Hugh E. Williams

Ideas, Engineering, Search Engines, Leadership, …

Ranking at eBay (Part #3)

Over the last two posts on this topic, I've explained some of the unique problems of eBay's search challenge, and how we think about using different factors to build a ranking function. In this post, I'll tell you more about how we use the factors to rank, how we decide if we've improved ranking at eBay, and where we are on the ranking journey.

Hand-tuning a Ranking Function

A ranking function combines different factors to give an overall score that can be used to rank documents from most- to least-relevant to a query. This involves computing each factor using the information that it needs, and then plugging the results into the overall function to combine the factors. Ranking functions are complicated: there's typically at least three factors in the most simple function, and they're typically combined by multiplying constants by each of the factors. The output is just a score, which is simply used later to sort the results into rank order (by the way, the scores are typically meaningless across different queries).

If you've got two, three, or maybe ten different factors, you can combine them by hand, using a mix of intuition, and experimentation. That's pretty much what happens in the public domain research. For example, there's a well-known ranking function Okapi BM25 that brings together three major factors:

1. **Term frequency**: How often does a word from the query occur in the document? (the intuition being that a document that contains a query word many times is more relevant than a document that contains it fewer times. For example, if your query is *ipod*, then a document that mentions *ipod* ten times is more relevant than one that mentions it once)
2. **Inverse document frequency**: How rare is a query word across the whole collection? (the intuition being that a document that contains a rarer word from the query is more relevant than one that contains a more common word. For example, if your query was *pink ipod nano*, then a document that contains *nano* is more relevant than a document that contains *pink*)
3. **Inverse document length**: How long is the document? (the intuition being that the longer the document, the more likely it is to contain a query word on t[illegible] probabilities. Therefore, longer documents need to be slightly penalized o[illegible]nate the results for no good reason)

Follow

How are these factors combined in BM25? Bm25 the community recommends that t inverse document frequency (a multiplica from different people, and it's pretty mucl what works. You'll often find that researcl they selected them; for example, in this 2(and the constants we chose.

lia page for Okapi ly higher than the recommendations approaches and see ey used, and how l25 variant we use

This all works to a certain point: it's possib intuitively understand, as long as you don

ction you can

Follow "Hugh E. Williams"

Get every new post delivered to your Inbox.

Join 2,664 other followers

Enter your email address

Sign me up

Build a website with WordPress.com

Training Algorithms to Combine Factors

At eBay, we've historically done just what I described to build the Best Match function. We created factors, and combined them by hand using intuition, and then used experimentation to see if what we've done is better than what's currently running on the site. That worked for a time, and was key to making the progress we've made as a team.

At some point, combining factors by hand becomes very difficult to do — it becomes easier to learn how to combine the factors using algorithms (using what's broadly known as machine learning). It's claimed that AltaVista was the first to use algorithmic approaches to combine ranking factors, and that this is now prevalent in industry. It's certainly true that everyone in the Valley talks about Yahoo!'s use of gradient boosted decision trees in their now-retired search engine, and that Microsoft announced they used machine-based approaches as early as 2005. Google's approach isn't known, though I'd guess there's more hand tuning than in other search engines. Google has said they use more than 200 signals in ranking (I call these factors in this post).

Let me give you an example of how you'd go about using algorithms to combine factors.

First, you need to decide what you're aiming to achieve, since you want to learn how to combine the factors so that you can achieve a specific goal. There's lots of choices of what you might optimize for: for example, we might want to deliver relevant results on a per query basis, we might want to maximize clicks on the results per query, we might want to sell more items by dollar value, we might want to sell more items, or we might want to increase the amount of times that a user uses the search engine each month. Of course, there's many other choices. But this is the important first step — decide what you're optimizing for.

Second, once you've chosen what you want to achieve, you need training data so that your algorithm can learn how to rank. Let's suppose we've decided we want to maximize the number of clicks on results. If we've stored (logged or recorded) the interactions of users with our search engine, we have a vast amount of data to extract and use for this task. We go to our data repository and we extract queries and items that were clicked, and queries and items that were not clicked. So, for example, we might extract thousands of sessions where a user ran the query *ipod*, and the different

item identifiers that they did and didn't click on; it's important to have both positive and negative training data. We'd do this at a vast scale, we're likely looking to have hundreds of thousands of data points. (How much data you need depends on how many factors you have, and the algorithm you choose.)

