Jobandtalent at RecSys Challenge 2016

Jose Ignacio Honrado Oscar Huarte Cesar Jimenez
Sebastian Ortega Jose R. Perez-Aguera
Joaquin Perez-Iglesias Alvaro Polo Gabriel Rodriguez

Jobandtalent (recommendation.team@jobandtalent.com)
Paseo de la Castellana, 93, Madrid, Spain

ABSTRACT

In this paper we describe the system built by the Jobandtalent Recommendation Team to compete in the RecSys Challenge 2016. The task consisted in predicting future interactions between Users and Items within the XING platform. The data provided by XING consists of users,items, plus interactions, and impressions of items showed to those users. We decided to apply a Learning to Rank approach to find the best combination of relevance features. We finally achieved the 11th position.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

Keywords

Recommendation Systems, Information Retrieval, Learning to Rank

1. INTRODUCTION

At *Jobandtalent* we try to find the perfect jobs for our users and, on the other way around, match the perfect candidates with the jobs posted in our platform. This task is, by definition, a recommendation system. Our approach is built on Learning to Rank techniques, which is a generic framework based on features that uses a discriminative training process. Our main day to day duty is to figure out and implement new features and gather data to continually feed the training process.

Recsys 2016 perfectly fits the challenges we face ourselves, hence this was a great opportunity to test our recommendation generation approach in a different environment, with different data and to compare our performance with other competitors using other approaches. Our focus was to reuse the tools and processes we already have as much as we could, but we also built some *ad-hoc* tooling to explore the data and to describe the best features based on previous data exploration. Then, we adapted our internal recommendation frameworks and implemented a training pipeline. Next sections describe in detail the steps we followed to fullfil our *Recsys 2016* participation.

RecSys Challenge '16, September 15 2016, Boston, MA, USA

DOI: http://dx.doi.org/10.1145/2987538.2987547

2. LEARNING TO RANK

The underlying idea of Learning to rank is to use machine learning techniques to build ranking models. State-of-the-art learning to rank techniques try to combine features to predict the ''best'' possible ranking of documents for a query[4]. The ranking is generated by a trained model based on previous observed data. We apply the ''listwise approach'', that produces a ranking of documents, while it optimises a metric which measures the quality of the ranking. Other approaches, in contrast, only consider an isolated document or a pair of documents as ''pointwise'' or ''pairwise'' techniques respectively[4]. To be able to carry on with a supervise-learning approach like Learning to Rank does, it is required to create a set of queries and its associated relevant documents. This set of fixed queries, or ground truth, allows to build the trained model, based on a specific ranking quality metric. In addition, a similar set of data, it is necessary to evaluate the performance of the previously trained model. This set of data is commonly named evaluation set.

3. LEARNING TO RANK IN RECSYS 2016

For the case of *Recsys 2016* and based on the supplied data, the input for our learning to rank framework would be Users, as queries, while the output would be a ranking of Items, as documents. Also, to be able to build a model, we required some relevance judgements, that is, the ground truth from which the algorithm will learn. In our case, relevance judgements consisted of the interactions between Users and Items. More precisely, we considered an Item relevant to a User when a positively interaction existed between them (click, bookmark, reply or apply). As the goal of the task was to predict the interactions between Users and Items for a given week (week 46), for which we don't have any data, we followed the same pattern to create the ground truth as we assume a higher similarity between week 45 and 46 than with any other week. Therefore we extracted the relevance judgements from the last available week (week 45), discarding previous interactions. To increase the precision of the trained model, we assumed that different type of interactions have different weights. For instance, we found more relevant a User replying to an Item that just clicking on it.

Once we had the relevance judgements, it was straightforward to build the training and evaluation dataset. We split the User queries following a random 80/20 distribution, that means 80% of the queries were used for training, while the rest 20% were used for model evaluation.

The final input for the training algorithm was a feature matrix. The feature matrix is a *NxM* matrix where *N* is the total results for all queries and *M* is the number of defined features. It is important to remark that all values for the features must avoid to use data from 45^{th} week as this will incur in overfitting, biasing the model and thus the obtained results.

There are a set of well known learning to rank algorithms based on different techniques as RankBoost[6], RankNet[1] , LambdaRank[3] or LambdaMART [2]. From our previous experience, we decided to use LambdaMART as it outperforms many other methods[5]. LamdbdaMART uses a listwise approach based on gradient tree boosting, further details about the algorithm are out of the scope of this paper. Finally, we should mention that we decided to use the evaluation metric NDCG[7] to train our model. Main advantage of this metric is that it is able to take into account different levels of relevance which, as we mentioned before, perfectly fits the different types of interactions shown by our users.

