Carotene: A Job Title Classification System for the Online Recruitment Domain

Faizan Javed[1], Qinlong Luo[1], Matt McNair[1], Ferosh Jacob[1], Meng Zhao[1], Tae Seung Kang[2]
[1]Data Science R&D, CareerBuilder
5550 Peachtree Parkway, Norcross (Greater Atlanta Area), GA 30092, USA
{Faizan.Javed, Qinlong.Luo, Matt.McNair, Ferosh.Jacob, Meng.Zhao}@CareerBuilder.com
[2]Department of Computer & Information Science & Engineering, University of Florida,
Gainesville, FL 32611, USA
tsk@cise.ufl.edu

***Abstract*—In the online job recruitment domain, accurate classification of jobs and resumes to occupation categories is important for matching job seekers with relevant jobs. An example of such a job title classification system is an automatic text document classification system that utilizes machine learning. Machine learning-based document classification techniques for images, text and related entities have been well researched in academia and have also been successfully applied in many industrial settings. In this paper we present Carotene, a machine learning-based semi-supervised job title classification system that is currently in production at CareerBuilder. Carotene leverages a varied collection of classification and clustering tools and techniques to tackle the challenges of designing a scalable classification system for a large taxonomy of job categories. It encompasses these techniques in a cascade classifier architecture. We first present the architecture of Carotene, which consists of a two-stage coarse and fine level classifier cascade. We compare Carotene to an early version that was based on a flat classifier architecture and also compare and contrast Carotene with a third party occupation classification system. The paper concludes by presenting experimental results on real world industrial data using both machine learning metrics and actual user experience surveys.**

Keywords—machine learning; text classfication; job title classification

I. INTRODUCTION

In many industries the advent of *Big Data* has resulted in a need to process and analyze vast amounts of data in a fast, efficient and scalable manner to extract business insights and improve customer experience. In the e-commerce and web domains, there is often a requirement for large-scale systems that can classify millions of items into thousands of categories to facilitate item catalog categorization. For many large e-commerce operations, successful organization and classification of items results in an improved end-user customer experience. This improved customer experience can consist of more fine tuned searching, browsing and recommendation of items which can potentially lead to an increase in volume of sales transactions. In the online job recruitment domain, classification of large datasets consisting of job ads (title, description and requirements) and resumes to predefined or custom occupation categories is important for many downstream applications such as job recommendations and human capital graph analytics. At CareerBuilder, classification of job ads and titles helps improve many downstream applications that in turn support the company's goal of empowering employment and helping job seekers find the jobs and training they need. For example, job title classifications enable data aggregation and analytics that is critical to our labor market analytics products such as Supply and Demand. These products deliver valuable insights to employers to help shape their recruitment strategies. Accurate classification of jobs and resumes also facilitates matching job seekers and employers by improving our search and recommendation products. To design a large-scale job title classification system we leverage text document classification and machine learning algorithms and frameworks.

Document classification has been widely studied in machine learning, data mining and information retrieval communities and has also been applied in various industrial settings. An automated approach to job title classification reduces to the problem of text document classification with machine learning. A machine learning-based document classification approach requires labeling documents with predefined classes to create a set of training data. This training data is then used to learn a model that can be utilized to assign one or more of the predefined classes to new documents. Several classification algorithms such as Support Vector Machines (SVM) [1] and k-Nearest-Neighbor (kNN) [2] have been applied to various industrial problems with good empirical results. Machine learning applications in industry frequently have the need to process very large datasets. A number of large-scale distributed machine learning frameworks have adapted machine learning algorithms to distributed and scale up architectures [3]. These frameworks are applicable to the increasingly common application scenarios that involve very large datasets and feature counts for computationally demanding training of complex learning models as well as near real-time inference constraints.

For job title classification, we have in the past relied on a third-party tool called Autocoder[1]. Autocoder classifies job related content such as resumes and job ads to a standardized

[1] http://www.onetsocautocoder.com/plus/onetmatch

978-1-4799-8128-1/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataService.2015.61

TABLE I. SOFTWARE DEVELOPERS, APPLICATIONS O*NET CATEGORY

Taxonomy Group	Code and Description
Major	15 - Computer & Mathematical Occupations
Broad	1130 – Software Developers and Programmers
Minor	1132 – Software Developers, Applications
O*NET Extension	00 – Software Developers, Applications

hierarchy of occupation categories known as the Occupational Information Network (O*NET[2]). O*NET was developed through the sponsorship of the U.S. Department of Labor/Employment and Training Administration (USDOL/ETA) and is an extension of the Standard Occupational Classification (SOC[3]) system developed by the U.S. Bureau of Labor Statistics. The current O*NET taxonomy is based on SOC version 2010 and is a four level hierarchy with 23 major top level groups, 97 minor groups in the second level, 46 broad occupations in the third level and 1,110 leaf level occupations. Table 1 shows the taxonomy breakdown of the *Software Developers, Applications* leaf level occupation which has the O*NET code 15-1132.00.