So, now we've got examples of what users do and don't click on a per query basis. Third, it's time to go an extract the factors that we're using in ranking. So, we get our hands on all the original data that we need to compute our factors — whether it's the original items, information about sellers, information about buyers, information from the images, or other behavioral information. Consider an example from earlier: we might want to use term frequency in the item as a factor, so we need to go fetch the original item text, and from that item we'd extract the number of times that each of the query words occurs in the document. We'd do this for every query we're using in training, and every document that is and isn't clicked on. For the query ipod, it might have generated a click on this item. We'd inspect this item, count the number of times that ipod occurs, and record the fact that it occurred 44 times. Once we've got the factor values for all queries and items, we're ready to start training our algorithm to combine the factors.

Fourth, we choose an algorithmic approach to learning how to combine the factors. Typical choices might be a support vector machine, decision tree, neural net, or bayesian network. And then we train the algorithm using the training data we've created, and give it the target or goal we're optimizing for. The goal is that the algorithm learns how to separate good examples from bad examples using the factors we've provided, and can combine the factors in a way that will lead to relevant documents being ranked ahead of irrelevant examples. In the case we've described, we're aiming for the algorithm to be able to put items that are going to be clicked ahead of items that aren't going to be clicked, and we're allowing the algorithm to choose which factors will help it do that and to combine them in way that achieves the goal. Once we're done training, we'd typically validate that our algorithm works by testing it on some data that we've set aside, and then we're ready to do some serious analysis before testing it on customers.

Fifth, before you launch a new ranking algorithm, you want to know if it's working sensibly enough for even a small set of customers to see. I'll explain later how to launch a new approach.

If you're looking for a simple, graphical way to play around with training using a variety of algorithms, I recommend Orange. It works on Mac OS X.

What about Best Match at eBay?

We launched a machine-learned version of Best Match earlier in 2012. You can learn more about the work we're doing on machine learning at eBay here.

We now have tens of factors in our ranking function, and it isn't practical to combine them by hand. And so the 2012 version of Best Match combines its factors by using a machine learned approach. As we add more factors — which we're always trying to do — we retrain our algorithm, test, iterate,

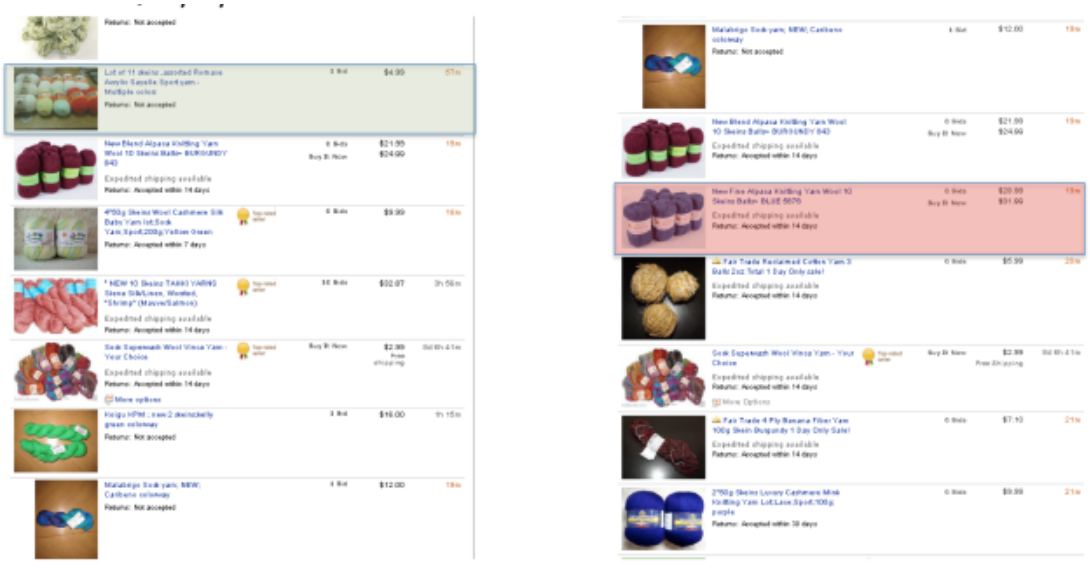
learn, and release new versions. We're adding more factors because we want to bring more knowledge to the ranking process: the more different, useful data that the ranking algorithm has, the better it will do in separating relevant from irrelevant items.

