Personalized E-commerce Search

A thesis submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

Electrical Engineering

by

Arnav Kansal

Entry No. 2013EE10440

Under the guidance of

Prof. Jayadeva

Department of Electrical Engineering,
Indian Institute of Technology Delhi.
April 2017.

Certificate

This is to certify that the thesis titled **Personalised E-commerce Search** being submitted by **Arnav Kansal** for the award of **Bachelor of Technology** in **Electrical Engineering** is a record of bona fide work carried out by him under my guidance and supervision at the **Department of Electrical Engineering**. The work presented in this thesis has not been submitted elsewhere either in part or full, for the award of any other degree or diploma.

Dr. Jayadeva
Professor, Department of Electrical Engineering
Indian Institute of Technology, Delhi

Abstract

In the past years there has been an explosive growth in the E-commerce sector. Improving user experience has a direct impact on the probability of the user returning to the same website. This B.Tech project aims at using machine learning methods to improve the Personalization in the E-commerce Search space by re-ranking the search results shown to the users. Various techniques have been explored such as using Classifier to Re-rank, Collaborative filtering and also Learning to Rank algorithms(LTR). Linear feature models belonging to the LTR class outperformed all other methods. Many search strategies have been used to gain speed up and ensure global maximum of the evaluation metric while learning the linear feature model. A regularization framework was introduced in the linear feature model. Finally a neural network architecture was tried to compare with the linear feature model. Also the function profile of the evaluation metric for the linear feature model was plotted by projecting the profiled points on 2D to give more insight about the metric.

Acknowledgments

I would like to thank all the people who helped me with this project. I acknowledge the support, encouragement and invaluable feedback given by my supervisor **Dr. Jayadeva**, Professor, Electrical Engineering Department during the project. I was continuously motivated by him and am extremely lucky to have a supervisor who cared so much for my work. I would also like to thank Mayank Sharma, Dr. Jayadeva's PhD student for his constant support. Further, I would like to express my gratitude towards the evaluation committee for their gainful suggestions.

Arnav Kansal

Contents

Chapter 1

Introduction

1.1 Introduction

Search engines used in the e-commerce space differ(benefit) over the usual search engines because of the availability of transactional data. In e-commerce websites users also show unique search patterns, based on various factors such as their likes, needs, purchasing power, etc. These unique patterns of the users can be learned and applied to re-rank the basic(unbiased) search results corresponding to a query.

This B.Tech project aims at using machine learning methods to :

- Improve the Personalization in the E-commerce Search space
- By re-ranking the search results shown to the users.

1.1.1 Personalization

Personalization is basically targeted marketing. Users of an e-commerce platform are unique, and have varied interests. The concept of an unbiased search does not completely fulfill the need of the users, unlike usual search engines. Thus targeted marketing results in enhanced user experience. To elucidate upon this, assume that there are 2 different users X and Y. One of them has recently purchased a smart phone, and the other one bought a bed. Now both of them enter a search query *cover*. It makes sense to show X mobile covers, and Y bed covers.

(a) User X shown mobile covers

(b) User Y shown bed covers

Figure 1.1: Personalized Results

1.2 Data set

The dataset used was given by *DIGINETICA* for CIKM Cup 2016 Track 2 [4]. This data has anonymized search and browsing logs, product data and anonymized transactions records. The anonymized data hides a lot of information for the purpose of this study, but after the AOL search leak that happened in 2005, it has been impossible to get real data that has not been anonymized. Further, the data is present in raw form, i.e. log files. To describe the transactions and product data, a total of 6 files have been provided in the data, which correspond to Queries and their corresponding search results, product meta data, click/view and purchase data. The data has been partitioned already in the train test split.

Train Queries	636,160
Test Queries	286,967

Table 1.1: Train-test split

In the data for a given query, a list of ranked results produced by the underlying(unbiased) search engine of the e-commerce is given. This list is generated based on only the relevance of product for a given query, and does not take into account any user preference. So our task is to incorporate user preference in this list and produce a new, re ranked list.

1.3 Mathematical Formulation

Given a set of queries Q, each query $q \in Q$ is associated with a user u issuing the query on the search engine. The search engine returns to the user u, a list of items l_q in answer to the query q.
The task at hand is to re rank items within l_q to provide optimum ranking results. This optimality is with respect to a ranking measure which takes an ordered list as an input and returns a scalar which depicts the extent of correct positioning of items according to relevance with respect to the query. For our problem lets assume we have a training and a test data. The training data must have a set of queries Q, with each query having a list of items associated with it along with meta information regarding the user and the items present in the list. Each item i in the list l_q corresponding to the query q, must have a scalar relevance attached to it. The task is then to reorder the lists of items present in the test set.

1.4 Evaluation Metric

Normalized Discounted Cumulative Gain(NDCG), the standard evaluation metric employed in any Information retrieval task is used here for calculating the accuracy of the re ranked list. NDCG is a function of order of items of a list and is determined by the positioning of items defined by their relevance. Relevance is a variable associated with each item of a list which may be known for the training dataset before hand. The Discounted cumulative gain is calculated for a given list of items corresponding to a query as follows.

$$DCG_{query} = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{1.1}$$

Here p denotes the number of products for the query.

$$rel_i = \begin{cases} 0, & \text{if product is never clicked} \\ 1, & \text{if product is shown to user in search and clicked} \\ 2, & \text{if product is shown, clicked and purchased by user} \end{cases} \tag{1.2}$$

This basically says that more important items should be placed in front of the list as items further away in the list will be penalized with a decaying factor of the index in the list at which the item is finally placed. The NDCG however, is calculated by dividing the DCG of a query with the maximum possible value of DCG which can be attained by shuffling the order of the current list. This definition of NDCG is valid for one list corresponding to a query in the dataset. The total metric is calculated by averaging this over all queries.

Chapter 2

Classifier Based Re-Ranking

2.1 Idea

Aim of the problem is to achieve a maximum NDCG score, on the final presented ranking by predicting correctly the relevance and re-ranking based on it. So, the following approach was tried at first.

- Finding a classifier to predict relevance of an item, given (query,user).
- and to use classifier outputs to Re-rank items for each query.