The O*NET taxonomy is not granular enough for our job title classification needs. For example consider the two SOC codes that represent the *Software Developers* category: 15-1132 (*Applications)* and 15-1133 (*Systems Software*). These two categories encompass all general as well as niche and emerging software engineering job titles such as *Machine Learning Engineer*, *Big Data Engineer*, and *Java Developer* among others. Because of this lack of granularity in job titles our search recommendation and analytics products are not able to drill down on metrics and analytic insights to the degree required to provide the best possible experience for our end users. This lack of fine-grained granularity is the primary motivation for our work in this area. Another reason is that O*NET and SOC taxonomies are not updated frequently enough to accommodate emerging job categories and titles. As of this writing the next update is scheduled for 2018, and then every ten years after that[4].

In this paper we describe Carotene, a semi-supervised job title classification and taxonomy discovery system currently in production at CareerBuilder. Carotene classifies job ads and resumes to our in-house job title taxonomy with the long-term goal of supporting classification to O*NET-SOC (henceforth stated as O*NET) codes as well. The taxonomy discovery component utilizes clustering techniques to discover job titles from job datasets. These job title sets are discovered for every SOC major group (henceforth referred to as SOC major). Carotene's classification component is composed of a hierarchical coarse and fine-level classifier cascade where a fine level classifier utilizes job title datasets for every SOC major depending on the classification results of the coarse-level classifier. We refer to this fine-level classification operation as *vertical classification* where the term *vertical* refers to one of the 23 SOC majors. The coarse-level classifier classifies to one SOC major while the fine-level classifier assigns the most relevant title from the job title set for that particular SOC major. In the rest of the paper, we discuss related work in Section II. Section III describes the architecture of Carotene as well as enhancements made to it compared to an earlier version called CaroteneV1. In Section IV we compare and contrast Carotene with Autocoder and CaroteneV1 using real world industrial datasets and show that using a cascade architecture for job title classification with a home grown taxonomy results in more accurate classification with finer granularity in the set of titles discovered. We conclude and discuss future work in Section V.

II. RELATED WORK

Systems in many diverse application domains such as medical text, e-commerce and web portals use concept hierarchies and taxonomies to organize their content according to certain functional characteristics. Some well-known examples of taxonomies are Amazon.com, Ebay.com, Wikipedia, International Classification of Diseases and the U.S. Library of Congress Classification. There has been a plethora of work in both industry and academia that leverages pre-existing hierarchies to build classification systems for proper categorization of new entities. Many works also use clustering techniques to overcome the paucity of labeled data for training classifiers and improve the classification process.

One of the first applications of SVMs as cascades for classifying large heterogeneous web content was discussed in [4]. The approach used SVMs as both coarse and fine grained classifiers with the condition that a threshold score had to be exceeded at the coarse level before a classification could be made at the fine grained level. The *Deep Classification* method by Xue et al [5] tackles hierarchical classification using a two-stage approach. In the first stage a category search is conducted to obtain a set of candidate documents for the query document. These candidate documents are used to prune the hierarchy and train a classifier to label the query document. A big drawback of this approach is the exorbitant computational requirement to train a classifier for every query document. In [6], Crammer et al use a cascading system of a k-best online learning algorithm and a rule-based and automatic coding policy for a multi-class, multi label system to assign clinical codes to free text radiology reports. To improve the classification accuracy of the system, the output of the rule-based system is used as features for the online learning algorithm. Similar to a classifier cascade, an "expert and gates" architecture in concert with a hierarchical array of neural networks was applied to the problem of medical records categorization in [7] and found to be more successful than flat classification.

In [8] a coarse-to-fine grained cascade is used for cancer detection using 3D medical images. The approach leverages a Relevance Vector Machine (RVM) as a coarse-level classifier to prune data samples which are far from the classification boundary. Clustering is performed on the data samples to group similar instances into the same clusters and then kNN is used as the fine-level classifier for the final classification. The SVM-kNN cascade is also used in [9] for predictive

[2] http://www.onetcenter.org/taxonomy/2010/list.html

[3] http://www.bls.gov/soc/major_groups.htm

[4] http://www.bls.gov/soc/revisions.htm

maintenance of industrial machinery. In this machine fault detection application, false negatives incur a high maintenance cost. False negatives are most likely to occur closer to the separating hyperplane between the two classes. To leverage their respective strengths, SVM is used for data points away from the hyperplane and kNN is used when data points lie closer to the separating hyperplane. As mentioned before, clustering has also been used to improve hierarchical cascade classification systems. Slonim and Tishby [10] reduce the feature space dimensionality by applying the information bottleneck principle to create word clusters. They show that this robust, low dimensional representation of the documents as word clusters improves text classification performance. CPHC [11] is a semi-supervised classification algorithm that alleviates the need to train a classifier by utilizing a pattern-based cluster hierarchy to directly classify instances.

