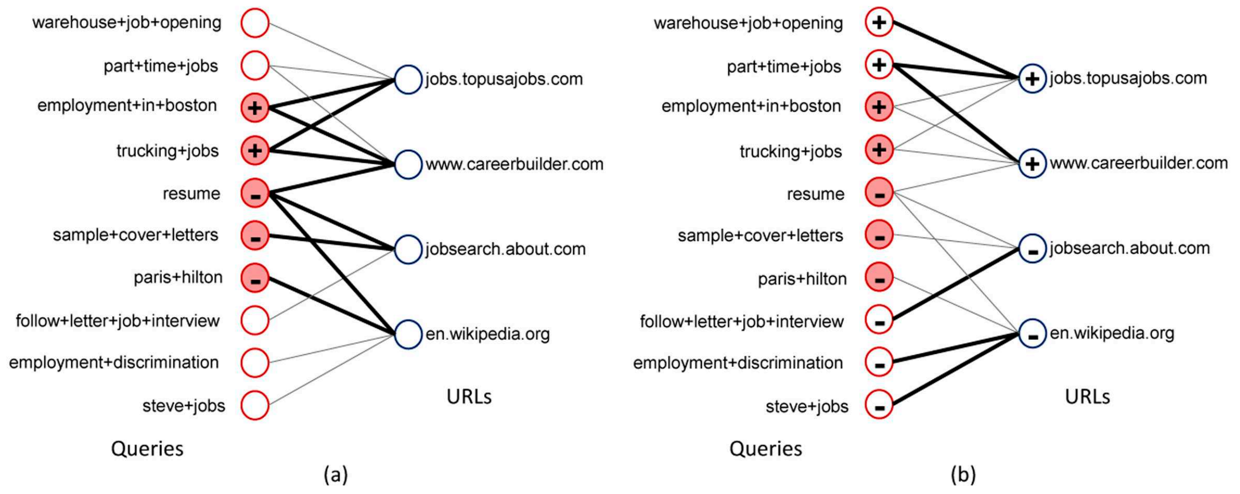

Figure 1: An intuitive example of learning *job intent* from a click graph. (a) A click graph where shaded (red) nodes represent seed queries labeled as + or -; (b) Label information +/- is propagated from seed queries to URLs and then to unlabeled queries. Note that queries like "steve jobs" and "employment discrimination" are correctly inferred as negative, as they both connect with "en.wikipedia.org" which is inferred to be a negative URL. In practice, the inference is performed iteratively using soft labels rather than hard ones.

A major contribution of this work is that we use a principled approach that automatically labels *a large amount* of queries in a click graph, which are then used in training content-based classifiers. Our approach is inspired by graph-based semi-supervised learning techniques [17, 21, 20, 12], and especially by [20]. The key idea is that we manually label a small set of *seed queries* in a click graph, and we iteratively propagate the label information to other queries until a global equilibrium state is achieved. An intuitive illustration of this idea is given in Figure 1. One technical novelty of our work is that we regularize the learning with content-based classification. When a click graph is noisy or sparse (as is usually the case [8]), such regularization would prevent a click graph from propagating erroneous labels.

We demonstrate the effectiveness of our algorithms in two applications, product intent and job intent classification, while our approach is general enough to be applied to other tasks. In both cases, we expand the training data with automatically labeled queries by over two orders of magnitude, leading to significant improvements in classification performance. An additional finding is that with a large amount of training data obtained in this fashion, classifiers using *only* query lexcial features can work remarkably well. The rest of the paper is organized as follows. Section 2 presents our learning algorithms that infer query intent from click graphs; Section 3 discusses related work; Section 4 describes our evaluation methodology, experiments and results, followed by concluding remarks in Section 5.

2. METHODOLOGY

As mentioned in the introduction, the ultimate goal of our work is to learn a content-based classifier. In other words, given a feature representation of queries, we desire to learn a classification function that not only correctly classifies observed queries, but also generalizes to unseen queries. In this work, we use a maximum entropy classifier which models the conditional probability as follows,

$$p_\lambda(y|x) = \frac{\exp^{\sum_j \lambda_j \phi_j(x,y)}}{\sum_y \exp^{\sum_j \lambda_j \phi_j(x,y)}} \qquad (1)$$

where x denotes an input query, y denotes classes (binary in our case), and $\phi_j(x, y)$, $j = 1, 2, \ldots$, represent a set of query lexical features. The classifier is parameterized by λ which can be estimated using a maximum likelihood objective. A detailed discussion of using maximum entropy model for text classification can be found in [14].