2.2 Feature Extraction from Raw Data

The first step would be to find relevant features to describe query-user-item tuple. Let the list of items associated with query q be l_q. For each item in this items list l, a query,item pair is chosen and some fixed set of features are extracted for the pair. So the query,item pair is now denoted by a feature vector X. The computed features can be broadly classified into these five categories.
User features: which describe the viewing and purchasing history of the user.
Query features: which contain the ranking measures of the list returned by the unbiased search engine, such as the initial value of NDCG of the list.
Item features: these describe the item transaction history, along with its meta-data. These also contain the descriptive statistics of ranking achieved by the item in all other searches.
Query-Item features: this contains a similarity measure (Jaccard Coefficient) of the query string and product meta-data.
Session features: These contain the average session duration, and average

amount of queries issued in the session.
The details of the extraction process from raw log files is given in the feature extraction appendix. A

2.3 Re-Ranking using Classifier

The feature extraction phase would be followed by training the classifier. This classifier should meet some special needs, such as the ability to output class probabilities instead of output label directly, as explained further. The final part would be to somehow use the classifier outputs to re rank the search results.
One option is to directly rank the items based on the output target, the relevance. But in our case the list of items per query is large, and the output relevance belongs to only three classes. So much of the items will get clustered according to their class outputs. And finding the correct order between elements of same class will not be done.
This problem has been solved using methods employed in [10]. Consider that the classifier trained for predicting relevance, outputs probabilities for different classes. So given the feature vector which describes the (query,session,user,item) tuple, the classifier gives out a probability of confidence for each of the relevance class, namely {0,1,2}. Now the aim of the problem is to maximize the NDCG metric.
Consider the NDCG metric, as given in (1.1), which is a monotonic decreasing function of i, which is the index in the list. The item score is then calculated by taking into account the probability of an item belonging to each the of three classes. So a scoring function is used as given by [12] which maximizes the expected value of the NDCG metric given the tuple described above.

$$E[\sum_{i=1}^{p}\frac{2^{rel_i}-1}{log_2(i+1)}] = \sum\frac{E[2^{rel_i}-1]}{log_2(i+1)} \tag{2.1}$$

As larger relevance labels must appear in the front of the list, and the denominator is decaying as i increases, the scoring function is chosen as

$$\text{score(item,(query,user,session))} = E[\sum_{i=1}^{p} 2^{rel_i} - 1] \tag{2.2}$$

which by the linearity of expectation breaks into

$$\text{score(item,(query,user,session))} = \sum_{i=1}^{p}[3 \times \text{prob(relevance(item)} = 1)+ \\ \text{prob(relevance(item)} = 2)] \tag{2.3}$$

It is shown that even though DCG errors are bounded by classification errors [10], the bound is only upper. Finally sorting items given by their score as calculated above will give us a ranking which tries to maximise NDCG.

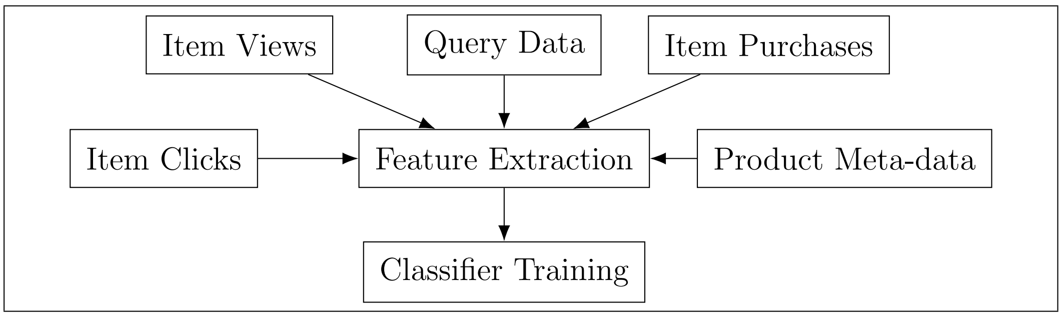


Figure 2.1: Training Pipeline

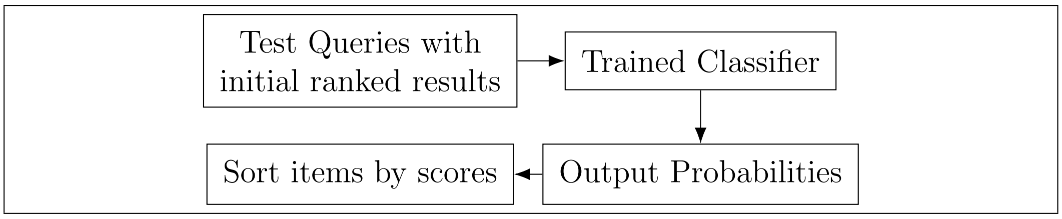


Figure 2.2: Testing Pipeline

2.4 Results

Classifier	Accuracy[Hold out set]	NDCG	Improvement (baseline)
Nearest Neighbors	95.7	0.442	15.7%
Linear SVM(SVC)*	91.7	0.467	22.25%
Decision Tree*	96.22	0.438	14.66%
Random Forest	93.4	0.449	17.53%
AdaBoost	93	0.388	1.56%
Logistic Regression	91.9	0.468	22.5%

Table 2.1: Classifier Re-ranking results

*Platt's scaling [14] for SVM and probability estimate for Decision Tree used for determining output probability.
Multiple classifiers which do have a measure of probability for class outputs were trained. As the training size of the data is close to 1.5M samples after performing preprocessing steps as shown in A. At first linear models were tried which started to give decent results compared to the baseline results given by the data provider. The classification task was performed using scikit library [13]. The baseline provided by the data provider is simply sorting based on score calculated as:

$$\text{score(item)} = \text{views(item)} + 2 \times \text{clicks(item)} + 3 \times \text{purchase(item)} \tag{2.4}$$

The baseline score given is NDCG: 0.382.

As is observed from the results, using this naive approach gives us significant improvements over the baseline. Out of the various classifiers, the logistic regression classifier gives us the best results for the computed features.

Chapter 3

Collaborative Filtering

The relevance prediction approach was followed by the standard techniques of recommender systems. Recommender systems are, as the name suggest, systems that recommend user items that match the users interests, and also the items that 'similar' users viewed or purchased.
So essentially, it takes into account not only the user's specific interests, but also the interests of users who have shown matching viewing or purchase patterns in our case.
The most widely used technique in recommender systems is Collaborative filtering(CF). The matrix factorization technique which belongs to the class of latent factor models, which became hugely popular after the Netflix Prize [1] was used for this purpose.

3.1 Matrix Factorization

Matrix Factorization is a latent factor based formulation of a collaborative filtering model. In this model a user item preference matrix is first designed. So for each (user,item) pair (X,y), there exists a preference rating for a user X to an item y. Let us call this matrix M. If the space of users is spanned by the vector U, and items by I then,

$$M : U \times I \to R \tag{3.1}$$

This matrix can comprise of explicit ratings in the form of like or dislike, like in the case of comments or posts on facebook, or these can be implicit, like in our case where there is no explicit ratings present. This implicit feedback in our case can be extracted from the provided transactional data, such as the click, view and purchase data. For example each view, click or purchase of an item y, by the user X, could contribute partially to the $M_{X,y}$. In the

