Learning to Rank for Better Search Results

By Andy Liu, Senior Data Engineer at Lucidworks

Improving search relevance is difficult. Learning to Rank (LTR) is an important and powerful technique which utilizes supervised machine learning to address the problem of search relevancy. An LTR approach leverages machine learning to automatically tune relevancy factors, which not only alleviates the pain associated with manual processes like boosts and blocks, but also promises significantly improved relevancy with the use of state of the art modeling techniques.

Recent versions of Solr include a Learning to Rank component, but there are still significant practical barriers to using LTR. First of all, Solr's LTR capabilities are library and API-level building blocks, with reference documentation that would only make sense to a very experienced search engineer. Secondly, Solr's LTR component does not actually train any models; it is left up to you to custom build a model training pipeline from scratch. Lastly, figuring out how all these pieces fit together to create an end-to-end LTR solution requires a significant amount of engineering and data science expertise.

To address these shortcomings, this tutorial will demonstrate how to implement an end-to-end Learning to Rank solution that greatly reduces the amount of effort required to implement and operationalize LTR. In addition to these practical benefits, we will demonstrate the power of the Fusion platform by combining LTR with insights derived from signals. When used together, this offers significant gains in search relevancy.

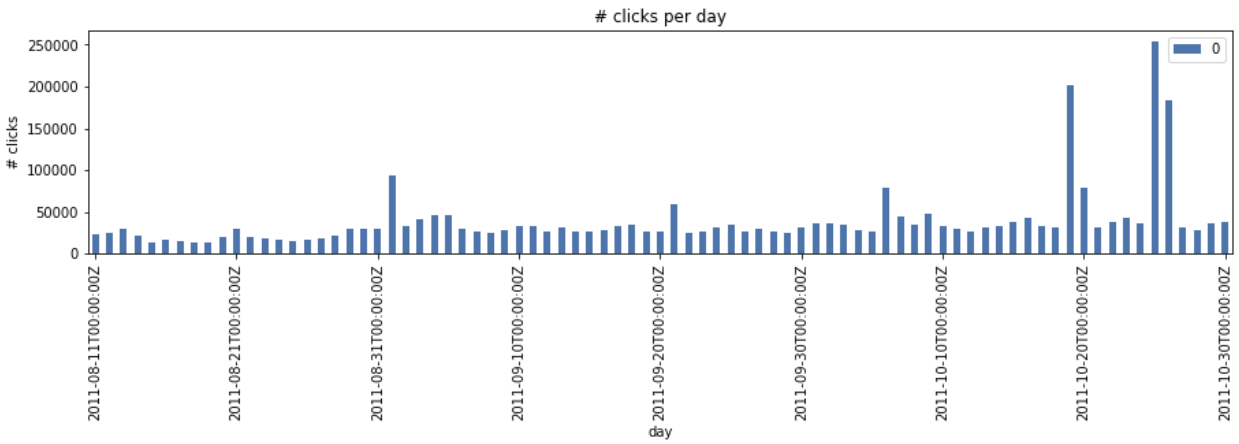
For a brief, yet effective introduction to the fundamentals of Learning to Rank and Solr's LTR capabilities, watch Diego Ceccarelli and Michael Nilsson's talk at https://berlinbuzzwords.de/17/session/apache-solr-learning-rank-win.

Learning to Rank Tutorial Setup

The end-to-end Learning to Rank tutorial code with setup instructions can be found at https://github.com/lucidworks/fusion-ltr-webinar.

Under **Fusion/Solr Setup**, we walk you through the initial setup process for enabling LTR for your Fusion/Solr instance. Fusion's Solr configuration administration UI makes configuring LTR far more convenient and straightforward.

For our tutorial, we use an ecommerce dataset provided by Kaggle and Best Buy which can be downloaded at https://www.kaggle.com/c/acm-sf-chapter-hackathon-big. Follow the instructions under **Index Data** to index the product catalog and click signals. The dataset contains 1,275,077 products, and 3,108,779 signal events— all clicks over a 2 month time period:



In order to train a Learning to Rank model, we need a set of relevance judgments, known as ground truth (for more information, see https://nlp.stanford.edu/IR-book/html/htmledition/information-retrieval-system-evaluation-1.html). Ground truth is necessary in order to measure and evaluate relevance in a repeatable and holistic manner. Ground truth is typically obtained in two ways:

- **Manually:** While producing relevance judgments may be time consuming, coming up with a representative sample of queries with high business value and identifying relevant documents for each usually takes just a few days and is absolutely necessary to measure the impact of any search relevancy changes, even if you don't use a Learning to Rank approach. In the case of LTR, a machine learning algorithm is used to optimize a performance metric (like Precision/Recall/F1) over your ground truth set with the available relevancy factors (called "features", discussed later in this tutorial).