Similar to works discussed above, Carotene utilizes a cascade architecture as well as a clustering component to improve classification efficacy. Carotene shares the application domain (online recruitment) with LinkedIn's job title classification system [12]. LinkedIn utilizes a heavily manual phrase-based classification system dependent on the near-sufficiency property of short-text. Their system also relies on an offline component that involves manually creating a controlled vocabulary, building standardized titles and crowd-sourced labeling of training samples. In terms of the approach taken, Carotene shares similarities with Ebay's work on large-scale item categorization [13]. While Carotene shares Ebay's hierarchical approach of cascading classifiers, unlike Ebay's system Carotene uses SVM as a coarse-level classifier and kNN as the fine-level classifier. This is similar to the systems described in [8] and [9] where SVM is first used to prune data samples to focus more on classification critical samples. This is done because in a high dimensional space selecting a distance metric and neighborhood size is non-trivial for kNN.

III. Design And Development Of Carotene

In this section we describe the design and development of Carotene. Carotene is a multi-class, multi-label, semi-supervised system composed of a clustering component for taxonomy discovery and a cascade classifier architecture. For a query text, it returns a list of job titles ranked by confidence scores. As discussed in [14], for large-scale taxonomies that are highly unbalanced a hierarchical classifier is preferred over a flat classifier. This is because a flat classifier has to make a single decision that is more difficult while a hierarchical classifier, though it has to make multiple decisions, experiences a less severe imbalance at each level of the hierarchy. On the other hand, hierarchical classifiers suffer from error propagation where the final classification decision relies on the accuracy of a series of previous decisions in the hierarchy and thus is more likely to make a misclassification. Pruning the taxonomy can mitigate this problem. In our case we only consider the top (SOC major) and ONET code levels of the four-level ONET taxonomy. Fig. 1 shows the distribution of job ads at the SOC major level at CareerBuilder over the past year. The job ads are skewed towards a number of occupation categories such as 15 Computer & Mathematical), 41 (Sales), and 43 (Office and Administrative) among others.

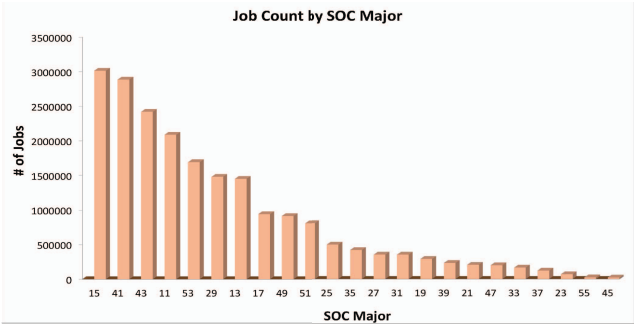

Fig. 1. Distribution of job ads by SOC major

Fig. 2 gives an overview of Carotene. The system relies on a training dataset that is used for both the clustering and the cascade classifier components explained in up coming sub-sections. The dataset initially consisted of 3.6m job ads collected from the CareerBuilder.com. To circumvent the difficulty of labeling the datasets we initially considered crowdsourcing. However crowdsourcing of large datasets is an expensive endeavor and is also susceptible to systematic and consistency errors [12]. Instead of crowdsourcing we used Autocoder to label the training dataset. Autocoder has an O*NET code classification accuracy of 80% for job titles and 85% for job ads[5]. While this puts a theoretical upper bound on Carotene's potential accuracy, considering the cost and effort efficiency gains in obtaining a labeled training dataset and our long term goal of matching or exceeding Autocoder's accuracy we proceeded with this approach for obtaining a training dataset. To tackle the class imbalance problem that exists in the dataset we under-sampled all the classes to a base count of around 150k jobs that decreased the training dataset size to approximately 2m jobs.