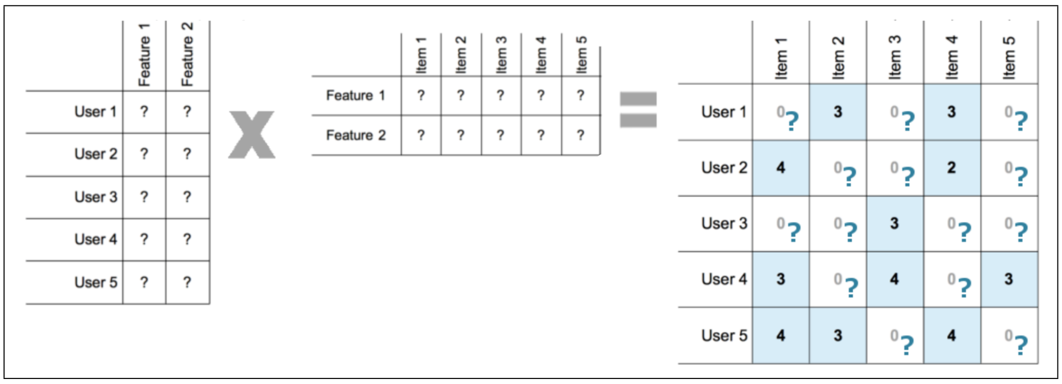


Figure 3.1: Matrix Factorization

classical matrix factorization approach as discussed in [9], users are mapped to a latent factor space of a fixed dimension. Items are also mapped to the same space. Now users and items can be compared as they are vectors of the same dimension in the same latent factor space.

Similarity between user X, and item y in this space is given by the dot product of the two transformed vectors. After transforming each user X to vector U_X, and each item to the vector I_y, the computed dot product is then said to be the original rating $M_{X,y}$.

$$M_{X,y} = U_X^T I_y \tag{3.2}$$

In this preference matrix, most of the entries are unknown, because of the nature of the transaction data. A user is very likely to have interacted with very few of the items present in the universe of items. Also the fact that a user did not buy/click an item also points to the fact that the user might actually, not like the product.
So the general methods of Singular Value Decomposition which is solved by stochastic gradient descent as done by [5] are not applicable in this case. And special methods as proposed by [6] for implicit feedback data sets has to be employed.

3.1.1 Alternating Least Squares

In the method talked above, there is a fundamental difference from the standard SVD formulation is the loss function. Comparing to the

$$\min(\sum_{X,y}((M_{X,y} - U_X^T I_y)^2 + \lambda|U_X|^2 + \mu|I_y|^2 \tag{3.3}$$

we now have our loss function as

$$\min(\sum_{X,y}(C_{X,y}(P_{X,y} - U_X^T I_y)^2 + \lambda|U_X|^2 + \mu|I_y|^2 \tag{3.4}$$

Here P is a matrix of one's and zeros with one's at positions where U_X has interacted with I_y. C is the matrix which denotes the confidence, that is in our case, the number of purchases/clicks of item I_y by user U_X. When any of U_X or I_y is fixed, the problem can be solved by quadratic optimization. So alternating least squares proceeds by fixing one U_X, finds the corresponding I_y. This is followed by fixing the new found I_y and finding the next U. This process is carried till convergence.

3.1.2 Data Preprocessing

For the case of our data, there are 141,127 unique users, and 81,160 items who have had interaction with the users in the form of clicks, views or purchases. The resulting matrix is a huge, and has potentially 10B entries. The interesting fact is that for our case, only about 468,752 of these entries are known, and the rest are unknown. So we are dealing with a matrix which has sparsity of about 99.995%.The entries are filled with the following weights.

$$C_{X,y} = \alpha_1 \times \text{num˙purchases} + \alpha_2 \times \text{num˙clicks} + \alpha_3 \times \text{num˙views}$$

3.1.3 Results

The values of α_i's were found by doing a grid search over a reduced space of [1,3,30,300,3000] evaluating the model's NDCG for each case. The maximum

value of NDCG was found to be 0.384 with the α values being [30,3,1].
The model here managed to perform just better than the baseline score, and did not perform as well as the classifier approach basically because of the absence of query information, and other product meta data, from the model.The above model suffered from the absence of extra available information of the given problem and modeled only the user-item interactions. As shown in [8], it is almost always better to use contextual information about the user-item pairs. In our case, the context might mean a lot of things, such as the given query, the ordering of the initial ranked list, and other settings such as the session characteristics. So other methods like like the Learning to Rank methods which employ underlying supervised learning algorithms were deployed next.

Chapter 4

Learning to Rank

Finally after trying the classical machine learning approach, and having gone over with the collaborative filtering techniques, the Learning to rank methodology was adopted.
Learning to Rank is a general scheme in which we construct a machine learning approach to construct a model which outputs ranks. So what we had done explicitly in the naive model in a two stage process by first finding a classifier and then using its outputs to re rank, hinging on the fact that the errors in NDCG would be bounded by the errors in classification, is somewhat taken care of intrinsically by the model itself.
In this methodology, the dataset is first divided into parts based on the specific query. Then for that query, and the list of items that are presented along with that query, meaningful features are used to learn the ranking of these items given the query and the extracted features. A loss function is then created by taking three different techniques. All of these three techniques have been discussed and have been put to task for our problem. Further RankLib [3] was used for performing the learning task.

4.1 Pointwise approaches

In this approach, the cost function is constructed in much the same way as any other regression, or classification formulation. The dataset which is divided into parts by query is taken, and for each query,item pair a relevance is learned. The final list of items is just sorted based on this relevance.
Notice that the approach used initially at the problem was much the same, only that instead of letting the model take care of the scoring, we had isolated the model from the scoring stage and each of the step was performed independently.

As not much has evolved of the point wise approaches for ranking, as it is very similar to basic regression, we use the Random Forest Ranking approach.

4.2 Pairwise approaches

In the pairwise approach, the cost function is altered to support taking all pairs of items from the list of returned items, given a query.

4.2.1 Ranknet

The underlying machine learning tool in the ranknet [2] procedure is a neural network. Now this neural network takes in a feature vector, and its output is called the score of the feature. This score is calculated for two items, is used in the cross entropy loss, where the cost function is probabilistic. This function converts the difference of outputs for two items (i,j) which is $\delta_{i,j}$ to a probability as follows.

$$\text{P(item-i comes before item-j in ranking)} = \text{logit}(\delta_{i,j}) \tag{4.1}$$

here $\delta_{i,j}$ is given by the difference of output of features given by item i, and item j given a query to the neural network. The cross entropy loss is given by:

$$C = -P'_{i,j}log(P_{i,j}) - (1 - P'_{i,j})log(1 - P_{i,j}) \tag{4.2}$$

where $P'_{i,j}$ is the probability calculated by the model, and $P_{i,j}$ is the true training value. After calculating the cross entropy loss using this function, the model training is quite similar to the regular training procedure of a neural network, in the sense that the forward propagation is carried out in the same way for both the samples with itemi and itemj, the backpropagation step differs a bit however. The cost is calculated for both samples, but the gradient of the cost sent for backpropagation is the difference of gradients for both samples. So finally, the weights are updated as follows as shown in [2]:

$$w_k \rightarrow w_k - \eta(\lambda_{i,j}(\frac{\partial out_i}{\partial w_k}) - \frac{\partial out_j}{\partial w_k})) \tag{4.3}$$

Where out_k is the output of neural network with feature corresponding to item k, and $\lambda_{i,j}$ is given by $\frac{\partial C(out_i - out_j)}{out_i}$ For our case one hidden layer was chosen with number of nodes in the layer equal to the number of features. As this training procedure was taking very long time to tune, the number of hidden layers or the number of nodes was not fine tuned. The unavailability of caffe like frameworks which could be heavily parallelized on GPU was limiting this model and the bare structure coded on java in Ranklib was not taken any further.