- **Implicitly:** Instead of producing relevance judgments manually, user-generated signals may be used as implicit labels. Using signals to train Learning to Rank models is a common practice and supported by academic literature (i.e. https://www.cs.cornell.edu/people/tj/publications/radlinski_joachims_05a.pdf), although is generally of lower quality and more noisy than human expert produced relevance judgments. It may be possible to improve the reliability of click-based labels by incorporating other sources of supervision using impression and position data.

For our tutorial, we will use signal clicks as ground truth. Here is a sample of 10 signal clicks:

query	doc_id	date
monster ibeat earbuds dre beats -white	9492426	2011-10-31T10:44:53.990Z
ps3	3158734	2011-10-31T10:43:50.680Z
sony waio	3519969	2011-10-31T10:43:37.920Z
blackberry torch	3011735	2011-10-31T10:42:32.680Z
gunnar	1177301	2011-10-31T10:42:07.730Z
web camera	1722057	2011-10-31T10:41:22.410Z
sony dock	2284337	2011-10-31T10:39:20.630Z
small refrigerators	9723536	2011-10-31T10:39:13.830Z
sony dock	3316142	2011-10-31T10:38:57.540Z
xbox live	1419578	2011-10-31T10:37:55.590Z

Each row corresponds to a click event where a user clicked on doc_id when searching for query. In using these clicks as training labels, we are asserting that the most relevant document for each query is the one that the user clicked. We will be using machine learning to learn a complex function that optimizes the rank of return for these queries such that the "relevant" documents are more likely to be ranked before irrelevant documents.

Learning To Rank Training Workflow

Steps for learning and deploying an LTR model:

1. Feature definition: Define features (in JSON) and upload to Solr
2. Feature extraction: Extract <query, document> feature vectors from Solr for each query in our ground truth set
3. Core model training:Learn model from feature vectors
4. Upload model: Upload model as JSON to Solr to for query-time reranking

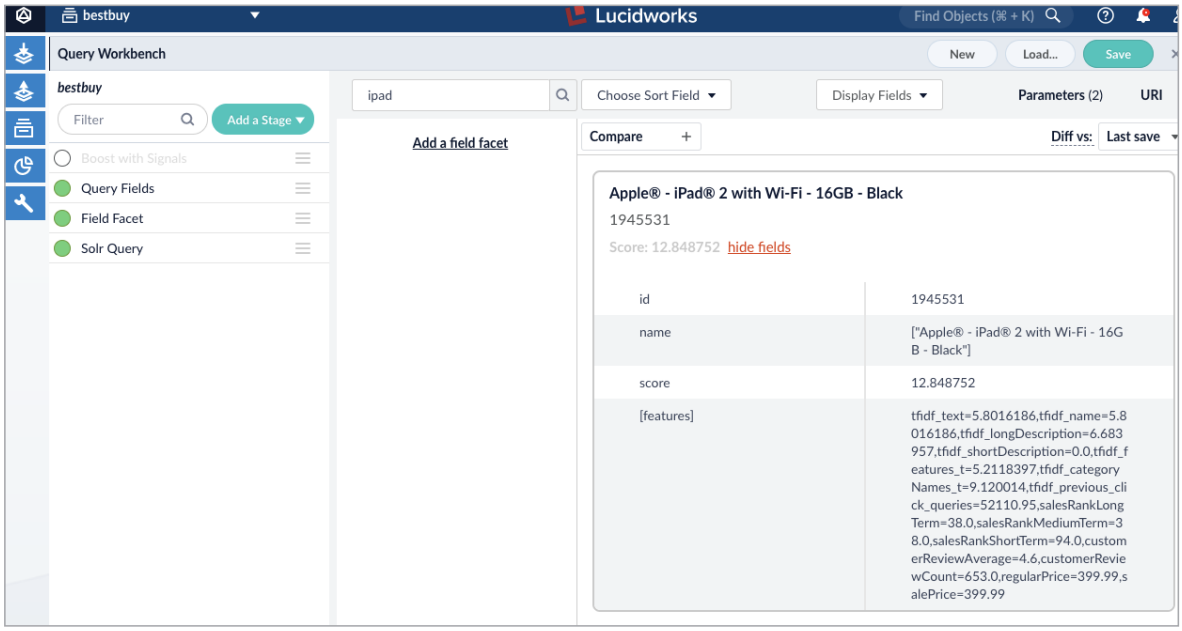
Feature Definition

Features are a list of relevancy factors our LTR algorithm automatically assigns weights to. Features are defined in JSON and uploaded to Solr. This can be done easily using Fusion—see **https://github.com/lucidworks/fusion-ltr-webinar#add-features**. The following are the features defined for this tutorial:

- tfidf_name: Solr score of <query, name field of product>
- tfidf_longDescription: Solr score of <query, longDescription field of product>
- tfidf_shortDescription: Solr score of <query, shortDescription field of product>
- tfidf_categoryNames: Solr score of <query, categoryNames field of product>
- tfidf_previous_click_queries: Solr score of <query, previous_click_queries field of product>, which is derived using Fusion's signals capabilities. See **https://github.com/lucidworks/fusion-ltr-webinar/blob/master/Data%20Setup.ipynb** for info on how this field is indexed.
- salesRankShortTerm: static value of salesRankShortTerm field indexed with each product
- salesRankLongTerm: static value of salesRankLongTerm field indexed with each product
- customerAverage: static value of customerAverage field indexed with each product
- customerReviewCount: static value of customerReviewCount field indexed with each product

Feature Extraction and Model Training

Follow the instructions at https://github.com/lucidworks/fusion-ltr-webinar#setup-fusion-query-pipeline-to-output-feature-vectors to configure your default Fusion query pipeline to output feature vectors for each query. If configured properly, submitting a query should yield an additional field returned for each document called "[features]", which contains the feature vector represented as a comma separated list of <key, value> pairs.



Now, Fusion is configured properly and you are ready to run the feature extraction and model training notebooks, referenced under https://github.com/lucidworks/fusion-ltr-webinar#execute-ltr-training-pipeline.

Feature extraction is a very CPU intensive operation, which requires Solr to calculate feature vectors for every ground truth query, which usually number in the thousands of queries. The feature extraction function for this tutorial demonstrates how to parallelize this process using Fusion's Spark cluster.

Core model training involves:

1. Pre-processing feature vectors
    - Downsampling negative class – Include N randomly sampled negative instances for each query_id
    - Z-score normalizing feature values, if necessary
2. Splitting feature vectors into train and test sets

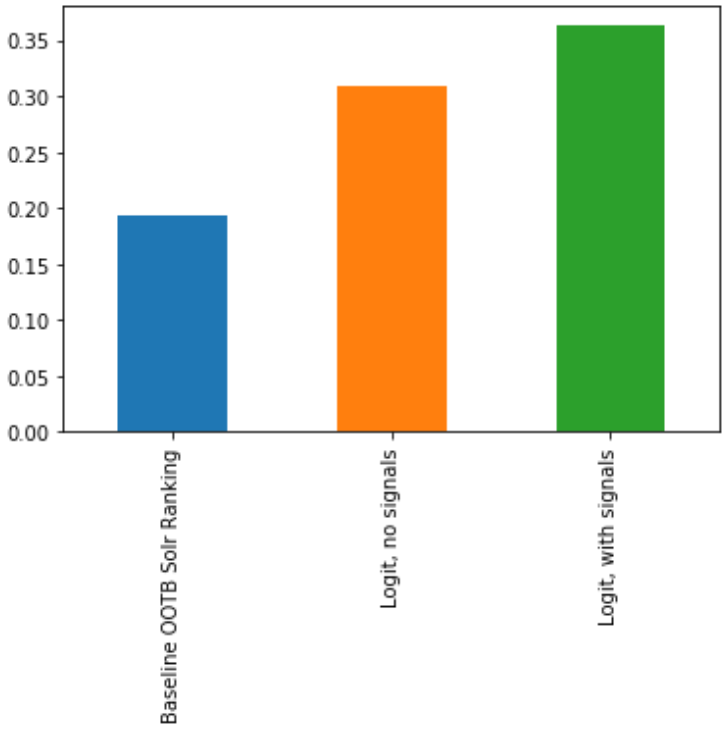
3. Training a simple pointwise Logistic Regression model