3.1 Data exploration

Prior defining a set of features that describe the behaviour of *Recsys 2016* Users, we did an exploratory analysis of available data.

A hypothesis we had was whether there was *a priori* probability of an Item being interacted, some kind of 'popularity' attribute we could measure. We discovered there were Items that had a significant higher probability of being interacted than the majority of Items.

We also wanted to know if there were terms within Items' titles with high probability of being interacted, that is, if there were some click-bait terms. Unfortunately, we were not able to find any correlation between interactions and Item terms.

Another analysis we performed was whether a *User* was keen to interact several times with the same Item. Interestingly, we found that there was a high probability of recurring interactions and that this probability, as expected, decayed over time, as it can be seen in Figure 1.

One interesting insight we obtained from data is that there was a really high number of interactions happening without a prior impression. Despite this fact, Items shown to Users still had a high probability of being interacted.

We tried to find out if there was any relationship between the probability of a User interacting with an Item and the geographical distante between both. To explore this, we inferred the location of a User by computing a simplified gravity center of his interacted Items. Latitude of this that gravity center was the arithmetic mean of all latitudes of the Items the User interacted with. Longitude of the User's gravity center was calculated in the same fashion. As expected, we observed that there were a lot of interactions within a 10 kilometres radius, but one interesting thing we found out is that there were a nearly perfect Gaussian probability distribution with an average of 250 kilometres and a really long tail of Items interacted in a radius above 1000 kilometres (see Figure 2). In our opinion, it looks like a synthetic joint probability distribution between a Gaussian with average 250 and a uniform probability distribution above 1000 kilometres.

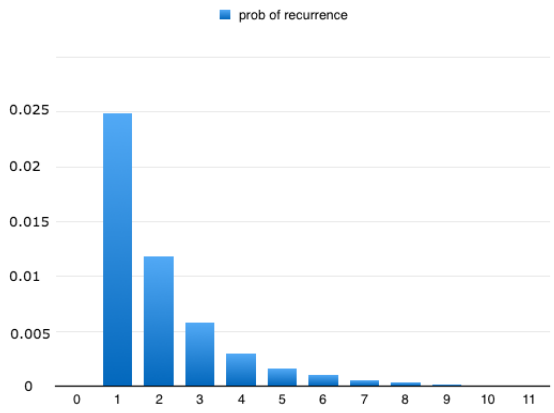

Figure 1: Probability or Recurrence

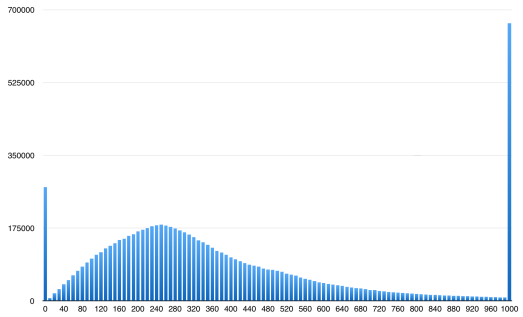

Figure 2: Interaction by Distance

Another path we took was to find out if similar Users interact with the same Items, where similar is a broad concept that can be modelled in different ways. We tried different definitions of similarity among Users, as Users with same job roles, same discipline and industry or same region and country. The most restrictive similarity model (*same job roles*) showed a better performance in terms of Item interaction prediction.

4. FEATURES

We have implemented several features to cover different aspects of *Items* relevance given a *User*. In our internal framework these features can be implemented in two different flavours: *retrievers* or *collectors*.

Following the features which were included in the final trained model are described:

4.1 Retrievers

Retrievers' responsibility is to retrieve a list of documents given a query, this list of documents is usually sorted as it assigns a score to each returned document. Therefore, retrievers are in charge of obtaining recall, as only documents retrieved by this type of features will be part of the list of results returned.

Job title match.

Content based feature that match the job roles of the *User* and the title of the *Item*. Only those *Items* that contain the whole set of job role terms of the User in their title are returned. Also, this retriever sets an score based in the matching degree using BM25.

Job tag match.