4.3 Listwise path

This is a more holistic answer to the given problem of ranking and indeed the results achieved with this technique speak for themselves. In this technique basically the loss is constructed for the entire list of items given the query.

4.3.1 ListNet

ListNet [16] is a model built up on RankNet itself. There are subtle differences however. Here the complete list of items for a query is given to the model as input. The probabilistic cost function now used is analogous to multi class logistic loss, where ranknet had its similar to 2 class logistic loss. Now the probabilistic cost function is defined for a list l which has n items and the scores of the neural network for each item in the list denoted by s is:

$$P_{s,l} = \prod_{i=1}^{n} \frac{e^{s_i}}{\sum_j e^{s_j}} \tag{4.4}$$

Also the new loss function defined is over all lists, corresponding to all queries. The loss for a particular query which has a list of items l, with neural net output scores s and actual labels z is given by:

$$L(s, z) = - \sum_{l' \in \text{perms(l)}} P_{z,l'} log(P_{s,l'}) \tag{4.5}$$

where perms(l) is all permutations of list l. The training procedure is also modified in a similar way in the sense that now gradient of cost used to update weights of the neural network are found by calculating cross entropy loss for the entire list of items. The model suffered from the same problem as Ranknet in terms of time taken for training the model.

4.3.2 Linear Feature Model

Lastly, we try a linear feature based model for ranking [11]. In this approach the model directly tries to optimize for the maximum value of the evaluation metric for the information retrieval task over the entire list. It tries to do so by assuming a scoring function which is linear in the features. The model is a linear feature model with weight vector associated with it **w**. The dimension of **w** is that of the constructed feature vector **X**. For every item i in a list l, the score of item i is calculated as taking the $\mathbf{w}^T\mathbf{X}$, with the the feature vector associated with the item i for that list. The next part of the algorithm simply reorders the items of the list according to this score.

Training the model

The task at hand is to produce the best ranking of items in the list according to an evaluation metric. Now our model must be such that maximum NDCG is achieved over the testing data. Scoring function is linear in the features.

$$\text{Score}(\mathbf{X}) = \mathbf{w}^T\mathbf{X} \tag{4.6}$$

w is found by maximizing the NDCG directly.

$$\mathbf{w} = \arg\max_{w'} \sum_{\text{train data}} \text{NDCG}(l_{w'}) \tag{4.7}$$

Where l_w is permutation of list l achieved by sorting l according to scores of items i in l by ***w****.* The following strategy is chosen for finding the best w.

Coordinate Ascent

The objective function is,

$$max(\sum_{\text{train data}} \text{NDCG}(l_{w'})) \tag{4.8}$$

This optimization process is completed with coordinate ascent, which basically is a local search procedure, in which all parameters except one in w are fixed, and it is varied to find maximum. This process is repeated multiple times, till convergence.

In the model it is assumed that the weight vector is non negative. This assumption is based on the fact that the features chosen are only positively contributing the relevance.

This is a very strong constraint and actually reduces the search space to the k-simplex where the feature is k+1 dimensional as shown in [11]. This constraint may be relaxed when using other local search procedures.

Some key points to note for the model are:

- Two models are equivalent if they produce the same ranking.
- Two models will guarantee the same ranking if they are scaled versions of each other. i.e. given **w1** and **w2**, **w1** = scaled(**w2**).
- Thus after every iteration of the optimization procedure **w** is normalized such that the optimization in one step is in R^d but overall the search space is contained in the d-simplex.

Given below is the pseudo code of the search strategy employed.

Algorithm 1 Coordinate Ascent

```
function CA($X$,num-iter,tol, num-restart)▷ Where X - data matrix obtained after feature extraction, dim - (no. queries* no. items) x feature dimension
    bestModelscore = 0
    for $k = 1$ to num-restart do
        w = $size(w)^{-1}$ * vector of ones of length feature dimension
        signs = -1,0,1
        startscore = scorer(w,X,Q)
        consecutive-fails = 0
        while consecutive-fails < len(w)-1 do
            Shufflefeats(w)
            for $j = 1$ to feat dimension do
                origwt = w[j]
                for $k = 1$ to len(signs) do
                    step = stepbase * w[j]
                    for $l = 1$ to num-iter do
                        w[j] = w[j] + signs[k]*step
                        if l < num-iter-1 then
                            step = step * stepScale
                        end if
                        score = scorer(w,X,Q)
                        if regularize then
                            score = score - penalty(w)
                        end if
                        if bestscore > score then
                            bestscore = score
                            succeeds = true
                        end if
                    end for
                    if succeeds then
                        break
                    else
                        w[j] = oldwt
                    end if
                end for
                if succeeds then
                    normalize(w)
                    bestWeight = w
                end if
            end for
        end while
    end for
end function
```

4.4 Results

The results of the above discussed learning to rank methods have been compiled as follows.

LTR Method	type	NDCG	Improvement(baseline)
RandomForest	PointWise	0.460542	20.4%
RankNet	Pairwise	0.349715	-8.6%
ListNet	ListWise	0.418020	9.42%
Coordinate Ascent	ListWise	0.480967	25.65%

Table 4.1: L2R results

Here on an average we were able to achieve better results than our initial methods, except one case of Ranknet. However we can observe, that from Ranknet to Listnet, i.e changing the formulation slightly showed drastic improvements in model quality. Also here again, linear models have shown best results.

4.5 Improving the Linear Feature Model solver

Finding **w** using Coordinate Ascent is slow and may not guarantee a global maximum. Even though the latter problem may be resolved partially by performing the search using different random restarts and also shuffling the set of weights randomly after every pass, there still is scope of improvement both in the time taken to reach a maxima and also the trueness of it.

For resolving these two issues two other search strategies were used.

4.5.1 Other search strategies

- Multi Trajectory Local Search [MTSLS]

- Global Optimization Through a Support Vector Machine based Adaptive Multistart Strategy [GOSAM]

4.5.2 Multi Trajectory Local Search [MTSLS]

In solving the linear feature model using CA, it was assumed that the weight vector was non negative. To relax this constraint Multiple Trajectory Search for unconstrained optimization was used [15]. To put it simply, it is a local search strategy. It is much similar to CA in the sense that it tries to do a line search for one dimension at a time.

The basic key points of the procedure are given as follows:

- Works like CA by considering one weight at a time.
- At any point in the search space hops back a step or front half a step adaptively setting step.
- Faster than Coordinate Ascent*.

* Faster per iteration over length of features.

Comparing local Search strategies

MTSLS is not limited by the non negative weight constraint found in the CA formulation and also is advantageous in terms of speed.

