in terms of regression problems.

The learning algorithm tries to fit a model for a set of samples $[A, B]$ in $\mathbb{R}^d$, together with the target probabilities P_{AB} which indicates the probability of document A being ranked higher than document B. Models are created such that $f: \mathbb{R}^d \longmapsto \mathbb{R}$ such that the rank order of a set of test samples is specified by the real values that f takes, specifically $f(x_1) > f(x_2)$ is equivalent to the order that x_1 is ranked higher than x_2 i.e. $x_1 \rhd x_2$.

In neural network there are two main operation: - forward propagation and backward propagation.

Before diving into the details of the algorithms, some notations need to be defined. w_{jk}^{l} would be used to denote the weight for the connection from the kth neuron in the $(l-1)$th layer to the jth neuron in the lth layer. b_{lj} would be used for the bias of the jth neuron in the lth layer. And we use a_{lj} for the activation of the jth neuron in the lth layer.

$$\mathrm{a}_{\mathrm{j}}^{\mathrm{l}} = \sigma\left(\sum_k w_{jk}^{l} a_k^{l-1} + b_j^l\right)$$

Let us assume a neural network with one input layer, 2 hidden layers and an output layer. Given training examples (x, y). For the ith training sample denote the outputs of net by o_i, the targets by t_i, let the transfer function of each node in the jth layer of nodes be g^j, and let the cost function be $\sum_{i=1}^{q} f(\, o_i,\ t_i)$. If α_k are the parameters of the model, then a gradient descent step amounts to $\delta\alpha_k = -\eta_k \frac{\partial f}{\partial \alpha_k}$, where η_k are the positive learning rates. The network embodies a function denoted by

$$o_i = g^3\left(\sum_j w_{ij}^{32} g^2\left(\sum_k w_{jk}^{21} x_k + b_j^2\ \right) + b_i^3\right) \equiv g_i^3$$

where w is the weights and b is the bias units, the upper indices represent the node layer and the lower index represents the node within the corresponding layer. Taking the derivatives of f with respect to the parameters gives

$$\frac{\partial f}{\partial b_i^3} = \frac{\partial f}{\partial o_i} g_i'^3 \equiv \Delta_i^3$$

$$\frac{\partial f}{\partial w_{in}^{32}} = \Delta_i^3 \mathrm{g}_n^2$$

$$\frac{\partial f}{\partial b_m^2} = \mathrm{g}_m'^2\left(\sum_i \Delta_i^3 \mathrm{w}_{im}^{32}\right) \equiv \Delta_m^2$$

$$\frac{\partial f}{\partial w_{mn}^{21}} = x_n \Delta_m^2$$

where x_n is the nth component of the input.

If the net had a single output neuron the above network could be brought down to the ranking problem. The cost function would become the function of the difference of the outputs of two consecutive samples $f(o_2 - o_1)$. Here it is assumes that the first is ranked higher than the second one. Now, feed forward step is computed on the first sample while storing all the parameters. Similarly the second parameter is also done. the gradient of the cost then become $\frac{\partial f}{\partial \alpha} = \left(\frac{\partial o_2}{\partial \alpha} - \frac{\partial o_1}{\partial \alpha}\right) f'$. Thus if $f' \equiv f'(o_2 - o_1)$, then

$$\frac{\partial f}{\partial b^3} = f'(g_2'^3 - g_1'^3) \equiv \Delta_2^3 - \Delta_1^3$$

$$\frac{\partial f}{\partial w_{in}^{32}} = \Delta_2^3 g_{2m}^2 - \Delta_1^3 g_{1m}^2$$

$$\frac{\partial f}{\partial b_m^2} = \Delta_2^3 w_m^{32} g_{2m}'^2 - \Delta_1^3 w_m^{32} g_{1m}'^2$$

$$\frac{\partial f}{\partial w_{mn}^{21}} = \Delta_2^2 g_{2n}^1 - \Delta_1^2 g_{1n}^1$$

Note that the terms always take the form of the difference of a term depending on x_1 and a term depending on x_2, 'coupled' by an overall multiplicative factor of f', which depends on both. Thus it can be seen that ranking using neural net or in short RankNet can be accomplished be a simple modification of the back-propagation algorithm.

Another interpretation

Some engineers at Microsoft came up with a more well-defined and complex algorithms which would be used in general for ranking. This was similar to the above described but was more well-defined and produced better results. This did away with many unnecessary steps that would take up more time and space slowing down computation. This is in a way a parallel drawn from the previously shown algorithm with more compact symbols making it easier to grasp. The previous derivation was a formulation of 4 layered network which has one input layer, 2 hidden layer and one output layer. However, the one proposed by the engineers at Microsoft is more generalized over n number of hidden layers. They provided the derivation for one layer, which can be further extended to n layered problem.

For this model, the training data is partitioned with respect to the query. At the point of training data, each input $x \in \mathbb{R}^d$, where d is the number of features, is assigned to a number $f(x) = s$ which is its score pertaining to the model. For a given query, each pair of documents or items are taken into account. Let the pair of items be their feature vectors x_i and x_j. Let the scores be defined by $s_i = f(x_i)$ and $s_j = f(x_j)$. Let $x_i \rhd x_j$ define that the ith item is more relevant for this query than the jth item. However, their relevance varies w.r.t. the query. For one query it may have the opposite relevance. The two scores are then mapped to a probability that ith item has a higher relevance than the jth item using a sigmoid function.

$$P_{ij} = P(x_i \rhd x_j) = \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

The cross-entropy cost function which penalizes the deviation of the model output probabilities from the desired probabilities: let $\bar{P}_{ij}$ be the known probability of the ranking for the pair (x_i, x_j). Then the cost is defined as:

$$C = -\bar{P}_{ij}\log(P_{ij}) - (1 - \bar{P}_{ij})\log(1 - P_{ij})$$

For a given query, $S_{ij} \epsilon \{0, \pm 1\}$ be defined to be 1 if ith document has a higher relevance than jth document, -1 if opposite and 0 if both have the same relevance. It has been assumed here that query-document pairs' rankings are known. Then, $\bar{P}_{ij}$ is defined to be $\bar{P}_{ij} = \frac{1}{2}(1 + S_{ij})$.

The above equations can be combined to form,

$$C = \frac{1}{2}(1 + S_{ij})\sigma(s_i - s_j) + \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

Till here, it follows the same principle that was followed earlier. It can be seen that the cost is highly symmetric, i.e. interchanging i and j and simultaneously changing the sign of S_{ij} would leave the cost invariant. One important thing to note is that when $s_i = s_j$, the cost is $\log 2$, so the model incorporates a margin, i.e. documents with different labels but to which the model assigns same scores are still pushed away in ranking. This gives:

$$\frac{\partial C}{\partial s_i} = \sigma\left(\frac{1}{2}(1 - S_{ij}) - \frac{1}{1 + e^{-\sigma(s_i - s_j)}}\right) = -\frac{\partial C}{\partial s_j}$$

This gradient would be used to update the weights $w_k \epsilon \mathbb{R}$ associated with the model so as to reduce the cost via stochastic gradient method:

$$w_k \longrightarrow w_k - \eta\frac{\partial C}{\partial w_k} = w_k - \eta\left(\frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k}\right)$$

where η is a positive learning rate. Explicitly,

$$\delta C = \sum_k \frac{\partial C}{\partial w_k}\delta w_k = \sum_k \frac{\partial C}{\partial w_k}\left(-\eta\frac{\partial C}{\partial w_k}\right) = -\eta\sum_k\left(\frac{\partial C}{\partial w_k}\right)^2 < 0$$

One important note that many other algorithms like gradient boosted trees like MART does is it directly computes $\frac{\partial C}{\partial s_i}$ without computing $\frac{\partial C}{\partial w_k}$ thus saving computational and memory space enabling faster and efficient computing.