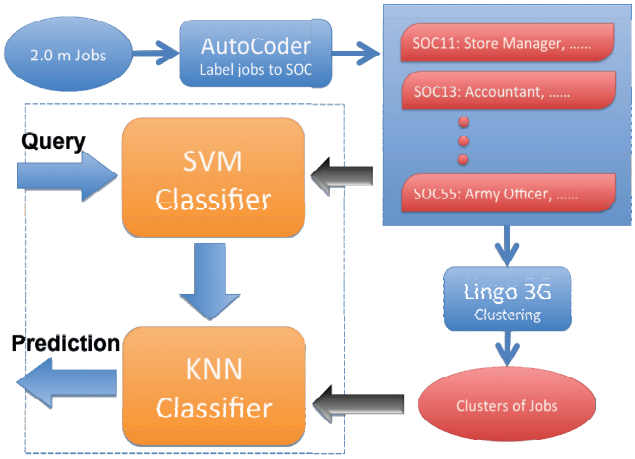

Fig. 2. Architecture of Carotene

5 http://www.onetsocautocoder.com/plus/onetmatch?action=guide

A. Taxonomy Discovery: Clustering to Discover Verticals

The first phase of Carotene discovers clusters of job titles in the labeled training dataset to infer a job title taxonomy organized by SOC major. All the jobs in the training set are pre-processed by removing extraneous markup characters, stop-words and noise. We ran a clustering process on each of these compartmentalized dataset segments and refer to this clustered taxonomy of job titles arranged by SOC major as a *vertical*. Every cluster in a vertical has a group title label called *glabel* that is the representative normalized label of the raw job titles in that cluster. These verticals are used by the fine-grained classifier component of the cascade that we discuss in section C.

For clustering we use Lingo3G, a proprietary clustering library based on the Lingo clustering algorithm [15]. Lingo balances both simplicity and accuracy by using a novel combination of numerical and phrase-based clustering methods that identify lexically meaningful and semantically distinct cluster labels before fulfilling cluster assignments. Lingo first applies singular value decomposition (SVD) on the tf-idf (term frequency-inverse document frequency) term-document matrix where left-singular vectors are used to identify cluster labels and singular values are used to determine the ideal amount of clusters. Documents are assigned to clusters based on cosine distance over a user-defined threshold to quantify resemblance. While Lingo has a lower misclassification rate than other clustering methods, it consumes more memory due to relatively frequent and high dimensional matrix operations especially on larger datasets [16]. Consequently we applied Lingo3G only on job titles and not job ads. Lingo3G uses a custom-built metaheuristic algorithm that is a more enhanced version of Lingo. We use Lingo3G because of its propensity to identify meaningful cluster labels that greatly assist in inferring a job title taxonomy from a dataset.

Below we discuss some of our parameter settings for Lingo3G which were selected based on experimental results:

1) max-cluster-size [0.0-1.0]: controls the maximum allowed cluster size. We use the default value of 0.4 which means that a cluster cannot contain more than 40% of all documents.

2) merge-threshold [0.0-1.0]: defines the cluster overlap threshold value at which clusters will be merged. A low merge-threshold value causes eager merging of clusters resulting in larger clusters that may contain irrelevant documents. Meanwhile a high merge-threshold value results in a large number of small clusters with more relevant documents. We set the value of merge-threshold to 0.5. This is a little lower than the suggested default of 0.7 because we found that the default value was creating too many useless or redundant job title clusters.

3) single-word-label-weight [0.0-1.0]: controls the propensity of the clustering engine to create single word cluster labels. A high value of this parameter will result in many single word cluster labels. We set this parameter to 0.1 which is much lower than the default of 0.5 because the default value was very aggressive in ignoring multi-word job titles which form a sizeable contingent of the job title taxonomy.

4) max-clustering-passes-top [0-10]: determines the maximum number of clustering passes that are performed to discover clusters. Every subsequent pass discovers more specific clusters than the first pass. When this parameter is set to 0, the algorithm only stops clustering when no more clusters can be created. We set this parameter to 0, which allows the clustering engine to create the maximum possible number of clusters. While this results in some noisy clusters, we also discover more niche clusters that are representative of emerging job titles that may otherwise not materialize as a cluster because of their low frequency count in the dataset.

5) max-hierarchy-depth [1-5]: controls the number of cluster levels to create. Since we are not creating hierarchical clusters, we set this parameter to 1 which disables hierarchical clustering and preserves small clusters.

6) min-cluster-size [0.0-1.0]: controls the minimum allowed cluster size. We set this value to 0.0005, which means that clusters cannot contain less than 0.05% of all documents. We have experimentally validated that this value strikes a good balance between niche and noisy clusters.

7) phrase-df-threshold-scaling-factor [0.0-5.0]: controls the Phrase-level Document Frequency cutoff scaling factor. Low values of this parameter preserve infrequent phrases at the cost of slower processing. We set this value to 0.0 because it preserves the more descriptive and specialized cluster name phrases (e.g., Nurse vs Emergency Nurse) which would otherwise be subsumed by more general clusters.

8) word-df-threshold-scaling-factor [0.0-5.0]: controls the World-level Document Frequency cutoff scaling factor. Low values of this parameter preserve infrequent words at the cost of slower processing. We set this value to 0.0 because it preserves the more descriptive and specialized cluster name words (e.g., Surgeon) that may otherwise be not considered as candidate cluster names.