- Each sub-iteration of CA involves 3 calls to external ranking function which goes over the entire data set.
- MTSLS cuts this to 2 calls in most of the cases.

MTSLS at every sub iteration only checks if objective is increasing by increasing or decreasing a weight, however in CA other than these two possibilities the case of dropping the feature of the particular dimension is also checked. This extra check has an extra call to the objective function which in turn is

Algorithm 2 MTSLS

```
1: function MTSLS($X$,num-iter,tol)    ▷ Where X - data matrix obtained after feature extraction, dim - (no. queries* no. items) x feature dimension
2:     w = random vector of length feature dimension
3:     oldscore = scorer(w,X,Q)
4:     step = MINSTEP
5:     improve = true
6:     for $i = 1$ to num-iter do
7:         if improve is false then
8:             step = step/2
9:             if step < MINSTEP then
10:                step = base1 + random(0,base2)
11:            end if
12:        end if
13:        for $j = 1$ to feat dimension do
14:            w[j] = w[j] - step
15:            afterscore = scorer(w,X,Q)
16:            if abs(afterscore-beforescore) < tol then
17:                w[j] = w[j] + step
18:            else
19:                w[j] = w[j] + 0.5 step
20:                afterscore = scorer(w,X,Q)
21:                if abs(afterscore-beforescore) < tol then
22:                    w[j] = w[j] - 0.5 step
23:                else
24:                    bestw = w
25:                end if
26:            end if
27:        end for
28:    end for
29: end function
```

Table 4.2: Speedup - Time comparison

Local Search Strategy	NDCG*	Time(300 iterations)
CA	0.480967	38m1.011s
MTSLS	0.47895	33m57.753s

defined over the entire dataset, so cutting one call every sub iteration results in a decent speedup.

* Max among 10 models obtained from different Random Restarts.

4.5.3 Global optimization to find Feature Weights

In MTSLS speedup has been gained from CA but still we are not very sure of the global maxima being attained by the local search procedures employed above. Thus Global Optimization Through a Support Vector Machine based Adaptive Multistart Strategy [GOSAM] [7] has been used. The main ideas behind GOSAM are that it:

- Tries to learn the structure of local maxima.
- Uses this information to find next start for Local Search.

Algorithm overview

1. For n points (xi), obtain local maxima f(xi) using Local Search and store in set S.
Do while termination criteria is met.

- Fit an SVR through S.
- Find maxima of the obtained SV regressor.
- Start local search from this new point.
- Add this point to S.

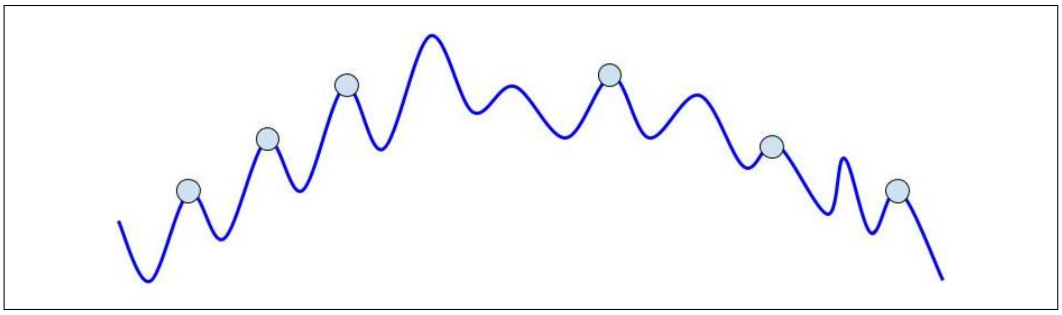


Figure 4.1: Illustrating the working of GOSAM - Function for global optimization. Initial local maxima found.

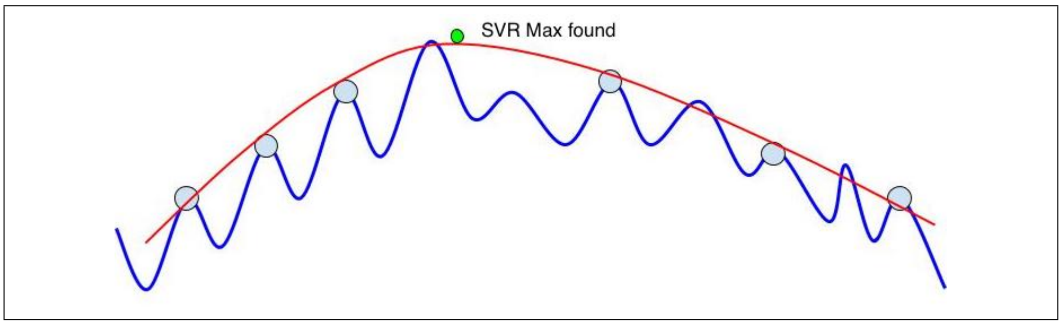


Figure 4.2: Fit SVR through the local maxima.

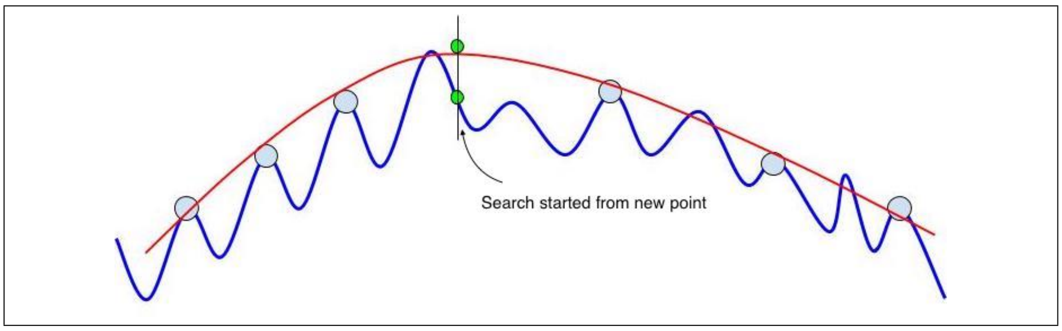


Figure 4.3: Maximum point of SVR found.

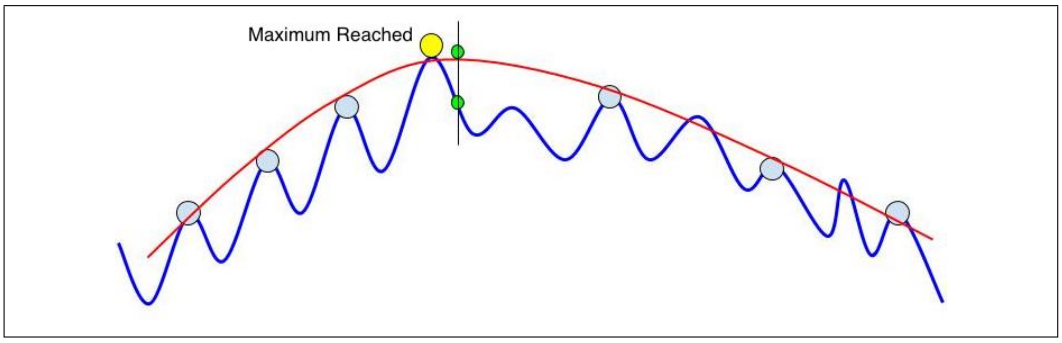


Figure 4.4: Starting local search from the SVR max.