The above equation can also be modelled as:

$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k} = \sigma\left(\frac{1}{2}\left(1 - S_{ij}\right) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right)\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right)$$
$$= \lambda_{ij}\left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k}\right)$$

where λ_{ij} has been defined as

$$\lambda_{ij} = \frac{\partial C\left(s_i - s_j\right)}{\partial s_i} = \sigma\left(\frac{1}{2}\left(1 - S_{ij}\right) - \frac{1}{1 + e^{\sigma(s_i - s_j)}}\right)$$

This method has some pros and cons in the ranking system. This is the way a neural network can be used for ranking. It is an intermixed of classification and regression but unlike both, the document doesn't solely belong to one class or one rank. Depending on the query, the features are generated which then help in ranking the document. This is an intermixing of regression and classification where initially it works as a regressor but then it acts like a classifier but given some other parameters and conditions. To conclude, each input that the model sees is a new input consisting of 2 query-document pairs which needs to be classified in either of the three classes: if the rank of first is better than the second or vice versa or are they equal in terms of their ranks. Based on all the ranking, a list called a ranking list is derived. On this the metrics mentioned before are calculated and further calibration is done on the basis of those.

Linear Regression

In the RankNet if a computational neuron is considered, which is simply a product with the weights of each of the features and their sum without any activation function, then the output of the neuron would be the score of the document with respect to the query. Once all the documents for a given query have been assigned scores then the ranking is done after which the ranks are changed in order to attain better rankings.

Drawbacks of RankNet

RankNet is optimizing for the number of pairwise errors, which is fine if that is the desired cost, but it does not match well with some other information retrieval measures. The idea of writing down the desired gradients directly, rather than deriving them from a cost, is one of the ideas underlying LambdaRank: it allows us to bypass the difficulties introduced by the sort in most IR measures. Note that this does not mean that the gradients are not gradients of a cost. In other words, in RankNet the optimization is done for gaining accuracy over the other information retrieval measures. However, this problem has to be eradicated somehow. The focus of the algorithm should be on maximizing the measure for information retrieval and not the measure for accuracy as told before. Here the concern is about the ranking and not the order of ranking.

Suppose two documents of same rank are shown to this algorithm in a specific way. If it is shown in the reverse way the ranking should not change but the order of presenting the documents has changed. This ordering would not have affected the measures for information retrieval but this is an inaccuracy in terms of measuring accuracy. This measure was taken care of in LambdaRank, another algorithm developed by Microsoft for better ranking of documents.

Transition from RankNet to LambdaRank

The major throwback and problem with the previous algorithms is that in all the algorithms there was an indirect way to pertain to the accuracy of the order of ranking. However, this is not a desirable trait when it comes to the order of ranking. This is because the main concern is not the order of ranking but the ranking itself. In LambdaRank, the cost with respect to the information retrieval measures is not considered. Here, the gradient of the cost with respect to the model scores are taken into consideration.

It has been experimentally shown that if λ_{ij} is modified by multiplying it with the size of the change in NDCG ($|\Delta_{NDGC}|$) given by swapping the rank positions of items i and j, good results are obtained. Thus λ_{ij} becomes:

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta_{NDGC}|$$

Now the task of C has changed. Preciously the task of C was to minimize the error. But now, there is a need to maximize the change in NDCG to obtain better results. Since here the task is to maximize C , the weight equation is changed to

$$w_k \longrightarrow w_k + \eta \frac{\partial C}{\partial w_k}$$

so that,

$$\delta C = \sum_k \frac{\partial C}{\partial w_k} \delta w_k = \sum_k \frac{\partial C}{\partial w_k}\left(\eta \frac{\partial C}{\partial w_k}\right) = \eta \sum_k \left(\frac{\partial C}{\partial w_k}\right)^2 > 0$$

Thus although Information Retrieval measures, viewed as functions of the model scores, are either flat or discontinuous everywhere, the LambdaRank idea bypasses this problem by computing the gradients after the items have been sorted by their scores. It has been shown empirically that, intriguingly, such a model actually optimizes NDCG directly. In fact it has been further shown that if there is a desire to optimize some other information retrieval measure, such as MRR or MAP, then LambdaRank can be trivially modified to accomplish this: the only change is that $|\Delta_{\text{NDGC}}|$ above is replaced by the corresponding change in the chosen IR measure.

MART

By the time the above methods were being discovered, significant work had been done on decision trees and its application in the fields of regressing and ranking as its working in the field of classification was already well known and established. It was combined with the process of boosting which is applied to the decision of the tree in consideration. To understand MART, decision trees and boosting needs to be understood. Decision tree is again a classification algorithm which makes decisions based on certain attributes of the data and classifies the data into various categories based on a lot of factors like depth of a tree, number of leaves per node. Based on the divisions of the data, it creates the model and then classifies the data. Boosting is the process in which the wrong outputs of a function are given higher priority while calibrating the parameters of the mode. This can be understood like if a person makes a mistake and is severely penalized at a step of the task then when he is asked to repeat the same take again he would take extra care not to repeat the same mistake again. Thus it can be well imagined how a combination of decision trees and boosting can be used for classification. Now the major question arises that how to use this model for ranking which involves regression and classification together. This problem was partially solved by the advent of MART and fully solved by the advancement in MART by getting in LambdaMART.

MART covers the regression and boosting part of ranking problem. With the help of decision trees, which was initially built for classification, can be used for regression analysis with some modifications. It partitions the data in terms of the number of leaves and space with respect to the features. The loss function or the error metric computed using MART is basically the mean error in determining the value of the feature in training. This is similar to the inaccuracy or the misclassification rate in classification using decision trees.\

Mathematical modeling

In short, MART or Multiple Additive Regression Trees is a boosted tree model in which the model is a linear combination of the outputs of a set of regression trees. A decision tree partitions the parameter space into disjoint regions $\mathbb{R}_k, k \epsilon\{1,2,\ldots,K\}, K =$ number of leaves. Formally, the regression model predicts a value using a constant γ_k for each region $\mathbb{R}_k$:

$$T(\boldsymbol{x};\Theta) = \sum_{k=1}^{K} \gamma_k l(\boldsymbol{x} \epsilon\, \mathbb{R}_k)$$

$\Theta = \{\mathbb{R}_k, \gamma_k\}_1^K$ describes the model parameters, $l(.)$ is the characteristic function (1 if the argument is true, otherwise 0) and $\gamma_k = mean(y_i | x_i \epsilon \mathbb{R}_k)$. Optimal parameters $\widehat{\Theta}$ are found minimizing the empirical risk:

$$\widehat{\Theta} = \arg\min \sum_{k=1}^{K} \sum_{x_i \epsilon \mathbb{R}_k} L(y_i, \gamma_k)$$

Here, $L(.)$ is the loss function which is defined as following:

$$L(y_i, \gamma_k) = (\gamma_k - y_i)^2$$

This loss function is also to be minimized as follows:

$$S_n = \sum_{j=1}^{N} \left(\sum\nolimits_{i \epsilon L} (\gamma_i - y_j)^2 + \sum_{i \epsilon R} (\gamma_i - y_j)^2 \right)$$

Here N is the total number of instances. L(R) is the sets of indices of samples that fall in the left or the right and γ represents the mean at the nth node. Thus there is an attempt to minimize the cost at each node resulting in the least cost of the overall tree.

Boosting combines a set of multiple weak learners to build a strong learner. A weak learner is one which has an error rate lower than random guessing. The approach for any boosting task is as follows:-

1. Apply a weak learner to iteratively modified data
2. Generate a sequence of learners
3. Build weighted values

The next work is to find a function $F^*(x)$ that maps x to y, such that the expected value of the loss function is minimized:

$$F^*(x) = \arg F(x) \min \mathbb{E}_{y,x}\left[L\big(y, F(x)\big)\right]$$

The boosting approximated $F^*(x)$ by an additive expansion

$$F(x) = \sum_{m=1}^{M} \beta_m h(x; a_m)$$

where $h(x; a_m)$ are simple functions of x with parameters $a = \{a_1, \dots, a_n\}$ defining the function h and β are the expansion coefficients. The expansion coefficients $\{\beta_m\}_0^M$ and the function parameters $\{a_m\}_0^M$ are iteratively fit to the training data:

1. $F_0(x)$ is set randomly
2. For each $m = 1, \dots, M$

$$(\beta_m, a_m) = \arg\min \sum_{i=1}^{N} L\big(y_i, F_{m-1}(x_i) + \beta h(x_i, a)\big)$$

and

$$F_m(x) = F_{m-1}(x) + \beta_m h(x; a_m)$$

The process of gradient boosting is mainly used for solving the differentiable loss functions.