The clustering process for each of the 23 verticals took around 5 minutes on average on a dual core i7 2Ghz MacBook Air with 8GB of RAM. After clustering, we applied several post-processing steps such as cluster merging, deletion, editing and glabel validation that required domain expertise and human validation. Lingo3G provides an option to influence the way cluster labels are chosen by configuring a label dictionary. The label dictionary can be used to prevent certain words and phrases (e.g., offensive language and company names) to be chosen as cluster labels. The label dictionary can also be used to promote and increase the likelihood of some words and phrases to be chosen as cluster labels (e.g., job title terms). For our case we do not boost any words or phrases but do exclude state and city names and state and country acronyms that

appear in the data set such as *{Atlanta, GA, San Jose, New York, etc.}*. These often appear in conjunction with job titles that state the location of the job such as in *Software Engineer wanted, San Diego, CA Area.* We also exclude overly general job titles such as *Co-op*, *Contractor,* and *Specialist* among others. Such job titles are more akin to job title modifiers. Lingo3G also provides an option to define a synonym dictionary. This dictionary indicates to the clustering process that certain words and phrases have the same meaning and should be treated as synonyms during clustering. For example, the set elements *{Senior Level, Senior-Level, Sr., Sr, Senior}* are synonyms for the job title modifier *Senior*. Similarly the set elements *{CFO, Chief Financial Officer, Chief Finance Officer, Chief Financial and Operating Officer}* contain synonyms for the job title *CFO*. By default the label dictionary and synonyms are configured manually as xml files. The clustering process is run from the command line and the results are persisted to files that then have to be analyzed manually.

Instead of performing these steps manually, we crowdsourced this task to our in-house team and created a tool called *DataBroom* which provides a user-friendly interface and abstraction to automate most of these tasks. Fig. 3 shows the DataBroom tool in action. The figure shows two display panes. The display pane on left is a list of all the clusters labels. Selecting a cluster from this list displays on the right pane all the documents that belong to this cluster. In the figure the *Copy Writer* cluster has been selected on the left and the right pane shows all the documents that belong to this cluster. The *Merge* option performs the synonym operation. There are also options to delete clusters as well as to run the clustering process iteratively. The results of the clustering process can be instantly viewed in the UI when the process is complete. By using the DataBroom tool we were able to finalize the job title taxonomy in less than half the time of the original estimate of three months while using a headcount of four people.

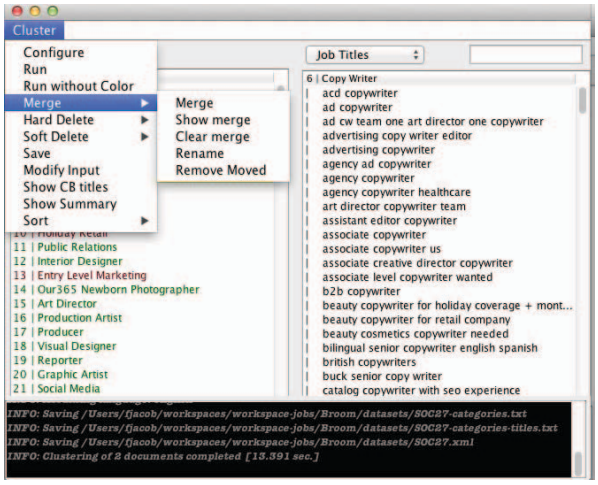

Fig. 3. DataBroom UI and cluster configuration menu

The taxonomy currently has 4,076 titles ranging from emerging titles such as *Data Scientist* and *Hadoop Engineer* to niche titles such as *National Luminologist Trainer*. Note that the O*NET taxonomy does not have a representation for such fine-grained titles. To incrementally augment the taxonomy and capture emerging and niche titles to better support our end users, the taxonomy discovery process can be run in quicker intervals (say, quarterly) than the scheduled five year O*NET updates.

B. SOC Major Classifier

As discussed in our previous work [17], our first iteration of a job title classification system (CaroteneV1) was a flat, *big bang* approach using a kNN classifier with a single vertical representing all of our initial set of 1,800 job titles across all the SOCs. Carotene was changed to a cascade architecture because: 1) a flat architecture using an instance-based classifier does not scale to ever increasing datasets and taxonomy, and 2) it cannot support a hierarchical classification system such as O*NET which is one of our long-term goals.