GOSAM again requires to find a maximum point of the Support vector Regressor. This again is a non linear optimization problem, however the decision boundary of the svr is thankfully a linear combination of the support vectors and its derivative can be found. Thus vanilla gradient descent can be used to find its maximum.

Thus the decision boundary f(x) can be written in terms of the Lagrange multipliers (α_i^+ and α_i^-) and the kernel function.

$$f(x) = \sum_{1}^{L} (\alpha_i^+ - \alpha_i^-) K(x, x_i) + b$$

Correspondingly the derivative of the surface is defined as follows for the RBF Kernel case.

$$\nabla(f) = \sum_{1}^{L} -2\gamma(\alpha_i^+ - \alpha_i^-) K(x, x_i)\mathbf{x}$$

Table 4.3: GOSAM SVR Max strategy comparison

Strategy [SVR max]	NDCG
MTSLS	0.468524
GD	0.478758

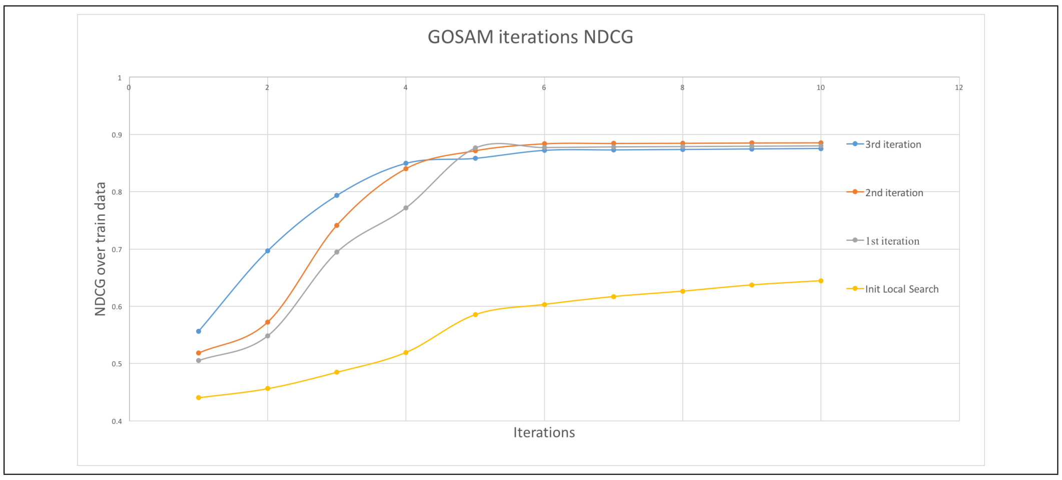


Figure 4.5: GOSAM over time

GOSAM solution gives better train score as solution closer* to global maximum is chosen. But still is a little behind the CA solution in the test scores. This may be some form of over fitting that seems to have crept in. Also the GOSAM procedure has too much of tunables as each iteration of involves fitting an SVR curve and svr fitting itself has so many parameters.

Algorithm 3 Gosam

```
1: function GOSAM($X$,num-iter,tol, num-restart, num-starts)  ▷ Where X - data matrix obtained after feature extraction, dim - (no. queries* no. items) x feature dimension
2:     for $i = 0$ to totPoints do
3:         YData[i],Xdata[i] = CA()          ▷ Initialise the set of X,Y Points
4:     end for
5:     k = totPoints
6:     while iteration < nMaxSvrIteration do
7:         regressor = SVMRegressor()
8:         newPt = SVRMax()
9:         YData[k],Xdata[k] = CA(newPt)
10:        k = k+1
11:        iteration = iteration + 1
12:    end while
13:    maxscore = 0
14:    for $i = 0$ to k do
15:        if Ydata[i] > maxscore then
16:            bestwt = XData[i]
17:            maxscore = YData[i]
18:        end if
19:    end for
20: end function
```

4.6 Regularizing the linear feature model

Finding the weights using the GOSAM procedure gave us a hint of overfitting and thus over and above the n-fold cross validation that was being regularization is adopted. The question was how to introduce regularization in this framework to ensure that we are not overfitting? A new method for regularization has been tried. It is done in the following way. At the stage where the evaluation metric is computed for a model state, this score is penalized by some form of regularizer. Many forms of regularizer are used and tested in this setting are described below: Old objective to be maximized was

$$\sum_{q \in Q} \text{NDCG}(l_w)$$

Where l_w is permutation of list l achieved by sorting l according to scores of items i in l by **w**.

This score was earlier maximized using CA, MTLSLS and also GOSAM procedures. Now for regularized case the objective would become:

$$(\sum_{q \in Q} \text{NDCG}(l_w)) - \alpha ||w||_k$$

where k could be 1 for L1 and 2 for L2 norm.

Other than these standard norm regularizer a different strategy was used for the regularizer.

Modified Regularizer

$$(\sum_{q \in Q} \text{NDCG}(l_w)) - \alpha ||w - reg||_k$$

Where **reg** = 1/numfeats*[1 1 ... 1] Here for k=1,2, the regularizer is called modified L1,L2. Again these objectives were maximized by using different solvers and the results are compiled as follows.

Table 4.4: Comparison between different Regularizers

Model	Solver	NDCG (5Fold CV)	NDCG(Max)
Unregularized	CA	-	0.480967
L1	MTSLS	0.4731 +- 0.0034	0.4774
L2	MTSLS	0.4744 +- 0.0027	0.4784
L2	CA	0.4837 +- 0.0027	0.4844
L1	Gosam	0.4689 +- 0.0059	0.4739
L2	Gosam	0.4759 +- 0.0053	0.4821

Motivation behind modifying version of the regularizer

- In the local search procedure the search was getting initiated from reg.
- The idea was to restrict going to far away from this initial solution.

- Also viewing from the model complexity the simplest model would be the normal vector which would simply score the data point as the sum of its features. Thus this regularizer can be viewed as the model closest to the simplest model which is w = 1/numfeats *[1 1 1 ... 1] by distance.

Modified L1/L2 scores

Table 4.5: Modified Regularizers results

Solver	NDCG(5FoldCV)	NDCG(Test)
CA L1	0.4804 +- 0.006	0.486541
CA L2	0.4826 +- 0.003	0.484879
Gosam L2	0.4727 +- 0.007	0.485773

4.6.1 Viewing the function profile

After having tried many optimization techniques to solve for the maximum, plotting the surface of the objective was attempted. For this purpose primarily two schemes were used.

- Projecting the input data to 2d and finding the function profile.
- Finding the function profile in the original feature dimension and then reducing its dimensionality to 2d.