1. Fit the function $h(x; a_m)$ by the use of least square method:-

$$a_m = \arg\min \sum_{i=1}^{N} [\widetilde{y_{im}} - h(x_i, a)]^2$$

   to the “pseudo”-residuals

$$\widetilde{y_{im}} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$$

2. Given $h(x; a_m)$, the β_m are

$$\beta_m = \arg\min \sum_{i=1}^{N} L\big(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m)\big)$$

The parameters a_m for K-terminal node regression trees are represented as $\mathbb{R}_{km}$.

$$h(x; \{\mathbb{R}_{km}\}_1^K) = \sum_{k=1}^{K} \overline{y_{km}}\, l(x \epsilon \mathbb{R}_{km})$$

with $\overline{y_{km}} = mean_{x_i \epsilon \mathbb{R}_{km}}(\widetilde{y_{km}})$ the tree predicts a constant value $\overline{y_{km}}$ in the region $\mathbb{R}_{km}$. The β_m is the prediction of the tree and is represented by γ_{km} for each $\mathbb{R}_{km}$:

$$\gamma_{km} = \arg\min \sum_{x_i \epsilon \mathbb{R}_{km}} L(y_i, F_{m-1}(x_i) + \gamma)$$

The approximation of F in stage m is then:

$$F_m(x) = F_{m-1}(x) + \eta\, \gamma_{km} l(x \epsilon \mathbb{R}_{km})$$

The parameter η controls the learning rate of the procedure.

The stopping parameters are the number of trees M.

To summarize, (which stands for Multiple Additive Regression Trees, or also known as Gradient boosted regression tree) is an approach utilizing a boosted tree model in which the output of the model is a linear combination of the outputs of a set of regression trees. MART is considered a class of boosting algorithms that may be viewed as performing gradient descent in function space, using regression trees. MART can be trained to minimize any general cost (classification, regression, ranking), however, underlying model up on which MART is build is the least squares regression tree.

Since MART belongs to the family of boosting algorithms, it runs a several rounds of boosting and in each step there is a regression tree added and its weight is determined. The final scoring (ranking) function is defined as follows

$$F_N(\overrightarrow{x_j}) = \sum_{i=1}^{N} \alpha_i f_i(\overrightarrow{x_j})$$

where, $f_i(.)$ is a ranking function of a single regression tree, α_i is a weight or learning rate and $\overrightarrow{x_j}$ is the feature vector of document d_j.

LambdaMART

Though MART can be used for ranking also, it tends to give better results than liner regression but lags behind SVM. The problem lies again in the calculation of the loss function and its inability to handle pairwise samples. Thus to take care of this problem, lambdaMART was introduced which was a child of lambdaRank and MART algorithm. It used the fact that MART had multiple decision trees and was a part of ensemble learning which helped in reducing the error and giving better results in terms of the output regression. LambdaRank's concept of using the IR measures was used here. Another problem that was seen in lambaRank was that it updated all its weights after each query was examined. But that would result in faulty updates as no two queries-document pair would have the same feature vector. This problem was solved in LambdaMART when the weight updates happened only on those nodes using all the data that falls on it, and so LambdaMART updates only a few parameters at a time (namely, the split values for the current leaf nodes), but using all the data (since every x_i lands in some leaf). This means in particular that LambdaMART is able to choose splits and leaf values that may decrease the utility for some queries, as long as the overall utility increases. This is the uniqueness that it provides over the LambdaRank.

Mathematical Modelling

Thus, MART algorithm is used along with specific gradients and Newton step. The gradients $\overline{y_i}$ are easy to compute as they are just λ_is from the LambdaRank. Unlike MART, here each tree models the λ_i for the entire dataset and not just a single query. For computation of the Newtonian step, all the $\lambda_i s$ are required.

$$\lambda_{ij} = \frac{-\sigma|\Delta Z_{ij}|}{1 + e^{\sigma(s_i - s_j)}},$$

Here Z corresponds to the information retrieval measure taken into consideration e.g. NDCG. Thus λ_i can be written as

$$\lambda_i = \sum_{j:\{i,j\}\epsilon I} \lambda_{ij} - \sum_{j:\{j,i\}\epsilon I} \lambda_{ij}$$

Here, for a particular item the cost function is defined as

$$C = \sum_{\{i,j\} \rightleftharpoons I} \left|\Delta Z_{ij}\right| \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

so that,

$$\frac{\partial C}{\partial s_i} = \sum_{\{i,j\} \rightleftharpoons I} \frac{-\sigma\left|\Delta Z_{ij}\right|}{1 + e^{-\sigma(s_i - s_j)}} \equiv \sum_{\{i,j\} \rightleftharpoons I} -\sigma\left|\Delta Z_{ij}\right|\rho_{ij}$$

where

$$\rho_{ij} \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}} = \frac{-\lambda_{ij}}{\sigma\left|\Delta Z_{ij}\right|}$$

Then,

$$\frac{\partial^2 C}{\partial s_i^2} = \sum_{\{i,j\} \rightleftharpoons I} \sigma^2\left|\Delta Z_{ij}\right|\rho_{ij}\left(1 - \rho_{ij}\right)$$

and the Newton step size for the kth leaf of the mth tree is given as

$$\gamma_{km} = \frac{\sum_{x_i \epsilon \mathbb{R}_{km}} \frac{\partial C}{\partial s_i}}{\sum_{x_i \epsilon \mathbb{R}_{km}} \frac{\partial^2 C}{\partial s_i^2}} = \frac{-\sum_{x_i \epsilon \mathbb{R}_{km}} \sum_{\{i,j\} \rightleftharpoons I} \left|\Delta Z_{ij}\right|\rho_{ij}}{\sum_{x_i \epsilon \mathbb{R}_{km}} \sum_{\{i,j\} \rightleftharpoons I} \sigma\left|\Delta Z_{ij}\right|\rho_{ij}\left(1 - \rho_{ij}\right)}$$

While implementing, it is highly convenient to simply compute ρ_{ij} for each sample x_i; then for computing γ_{km} for any leaf node would just involve summing and dividing. Similar to logistic regression, for a given learning rate η the choice of σ will not make any difference to the training, since the γ_{km}'s scale as $\frac{1}{\sigma}$, the model score is always incremented by $\eta\gamma_{km}$ and then multiplied by σ.

Though computationally, LambdaMART is expensive but in the long run it provides the best results when compared to all the models mentioned earlier.

Dataset description with an example on pre-processing

The dataset referred here is the Reuters21578 dataset. The reuters-21578 dataset was compiled by David Lewis and originally collected by the Carnegie group from the Reuters newswire in 1987. It contains 21578 news articles, each belonging to one or more categories. The frequency of occurrence of documents varies greatly from category to category.