The lexical features used in this work are n-grams, although they can be easily substituted by other features. An n-gram refers to a consecutive n word tokens that appear together, and we treat sentence start "<s>" and sentence end "</s>" as two special word tokens. For example, the query "trucking jobs" will activate a number of features including 1) unigrams: "trucking" and "jobs"; 2) bigrams: "<s>+trucking", "trucking+jobs" and "jobs+</s>"; while higher-order n-grams can be derived similarly. An n-gram is naturally smoothed with its lower-order counterparts via the linear interpolation $\sum_j \lambda_j \phi_j(x, y)$. Such lexical features, despite their sparseness, are a relatively unbiased representation of queries. One added advantage of using such features is that classification can be done *prior to* information retrieval [4]. In fact, using query lexical features can yield remarkable classification performance given abundant training data, as will be demonstrated in Section 4.4.

In this section, we focus our attention on how to obtain a large amount of training data in an automated fashion by the use of click graphs. We formally formulate our learning objectives, and provide solutions with two simple algorithms. Additionally, we discuss practical considerations when learning with click graph.

2.1 Learning with click graphs

From a collection of click-through data, we can construct a bipartite graph $G = (X \bigcup Z, E)$, where $X = \{x_i\}_{i=1}^m$ represents a set of queries and $Z = \{z_k\}_{k=1}^n$ a set of URLs (or clustered URLs). Each edge in E connects a vertex in X with one in Z, and there is no edge between two vertices in the same set. Let W represent an $m \times n$ weight matrix, in which element $w_{i,k}$ equals the click count associating vertices x_i and z_k. Furthermore, we assume that a small set of *seed queries*, denoted as X_L, are manually labeled as positive or negative with respect to a specific intent. How to select seed queries is task-dependent, which will be discussed in Section 4. Given the click graph and the labeled set X_L (an example of which is given in Figure 1 (a)), our goal is to automatically assign labels to queries in the set $X \setminus X_L$.

The problem of learning from labeled and unlabeled data has been investigated in a number of works [17, 21, 20, 12]. Our objective and algorithm presented in this section are heavily influenced by the work of [20]. Formally, we let F denote an $m \times 2$ matrix, in which element $f_{i,y}$ is a non-negative, real number indicating the "likelihood" that x_i belongs to class y. Given F, the posterior probability $p(y|x_i)$ can be computed by $f_{i,y}/(f_{i,+1} + f_{i,-1})$. We further use F^0 to denote an instantiation of F that is consistent with manual labels: for $x_i \in X_L$ that are labeled as positive, we have $f^0_{i,+1} = 1 - f^0_{i,-1} = 1$; for those labeled as negative, we have $f^0_{i,+1} = 1 - f^0_{i,-1} = 0$; and for all unlabeled queries $x_i \in X \setminus X_L$, we have $f^0_{i,+1} = f^0_{i,-1} = 0$. Our goal, consequently, becomes to estimate F given G and F^0.

Furthermore, we define a *normalized click count matrix* $B = D^{-1/2}W$. Here D is a diagonal matrix in which element $d_{i,i}$ equals the sum of all elements in the i^{th} row (or column) of WW^T. Intuitively, $d_{i,i}$ can be understood as the "volume" of all length-of-two paths that start at x_i. The reason we use such a normalization will be evident shortly. Given the definition, our algorithm works as follows,

```
Algorithm 1
Input: matrix $F^0$ and matrix $B = D^{-1/2}W$
Output: $F^*$
 1: Initialize $F$ by $F^0$;
 2: repeat
 3:     Compute $H^i = B^T F^{i-1}$;
 4:     Compute $F^i = \alpha B H^i + (1-\alpha)F^0$, where $\alpha \in [0, 1)$;
 5: until the sequence $F^i$ converges to $F^*$
```

To show the convergence, we see that in the linear algebra sense $A = D^{-1/2}WW^TD^{-1/2}$ is *similar* to the stochastic matrix $D^{1/2}WW^T$. Therefore, the largest eigenvalue of A is 1, and all other eigenvalues are in $[0, 1)$ (since A is also positive semi-definite). Consequently, it is easy to see that the sequence of F^i converges to $F^* = (1-\alpha)(1-\alpha A)^{-1}F^0$ asymptotically.

Notice that line 3 and line 4 of Algorithm 1 can be merged into a single step $F^i = \alpha A F^{i-1} + (1-\alpha)F^0$ where $A = BB^T$. But when $n \ll m$ (far fewer URLs than queries), the two-step approach is computationally more efficient. Moreover, since B is a sparse matrix (most elements are zero), the computation complexity of each iteration of Algorithm 1 is linear in the number of graph edges.

