Learning to Rank for Business Matching

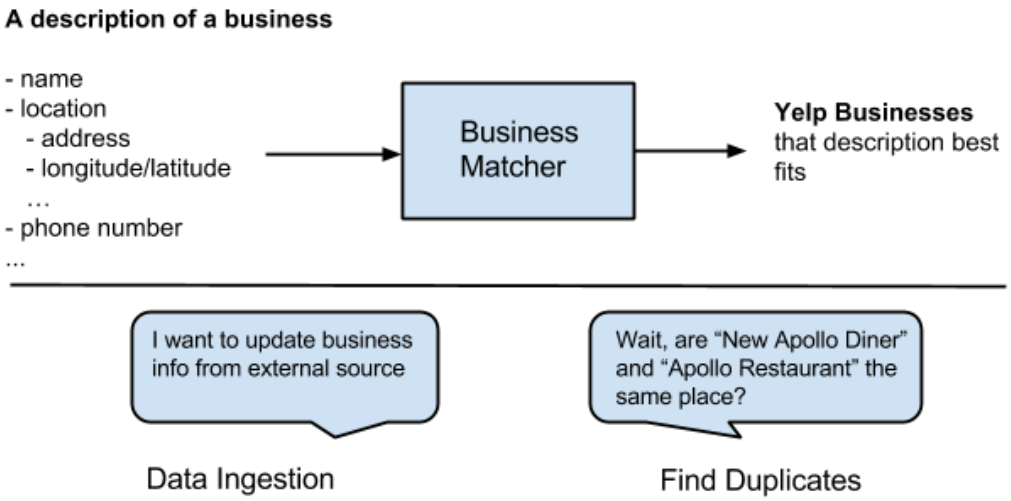
Rui W., Software Engineer
Dec 10, 2014

At Yelp, we solve a lot of different information retrieval problems ranging from highlighting reviews for businesses to recommending nearby restaurants. We recently improved one system called business matching, moving from a naive ranking function to a machine learned approach using Learn to Rank.

The Business Matching Problem

Business matching is a system that accepts a description of a business (e.g. name, location, phone number) and returns the Yelp businesses that description best fits. We can use this in several different contexts. For example, in 2012 Yelp announced a partnership spearheaded by the City of San Francisco to allow municipalities to publish restaurant inspection information to Yelp. The data feed shared by the City of San Francisco contained restaurants with business pages on Yelp but we didn't know the exact business IDs needed to correctly update their information.

Originally, we had a number of similar ad-hoc implementations across different engineering teams but, as we grew, concerns like codebase complexity, testability and maintenance costs, as well as matching quality started coming up. To solve these problems, we built a unified system to replace all the existing ad-hoc implementations. The system, similar to other information retrieval systems, takes a semi-structured description of a business (e.g. name, location, phone number) as input, searches through our database and returns a ranked list of Yelp businesses with scores that measure how good they match the description.



Architecture

We use Elasticsearch a lot at Yelp. It should come as no surprise that here we use it as our core

engine. Originally, the system architecture looked fairly simple: it normalized the input, built a big Elasticsearch query, searched the index, got the results, and did some filtering. In order to get the most relevant results, we needed to leverage information included in the input for each component. Components can be things like “name,” “location text,” “geo-distance” “phone number,” etc., which meant that each query we sent to Elasticsearch contained subqueries for each component. For example, we could build a name subquery that matches business names using TF-IDF scoring logic and a distance subquery that gives businesses higher scores if they are closer to the input location. The final query logic would linearly combine scores for each subquery or component and output a final score for each business candidate.

To measure the effectiveness of the system, we built an evaluation framework for it. A sample of some past business matching requests were pulled and we manually labeled the returned businesses as correct or incorrect matches. These results created the “gold dataset” and the system was then evaluated by running against this dataset and recording whether it returns the correct matching businesses for each request. Some standard information retrieval metrics like Precision/Recall/F1 are used to measure the matching quality.

Addressing The Downsides

The system worked fairly well but there’s always room for improvement. As you can imagine, there are many things we need to balance in the scoring logic here:

- How much weight should we give for address text matching?
- Is phone number a good indicator for matching?
- Is linearly combining each component score a good solution?

Originally, we addressed these questions by doing ad hoc experiments manually. We would poke around with different ideas, changing values and seeing if the F score improved. If we found a change where the evaluations looked good, we would push this change. However, performing these manual ad-hoc experiments is expensive and time-consuming.

