Optimizing Search Relevance for Enterprise Recruiting

Parag Namjoshi, Hamdi Jenzri, Adam Baker, Namrata Ghadi, Manjunath Balasubramaniam, and Harikrishna Narayanan

Workday Inc., Pleasanton CA 94588, USA
parag.namjoshi@workday.com, hamdi.jenzri@workday.com, adam.baker@workday.com, namrata.ghadi@workday.com, manju.balasubramaniam@workday.com, harikrishna.narayan@workday.com

Abstract. Workday's recruiting product is used by recruiters to find candidates that best match a given job requisition and by candidates to find jobs that match their profiles. Surfacing precise results, in addition to good ranking, is a critical product need. We have integrated several custom–built Named Entity Recognition (NER) models as part of query pre–processing to dramatically improve precision (and ranking) of search results. Furthermore, Workday is a multi–tenanted system with strong privacy and isolation guarantees. Workday does not have direct access to tenant data, including query data. This makes traditional approaches to NER training and validation impossible. We propose a novel scheme to train, validate, and calibrate our models for every one of our over eight hundred tenants. We show that our best models achieve significant error reduction compared to baseline NER models.

Keywords: Named Entity Recognition · Search Relevance · Bayesian Models · Conditional Random Fields · Recruiting Search

1 Introduction

Workday's recruiting solution caters to a variety of users such as recruiters looking for candidates and candidates looking for jobs. Job requisitions and candidate profiles are indexed in tenanted Elasticsearch clusters and have structured fields and free–text documents against which queries are run. For example, candidate profile has structured fields for Name, Phone Number, Email, Current Job title, Past Job titles, Skills, Degrees, Majors, Schools, Location etc. Candidate profiles also include unstructured documents such as Resumes and cover letters. Data of each customer (tenant) — candidate profiles, job requisitions, and search queries — is logically isolated and enjoys strong privacy guarantees. Hence, unlike in consumer web SAAS providers, Workday does not directly examine or use tenant data for labeling. Instead, we use independently sourced tables of entities of interest to build our models. We present a novel scheme to validate and calibrate our models for each tenant while satisfying privacy and isolation requirements.

Search queries are entered in a single search box and users prefer not to have to tag the queries themselves, i.e., users would prefer to type *Software Engineer Rust* rather than *job-title: "Software Engineer" and skill: "Rust"*.

User research has revealed three major types of queries issued to our system.

1. Recruiters trying to find a specific candidate's profile (**candidate query**)
2. Recruiters looking to identify suitable candidates, from the pool of profiles already in the tenant, for specific job requisitions (**sourcing query**)
3. Candidates searching for jobs suited to their skills, experience, and location (**job query**)

For candidate queries, recruiters search by name, phone number, or email and expect exactly one result. Following the taxonomy of Broder [1], candidate query is a *navigational query.*

Sourcing queries are issued by recruiters looking to identify candidates for specific job requisitions. They typically are large Boolean queries, involving multiple job titles, skills, and perhaps, a location. They are sometimes augmented by other entities like degrees, majors, companies, and universities. Following Broder's taxonomy [1], a sourcing query is an *informational query.* For sourcing queries, our customer research showed that recruiters strongly preferred highly precise results. They interactively refine the search query until results are narrowed to one or two hundred candidates. They then look at each profile before making their decisions about candidates' suitability. Recruiters prefer a very high degree of control during this process. For example, a likely target of query *Software Engineer Rust* is a candidate whose job–title is "Software Engineer" and who lists the "Rust programming language" as a skill; not a mechanical engineer who specializes in corrosion control and mentions word "software" in her profile. We need to develop a very precise understanding of users' intent for each phrase in the query to narrow the search results to the appropriate set.

Job queries are traditional *informational queries* where relevance ranking is the primary concern. Serving precise results, while not critical, is still useful.

We will review relevant literature in section 2. We will describe our notation, training dataset in section 3. We will also describe three models: a baseline model, a Bayesian model based on occurrence of different words in different corpora, and a model based on Conditional Random Fields (CRFs). Finally, in section 4 we present our novel validation methodology and compare the results of different models and present our conclusions in section 5.

2 Related Work

Query understanding has long been recognized as a critical component for a successful search engine. This problem is closely related to the well–researched Named Entities Recognition (NER) problem in NLP [4].

Common schemes for query understanding rely upon relevance feedback mechanisms, e.g., [7, 9, 3, 8]. Our approach is complimentary to these approaches and addresses precision issues in a very direct way.

Table–driven query tagging approach of [11] is most similar to our approach. They do depend on access to query log data which, is not available to us due to privacy constraints.