At the coarse-level of the cascade, jobs are classified to the appropriate SOC major. The SOC major assigned is then used at the fine-level classifier level (explained in the next section) to narrow the space of potential labels for the job query. The current iteration of Carotene incorporates SVM as a coarse-level *SOC Major* classifier. SVMs are robust on sparse and high-dimensional data such as job title datasets and they do not over-fit data [18]. Consequently, SVMs excel at text classification and are an appropriate fit for our problem. The SVM implementation used for the SOC Major classifier is LIBLINEAR [19]. LIBLINEAR supports the one-vs-all strategy for multi-class classification. The SOC Major classifier was trained on the 2M jobs dataset that was augmented by 225 most frequently searched terms by users of our Supply & Demand portal. These terms had either incorrect (based on the existing Carotene model) or no classification results. These high frequency search terms included titles and terms such as *Halloween Associate*, *PMP*, *OBIEE* and *Motorhand.* Based on experimentation we chose frequency-based feature representation and removed any feature that occurred in fewer than two training instances. The training dataset had 5.8M features with approximately 280 non-zero features per instance. Because this is a large-scale, sparse data-learning problem where the number of features is far greater than the number of training instances, we chose the L2-regularized L2-loss SVC (dual) solver for training the model. We experimentally verified this solver's accuracy and training time efficacy compared to other options in LIBLINEAR and refer the reader to [19] for more details on the solvers in LIBLINEAR. It took less than 90 minutes to train the SOC Major classifier on an m3.2xlarge EC2 instance with 30GB of RAM. While the trained model is 1GB (uncompressed) in size, single node classification performance is extremely fast at less than 40ms per instance.

The SOC Major classifier also has a distributed, batch classification mode. On our in-house 60-node Hadoop[6] cluster, the SOC Major classifier can process over 4M jobs within a few minutes. The SOC Major classifier is a multi-class, multi-label classifier that uses Platt's scaling [20] to return a list of classifications ranked by confidence. However in Carotene's

[6] http://hadoop.apache.org

current implementation only the top ranked classification returned by the SOC Major classifier is selected for further processing in the fine-grained classification stage.

For a multi-class classifier the appropriate performance metrics are average precision, average recall and macro-average F-score [21]. To estimate the accuracy of the model we ran a k-fold cross validation test with k set to 10 because it provides the best balance of overlap between test sets and the size of test sets [22]. The SOC Major classifier's cross validation metrics for average precision, recall and F-score were 95.54%, 95.33% and 95.43% respectively. Global accuracy was 95.8% and coverage at 99%. On a separate dataset of 2.1m jobs, using Autocoder as the gold standard, the SOC Major classifier achieved an average accuracy of 90%.

C. Proximity-based Vertical Classification

The fine-level classifier component of Carotene is an implementation of a multi-class, multi-label proximity-based classifier. Proximity-based classifiers leverage the property that documents belonging to the same class are more likely to be close to one another based on a distance metric such as cosine distance [23]. They may either find the k-nearest neighbors of a query instance, or pre-process the data into clusters and use cluster meta-data as k-nearest neighbor documents to improve classification efficiency. Carotene uses kNN as a fine-level classifier with k empirically set to 20. kNN is considered a slow classifier because of its lack of information abstraction from training data and reliance on instance-time classification. However our implementation uses an open-source, industrial strength search engine library called Lucene[7] that results in classification response time of less than 100ms. We use Lucene queries with a combination of unigram and bigram terms to construct the queries because job titles can be composed of both single and multiple terms. Indexed documents are represented as term vectors and are ranked using a tf-idf and cosine-similarity scoring model. CaroteneV1 was an implementation of this kNN classifier that queried a single vertical composed of the initial set of 1,800 titles. In Carotene the kNN classifier has access to 23 verticals and chooses the appropriate vertical to use based on the classification it receives from the SOC Major classifier. Note that in Carotene, model training time is only required for the SOC Major classifier. In Carotene the kNN architecture has been improved in a number of ways that we detail in the following subsections.

1) Weighted Voting KNN: As described above, kNN uses a *majority voting* strategy which is susceptible to skewed class distributions. For example, the query *Big Data Developer* will return eight *Big Data Developer* and twelve *Big Data Engineer* (k=20) nearest neighbor instances. The kNN prediction will be *Big Data Engineer* even though the query is *Big Data Developer* and a *Big Data Developer* class exists. To rectify such misclassifications, the kNN voting strategy was changed to *weighted voting* where the absolute value of Lucene scores are used to assign a weight to each neighbor as shown in (1):

$$w_i = \frac{s_i}{\sum_{j=1}^{k} s_j} \quad (1)$$

where w_i is the weight of the i[th] neighbor and S_i is the Lucene score of the i[th] neighbor of the query. The confidence score for each class is defined in (2), where f_n is the confidence score for class n and C_n represents the set of neighbors in class n.

$$f_n = \frac{\sum_{i \in C_n} w_i}{\sum_{j=1}^{k} w_j} \quad (2)$$

2) Index glabels: As discussed in section A, a vertical is composed of clusters and every cluster has a normalized title called a glabel which represents the raw job titles in the cluster. In CaroteneV1 only the raw job titles were indexed and searched on. However raw job titles are noisier than glabels and in Carotene both the raw job title and glabels are indexed as *title* and *label* fields respectively. A Lucene multi-field query is used to query against these fields with a pre-defined boost factor. The *title* field has the default boost factor of 1.0 and a boost factor of 0.9 for the *label* field gives the best results.