4. Evaluating results using recall@K on held-out test set as evaluation metric

In the model training notebook, we demonstrate the differences between these models:

- Solr out-of-the-box BM25 ranking using textual features only
- Logistic Regression using all features except the signals feature
- Logistic Regression using all features

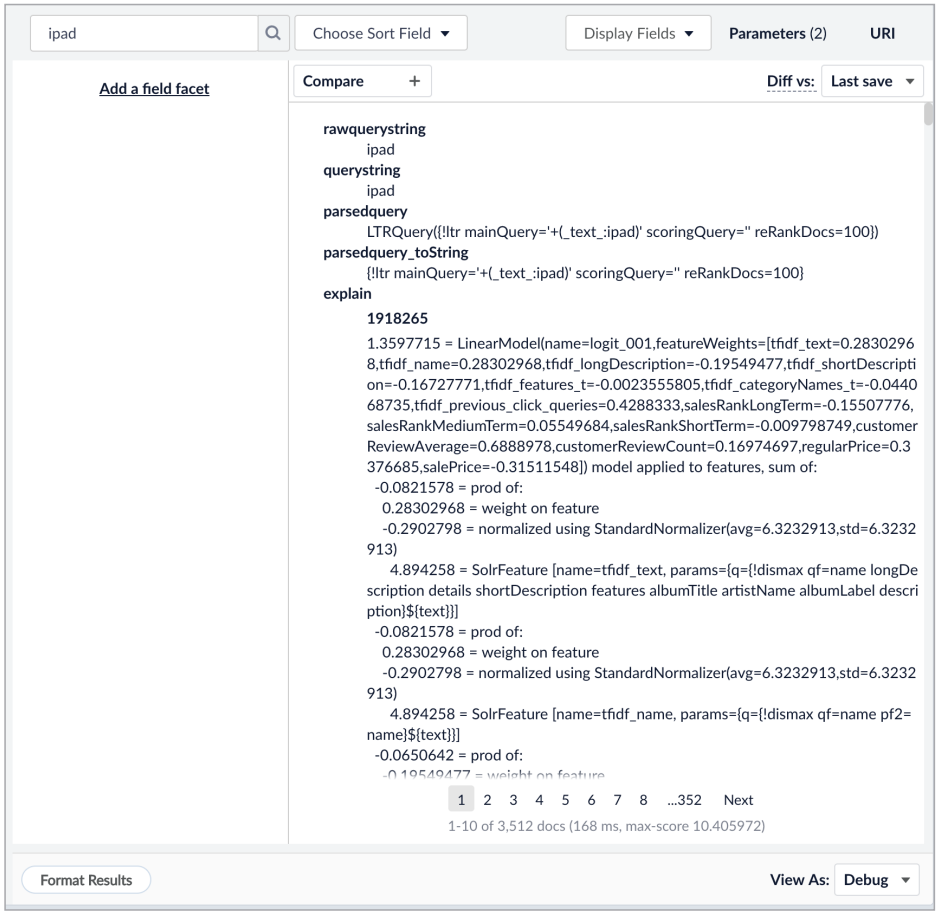
As the following figure shows, there are significant relevance improvements observed over OOTB Solr BM25 ranking, and even over the Logistic Regression trained model when using signal features. While the goal of this tutorial is to demonstrate how to implement Learning to Rank, and not necessarily how to produce the most accurate model, it is useful to note that leveraging Fusion's signals, even in a relatively casual way, can easily yield significant gains in relevance.



Once the model is trained, the best model is uploaded to Solr so that it can be made available to the Solr Learning to Rank component for query-time reranking of search results.

Enable Query-time Reranking

To enable query-time reranking, follow the instructions at https://github.com/lucidworks/fusion-ltr-webinar#update-fusion-query-pipeline-to-enable-query-time-reranking-using-trained-ltr-model. Enabling query-time reranking involves adding an additional query pipeline stage that adds a "rq" parameter that references the model that was just uploaded. To verify that query-time reranking is active, run queries in debug mode and verify that the scoring explanations make reference to "LinearModel":



Conclusion

Learning to Rank is a powerful tool in the toolbox of any search engineer looking to improve search relevance without a series of manual interventions. By using signals of user behavior (such as what they clicked) and features of your documents, you can teach the machine to do the relevance tuning for you and achieve better results than other techniques alone.

Get Started or Learn More

For more information or to start using Lucidworks Fusion, contact us today to learn more at lucidworks.com/contact or call **415-329-6515**.