It has been shown in [20] that F^* is an optimal solution of minimizing the following objective,

$$Q(F) = \alpha Q_1(F) + (1-\alpha)Q_2(F) \qquad (2)$$

where

$$\begin{aligned} Q_1(F) &= \frac{1}{2}\sum_{y=\pm1}\sum_{i,j=1}^{m}\left(\sum_{k=1}^{n} w_{i,k}w_{j,k}\right)\left\|\frac{f_{i,y}}{\sqrt{d_{i,i}}} - \frac{f_{j,y}}{\sqrt{d_{j,j}}}\right\|^2 \\ Q_2(F) &= \frac{1}{2}\sum_{y=\pm1}\sum_{i=1}^{m}\|f_{i,y} - f^0_{i,y}\|^2 \end{aligned}$$

The objective consists of two terms. First, minimizing $Q_1(F)$ alone gives $AF^*_1 = F^*_1$. This means both columns of F^*_1 are the principle eigenvector of A, recalling the fact that the largest eigenvalue of A is 1. From a different perspective, if we replace line 4 of Algorithm 1 by $F^i = BH^i$, the sequence F^i will converge to F^*_1. This is true for any value of F^0 as long as F^0 and F^*_1 are not orthogonal. In other words, $Q_1(F)$ asks F to be in the equilibrium state of the click graph. Secondly, $Q_2(F)$ regularizes F towards F^0. In this regard, $Q(F)$ is a tradeoff between the consistency with the intrinsic structure of the click graph and the consistency with manual labels.

Once F^* is obtained, we normalize its elements to obtain posterior probabilities $p(y|x_i)$, $i = 1..m$. In training the final maximum entropy classifier, we can either use these posterior probabilities directly, or we can clamp them to 0/1 values. In our experiments in Section 4 we chose the latter method for simplicity.

2.2 Content-based regularization

A click graph can be sparse — there might be missing edges between queries and URLs that are relevant. Moreover, user clicks are often noisy, which result in edges between irrelevant queries and URLs. Missing edges prevent correct label information from being propagated, while classification errors may arise from spurious edges.

To compensate for the sparsity and noise of a click graph, we regularize click graph learning by content-based classification. Specifically, we use $F^c(\lambda)$ to denote an $m \times 2$ matrix, representing the output of the maximum entropy classifier; each element $f^c_{i,y} = p_\lambda(y|x_i)$ is a classification function defined in Equation (1). Then we treat F^c as a prior of F and modify our objective in Equation (2) accordingly,

$$Q(F, \lambda) = \alpha Q_1(F) + (1-\alpha)Q_2(F, \lambda) \qquad (3)$$

where $Q_1(F)$ is the same as that in Equation (2) and $Q_2(F, \lambda)$ has the following form,

$$Q_2(F, \lambda) = \frac{1}{2}\sum_{y=\pm1}\sum_{i=1}^{m}\|f_{i,y} - f^c_{i,y}(\lambda)\|^2 \qquad (4)$$

The new objective $Q(F, \lambda)$ asks F to be consistent with the output of the maximum entropy model, while keeping with the intrinsic structure of the click graph.

The objective in Equation (3) can be optimized in an iterative fashion. Given an estimate F^*, the problem is reduced to estimating maximum entropy model parameters λ that minimizes the quadratic loss in Equation (4). This is can be solved using stochastic gradient descent or other numerical methods. Next, given an estimate λ^*, the objective essentially becomes that in Equaton (2) except that F^0 is replace by $F^c(\lambda^*)$. Thus, we can optimize F and λ alternatively as follows.

Algorithm 2

```
Input: matrix $F^0$ and matrix $B = D^{-1/2}W$
Output: $F^*$ and $\lambda^*$
1: Initialize $F^* = F^0$, and initialize $\lambda$ as random;
2: repeat
3:     Find $\lambda^* = \underset{\lambda}{\operatorname{argmin}}\, Q(F^*, \lambda)$ using stochastic gradient descent;
4:     Find $F^* = \underset{\lambda}{\operatorname{argmin}}\, Q(F, \lambda^*)$ using Algorithm 1, where the input are $F^c(\lambda^*)$ and $B$;
5: until the value $Q(F^*, \lambda^*)$ converges
```

In contrast to Algorithm 1, this algorithm jointly optimizes F and λ, and the output λ^* gives the final maximum entropy classifier. The convergence is guaranteed since $Q(F, \lambda)$ is lower-bounded and its value is decreased every iteration. In practice, we use a more relaxed stopping criterion: at each iteration, we make binary classification for $\{x_i\}_{i=1}^m$ based on normalized F^*, and the algorithm stops when we observe no more change in our predictions.