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"
OLDID="5552" NEWID="9">
<DATE>26-FEB-1987 15:17:11.20</DATE>
<TOPICS><D>earn</D></TOPICS>
<PLACES><D>usa</D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0762&#31;reute
r f BC-CHAMPION-PRODUCTS-&lt;CH   02-26 0067</UNKNOWN>
<TEXT>&#2;
<TITLE>CHAMPION PRODUCTS &lt;CH> APPROVES STOCK SPLIT</TITLE>
<DATELINE>    ROCHESTER, N.Y., Feb 26 - </DATELINE><BODY>Champion
Products Inc said its
board of directors approved a two-for-one stock split of its
common shares for shareholders of record as of April 1, 1987.
    The company also said its board voted to recommend to
shareholders at the annual meeting April 23 an increase in the
authorized capital stock from five mln to 25 mln shares.
 Reuter
&#3;</BODY></TEXT>
</REUTERS>
```

Figure 3: Example of the Reuters21578 with topic as "earn"

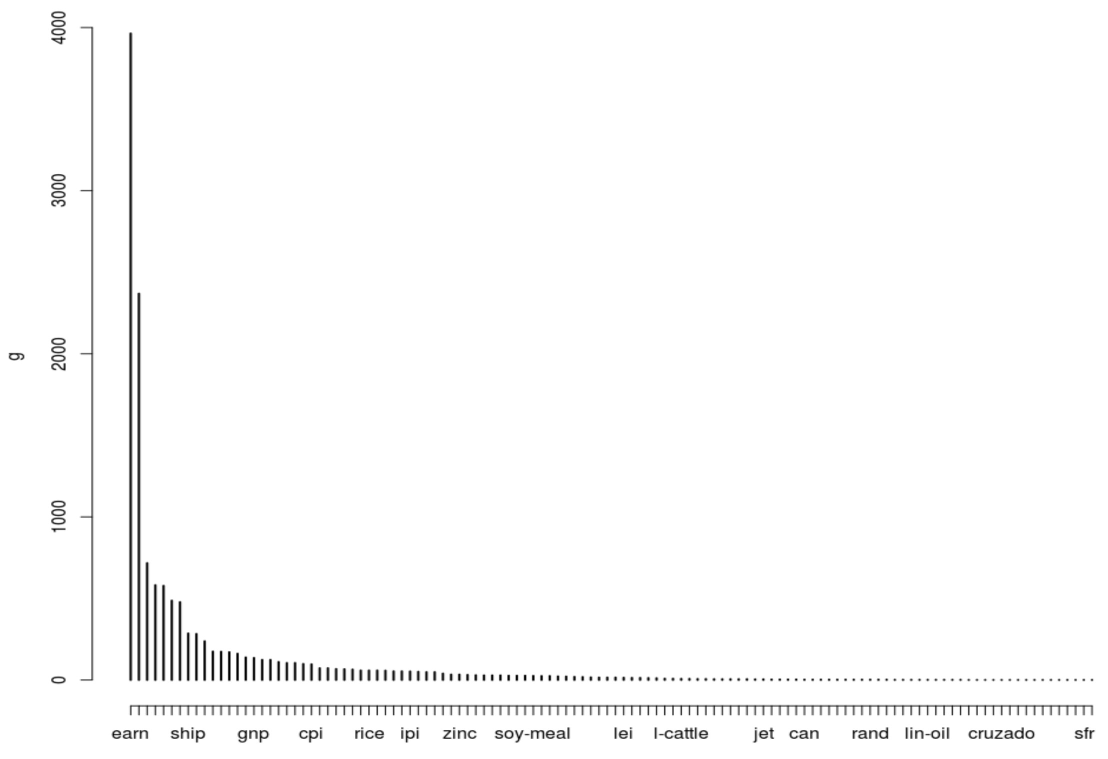

Figure 4: Graphical representation of topics VS frequency

Here it can be seen that most of the data is present in the first few topics. So heuristically top 10 topics were taken. The below table would be a better description of the above figure.

	Topic	Frequency
1	Earn	3964
2	Acq	2369
3	money-fx	717
4	Grain	582
5	Crude	578
6	Trade	487
7	Interest	478
8	Ship	286
9	wheat	283
10	corn	238

Just as an example of how the pre-processing actually works out, the data in the above shown figure was taken for analysis.

1. The raw data
   Champion
   Products Inc said its
   board of directors approved a two-for-one stock split of its
   common shares for shareholders of record as of April 1, 1987.
   The company also said its board voted to recommend to
   shareholders at the annual meeting April 23 an increase in the
   authorized capital stock from five mln to 25 mln shares.
2. After lowering the case
   champion
   products inc said its
   board of directors approved a two-for-one stock split of its
   common shares for shareholders of record as of april 1, 1987.
   the company also said its board voted to recommend to
   shareholders at the annual meeting april 23 an increase in the

authorized capital stock from five mln to 25 mln shares.

3. After removing numbers
   champion
   products inc said its
   board of directors approved a two-for-one stock split of its
   common shares for shareholders of record as of april , .
   the company also said its board voted to recommend to
   shareholders at the annual meeting april an increase in the
   authorized capital stock from five mln to mln shares.
4. After removing punctuations
   champion
   products inc said its
   board of directors approved a two-for-one stock split of its
   common shares for shareholders of record as of april
   the company also said its board voted to recommend to
   shareholders at the annual meeting april an increase in the
   authorized capital stock from five mln to mln shares
5. After removing stop words
   champion
   products inc said
   board directors approved twoforone stock split
   common shares shareholders record april
   company also said board voted recommend
   shareholders annual meeting april increase
   authorized capital stock five mln mln shares
6. After stemming (R software was used)
   champion product inc said board director approv twoforon stock split common share sharehold record april compani also said board vote recommend sharehold annual meet april increas author capit stock five mln mln share

Now all these words were taken and weighed using TF-IDF weighing scheme. One problem that lies with this kind of problem is that the matrix that would be produced would be highly sparse. The main reason is that there are many words which are present in some documents and not in others. Thus the matrix has 0 in those places. Using the mod Apt split mentioned before and post-pre-processing the training set had a non-sparsity of around 0.27% while the testing had around 0.41% which is suggestive of the fact that most of the contents are zero.

The Reuters21578 was used for classification and clustering. For the ranking of documents, another dataset by the name of MQ2007 was used. MQ2007 is a query set made from Gov2 web page collection (~25M pages) and query sets from Million Query track of TREC 2007. There are about 1700 queries in MQ2007 with labelled documents. Each row is a query - document pair. The first column is relevance label of this pair, the second column is query id, the following columns are features, and the end of the row is comment about the pair, including id of the document. The larger the relevance label, the more relevant the query- document pair. A query-document pair is represented by a 46-dimensional feature vector. Here are several example rows from MQ2007 dataset:

