

YetiRank: Everybody Lies

Andrey Gulin
Yandex
gulin@yandex-team.ru

Igor Kuralenok
Yandex
solar@yandex-team.ru

Annotation

Our approach consists of new pairwise algorithm for learning the ranking function that is based on Friedman's gradient boosting. The main feature of our approach is modeling the imprecision of the judgments. In particular, we build a probabilistic transition model of one judgment into another when the same query-url pair is reviewed several times, and use the resulting transition matrix to weigh the pairs of urls used for training. Since the transition matrix for the competition data wasn't available, we emulated the data to construct it.

Learning algorithm

We used pairwise learning idea and Friedman's gradient boosting approach [1] with trees as a weak learner as a basis for our approach. The objective function for the optimization was:

$$\sum_{(i,j)} w_{ij} \log \frac{e^{x_i}}{e^{x_i} + e^{x_j}}, \quad (1)$$

where index (i, j) goes over all pairs of urls retrieved for a query such that the judgment for the document i is higher than the judgment for the document j , w_{ij} is the weight of a given pair of urls and x_i, x_j are predictions of the ranking function on the urls. To maximize the objective function we need to differentiate it. In the gradient boosting paradigm, the function optimization is performed on every step in the direction maximally aligned with the gradient. In our case, we would like to have the best alignment for every term in the sum that could be achieved by variable substitution to $x_i - x_j$:

$$\operatorname{argmax}_{dx_i, dx_j} \sum_{(i,j)} w_{ij} \left(d(x_i - x_j) - \frac{\partial}{\partial(x_i - x_j)} \log \frac{e^{x_i}}{e^{x_i} + e^{x_j}} \right)^2$$

As a result, we need to find dx_i :

$$\operatorname{argmin}_{dx_i, dx_j} \sum_{(i,j)} w_{ij} \left(dx_i - dx_j - \frac{e^{x_j}}{e^{x_i} + e^{x_j}} \right)^2 \quad (2)$$

After we know dx_i for all urls for a query, optimization of (1) consists of making a step of gradient boosting in the direction of dx_i .

To compute dx_i , we used oblivious decision trees that unlike traditional ID3, MD4.5, CART [2] decision trees, use the same split for all nodes at a given depth. The choice of oblivious trees reduces our algorithm to splitting the space down into cells in a way that optimizes equation (2). The splitting is done greedily. First, the space is split into two halves, then into 4 quadrants, using the first split as a starting point, and so on until the limit on the depth of the tree is reached. Our final solution for the contest had a depth limit of 6. To improve the result we used bagging. It is important to note, that the solutions of the mean squared equation for each checked condition can be postponed until the time when the data is split into 2^m cells, and the optimization can work in the dimensions of $2^m \times 2^m$ where m is the tree depth, instead of $n \times n$ where n is the number of urls.

The key role in the effectiveness of the proposed solution is played by how the weight of the pair of urls is computed. We used the following intuition:

- the weight of the pair is dependent on its potential position in ranking
- only pairs that could be found in ranking are important (i.e. documents from the top rarely form pairs with documents from the bottom)

- the confidence in the correctness of the order of urls in the pair is important

As a result we considered the following model: $w_{ij} = d_{ij}c(r_i, r_j)$ where d_{ij} is the combined penalty for the pair rank and the probability of this pair's occurrence, r_i – relevance judgment on the url i , $c(r_i, r_j)$ is the degree of confidence that documents with r_i and r_j marks are in correct order.

To introduce the notion of “potential” position in ranking, we used the random transformations of the current ranking function $x'_i = x_i \frac{\xi}{1-\xi}$, where ξ is uniformly distributed in $[0, 1]$. Sampling this random transformation leads to some ranking which forms number of consequent pairs. Then we can use these pairs to predict mean position and the probability of the occurrence of a pair $d_{ij} = \frac{1}{n} \sum_{t=1}^n \frac{1}{index_t((i,j))}$ where $index_t((i,j))$ is the index of the pair in a sample ranking t if it was present, ∞ otherwise, and n is the number of samples.

It is widely accepted that the expert judgments are unstable. For instance, even after issuing a judgment today, the expert may change it tomorrow. This fact bring us to representation of a judgment as a random variable. As a result, the right order of documents is also a random variable, and we can introduce a function we call “confidence in the ordering”. We used the following equation to estimate the confidence:

$$c(r_i, r_j) = \sum_u \sum_v sign(u - v) p(J_u | r_i) p(J_v | r_j)$$

where u and v run over indexes of all judgments J in the increasing order of relevance.

Transition matrix

Finally, we address the way of building the distribution $p(J|r_i)$. When we have access to the experts, it makes sense to conduct a direct experiment and collect several judgments per query-url pair. However, in the contest we emulated the process of judgment collection. The classifier built on the first half of the training set and applied to the second half, was used a second judge. The table below illustrates the distribution we used after some hand-tuning:

	bad	fair	good	excellent	perfect
bad	0.48	0.40	0.10	0.02	0
fair	0.08	0.68	0.21	0.03	0
good	0.01	0.44	0.47	0.08	0
excellent	0	0.07	0.54	0.38	0.01
perfect	0	0.01	0.09	0.43	0.48

At first we'd tried to use transition matrix built on first track (to employ transfer-learning) but experimental results on it were worse then those on original track 2 data.

Conclusions and future work

In conclusion, we presented a method for learning the ranking function that uses the additional information about the uncertainty of judgments. It performed well on the competition task and likely deserves more attention and further exploration. One of the possible directions for future research consists of exploring methods of noise reduction in building the transition matrix.

References

- [1] Jerome H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine", 2000, pp. 1189-1232, (29) Annals of Statistics
- [2] Quinlan, J. R., "Induction of Decision Trees", 1984, pp. 81-106, (1), Mach. Learn.