Fig. 3 shows the results of experimenting with the boost factor of the *label* field and its effect on Carotene's accuracy. If the boost factor is too small, indexing the *label* field will not help in fine tuning the classification results. On the other hand if the boost factor is too large, it will be equivalent to directly looking up normalized job titles to answer queries instead of querying a learning system that can generalize from training samples. The intent here is to improve the quality of classification in close boundary decisions. For certain job title queries we also calibrate the results to return a more related match if it exists in the ranked list of classification results. For example, for the query *Senior Java Software Engineer,* the classifier returns *Software Engineer* as the top result followed by *Senior Java Software Engineer*. This is because the *Software Engineer* cluster is larger and more general than the *Senior Java Software Engineer* cluster and also contains titles that belong to the *Senior Java Software Engineer* cluster. To rectify this, when the list of ranked classifications is being constructed, provided it exists in the classification result set, a high calibration boost value is added to a label which exactly matches the query.

IV. Experimental Results

We conducted several experiments to compare Carotene with both CaroteneV1 and Autocoder. For comparisons with CaroteneV1, we used two datasets: normalized titles and job ads. Normalized titles are a set of 570 job titles that exist in both CaroteneV1 and Carotene taxonomies. Job ads are a sample of 100 jobs randomly chosen from CareerBuilder's website and are distinct from the training dataset. We used these two datasets because internal teams at CareerBuilder use Carotene for classifying large text such as job ads as well as short-text composed only of job titles. Hence it is important to

[7] http://lucene.apache.org

test the accuracy of Carotene for both scenarios and to ensure that at the very least, for job titles in its own taxonomy, Carotene returns accurate results.

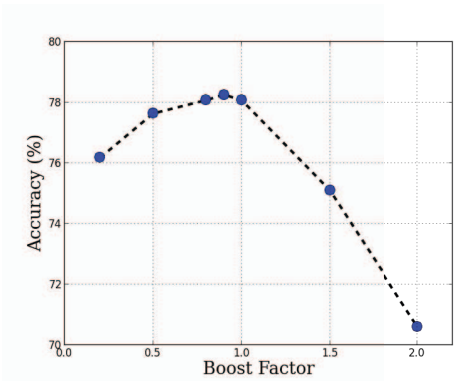

Fig. 3. Carotene accuracy and boost factor for the label field

TABLE II. ACCURACY OF CAROTENEV1 AND CAROTENE

Dataset	Accuracy	
	Carotene V1	*Carotene*
Normalized Titles	68%	80%
Job Ads	77%	79%

TABLE III. USER SURVEY RESULTS OF AUTOCODER VS CAROTENE

Classifier	Precision	Confidence
Carotene	68%	99%
Autocoder	64%	99%

Table II shows results of comparing CaroteneV1 and Carotene on the two datasets. For the normalized titles dataset, the enhancements in Carotene result in a 12% increase in accuracy from 68% to 80%. For the job ads dataset, the accuracy increase is more modest at 2%, increasing from 77% to 79%. This can be attributed to the fact that job ads are noisier than job titles. While the extra content in job ads can potentially provide more information to improve classification accuracy, in many cases job ads contain irrelevant general company text content such as mission statements and marketing pitches that can hinder classification accuracy. As both CaroteneV1 and Carotene are multi-class and multi-label systems, we also compared their hamming loss on the normalized titles test set. Hamming loss is the fraction of labels that are predicted incorrectly [24] with the optimal value being 0. On the normalized titles set, CaroteneV1 and Carotene had hamming loss values of 0.011 and 0.007 respectively, indicating that Carotene has higher quality classification labels than CaroteneV1. We are currently conducting more experiments on larger job ads datasets as well as job seeker resumes.

As Carotene and Autocoder do not share the same taxonomy, to compare the two we ran a user survey of users registered on CareerBuilder.com. In the survey we classified the most recent work history listed in the users resume using both Carotene and Autocoder. In the survey emails that were sent to solicit user responses, users were requested to rate the classifications from both Carotene and Autocoder on a scale of 1 to 5 where a rating of 1 implies very poor classification that totally missed the mark and 5 is for perfect classification. Formally, assuming the true precision of the system is P, and h is the error rate of estimated precision $\hat{p}$ at confidence level C [25] that satisfies

$$\Pr\left(\left|\frac{\hat{p}-P}{P}\right| < h\right) = C, \qquad (2)$$

where given sample S (note that implicitly, we assume all titles presented are *positive*), and its true positive set $S_{TP} \subseteq S$, the sample precision can be calculated as

$$\hat{p} = \frac{\sum_i I_{S_{TP}}(x_i)}{\sum_i I_S(x_i)}. \qquad (3)$$

Therefore, the sample size n of the random sample S from a population of size N is determined such that

$$\begin{cases} n_0 = \dfrac{z_\alpha^2(1-P)}{h^2 P} \\ n = \dfrac{n_0}{1+(n_0-1)/N} \end{cases}, \qquad (4)$$