```
=================================

2 qid:10032 1:0.056537 2:0.000000 3:0.666667 4:1.000000 5:0.067138 ...
45:0.000000 46:0.076923 #docid = GX029-35-5894638 inc = 0.0119881192468859 prob =
0.139842
0 qid:10032 1:0.279152 2:0.000000 3:0.000000 4:0.000000 5:0.279152 ...
45:0.250000 46:1.000000 #docid = GX030-77-6315042 inc = 1 prob = 0.341364
0 qid:10032 1:0.130742 2:0.000000 3:0.333333 4:0.000000 5:0.134276 ...
45:0.750000 46:1.000000 #docid = GX140-98-13566007 inc = 1 prob = 0.0701303
1 qid:10032 1:0.593640 2:1.000000 3:0.000000 4:0.000000 5:0.600707 ... 45:0.500000
46:0.000000 #docid = GX256-43-0740276 inc = 0.0136292023050293 prob = 0.400738
=================================
```

The 46 features in the dataset are extracted as follows:-

Column	Description
1	TF(Term frequency) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of the whole document
6	IDF of body
7	IDF of the anchor
8	IDF of the title
9	IDF of the URL

10	IDF of the whole document
11	TF*IDF of body
12	TF*IDF of anchor
13	TF*IDF of title
14	TF*IDF of URL
15	TF*IDF of whole document
16	DL(document length) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	BM25 of anchor
23	BM25 of title
24	BM25 of URL
25	BM25 of whole document
26	LMIR.ABS of body
27	LMIR.ABS of anchor
28	LMIR.ABS of title
29	LMIR.ABS of URL
30	LMIR.ABS of whole document
31	LMIR.DIR of body
32	LMIR.DIR of anchor
33	LMIR.DIR of title
34	LMIR.DIR of URL
35	LMIR.DIR of whole document
36	LMIR.JM of body
37	LMIR.JM of anchor
38	LMIR.JM of title

39	LMIR.JM of URL
40	LMIR.JM of whole document
41	PageRank
42	InLink number
43	OutLink number
44	Number of slashes in URL
45	Length of URL
46	Number of child page

Some details about these features are listed as below.

1. For the language model features, the implementation and suggested parameters given in Zhai et al. (2001) were used: for LMIR.ABS features, parameter was set d = 0.5; for LMIR.DIR features, parameter was set l = 50; for LMIR.JM features, parameter was set k = 0.5.
2. For BM25 feature, its parameters was set as k1 = 1.2, k3 = 7 and b = 0.75.

We partitioned each dataset into five parts with about the same number of queries, denoted as S1, S2, S3, S4, and S5, for five-fold cross validation

Folds	Training set	Testing	Validation
Fold1	{S1,S2,S3}	S4	S5
Fold2	{S2,S3,S4}	S5	S1
Fold3	{S3,S4,S5}	S1	S2
Fold4	{S4,S5,S1}	S2	S3
Fold5	{S5,S1,S2}	S3	S4

Results

The clustering results were not very prominent and didn't give any vital information. K-means and GMM both produced similar results. It is said to vary the value of k and find the "knee" point or the "elbow" point. The following graph was produced from varying k in k-means and similar was for GMM.

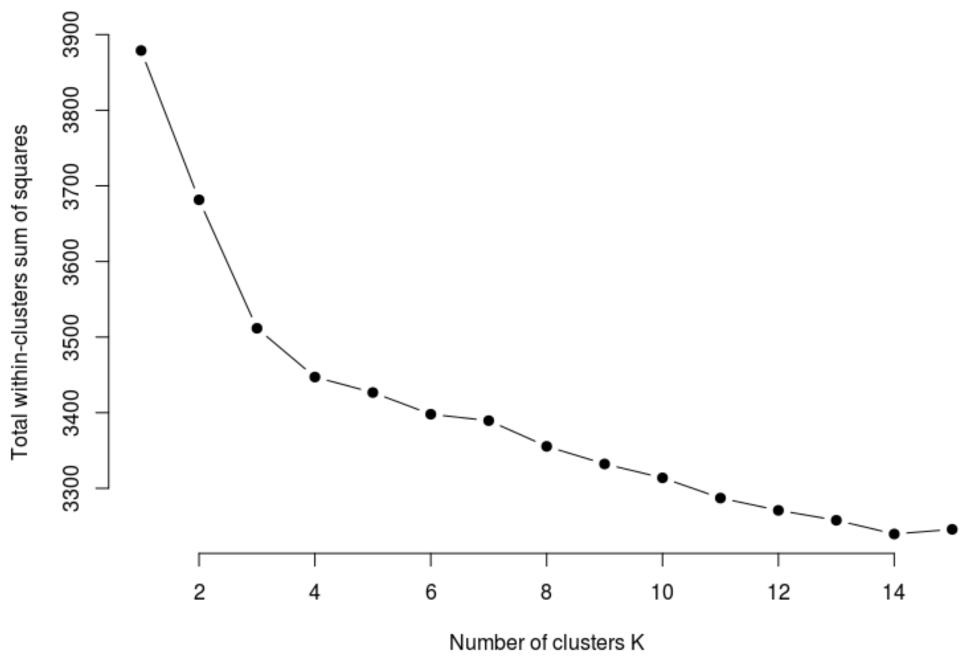

Figure 5: Total within-cluster sum of squares VS no of clusters

From the graph it can be forecasted that the value of k should be either 3 or 4. However, while viewing the clusters formed it was unable to differentiate between topics. Further analysis would be needed to figure out what is it basing its clusters on. It was the same case with GMM also.

The focus was shifted onto classification. KNN with the two distance metrics were done and the results that were obtained were not what was assumed.

For KNN with cosine distance metric the maximum accuracy achieved was 55.6%. The graph below shows the accuracy versus the k where the range of k is from 1 to 101 with increments by 2.

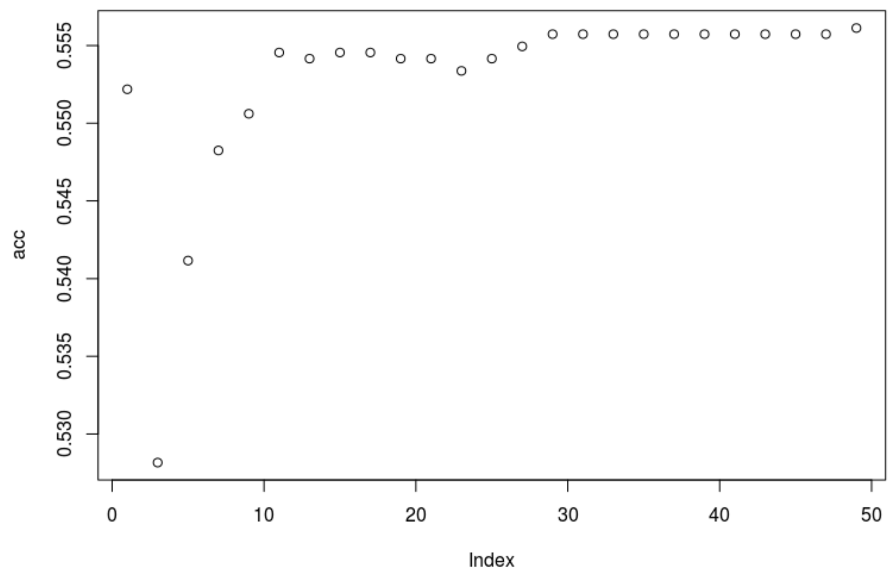

Figure 6: Cosine distance VS K

This was compared with Euclidian distance. The graph below shows the accuracy.

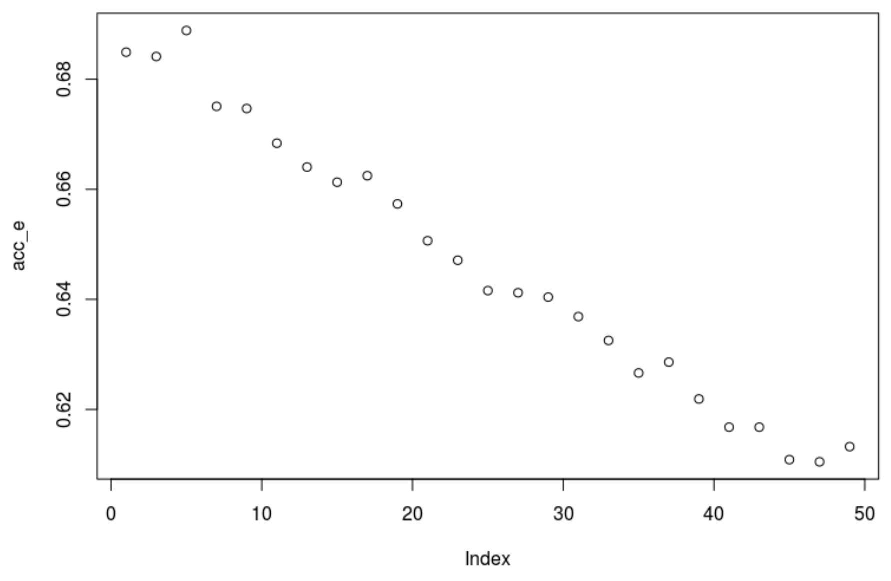

Figure 7: Euclidian VS K

It was found that accuracy of the second type was better than that by cosine metric. However, when random forest was involved, the accuracy of the training set was 90% while of the testing set was 86.53%.

When it comes to ranking, the tables below showcase the results.

LambdaMART

tree	leaf	learning rate	training	validation	testing
1500	6	0.1	0.48292	0.45228	0.44894
500	6	0.1	0.48292	0.45228	0.44894
100	6	0.1	0.47558	0.45122	0.44864
10	6	0.1	0.43802	0.43444	0.43128
20	6	0.1	0.44822	0.43962	0.43618
50	6	0.1	0.4615	0.44822	0.44326
75	6	0.1	0.4688	0.4493	0.44656
90	6	0.1	0.47148	0.45024	0.44722
200	6	0.1	0.48292	0.45228	0.44894
150	6	0.1	0.48292	0.45228	0.44894
125	6	0.1	0.48252	0.45224	0.44918

RankNet

Epoch	layer	node	training	validation	testing
100	1	10	0.42394	0.42646	0.42274
100	1	20	0.42632	0.42792	0.42448
100	1	30	0.42462	0.42864	0.42364
100	1	40	0.425	0.42756	0.42578
200	1	40	0.42628	0.4286	0.42322
100	1	50	0.42582	0.43006	0.42394