Conditional Random Field (CRF) [12] is a well–known technique for sequence tagging. CRFs have been applied to web query tagging e.g., [5] using a semi-supervised approach for training. Unlike [5], we construct synthetic queries as described in section 3.5 due to unavailability of query logs.

3 Query Intent Models

Understanding the user's intent allows us to improve the formulation of the Elasticsearch query to improve search results. For instance, understanding that a query is a candidate's name would allow us to search in the *name* field only in Elasticsearch, delivering more relevant and precise results than if we were to search across all fields.

The tagger implements learning models trained to recognize entities like *company, degree, job title, location, name, school, skill, ...* while reporting a confidence score for the parse. It also uses regular expressions to detect *email*s and the *libphonenumber* library [2] to detect *phone* numbers. It assumes that queries are **dense** with named entities, since we expect recruiters to query "java developer seattle" rather than "I need a Java developer in the Seattle area".

3.1 Notation

We introduce the following notation when describing all models in this section:

- $\mathcal{Y} = \{Name, JobTitle, Skill, Location, \ldots\}$, the set of intent tags.
- A query is a sequence of tokens $x_1, \ldots, x_n$. We tokenize a query by splitting on white-space.
- $\|s\|$ is the number of tokens in a sub-query s.
- A segment $s = x_i, \ldots, x_{i+\|s\|-1}$ is a sub-query denoting a single entity that will receive a single intent tag.
- We use the shorthand $P(y \mid x) = P(Y_i = y \mid X_i = x)$ where ys refer to values of inferred tag variables and xs refer to observed variables.
- For CRF evidence feature vectors (section 3.5), we have k features per tag f_1 through f_k. We use
$$\big[f_1(y); \ldots; f_k(y)\big]_{y\in\mathcal{Y}} \tag{1}$$
to mean the vector with each feature f_i repeated for each $y \in \mathcal{Y}$. So
$$\begin{aligned} f(x_i) &= \big[P(x_i \mid y); P(x_{i-1}, x_i \mid y)\big]_{y\in\{y_1,y_2\}} \\ &= \big[P(x_i \mid y_1); P(x_{i-1}, x_i \mid y_1); P(x_i \mid y_2); P(x_{i-1}, x_i \mid y_2)\big] \end{aligned} \tag{2}$$

3.2 Training data

Our training data comprises of large (but somewhat noisy) **intent tables** of Name, Job title, Skill, Degree, Major, School, Company, and Location and their approximate frequencies. The dataset is proprietary and represents a snapshot which reasonably approximates real-world frequencies of various entities. The data was sourced independent of all tenants.

We estimate $P(x \mid y)$ as the relative unigram frequency of the token x in the table for tag y. Similarly, we estimate $P(\|s\| \mid y)$ for a segment s as the frequency of entities with $\|s\|$ tokens in the table for tag y. For bigrams, we add a boundary token, denoted #, to the beginning and end of each entity phrase and estimate $P(x_{i-1}, x_i \mid y)$ as the relative frequency of the bigram $x_{i-1}x_i$ in table y.

Because all models described below rely on looking up tokens in our entity tables, we have no evidence for classifying tokens that occur in none of our tables. We label such tokens *Unknown* and fall back to the general document search with that token matching all fields.

3.3 The Baseline

Our simple baseline model uses unigram probabilities to assign the most likely label independently to each token in the query. Bayes' rule gives us a probability distribution over tags for token x:

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{\sum\limits_{y \in \mathcal{Y}} P(x \mid y)P(y)} \tag{3}$$

We choose the most probable tag for each token independently and then combine adjacent tokens with the same tag into a single segment. The segment scores are the product of the tag probability for each token in the segment. For our baseline model we use a uniform prior.

3.4 Naive Bayes

Suppose a user quotes a sub-part of a query. Now we know that this phrase is a coherent segment with a single label, so the token labels must agree. We also know the length of the phrase and can condition on that, since a person's name will typically be shorter than a job title. We can use a naive Bayes classifier to label the quoted segment s according to the formula:

$$P(y \mid s) = \frac{P(y)P(\|s\| \mid y) \prod\limits_{x \in s} P(x \mid y)}{\sum\limits_{y \in \mathcal{Y}} P(y)P(\|s\| \mid y) \prod\limits_{x \in s} P(x \mid y)} \tag{4}$$

To generalize to unquoted queries, we need to consider every possible segmentation. We tried a Viterbi-like algorithm for finding the best possible segmentation, but found our unigram data insufficient to overcome the prior length expectations implied by the model, resulting in poor labelings. Instead we marginalize out the segmentation information to obtain probability distributions over tags for each token, and combine the token tags using the same heuristic as the baseline model. We find the segment length information still improved the model over the baseline.