2.3 Click graph construction

Having described the core algorithms, we now turn to practical issues on click graph construction. In practice, it is sometimes inefficient and unnecessary to apply our learning algorithms to a gigantic click graph constructed from the entire collection of click-through data. In this section, we present a practical method of building a compact click graph and iteratively expanding it, if necessary, until it reaches a desired size. We first present two pre-processing steps *i.e.* removing *navigational queries* and clustering URLs; then we discuss our method on click graph construction.

A query is considered navigational when a user is primarily interested in visiting a specific web page in mind. For example, "youtube" is likely to be a navigational query that refers to the URL "www.youtube.com." Such a query usually has a skewed click count on one URL, and the class membership of that URL can be excessively influenced by this single query. To avoid their adverse effect on our learning algorithms, we identify navigational queries based on measures proposed in [11] and remove them from our click graphs.

Secondly, we merge related URLs into clusters to compensate for the sparsity of a click graph. Specifically, if a set of URLs have exactly the same *top-*, *second-* and *third-level domain names*, we group them into a single node and add up their click counts accordingly. For example,

```
   nurse.jobs.topusajobs.com/*
 finance.jobs.topusajobs.com/*
   miami.jobs.topusajobs.com/*
```

are all grouped to a URL cluster **jobs.topusajobs.com**.

Finally, since the most reliable information of query classes resides in seed queries, it would be more efficient to apply our algorithms *only* to a relatively compact click graph that covers these queries. To this end, we start from the seed queries and iteratively expand click graph in the following fashion,

1. Initialize a query set $X' = X_L$ (seed query set), and initialize a URL set $Z' = \emptyset$;
2. Update Z' to be the set of URLs that are connected with X';
3. Update X' to be the set of queries that are connected with Z';
4. Iterate 2 and 3 until X' reaches a desired size;

In each iteration, we can prune away queries and/or URLs with edges fewer than a threshold. The final click graph to which the learning algorithms are applied consists of X', Z' and edges connecting between them.

3. RELATED WORK

In recent years, many research efforts in query classification have been devoted to enriching feature representation of queries. The 2005 KDD Cup inspired the use of the World Wide Web for query enrichment. The winning solution by Shen et al. [16] used search engine results as features, including pages, snippets and titles, and built classifiers based on a document taxonomy; then classifications in the document taxonomy were mapped to those in the target taxonomy. Broder et al. [7] transformed the problem of query classification to that of document classification which was solved directly in the target taxonomy. A comprehensive comparison of these methods can be found in [4]. Another way to enhance feature representation is the use of word cluster features [15, 2]. In such an approach, semantically similar words can be grouped into clusters, either by domain knowledge or by statistical methods, and be used as features to improve the generalization performance of a classifier.

On the other hand, there has been a surge of interest in semi-supervised learning that leverages both labeled and unlabeled data to improve classification performance. One widely used approach is *self-training* (or *bootstrapping*) [19]. This method iteratively trains a seed classifier using the labeled data, and uses high-confidence predictions on the unlabeled data to expand the training set. *Co-training* [6] improves over self-training by learning two separate classifiers on two independent sets of features; each classifier's predictions on unlabeled data are used to enlarge the training set of the other. Moreover, Beitzel et al. [5] proposed a semi-supervised approach tailored to query classification based on selectional preference. Another important school of semi-supervised learning method is based on graphs, including Markov random walks [17], label propagation [21], learning with local and global consistency [20] and manifold regularization [12]. While these works differ in their optimization objectives, they all share the same underlying assumption that if two samples are close in the intrinsic geometry of an input space, their conditional distributions will be similar. Our Algorithm 1, which classifies unlabeled data based on a bipartite graph, is technically inspired by the work of [20]. Moreover, our objective in Equation (3), which jointly performs graph-based learning and maximum entropy model training, is closely related to the learning paradigm proposed in [12]. The key difference is that we combine two orthogonal views in this learning paradigm — while learning with click graphs focuses on user click information, the maximum entropy model is constructed on the basis of query content information.

Another group of related work aims to combine content information with click information in clustering and classification. Beeferman et al. [3] and Wen et al. [10] used click patterns as features in complement to query terms for clustering. Xue et al. [18] proposed an *iterative reinforcement algorithm* on a click graph for document classification.