The former method did not perform so well and in fact the maximum NDCG attained by it was lower than the baseline. This means that the first two principle components of the data were not enough for the task.
The second method yielded some interesting plots. The trend line plotted in pink color shows the trajectory taken by the search in the direction that causes the maximum of the objective. This 2d line was back projected to the original feature space. The vector obtained looked like the following. The 19th feature id is in fact the one causing most change in the NDCG, and was also verified by logging the search again by noting the change incurred in the NDCG by changing the feature weight. The 19th feature represents the median price. The median price is the median price of all items for a query.

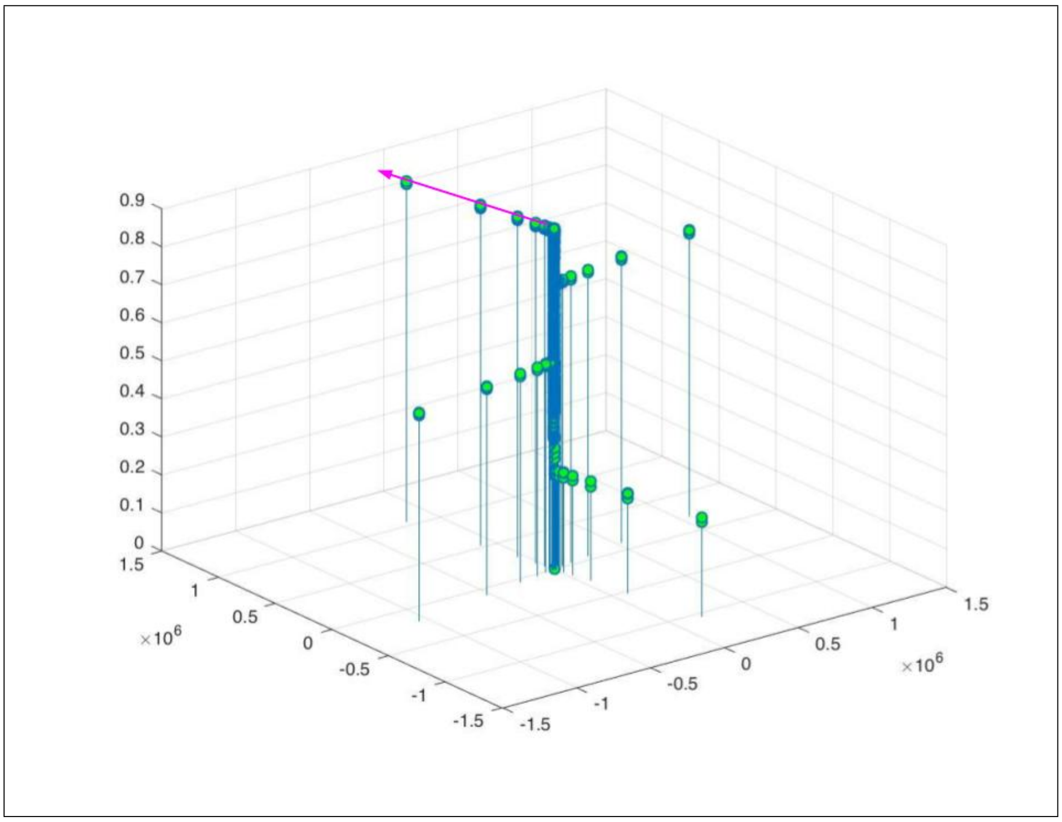


Figure 4.6: Function profile stem plot with maximum increase line

4.7 Adding Non-Linear Features

In the context of directly optimizing for the NDCG the linear feature models have been showing the best results. Further non linear features should be included. In a direct optimization setting a neural network structure was made with one hidden layer and the same number of nodes in it as the input layer. To actually add non linearity, the hidden layer nodes had a RELU activation. The output from the final layer was used as the score of the input.

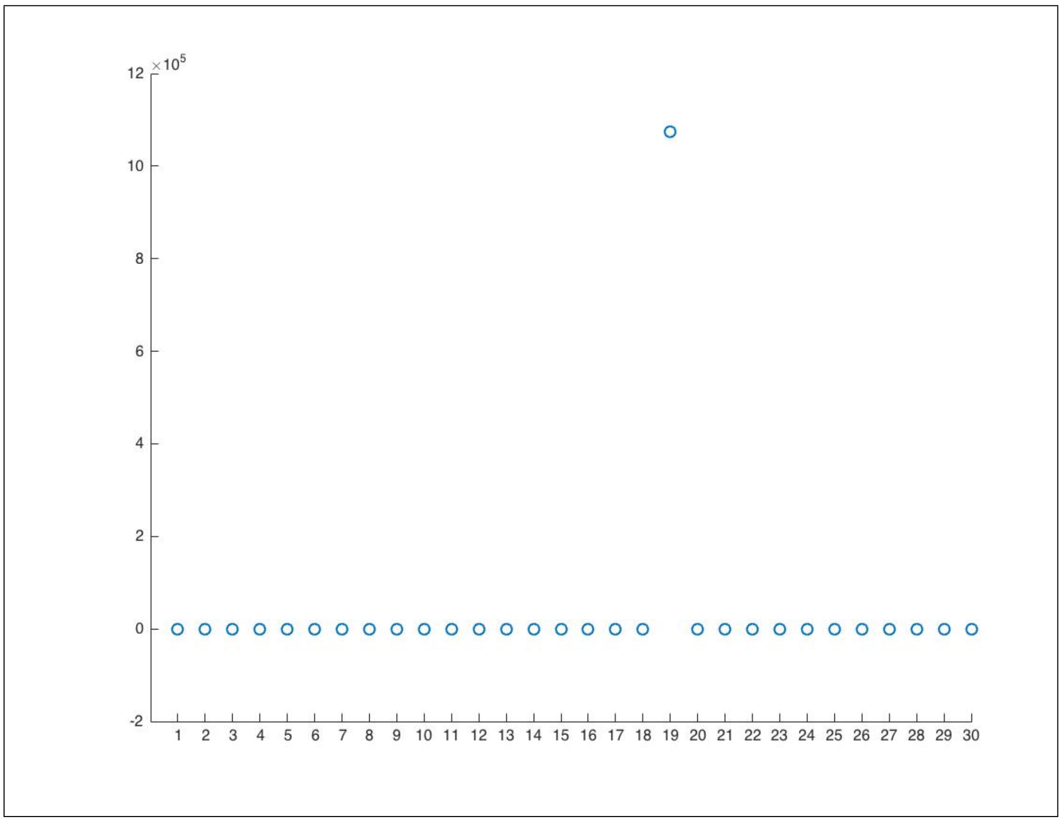
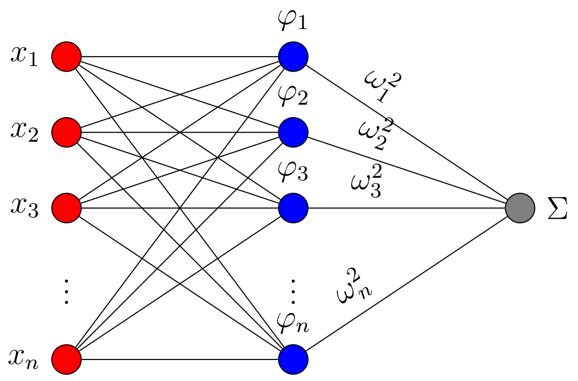