We don't talk about what target we're optimizing for, nor have we explained in detail what factors are used in ranking. We might start sharing the factors soon — in the same way Google does for its ranking function.

Launching a New Ranking Algorithm

Before you launch a new ranking function, you should be sure it's going to be a likely positive experience for your customers. No function is likely to be entirely better than a previous function — what you're expecting is that the vast majority of experiences are the same or better, and that only a few scenarios are worse (and, hopefully, not much worse). It's a little like buying a new car — you usually buy one that's better than the old one, but there's usually some compromise you're making (like, say, not quite the right color, you don't like the wheels as much, or maybe it doesn't quite corner as well).

A good place to start in releasing a new function is to use it in the team. We have a side-by-side tool that allows us to see an existing ranking scheme alongside a new approach in a single screen. You run a query, and you see results for both approaches in the same screen. We use this tool to kick the tires of a new approach, and empirically observe whether there's a benefit for the customers, and what kinds of issues we might see when we release it. I've included a simple example from our side by side tool, where you can see a comparison of two ranking for the query *yarn*, and slightly different results — the team saw that in the experiment on the left we were surfacing a great new result (in green), and on the right in the default control we were surfacing a result that wasn't price competitive (in red).



Side by side results for the query yarn. On the left, an experiment, and on the right is the default experience.

If a new approach passes our bar as a team, we'll then do some human evaluation on a large scale. I explained this in this blog post, but in essence what we do is ask people to judge whether results are relevant or not to queries, and then compute an overall score that tells us how good our new algorithm is compared to the old one. This also allows us to dig into cases where it's worse, and make sure it's not significantly worse. We also look at the basic facts about the new approach: for example, for a large set of queries, how different are the results? (with the rationale that we don't want to dramatically change the customer experience). If we see some quick fixes we can make, we do so.

Once a new algorithm looks good, it's time to test it on our customers. We typically start very small, trying it out on a tiny fraction of customers, and comparing how those customers use search relative to those who are using the regular algorithms. As we get more confident, we increase the number of customers who are seeing the new approach. And after a few week's testing, if the new approach is superior to the existing approach, we'll replace the algorithm entirely. We measure many things about search — and we use all the different facts to make decisions. It's a complex process, and rarely clear cut — there's facts that help, but in the end it's usually a nuanced judgement to release a new function.

Hope you've enjoyed this post, the final one in my eBay ranking series. See you again next week, with something new on a new topic!

Share this:

Tweet 17 Like 7 Share 7 Share 5 Email

★ Like


3 bloggers like this.

Related

Ranking at eBay (Part #2)
In "search engines"

Ranking at eBay (Part #1)
In "search engines"

Query Rewriting in Search Engines
In "search engines"

This entry was posted in search engines and tagged ebay, machine learned ranking, ranking, search, search engines on May 15, 2012 [http://hughewilliams.com/2012/05/15/ranking-at-ebay-part-3/] .

4 thoughts on "Ranking at eBay (Part #3)"

Ardent Logophile
May 15, 2012 at 12:27 pm

Thoroughly enjoyed reading the post! Thank you!

Pingback: Quick Note on the Pros and Cons of Changing Ranking Algorithms « Alan Kent's Blog

Mark
May 19, 2012 at 7:42 am

the future of ebay http://mashable.com/2012/05/18/threadflip/

tomatotalk
December 19, 2012 at 4:43 am

That was interesting,thank you. This will surely make eBay better. It is a bit similar to Amazon who also seems to rank by text and user behaviour, including sales and likes and reviews.