In their work, a content-based classifier was used to produce initial posterior probabilities, which were then interpolated with click graph predictions at each iteration step. One major distinction of our work is that we minimize a joint objective function with respect to a click graph and a content-based classifier. In our iterative algorithm (Algorithm 2), the click graph output is used as training data for the maximum entropy model, whose output is in turn used to regularize click graph learning.

4. EVALUATION

4.1 Applications

The query intent classifiers discussed in this work are making binary decisions regarding whether a query contains an intent that qualifies for a domain-specific search. At application time, they serve as frontend components of a search engine that have both general-purpose and vertical search functionalities. When a query is classified as positive with respect to a specific domain, the corresponding domain-specific search will be conducted at the backend, and will ideally return to user the most relevant and essential answers. In this work, we evaluate our learning algorithms in two applications, product intent and job intent classification, while our approach is general enough to be applied to other applications as well.

Product intent classification. A query with product intent is one that refers to any tangible product, or a class of products, which can be purchased in store or online. A number of positive examples (ignoring cases) are "ipod nano", "nike shoes on sale", "mercedes floormats", while negative ones are like "www.amazon.com", "apple inc", "soccer world cup." Query log analysis shows that approximately 5%-7% of the *distinct* web search queries contain product intent, although this number depends on time and search-engine. A major challenge to product intent classification is that product queries are rather diversified. Without substantial training data, learning a classifier using query lexical features only is likely to overfit, as will be shown in Section 4.4.

Job intent classification. We define a query to have job intent if the user is interested in finding certain types of job listings. For example, "trucking jobs", "employment in boston" are positive queries whereas "steve jobs", "employment discrimination" are not. Note that we treat queries such as "resume" and "sample cover letters" as *negative* though they are indirectly related to jobs. We estimate that roughly 0.2%-0.4% of the *distinct* web search queries have job intent. Unlike product queries, a vast majority of job queries contain key words or phrases such as "jobs" and "employment." When using such patterns in classification, however, one needs to be careful since they can also appear in negative examples, *e.g.* "steve jobs."

4.2 Data preparation

4.2.1 Click-through data

For semi-supervised learning, we collected a set of click-through data over a continuous period of time from the *Live Search* query log. After removing navigational queries and applying URL clustering as described in Section 2.3, this data set consists of 8 million distinct queries that have resulted in clicks, and 3 million distinct URL clusters that have been clicked on. There are 15 million distinct clicks (or query-URL pairs) which account for 32 million clicks in total. From this set of click-through data, we selected seed query sets for manual labeling, and constructed click graphs as described in Section 2.3, which we will present in detail shortly. In addition, we prepared a second set of click-through data, collected over a different time period, for measuring classification performance.

Amounts	Product intent	Job intent
Seed queries	2K (20% pos.)	300 (35% pos.)
Click graphs		
Query nodes	1,200K	600K
URL nodes	380	120
Query-URL edges	1,400K	700K
Total click count	3,200K	1,500K
Evaluation set queries	20K (6% pos.)	3K (30% pos.)
Expanded train set queries	300K (15% pos.)	60K (5% pos.)

Table 1: Configurations of seed query sets, click graphs, evaluation sets and expanded training sets for product intent and job intent classification. The number of "URL nodes" corresponds to that after clustering and pruning.

4.2.2 Seed query sets

Recall that we first need to manually label a small set of seed queries in order to apply our learning algorithms. Since seed query sets are presumably small, randomly sampling from query logs would lead to few positive queries, especially for job intent classification where the percentage of positive examples is only 0.2%-0.4%.

To compensate for this problem, we selected seed queries for training job intent classifiers in the following fashion. First, we obtained a number of job related websites (these can be easily obtained from the World Wide Web), and we collected queries that have landed on these websites. We then randomly sampled 200 of them for manual labeling. To mitigate the bias of our selected seed queries, we additionally sampled and labeled another 100 general web search queries (where an expectedly small portion are positive), and added them to the seed query set. In the end, we obtained 300 seed queries for training job intent classifiers, in which 35% are positive. Similarly, we obtained 2K seed queries for training product intent classifier in which 20% are positive.

It is worth mentioning our manual labeling procedure. In both applications, a user interface was created for human annotators where each query was presented along with retrieval results from two major search engines. Then human annotators looked at both results to make a binary judgment.

4.2.3 Click graphs

From the seed queries, we constructed click graphs using the method described in Section 2.3. We performed two such iterations for building the click graph for product intent classification, and only one iteration for the job case. In each iteration, we pruned away all URLs that have fewer than 3 edges, which greatly reduced the size of the resulting click graphs. The second row of Table 1 shows configurations of the resulting click graphs. Note that we are dealing with 1.4M/700K edges in product/job classification, which is the computation complexity of each iteration of Algorithm 1.