200	1	50	0.4252	0.4296	0.42234
100	2	10	0.38002	0.3763	0.37968

Linear regression

L2 regularisation	training	validation	testing
1.00E-10	0.43274	0.42722	0.42772
1	0.43306	0.42806	0.42752
1.00E+05	0.40606	0.40466	0.4049

SVM using Nystrom Approximation

N	C	gamma	training	validation	testing
440	2	2^-6	0.43006	0.41612	0.42044
400	-2	2^-7	0.42272	0.41456	0.41792
400	3	2^-7	0.4331	0.42006	0.42064
500	-2	2^-7	0.42126	0.41262	0.41742
500	3	2^-7	0.43368	0.41552	0.41984
600	-2	2^-7	0.4214	0.41438	0.41714
600	3	2^-7	0.43014	0.41514	0.42104
700	-2	2^-7	0.42126	0.41436	0.41852
700	3	2^-7	0.4327	0.42066	0.42162
800	-2	2^-7	0.42176	0.41298	0.41772
800	3	2^-7	0.43526	0.42044	0.41906
900	-2	2^-7	0.422	0.41318	0.41772
900	3	2^-7	0.43388	0.42262	0.4202
1000	-2	2^-7	0.4218	0.41258	0.41852
1000	3	2^-7	0.4378	0.41946	0.41924
1500	-2	2^-7	0.4218	0.413	0.41852
1500	3	2^-7	0.43718	0.4218	0.41886
2000	-2	2^-7	0.4218	0.41318	0.41832
2000	3	2^-7	0.43782	0.42046	0.42164
2000	-2	2^-5	0.43414	0.42186	0.41988

SVM using Random Fourier Features

N	C	gamma	training	validation	testing
440	2	2^-6	0.42074	0.41516	0.4183
400	-2	2^-7	0.43	0.4161	0.41986
400	3	2^-7	0.42062	0.47476	0.41852
500	-2	2^-7	0.43308	0.42102	0.42004
500	3	2^-7	0.42114	0.41438	0.41812
600	-2	2^-7	0.43414	0.42144	0.43144
600	3	2^-7	0.42174	0.41378	0.41792
700	-2	2^-7	0.4344	0.43938	0.41948

700	3	2^-7	0.42128	0.4203	0.42502
800	-2	2^-7	0.4323	0.41374	0.41768
800	3	2^-7	0.42184	0.4124	0.41852
900	-2	2^-7	0.43478	0.41984	0.423
900	3	2^-7	0.42142	0.41258	0.41852
1000	-2	2^-7	0.43554	0.41866	0.42064
1000	3	2^-7	0.4218	0.41318	0.41832
1500	-2	2^-7	0.43734	0.41868	0.41946
1500	3	2^-7	0.4218	0.41318	0.41832
2000	-2	2^-7	0.44048	0.41928	0.42064
2000	3	2^-7	0.43538	0.4179	0.41966
2000	-2	2^-5	0.42074	0.41516	0.4183

It was seen and observed that despite all the models available for usage, LambdaMART provided the best result due to the major reason that it is a ranking equivalent of an ensemble of decision stumps or random forest.

Working on a toy dataset

Document or text

1. This dog is my pet, Lucy. She eats pedigree. She is healthy for past 2 years. She is 2.5 years old. She is a german shephard.
2. This cat is Daisy. She drinks milk. She hates fish. She roams around the house. She doesn’t like to take a bath.
3. This dog is David. He barks a lot. He is my best friend. He drops me to school. He eats a lot.
4. This parrot is Sunny. He can mimic anyone. He loves to talk to people. He eats mostly grapes and nuts. He is not kept in a cage. Sunny is a free bird.
5. This is my fish Goldy. She swims freely in the tank. She makes friends with all kinds of fishes. She is never shy of anyone.

Query

1. Dog food
2. Cat
3. Pet dog

4. Animals

Pre-processing

Same as in the case of Reuters

Features

1. Number of words in query
2. Number of words in document
3. TF
4. IDF
5. TF-IDF
6. Boolean retrieval

TF of the documents after pre-processing

	anyon	around	bark	bath	best	bird	cage	can	cat	daisi	david	doesnt	dog	drink	drop	eat	fish	fr
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	
2	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0	0	1	
3	0	0	1	0	1	0	0	0	0	0	1	0	1	0	1	1	0	
4	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	
5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	

TF-IDF

	anyon	around	bark	bath	best	bird	cage	can	cat	daisi	david	doesnt
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.1786	0	0.1786	0	0	0	0	0.1786	0.1786	0	0.1786
3	0	0	0.2322	0	0.2322	0	0	0	0	0	0.2322	0
4	0.0778	0	0	0	0	0.1366	0.1366	0.1366	0	0	0	0
5	0.1102	0	0	0	0	0	0	0	0	0	0	0

But there are many columns so one feature that is taken is the row means, which boils it down to only one variable.

Row means of the TF is 0.2222, 0.24074, 0.185185, 0.3148148 and 0.222222.

Row means of the TF-IDF is 0.039, 0.04157, 0.03634, 0.040183 and 0.03683.

Row means of the DF is 1.331, 1.4262, 1.01, 2.5635 and 1.2699.

Boolean retrieval would give values like

	Document 1	Document 2	Document 3	Document 4	Document 5
Query 1	4	3	2	3	2
Query 2	0	1	0	0	0
Query 3	2	0	1	0	0
Query 4	2	2	1	1	2

Now we have all the parameters. We can apply them to one model at a time.

The query 1, 2 and 3 would be used for training and 4 would be used for testing. The matrix would look like this:

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Rank
Q1 D1	2	12	0.2222	1.331	0.039	4	1
Q1 D2	2	13	0.24074	1.4262	0.04157	3	0
Q1 D3	2	10	0.185185	1.01	0.03684	2	1
Q1 D4	2	17	0.314815	2.5635	0.040183	3	0
Q1 D5	2	12	0.2222	1.2699	0.03683	2	0
Q2 D1	1	12	0.2222	1.331	0.039	0	0