To make things even harder, different clients may want to use the business matching system to solve different problems. Each client might want a slightly different ranking logic. Combining this time consuming task with the expense of ad-hoc manual parameter tuning led to a large amount of human time being spent on tuning the system. We reached a point where we felt that improving ranking logic by running ad-hoc experiments manually was reaching a bottleneck. In order to optimize the system further we wanted to have some automated, algorithmic solution.

Learning to Rank

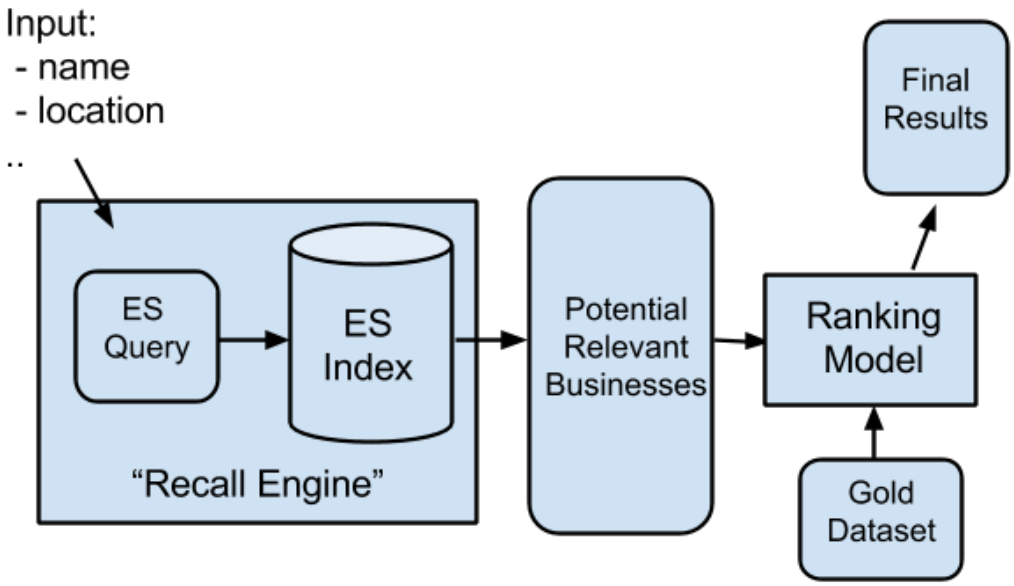
Learning to Rank is a method of automatically creating a ranking model by using machine learning. In our business matching problem, we wanted to build a machine learning model that would optimize our ranking algorithm in an automatic and systematic way.

How it works

Given the nature of our problem, we adopted the “ Pointwise Approach,” which approximates matching businesses as a regression problem. More specifically, for each candidate business we want the machine learning model to predict a numeric score, which serves as the estimation for how good this business matches the input description. As for features, we included the scores for each component returned from Elasticsearch. Some additional features reflect the original ranking given by Elasticsearch are included as well. We generated the training data by running regular evaluation on the gold dataset and recording relevant information returned from Elasticsearch.

Putting all of this together, we now have our new architecture. We still construct the query to Elasticsearch and get results containing a list of candidate businesses with scores for each component/subquery. But instead of linearly combining them, we throw them into the trained

rescoring model and, finally, a ranked list of rescored business candidates is returned. Essentially, for this system we're using Elasticsearch (which is really good at getting a pool of potential relevant candidates) as a "recall engine". We extract the core ranking logic out of Elasticsearch and use the more powerful machine learning algorithms to achieve better matching quality.



Improvements

Instead of searching for a set of optimal parameters of a linear ranking function manually, the learning algorithm makes this optimization process flexible. Now the ranking function can be either linear or non-linear by applying different learning algorithms with the parameters of the ranking function being learned in an automatic and systematic way. Moreover, since we are frequently updating the database, the TF/IDF and other scoring factors change over time. The new machine learning ranking model provides certain stability on top of Elasticsearch. Our evaluation results showed that our new learning to rank approach boosted F1 score from 91% to 95%.

With these improvements, we can treat our business matching system as a general business retrieval system framework that can be configured for new problems or clients, solving a much broader set of problems. For example, we are currently working on improving our business deduplication system, which discovers and automatically merges duplicate businesses in our data. To use business matching system we just need to retrain the machine learning model with a modified set of features on a different training set.

By using machine learning to approach our business matching problem, our retrieval system's matching quality significantly improved and also became more flexible, more stable and more powerful.

Back to blog