4.2.4 *Evaluation sets*

To measure classification performance, we separately prepared evaluation sets for human annotators to label, and we ensured that queries in these evaluation sets do not overlap with *any* seed queries. Furthermore, since the evaluation sets were sampled from a second set of click-through data, many queries therein do not even exist in the click graph used in semi-supervised learning. The amounts of evaluation data are shown in the third row of Table 1.

4.3 Evaluation metrics

The maximum entropy model output $p(y = 1|x_i)$, as defined in Equation (1), is the posterior probability that a query is positive with respect to a specific intent. We use an adjustable threshold γ on this output to balance the precision and recall. In other words, a query is considered positive when $p(y = 1|x_i) > \gamma$ and negative otherwise. By changing γ from 0 to 1, we are able to obtain the entire precision-recall curve which is used as a performance measure in our experiments. In some cases, we only report optimal F_α values as well as its precision and recall values owning to the lack of space. Here F_α is computed as

$$F_\alpha = (1 + \alpha) \cdot \text{precision} \cdot \text{recall}/(\alpha \cdot \text{precision} + \text{recall})$$

In our applications, we want to weight precision significantly higher than recall, as false positives are more costly than false negatives regarding user experience — it would appear strange to a user if the system displays vertical search results when the user does not have that intent. In our work, we use $F_{\alpha=0.2}$ values (meaning precision is weighted 5 times higher than recall) in Table 2 and Table 3, although using other comparable α's resulted in a similar trend.

4.4 Experiments and results

4.4.1 *Algorithm comparisons*

The main interest of work is to study how automatically labeled training data would impact classification performance. For a fair comparison, we used the same feature representation, specifically query lexical features, in all methods presented in this experiment. In product intent classification we used n-gram features with $n = 1, 2, 3$, while in the job case we used $n = 1$ and 2. Given the same feature representation, we compare classifiers which were trained on different types of training data:

1. **Manual labels (seed queries)**. We trained a maximum entropy classifier using seed queries only.

2. **Self-training**. We implemented the method of [19] based on query n-gram features. We first trained a maximum entropy classifier using seed queries, and then iteratively used high-confidence predictions on unlabeled queries to expand the training set.

3. **Algorithm 1**. We used a regularization coefficient $\alpha = 0.75$ in Equation (2). Note that in both Algorithm 1 and 2 we experimented with other regularization coefficients and found that $\alpha \in (0.5, 0.9)$ yielded similar good results. It took around 20 iterations for Algorithm 1 to reach a fixed point, and queries with high-confidence classification were selected and combined with seed queries to train a maximum entropy classifier.

4. **Algorithm 2**. As above, we used $\alpha = 0.75$ in Equation (3). Algorithm 2 ran 3 iterations (in the outer loop) before the predictions on queries stabilized. Again, high-confidence queries and seed queries were merged to train a maximum entropy classifier.

Figure 2: Product intent classification with training sets expanded by different algorithms

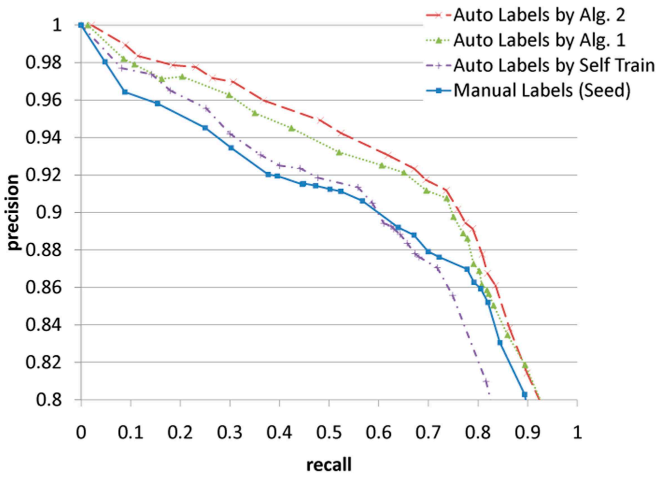

Figure 3: Job intent classification with training sets expanded by different algorithms

There are a number of implementation issues worth attention. First, in cases of self-training, Algorithm 1 and Algorithm 2, we all needed to set confidence thresholds to select the most likely positive and negative queries to expand the training sets, and such thresholds were chosen by cross-validation in our experiments. Secondly, since the selected queries were often excessively skewed toward negative ex-

# Distinct Queries	0	37K	75K	150K	300K
Precision	0.67	0.77	0.81	0.83	0.84
Recall	0.25	0.40	0.42	0.43	0.46
$F_{\alpha=0.2}$	0.53	0.67	0.7	0.72	0.74

Table 2: Product intent classification by varying the amount of automatically labeled queries by Algorithm 1. The first column corresponds to using 2K seed queries as training data.