3.5 Conditional Random Fields model

Conditional Random Fields (CRFs) [12] are undirected discriminative graphical models that efficiently capture the inter-dependency between the features in natural language data. This makes CRFs a good candidate to perform named entity recognition on sequential data, such as recruiting queries.

Training CRFs requires high quality labeled input dataset. Constrained by the restrictions on the use of production recruiting queries for training, we build an input data generation framework to simulate labeled train and test set. We specify a query mix, which is a distribution over sequences of entity types. Queries are constructed by randomly sampling entities from our training tables and concatenating them as necessary to create queries in the proportions specified by the query mix. For example, if the query mix specifies 20% $Name$ queries, and 10% $JobTitle, Skill$ queries (among other query types), and we wanted 50 queries, we'd sample 10 names, 5 job titles, and 5 skills from the training tables, and use the names on their own as training queries and concatenate each of the 5 job titles with one of the 5 skills for a training query.

We define a CRF model template using the Factorie [6] framework. Our model template defines two types of weights to infer: 1) transition weights between tags and 2) evidence weights between a token and its corresponding tag. To infer segmentation and tags in one pass we use a Begin, Inside, Outside (BIO) [10] encoding for our CRF tags. The CRF tag set is

$$\mathcal{Y}_{CRF} = \mathcal{Y} \times \{Begin, Inside\} \cup \{Outside\} \tag{5}$$

Because we assume queries are dense, we interpret anything $Outside$ to actually be unknown. We reserve $Unknown$ for out-of-vocabulary tokens (section 3.2). It's also used when the tags inferred for a query contain an $Inside$ tag without a corresponding preceding $Begin$ tag. All such illegal $Inside$ tags become $Unknown$ and are searched against all fields.

We train two types of CRF with different sets of evidence features for each token. The first model is a $Unigram$ model, which defines the evidence feature vector for the ith token x_i as:

$$f_{Uni}(x_i) = \big[\log P(x_{i-1} \mid y); \log P(x_i \mid y); \log P(x_{i+1} \mid y)\big]_{y \in \mathcal{Y}} \tag{6}$$

For the first and last tokens, the leading and trailing features are N/A, respectively.

The Bigram model defines the evidence feature vector for x_i as:

$$\begin{aligned} f_{Bi}(x_i) = \big[&\log P(x_i \mid y); \log P(\#, x_i \mid y); \log P(x_i, \# \mid y); \\ &\log P(x_{i-1}, x_i \mid y); \log P(x_i, x_{i+1} \mid y)\big]_{y \in \mathcal{Y}} \\ &\oplus \big[\log \textstyle\sum_{y \in \mathcal{Y}} P(x_{i-1}, \# \mid y)\big] \end{aligned} \tag{7}$$

where $\oplus$ denotes vector concatenation.

To train the model, we used LBFGS algorithm to optimize for the Hamming distance objective with L2 regularization. We use the Viterbi variant of belief propagation for tag inference.

4 Results

4.1 Model Validation and Calibration Methodology

Our models directly rely on tokens from vocabulary (intent tables) and this fact poses interesting challenges for model validation and calibration. Models perform poorly when evaluated against out–of–vocabulary tokens. So, it is a challenge to construct a meaningful test/holdout sets needed by traditional cross–validation methodologies. Ideally, we would validate and calibrate our models using real user search queries annotated with tags. Unfortunately, user queries are considered tenanted (private) data and we are not allowed to look at them, let alone manually tag them for validation.

Instead, we use structured tenant data as a proxy for queries issued in that tenant. Every week, we automatically synthesize tagged queries per tenant, to serve as a validation and calibration set (similar to our construction of synthetic training queries for the CRF described in section 3.5). Queries are created by randomly sampling appropriate structured fields to match a query mix defined *a priori* based on user research. There are two reasons for running separate validation and calibration jobs per tenant: 1. commingling data across tenants is contractually prohibited and 2. different tenants have different data distributions. Once the validation and calibration set is generated, the query intent model is automatically scored based on how well it segments and labels each synthetic query.

4.2 Performance Metrics

Following [13], we measure precision, recall, and F-measure for our query tagger. Precision is the proportion of predicted segments that are correct, and recall is the proportion of segments in the ground truth that were correctly segmented and tagged, as in [13]. In contrast to [13], we calculate each metric per tag. We calibrate the models per tag per tenant by determining the threshold that optimizes for F-measure.