Another content based feature that matches the job roles of the *User* and the tags of the *Item*. Only *Items* that contain the whole set of job role terms of the User in their tags are returned. This retriever also sets an score based in the matching degree using BM25.

Discipline and region retriever.

Return those *Items* with the same discipline and region as the *User*. This feature was part of the proposed baseline.

Industry and region retriever.

Return those *Items* with the same industry and region as the *User*. This feature was part of the proposed baseline.

Relevant items per interaction.

This is a family of retrievers based on the *User* behaviour. Generally, it returns *Items* similar to the ones interacted by the *User*. We defined 6 different similarity functions between *Items*, based on matching between titles and depending on the type of interaction: Click, Bookmark or Reply. For all cases, the retriever sets a score based on BM25.

Interactions made by user.

Behaviour based retriever that returns the most recent *Items* interacted by the User. By interacted we mean those Items clicked, bookmarked or replied by the user (deletions are not considered). The score set by the feature will be the number of the week where the Item was interacted.

Impressions shown to user.

Retriever which returns the most recent Items shown to the User. The score set by the feature will be the number of the week where the Item was impressed.

User to user collaborative filtering.

Collaborative filtering retriever trying to infer the User preferences by examining the behaviour of similar *Users*. By similar we mean *Users* with exactly the same Job Roles as another *User*. It fetches all Items interacted by those similar *Users* (all types of interaction except 'deletes'), sorted by the number of times each Item has been interacted and returns the top N elements of the rank. The score of the retriever is equivalent to the number of times each Item was interacted.

Item to item discipline and region collaborative filtering.

Collaborative filtering retriever that tries to infer Items the User could be interested in by examining his behaviour. It fetches the Items interacted by a User (all types of interaction except 'deletes') and it looks for the most recurring discipline and region tuples within the Interactions. The retriever will use the inferred discipline and region tuples to retrieve Items matching them. Items got a score based on the tuple, discipline and region, frequency.

Clustered Users.

Collaborative filtering retriever based on cluster of similar Users. To decide whether a user is a member of a cluster, the following features are applied: career level, education degree, experience, location, industry, discipline and activity.

For clustering we decided to use k-means, fixing the number of centroids to 3000. Finally, this retriever returns the most popular interaction for the cluster the input User belongs to. The retriever assigns a score based on the popularity of the Item within the cluster.

4.2 Collectors

Collectors assign scores to the documents retrieved by the *retrievers*, thus this type of features are focused on precision.

Freshness.

Annotates each document with the number of seconds since its most recent interaction. Items never interacted have a predefined score equivalent to a very old interaction.

Item Popularity.

For each retrieved Item, it calculates the number of times the Item has been interacted.

User Behaviour.

This family of collectors assign a set of flags indicating in which case the user interacted with the Item. We decided to add this flag for all the interaction types, assuming these as a relevance signal. A total of 4 features are computed, one for each type of interaction.

Each collector assigns a value of 1 if the User interacted (clicked, bookmarked, replied or applied, deleted) with Item and 0 in other case.

5. LEARNING PROCESS

In a nutshell, main purpose of the training process is to find the optimal way to combine the available features by assigning them a weight. The learning step takes the feature matrix as input and, as it is a quite memory consuming process, is sensitive to the size of the matrix. In order to speed up the training process, we tried to reduce the feature matrix size. This can be achieved in a double fashion: reducing the number of elements used to train and/or reducing the number of features applied. Bear in mind that for some of our cases the feature matrix achieved a size over 150GB, which made unfeasible to train with all the features/queries available.

Reducing the number of features can be achieved by means of Principal Components Analysis methods, detecting those features which strongly correlate or overlap with others and thus its contribution to the final score would be minimum. A different approach to reduce the size of our feature matrix was to retain a sample of the whole set of Items returned for each query. We call this process *head shrinking*[1] and, although it slightly penalises the performance of the trained model, it allowed us to run many more experiments by significantly reducing the training time.

The learning process was focused on optimising NDCG@30 metric. We decided to cut at a ranking size of 30, as this was the maximum number of results per query defined in

[1]the technique of head shrinking practiced by the Jivaroan tribes

Table 1: LambdaMART configuration parameters

Parameter	Value
No. of trees	500
No. of leaves	10
No. of threshold candidates	256
Min lead support	1
Learning rate	0.1

Recsys 2016 rules. Also, NDCG revealed itself as a better choice for this specific task than MAP@30.