# Distinct Queries	0	7.5K	15K	30K	60K
Precision	0.88	0.89	0.9	0.9	0.91
Recall	0.67	0.71	0.71	0.72	0.73
$F_{\alpha=0.2}$	0.84	0.86	0.87	0.87	0.88

Table 3: Job intent classification by varying the amount of automatically labeled queries by Algorithm 2. The first column corresponds to using 300 seed queries as training data.

amples, we discarded all negative queries that occurred only once. The resulting expanded training sets (which included seed queries) are shown in the last row of Table 1. As can been seen, we were able to automatically expand the training data by over two orders of magnitude in both applications.

Now we inspect the classification performance of the above algorithms. As shown in Figure 2 and Figure 3, Algorithm 1 & 2 significantly outperformed the other two methods in both applications. Specifically, Algorithm 2, *i.e.*, regularizing by content-based classification, showed significant advantage over Algorithm 1 in job intent classification, but not in the product case. This is largely due to the fact that the job intent classifier trained on seed queries already had reasonably good performance, thus providing a relatively good prior in Algorithm 2. In product intent classification, on the other hand, the seed query set was relatively small with respect to the large variety of product queries, resulting in a poor seed classifier and hence an unreliable prior.

Another observation in this experiment is that self-training did not work well in our query intent classification tasks. In self-training, if some unlabeled data that can be confidently classified by the seed classifier contain unseen discriminative features, then the algorithm would leverage these new features to improve classification. In query classification, however, since queries are short, it is difficult for the self-training algorithm to discover new discriminative features that generalize to other queries. In Section 4.4.3, we experiment with more generalizable features and compare them with query lexical features.

4.4.2 *Varying the amounts of training data*

Another interesting experiment is to see the impact of different amounts of automatically labeled queries on classification performance. To this end, we kept the same seed query sets, and randomly sampled other queries (according to their frequencies) in the click graphs described in Table 1. We experimented with different sampling rates, resulting in varying sizes of click graphs and hence of the final expanded training sets. Table 2 and Table 3 report precision, recall and $F_{\alpha=0.2}$ values for product and job intent classification

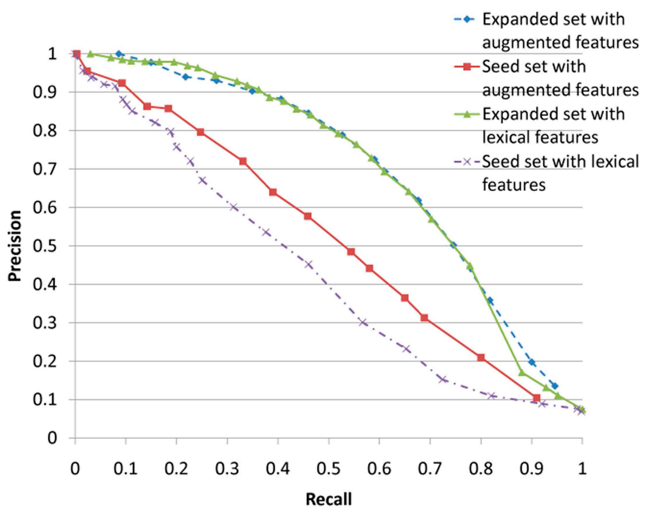

Figure 4: Product intent classification using different feature representations and different amounts of training data

respectively, where we exponentially increased the amounts of training data obtained by either Algorithm 1 or Algorithm 2. In both applications, we observed a monotonic increase in classification performance, although the increase slowed down when we used all click graph data available. Based on this experiment, it is reasonable to believe that classification performance could be further enhanced if we leverage a significantly larger collection of click-through data (which we did not explore in this work).

4.4.3 *Using more generalizable features*

As mentioned in Section 3, a significant amount of research efforts in query classification has been contributed to augmenting queries with external knowledge. Here we experiment with one augmented feature representation in product intent classification, and we compare its performance with that using query lexical features. More importantly, we inspect their respective performance when we change the amounts of training data.