Figure 4.7: Feature direction showing maximum increase in original space



For training this neural network for direct NDCG maximization the total set of weights was flattened and put in a one dimensional array.

$$\boxed{\omega_{11}^1}\ \boxed{\omega_{12}^1}\ \boxed{\cdots}\ \boxed{\omega_{nn}^1}\ \boxed{\omega_1^2}\ \boxed{\cdots}\ \boxed{\omega_n^2}$$

The score of a data point instead of being $\mathbf{w}^T\ \mathbf{X}$ is now:

$$\text{out} = \sum_i \omega_i^2 \text{max}(\sum_j (\omega_{ji}^1 x_j), 0) \tag{4.9}$$

So the final objective is now maximized with respect to these weights. This objective maximization is taken care of by MTSLS as the number of parameters (the number of elements in the flattened array) are O(d^2) where d is the dimension of the input data. The results clearly show that linear feature

Table 4.6: Linear Feature Model v/s NN

Model	NDCG
Linear Feature Model	0.4865
NN trained using MTSLS	0.459778

model was performing better.

Chapter 5

Conclusion

The problem of e-commerce search personalization was viewed at from different angles, and various techniques that are and have been used to solve this problem were explored.
The first approach, was a new style of solving this problem, but was limited by the specificity of the model, and it was heavily dependent on the choice of evaluation metric. For example changing the metric from NDCG to some other information retrieval measure may not give similar results for the naive model. In this approach linear models, such as linear SVM, and logistic regression showed better results.
Other techniques such as collaborative filtering were also tried, but the type of sparse data in terms of number of users and items, in comparison to the number of user-item interactions was very less, so any matrix/tensor factorization related technique from CF would have to deal with a very high sparsity of about 99.995%. This is why the results obtained from that procedure were insignificant. Although this is the technique, most large industries which rely on recommender systems use.
Finally the learning to rank approach, which is employed in various search engines, for the sole purpose of personalization gave significant results. It is seen through this experiment that even in the Learning to Rank methods, linear methods outperformed all other methods. The regularized linear feature model solved using Coordinate Ascent method gave the best result so far in all approaches tried with an NDCG of 0.48, which is a 25.65% increase in search quality from the baseline given the search metric of NDCG. Given the basic features mined from the raw logs, this regularized linear feature model solved using CA gives us an NDCG that puts us on the 3rd position of the CIKM CUP2016, Track 2.

Bibliography

[1] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, pages 89–96. ACM, 2005.

[3] V. Dang. The lemur project-wiki-ranklib.

[4] Diginetica. Cikm cup 2016 track 2: Personalized e-commerce search challenge. `http://competitions.codalab.org/competitions/11161`, 2016. Online; accessed 20 October 2016.

[5] Simon Funk. Netflix update: Try this at home. =http://sifter.org/ simon/journal/20061211.html, Dec 2006.

[6] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. Ieee, 2008.

[7] Jayadeva, Sameena Shah, and Suresh Chandra. *Learning Global Optimization Through a Support Vector Machine Based Adaptive Multistart Strategy*, pages 131–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[8] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010.

[9] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[10] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2007.

[11] Donald Metzler and W Bruce Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3):257–274, 2007.

[12] C. Bourguignat P. Masurel, K. Lefvre-Hasegawa and M. Scordia. Dataikus solution to yandex personalized web search challenge. Technical report, 2014.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[15] Lin-Yu Tseng and Chun Chen. Multiple trajectory search for large scale global optimization. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3052–3059. IEEE, 2008.

[16] Tie-Yan Liu Ming-Feng Tsai Hang Li Zhe Cao, Tao Qin. Learning to rank: From pairwise approach to listwise approach. Technical report, April 2007.

Appendix A

Feature Extraction from raw log files

The following class of features was extracted from the data

- User Features
- Query Features
- Item Features
- Query-Item Features
- Session Features

User Features

- Category distribution of Purchase/View/Click history
  - 1217 Categories
  - vector of length of number of Categories for each purchase/view/click features.
  - ith element giving user purchase/view/click counts.
- Mean price of category wise purchase

These features model different users by capturing the historical user actions.

Query Features

- Mean Reciprocal Rank (MRR)*
- Original NDCG for list*
  - Reciprocal rank of item with relevance = 2(max)
  - Three levels of relevance [0,1,2] (from mid term slide)
- Min/Max/Mean/Median price of list of items of Query*
- Length of Query*
  - For modeling query ambiguity.
- Number of items for query*

Item Features

- Item Category
  - Categorical Variable - One hot encoded [Sparse Vector]
- Item Price*
- Item Relative price in category*
  - $\frac{price-meanp}{meanp}$
- Item Min/Max/Mean/Median Rank across all queries*
- Item Name Size*
- Unique views/purchases*
- Item click/view/purchase counts*
  - Measures popularity

Query-Item Features

- Similarity Measure
    - Cosine Similarity
    - Jaccard Cofficient[Easy to compute]*
- Original Rank*
    - An important indicator of query-item relevance, as data is anonymized.

Session Features

- Duration of session*
- Avg number of queries per session*
- NDCG average per session*

*** marked features were used only for Learning to Rank Models**

A.1 Train Test statistics

For the case of Naive Classifier, and L2R methods, the raw data was filtered to give all query-item pairs, with known user, for feature extraction as mentioned above. Train data was now found to be 1,443,013 samples large, and test data consisted now of 326,651. For collaborative filtering the user-item preference matrix was 141127×81106 large, as it only consisted of (unique users) $\times$ (unique items).

A.2 Some Statistics of raw data

Some statistics of the data present in the raw log forms.
The scorer function is used in all the optimisation techniques and its pseudo code is given as follows.

Table A.1: Raw Data Statisitcs

Sessions	573,935
Products	134,319,529
Products viewed from search	2,451,565
Clicks on products	1,877,542
Avg. click per session	3.271
Products purchased from search	68,818

Algorithm 4 scorer

```
1: function SCORER(w,X,Q)    ▷ Where w is weight vector (array of dimension - feature dimension), Q is set of queries
2:     score = 0
3:     for q in Q do
4:         $X_q$ = partition(X,q)    ▷ partition dataset wrt queries
5:         n = size($X_q$)
6:         for $j = 1$ to n do
7:             score$_q$[j] = $w^T X_q[j]$    ▷ score all items in a list for a given query
8:         end for
9:         sort($X_q, score_q$)    ▷ Reorder list according to scores
10:    end for
11: end function
```

List of Figures

List of Tables

List of Algorithms