Q2 D2	1	13	0.24074	1.4262	0.04157	1	1
Q2 D3	1	10	0.185185	1.01	0.03684	0	0
Q2 D4	1	17	0.314815	2.5635	0.040183	0	0
Q2 D5	1	12	0.2222	1.2699	0.03683	0	0
Q3 D1	2	12	0.2222	1.331	0.039	2	1
Q3 D2	2	13	0.24074	1.4262	0.04157	0	0
Q3 D3	2	10	0.185185	1.01	0.03684	1	1
Q3 D4	2	17	0.314815	2.5635	0.040183	0	0
Q3 D5	2	12	0.2222	1.2699	0.03683	0	0

Q4 D1	1	12	0.2222	1.331	0.039	2	1
Q4 D2	1	13	0.24074	1.4262	0.04157	2	1
Q4 D3	1	10	0.185185	1.01	0.03684	1	0
Q4 D4	1	17	0.314815	2.5635	0.040183	1	0
Q4 D5	1	12	0.2222	1.2699	0.03683	2	1

Ranking via SVM is best understood when seen its working visually. For that, maximum of 2D data can be allowed and not more as it would be difficult to plot. To start with, a dataset is created in which the target values consist of three graded measurements Y = {0, 1, 2} and the input data is a collection of 30 samples, each one with two features.

The set of comparable elements (queries in information retrieval) will consist of two equally sized blocks, $X = X_1 \cup X_2$, where each block is generated using a normal distribution with different mean and covariance. In the figures, X_1 is represented with round markers and X_2 is represented with triangular markers.

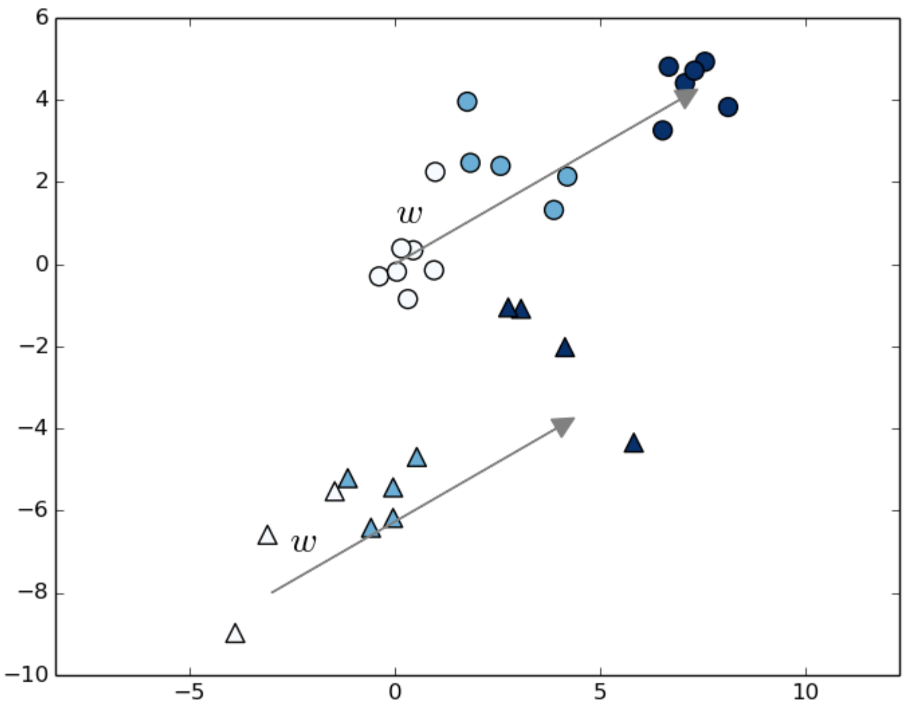

Figure 8: 2 sets of data and the same weight vector

From the plot it is quite evident that there exists a weight vector w on which if the data is projected would give a near perfect ranking. Assuming that the ranking model can be drawn as a parallel to regression model, the first trial would be to go for linear regression. One factor that linear regression would assume is that all the data in the two blocks are comparable. This is the point where it will make a blundering mistake. Thus if a weight vector is to be assumed to rank the above data, the using L2-regularization it can be found out. For this purpose ridge regression would be used. Ridge regression is a method that seeks to reduce the MSE by adding some bias and, at the same time, reducing the variance. From an equation standpoint, it can be thought of as a method that seeks to find the coefficients that minimize the sum of the squares of the residuals. Ridge regression adds an additional term that needs to be minimized. so when performing ridge regression one is minimizing the sum of the squares of the residuals as well as adding in a constraint on the sum of the squares of the regression coefficients. This second term, the sum of the squares of the regression coefficient is how the bias in introduced into the model.

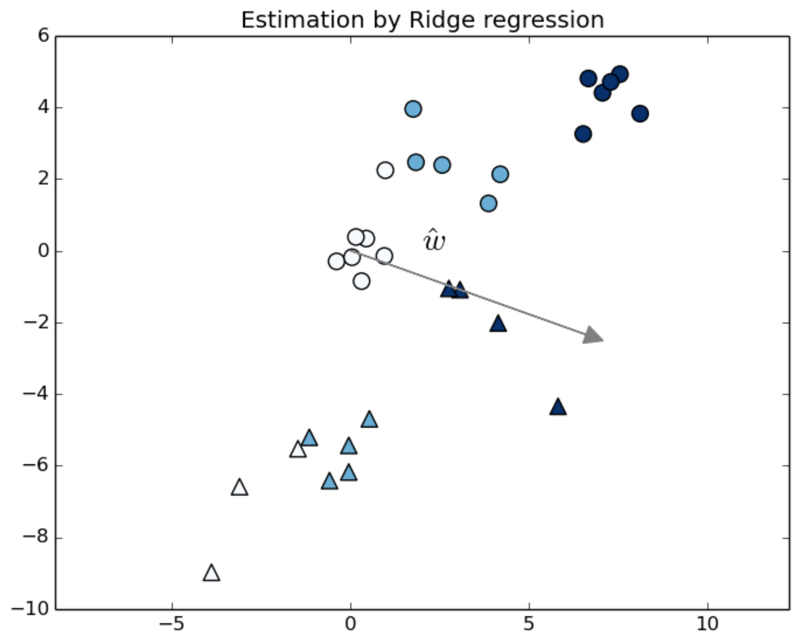

Figure 9: Estimation by ridge regression or linear regression

To access the quality of the model here, one metric that can be used for a small data set is the Kendell's tau correlation coefficient. It is defined as $\tau = \frac{P-Q}{P+Q}$, where P is the number of concordant points and Q is the number of discordant points. Any pair of points (x_i, y_i) and (x_j, y_j) where $i \neq j$ are said to be concordant if the rank of both x and y agree, i.e. if $x_i > x_j$ and $y_i > y_j$ or vice-versa then they are concordant. For all other cases when the order differs they are discordant. If the ranks are equal in both cases then that is not counted either as concordant or discordant.

For ridge regression for block 0 i.e. triangular component in the graph τ is 0.71122 and for the block 1 i.e. circular component τ is 0.84387.

This is needed for the comparison with RankSVM which is a pairwise transformation of the above model. As mentioned earlier pairwise transformation is done in the said manner $(x'_k, y'_k) = \left(x_i - x_j, sign\left(y_i - y_j\right)\right)$ for all comparable pairs. If the targets are same or of the different block then that pairing is skipped. The plot shows the same. The color is the magnitude of difference in the y part of the pair, i.e. the rank.

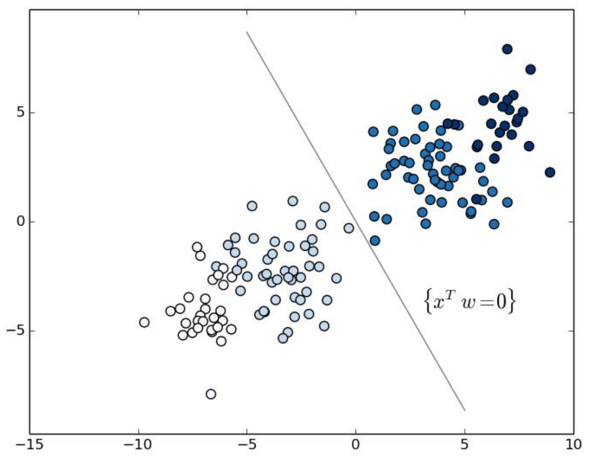

Figure 10: Linearly separable in terms of ranks and color showing the difference in rank

It can be easily seen from the graph that it is a linearly separable. It is possible as there are no order inversions in the in the training data. Thus a linear classifier is used. The graph below would be better in terms of determining the difference in ranks.

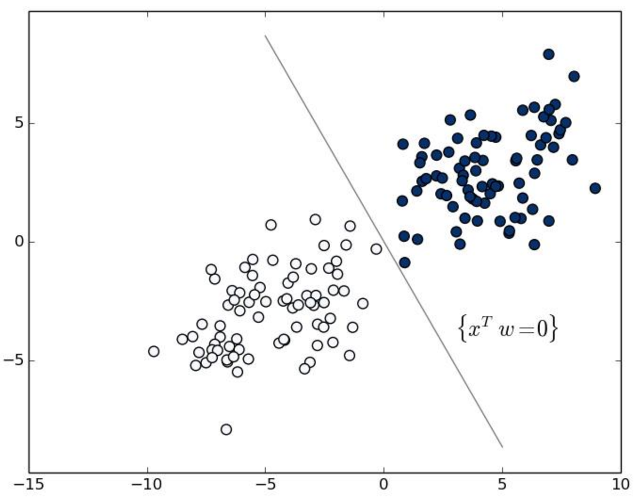

Figure 11: Linearly separable in terms of ranks and color showing the different signs

One side of the graph is positive and the other side is negative. This is linearly separable. Thus a linear classifier of SVM would suffice. Any other might overfit the data. The weight vector was got from that and plotted to show the weight vector.