Specifically, we augmented lexical features with word cluster features [15, 2] to improve the generalization ability of the product intent classifier. The idea is to group semantically similar words or phrases into clusters, and use these clusters, in addition to regular words, to create n-gram features. In this work, we created word/phrases clusters such as "<brand>" and "<type>" from a product database. The <brand> cluster consists of brand names, *e.g.*, "canon" and "nike", while the <type> cluster consists of product types, *e.g.*, "laptop" and "golf shoes". Then regular n-grams are augmented by these clustered n-grams in forms like "<brand> +camera" and "<type>+</s>", which are used as features in classification. One advantage of such features is that they can generalize to words or phrases that have not occurred in training data. The motivation here is akin to other query augmentation methods, *e.g.* by using linguistic knowledge [5] or using search engine results [16, 7].

As shown in Figure 4, when we used seed queries as training data, the performance of augmented features outperformed that of query n-gram features by a large margin. However, when we expanded our training set with automatically labeled queries, the difference in performance became negligible. This confirms our argument that with abundant training data, using only query words and phrases as features can work remarkably well.

5. CONCLUSIONS AND FUTURE WORK

In this work, we presented a semi-supervised learning approach to query intent classification with the use of click graphs. Our work differs from previous works on query classification in that we aim at drastically expanding the training data in an attempt to improve classification performance. This allows us to use relatively unbiased features, namely words/phrases in queries themselves, despite their sparseness. We achieved this goal by mining a large amount of click-through data, and inferring class memberships of unlabeled queries from those of labeled ones in a principled fashion. Moreover, we used content-based classification to regularize this learning process, and jointly performed graph-based learning and content-based learning in a unified framework.

In the future, we would like to investigate the impact of seed queries, *e.g.*, how to choose them and how much to choose, on our learning algorithm performance. Furthermore, it would be interesting to apply our approach to a even larger collection of click-through data, and to other tasks such as faceted query classification [13].

Acknowledgments. We are grateful to Jigar Mody, Samir Lakhani, Amit Gupta, Srinivas Bobba and Misha Mikhailov for providing data and for useful discussions.

6. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR'06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*, pages 19–26, 2006.

[2] L. D. Baker and A. McCallum. Distributional clustering of words for text classification. In *SIGIR'98: Proceedings of the 21st Annual International ACM SIGIR conference on Research and development in information retrieval*, pages 96–103, August 1998.

[3] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Knowledge Discovery and Data Mining*, pages 407–416, 2000.

[4] S. Beitzel, E. Jensen, A. Chowdhury, and O. Frieder. Varying approaches to topical web query classification. In *SIGIR'07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development*, pages 783–784, 2007.

[5] S. Beitzel, E. Jensen, O. Frieder, D. Lewis, A. Chowdhury, and A. Kolcz. Improving automatic query classification via semi-supervised learning. In *ICDM'05: Proceedings of the 5th IEEE International Conference on Data Mining*, 2005.

[6] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory*, 1998.

[7] A. Broder, M. Fontoura, E. Gabrilovich, A. Joshi, V. Josifovski, and T. Zhang. Robust classification of rare queries using web knowledge. In *SIGIR'07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, July 2007.

[8] N. Craswell and M. Szummer. Random walk on the click graph. In *SIGIR'07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 239–246, July 2007.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[10] J.-Y. N. J.-R. Wen and H.-J. Zhang. Clustering user queries of a search engine. In *Proceedings of the 10th International World Wide Web Conference*, 2001.

[11] U. Lee, Z. Liu, and J. Cho. Automatic identification of user goals in web search. In *WWW2005: The 14th International World Wide Web Conference 2005*, 2005.

[12] V. S. M. Belkin, P. Niyogi and P. Bartlett. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7(Nov), 2006.

[13] B. Nguyen and M. Kan. Functional faceted web query analysis. In *WWW2007: 16th International World Wide Web Conference*, 2007.

[14] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *IJCAI'99: Workshop on Machine Learning for Information Filtering*, pages 61–67, 1999.

[15] F. C. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *30th Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

[16] D. Shen, J. Sun, Q. Yang, and Z. Chen. Building bridges for web query classification. In *SIGIR'06: Proceedings of the 29th annual international ACM SIGIR conference on research and development in information retrieval*, pages 131–138, 2006.

[17] M. Szummer and T.Jaakkola. Partially labeled classification with Markov random walks. In *Advances in Neural Information Processing Systems*, volume 14, 2001.

[18] G.-R. Xue, D. Shen, Q. Yang, H.-J. Zeng, Z. Chen, Y. Yu, W. Xi, and W.-Y. Ma. IRC: An iterative reinforcement categorization algorithm for interrelated web objects. In *Proceedings of the 4th IEEE International Conference on Data Mining*, 2004.

[19] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, 1995.

[20] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, 2003.

[21] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02, Carnegie Mellon University, 2002.