As part of the training and evaluation process, we developed a tool to observe the performance of each feature. This tool was designed to measure the purity of each node of the output model. As LambdaMART uses a tree gradient boosted algorithm, the output model is a sequence of trees. The tool computed the purity of each node based on how good its related feature split relevant and non relevant Items. Using this information we could remove features that were not good enough or were not adding much value.

Finally, we detected a strong correlation between the performance of our model and the recall it was able to produce. This fact lead us to implement a recall tool to inform us the number of relevant documents returned by each feature, despite their position in the ranking. Given this information, we could attempt to extract patterns from documents that were not retrieved to improve recall.

5.1 Internal evaluation

As the number of submissions was limited and we wanted to speed up the training-evaluation loop, we added an internal evaluation stage with the main purpose of allowing us to perform evaluations before submissions. The internal evaluations were computed by running the trained model with a subset of queries (Users) and their interactions that happened in the week 45. The output obtained was evaluated using the same evaluation metric proposed by *Recsys 2016* challenge.

6. RESULTS

This section will give a brief overview over the results we obtained during our participation in *Recsys 2016* challenge. First, after some research we found the best LambdaMART parametrisation (see Table 1) for our use case, that we then used in all experiments we performed.

Table 2 shows the most significant submissions we made. Our initial attempt was the proposed baseline and then, as a quick optimisation, we changed the matching function to BM25 with some basic parameter tuning. BM25 with the right parameter configuration showed itself as a better matching feature than simple overlapping, as default baseline suggests.

After that, we added the *Interactions made by user* retriever which increased our score in more than a 500% percent. At that point, we added training that lead us to a little score boost. But the more important value training gave us was that it allowed us to combine and weight features avoiding the need to weight each feature by ourselves (a task that soon became unfeasible). In same path, we added the *Impression shown to user*, that also gave us a significant score boost (80% increase). Probability of recurrence over Items and impressions were, by far, the best performers. Adding

Table 2: Submission scores

Run	Internal	Official
Baseline	22849	30721
Baseline BM25 (b=0.2 / k1=1.2)	23591	36857
Interactions made by user	162846	229949
Trained model	176312	241422
Impressions with decay	400217	434433
Item Popularity	435325	475940
User behaviour and Freshness	442049	483732
Clustering (1500 centers)	447674	488192
Collaborative Filtering	457391	507022
Best effort @ 2000	480396	533232
Best effort @ 9000	482083	535899

a decay factor over time was a simple but effective way to sort previous interactions and impressions.

Then we added *Item popularity* collector. This was also a simple, but very effective, way to retrieve *Items* that have high probability of being interacted. It added a boost factor based on popularity to any item that was retrieved by a filter.

Clustering and collaborative filtering were our last attempts to increase the performance of the model. The main focus of both features was to improve recall, since these features are not content based as the rest of the features were.

Finally, we tried to improve the results by increasing the number of items retrieved by each feature (*Best effort @*), allowing LambdaMART to observe more Items in the training step. This had the objective of improving the precision of the model. As a consequence of this, the size of the returned rankings increased and that had a huge impact in the time needed to train new models.

7. CONCLUSIONS

After trying with many different approaches, we found out that the features which contributed the most were the ones related with previous interactions. We obtained more than 400k points by just returning previously interacted or impressed items sorted by time dimension.

We were able to considerably speed up our submission process by means of internal evaluation. Some experiments were discarded after the internal evaluation without having to generate the official submission file. Using last week of the supplied data to internally evaluate our model yielded results that strongly correlates[2] with the official evaluation carried out by the organisation.

We believe the biggest downside of our approach was limiting the number of documents supplied to LambdaMart. Increasing the size of the lists of documents retrieved made unfeasible to train a new model due to feature matrix size. Maybe, a better approach would be a setup of maximum recall where all items would be analysed, but for this case it was clear our implementation of LambdaMART could not deal with it.

We managed to execute all phases of our pipeline in parallel, except training the model. We strongly believe this fact prevented us of trying alternative features that would allow us to obtain a more precise model. As it is clear, a

[2]Linear correlation between internal and official evaluation exceeds 0.9

possible solution to this issue would be to develop tools and algorithms with parallelisation in mind.

8. REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[2] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. Technical report, June 2010.

[3] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 193–200. MIT Press, 2007.

[4] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview.

[5] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Yahoo! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010*, pages 1–24, 2011.

[6] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, Dec. 2003.

[7] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, Oct. 2002.