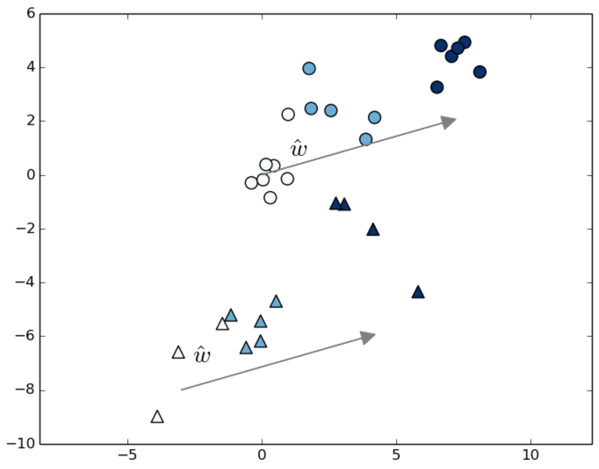

Figure 12: Linearly separable data on SVM with the weight vector

The ranking coefficient using Kendell's tau was calculated which gave 0.83627 for block 0 and 0.84387 for block 1. It is seen that block 1 was indifferent to linear regression and rankSVM, but there was significant improvement in terms of block 0.

For the other algorithms, the data created above would be the base of it all.

First of all is the RankNet. The figure below shows the neural network used for the toy dataset. For the original one the network was similar but more complex and more deep analysis is needed to fully understand it.

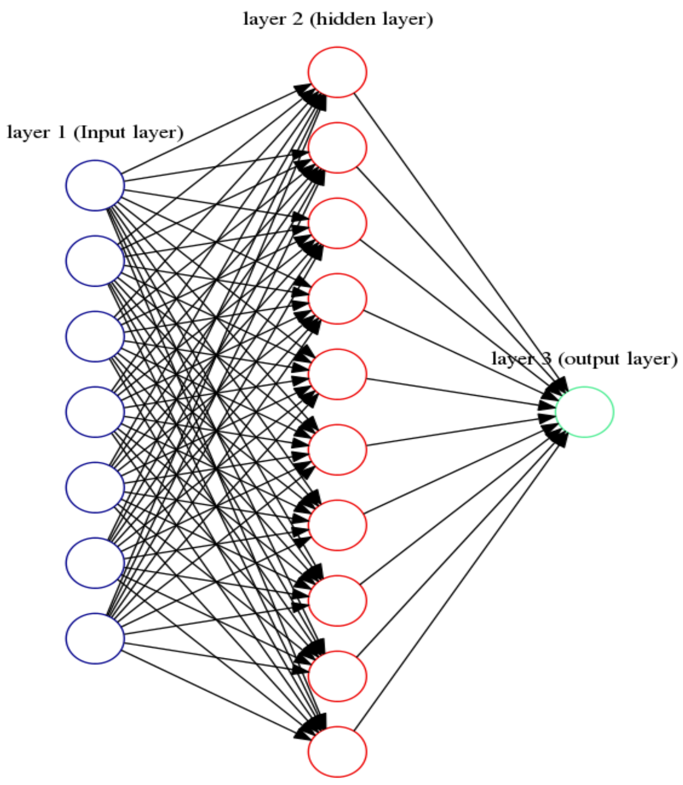

Figure 13: Model used for RankNet

In the above toy dataset, the weights that were formed from the input layer to hidden layer were

-0.03914	-0.05816	-0.0958	-0.0598	0.074734	-0.0495	-0.0077	0.08940	0.06486	0.004
-0.00797	0.06131	0.0159	0.0216	-0.0494	0.0265	-0.0355	0.09212	0.05001	0.0032
0.032212	0.067862	-0.0133	0.0185	-0.07537	-0.0843	0.04234	-0.0292	-0.0486	0.03719
0.023144	0.06066	-0.0329	0.01316	-0.08631	0.04921	0.07677	-0.0387	0.07798	0.05136
-0.04638	0.0973	0.04872	0.02543	-1.93E-4	-0.0989	-0.0183	0.03203	-0.0976	-0.0628
-0.08495	-0.03636	-0.0363	0.0854	0.02553	0.01183	0.0094	0.02364	-0.0781	0.06168
0.0157	-0.0074	0.0235	0.002	0.0684	0.03007	0.04464	-0.0326	-0.0628	0.04857

The weights for the hidden layer to output layer were:-

-0.0875	0.0153	0.0039	-0.0771	0.0609	0.0031	-0.00611	0.045747	-0.02138	0.09163

The NDCG obtained from the test case was 0.7320.

When linear regression was applied to the same dataset 7 weights were obtained. The additional weight is nothing but the bias.

0.3	0.3	260.417	-14069.7	0.0991	102.5418	-2.51399

The NDCG obtained from the test case was 0.6183.

While applying LambdaMART, 10 trees with binary separation was obtained.

Tree no	Feature	Threshold	Left	Right	Weight
1	6	3	-0.4060202	2.0	0.1
2	6	3	-0.35845292	1.7861544	0.1
3	6	3	-0.28415188	1.5218648	0.1
4	6	3	-0.3184343	1.6344081	0.1
5	6	3	-0.2543091	1.4356358	0.1
6	5	0.04	-0.46357542	1.3221567	0.1
7	5	0.04	-0.77627546	1.2945518	0.1
8	5	0.04	-0.90781724	1.4976496	0.1
9	5	0.04	-0.67229265	1.1416975	0.1
10	5	0.04	-0.58660144	1.0191938	0.1

The NDCG obtained for the test case was 0.9469.

It can be very well seen that the NDCG score and ranking is highly appropriate in LambdaMART. However on increasing the number of hidden layers and number of neurons in the hidden layer, better NDCG was obtained. Same goes for LambdaMART. However, if the learning rate of linear regression was altered, no change was seen. This indicated that linear regression is meant and made only for regression. It doesn't work well as a classifier. So it can be understood that to be a good ranker a model needs to be a good regressor and also a good classifier. The fundamental concept of lambdaMART is derived from random forest and decision trees. Since they are excellent classifiers, the regressor part is derived from MART which is a modified decision tree.

References

1. R. Herbrich, T. Graepel , et al. Support Vector Learning for Ordinal Regression, ICANN1999.
2. T. Joachims, Optimizing Search Engines Using Clickthrough Data, KDD 2002.
3. Y. Freund, R. Iyer, et al. An Efficient Boosting Algorithm for Combining Preferences, JMLR 2003.
4. R. Nallapati, Discriminative model for information retrieval, SIGIR 2004.

5. J. Gao, H. Qi, et al. Linear discriminant model for information retrieval, SIGIR 2005.
6. C.J.C. Burges, T. Shaked , et al. Learning to Rank using Gradient Descent, ICML 2005.
7. Tsochantaridis , T. Joachims, et al. Large Margin Methods for Structured and Interdependent Output Variables, JMLR, 2005.
8. F. Radlinski, T. Joachims, Query Chains: Learning to Rank from Implicit Feedback, KDD 2005.
9. I. Matveeva, C.J.C. Burges, et al. High accuracy retrieval with multiple nested ranker, SIGIR 2006.
10. C.J.C. Burges, R. Ragno, et al. Learning to Rank with Nonsmooth Cost Functions , NIPS 2006
11. Y. Cao, J. Xu, et al. Adapting Ranking SVM to Information Retrieval, SIGIR 2006.
12. Z. Cao, T. Qin, et al. Learning to Rank: From Pairwise to Listwise Approach, ICML 2007.
13. Chen, Kai and Li, Rongchun and Dou, Yong and Liang, Zhengfa and Lv, Qi, Ranking Support Vector Machine with Kernel Approximation, Computational Intelligence & Neuroscience, 2017
14. Rahimi, Ali, and Benjamin Recht. "Random features for large-scale kernel machines." Advances in neural information processing systems. 2008.