4.3 Tenant–specific Model Calibration

We compute validation and calibration metrics weekly for every tenant. Validation checks the F-measure for each tenant, and when the metrics are below specified threshold, all queries for the tenant fall back to default method of searching across all fields. Calibration searches for a threshold for each tag and tenant that optimizes an F-measure objective for that tag. The β parameter for the F-measure differs for each tag based on our estimates of the risk trade–offs of running too narrow or too broad a query.

When the validation results are high enough for a tenant, model provides a parse along with the optimal calibration thresholds for every tag. Confidence score for each segment is compared to it's corresponding threshold. If confidence score for a segment is above the threshold, we restrict search for that segment to specific field(s) (improved precision). A significant side–benefit of restricting search to a subset of fields is improved latency and throughput. If the confidence score for a segment is below the threshold, we apply appropriate boosts to the matches in those fields but also search all other fields (improved ranking).

4.4 Model Performance Comparisons

Table 1 summarizes the percentage of error reduction across our tags averaged over eight hundred production tenants, for the Bayes, CRF-Unigram and CRF-Bigram models compared to the baseline model. We report the outcome of running the validation and calibration jobs using the candidate profiles' data. In

Table 1. Average % of error reduction compared to the baseline model

Tag	**Naive Bayes**	**CRF-Unigram**	**CRF-Bigram**
company	**30.79**	5.37	-44.73
degree	**45.48**	-353.15	-547.32
job–title	**52.34**	36.41	21.84
location	**75.54**	36.68	-7.00
name	45.66	**59.45**	-190.96
school	30.99	**58.04**	24.13
skill	**43.35**	-112.06	-86.89

our current training setup, the naive Bayes model had the highest average reduction in error over the baseline model. CRF models did not show as much improvement over baseline due to the way they were trained.

5 Conclusion

In this paper, we presented a NER–based query understanding system for optimizing recruiting search. We identified the priority of highly precise results over just good ranking. We proposed and evaluated three types of NER models based on a large list of entities and their frequencies. First model is a simple unigram token tagger that we use as baseline. Our second model is based on naive Bayes algorithm that uses token length information, in addition to token frequencies, to improve segmentation and tagging. We also experimented with a CRF–based model for NER. We also proposed a novel method for validating and calibrating our models in a multi–tenant system with strong privacy and isolation requirements. Finally, we presented results achieved by our models in a production setting for more than eight hundred tenants.

References

1. Broder, A.: A taxonomy of web search. SIGIR Forum **36**(2), 3–10 (Sep 2002)
2. Google Inc: Google libphonenumber. https://opensource.google.com/projects/libphonenumber
3. Homa Baradaran Hashemi, Amir Asiaee, R.K.: Query intent detection using convolutional neural networks. In: International Conference on Web Search and Data Mining, Workshop on Query Understanding (2016)
4. Jurafsky, D., Martin, J.H.: Speech and Language Processing (2nd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2009)
5. Li, X., Wang, Y.Y., Acero, A.: Extracting structured information from user queries with semi-supervised conditional random fields. In: SIGIR. pp. 572–579 (2009)
6. McCallum, A., Karl, S., Singh, S.: FACTORIE: Probabilistic programming via imperatively defined factor graphs. In: NIPS (2009)
7. Mendoza, M., Zamora, J.: Identifying the intent of a user query using support vector machines. In: Karlgren J., Tarhio J., H.H. (ed.) String Processing and Information Retrieval, vol. 5721. Springer (2009)
8. Pournajaf, L., Aljadda, K., Korayem, M.: Long tail query enrichment for semantic job search. In: IEEE International Conference on Data Mining Workshops (ICDMW). vol. 00, pp. 215–220 (Nov 2017)
9. Radlinski, F., Szummer, M., Craswell, N.: Inferring query intent from reformulations and clicks. In: Proc. 19th Annual International World Wide Web Conference. pp. 1171–1172. Association for Computing Machinery, Inc. (April 2010)
10. Ramshaw, L.A., Marcus, M.P.: Text Chunking Using Transformation-Based Learning, pp. 157–176. Springer Netherlands, Dordrecht (1999)
11. Sarkas, N., Paparizos, S., Tsaparas, P.: Structured annotations of web queries. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. pp. 771–782. SIGMOD, ACM (2010)
12. Sutton, C., McCallum, A.: An introduction to conditional random fields. Foundations and Trends in Machine Learning **4**(4), 267–373 (Apr 2012)
13. Tjong Kim Sang, E.F., De Meulder, F.: Introduction to the conll-2003 shared task: Language-independent named entity recognition. In: HLT-NAACL. CONLL, vol. 4, pp. 142–147. Association for Computational Linguistics (2003)