where $\alpha = (1-C)/2,$ and z_α is the upper $\alpha/2$ quantile of the standard normal distribution such that

$$\Pr(Z < z_\alpha) = 1 - \frac{\alpha}{2}, \quad Z \sim N(0,1).$$

In our evaluation process, we set $h \equiv 0.03$ and $C \equiv 99\%$. This means the data-driven evaluation is implemented such that 99% of the time the estimated sample precision is no more than 3% different from the true population precision. Table III shows the results of the survey of 2M active users. Carotene has a 4% higher precision than Autocoder at classifying work histories of job seekers. We are also in the process of conducting a job poster survey to compare the quality of CaroteneV2 with Autocoder at classifying job ads.

V. CONCLUSION AND FUTURE WORK

In this paper we discussed the architecture of Carotene, a job title classification system currently in production at CareerBuilder. Carotene has a classifier architecture composed of an SVM-kNN cascade as well as a clustering component that is used in taxonomy discovery. We compare and contrast Carotene with a 3rd-party occupation classification system called Autocoder as well as an early version of Carotene that was based on a flat classifier architecture. Experimental results show that Carotene provides better classification results with a finer level of granularity in titles.

As an active internal project at CareerBuilder, there are a number of improvements and extensions to Carotene being worked on such as: 1) strategies for internationalizing Carotene to support multiple international markets, 2) extending the architecture to support classification at the O*NET code level, 3) investigating leveraging more than one coarse-level SVM classifications at the kNN level, and 4) conducting research to incorporate enrichment vectors for more semantically aligned classifications.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their useful feedback and comments. We would also like to thank Mark Rzeznik of CareerBuilder Site Usability team for his assistance with building the Carotene taxonomy and Kara Johnson of CareerBuilder Product Development team for soliciting regular customer feedback on Carotene's classification performance across many of our products.

REFERENCES

[1] T. Joachims, "Transductive inference for text classification using support vector machines," In Proceedings of ICML 1999, pp. 200-209, 1999.

[2] G. Guo, H. Wang, D. Bell, Y. Bi and K. Greer, "Using kNN model for automatic text categorization," Soft Computing, vol. 10(5), pp. 423-430, 2006.

[3] R. Bekkerman, M. Bilenko and J. Langford, Scaling up machine learning: parallel and distributed approaches. Cambridge University Press, 2011.

[4] S. Dumais and H. Chen, "Hierarchical classification of web content," In Proceedings of ACM SIGIR'00, pp. 256-263, 2000.

[5] G-R Xue, X. Dikan, Q. Yang, and T. Yu, "Deep classification in large-scale text hierarchies," In Proceedings of ACM SIGIR '08, pp. 619-626, 2008.

[6] K. Crammer et al., "Automatic code assignment to medical text," In BioNLP'07, Association for Computational Linguistics, pp. 129-136, 2007.

[7] M. E. Ruize and P. Srinivasan, "Hierarchical text categorization using neural networks," Information Retrieval, vol. 5(1), pp. 87-118, 2002.

[8] M. Liu, L. Lu, X. Ye and S. Yu, "Coarse-to-fine classification via parametric and nonparametric models for computer-aided diagnosis," In Proceedings of ACM CIKM'11, pp. 2509-2512, 2011.

[9] A. B. Andre, E. Beltrame and J. Wainer, "A combination of support vector machine and k-nearest neighbors for machine fault detection," Applied Artificial Intelligence, vol. 27(1), pp. 36-49, 2013.

[10] N. Slonim and N. Tishby, "The power of word clusters for text classification," In 23rd European Colloquim on Information Retrieval Research, 2001.

[11] H. H. Malik and J. R. Kender, "Classification by pattern-based hierarchical clustering," In From Local Patterns to Global Models Workshop (ECML/PKDD 2008), 2008.

[12] R. Bekkerman and M. Gavish, "High-precision phrase-based document classification on a modern scale," In Proceedings of the 17th ACM SIGKDD-KDD'11, pp. 231-239, 2011.

[13] D. Shen, J-D Ruvini and B. Sarwar, "Large-scale item categorization for e-commerce," In Proceedings of ACM CIKM'12, pp. 595-604, 2012.

[14] R. Babbar and I. Partalas, "On flat versus hierarchical classification in large-scale taxonomies," Neural Information Processing Systems (NIPS), pp. 1-9, 2013.

[15] S. Osiński and D. Weiss, "A concept-driven algorithm for clustering search results," IEEE Intelligent Systems, vol. 3(20), pp. 48-54, 2005.

[16] S. Osiński, Dimensionality reduction techniques for search results clustering. Masters thesis, Department of Computer Science, The University of Sheffield, UK. 2004.

[17] F. Javed, M. McNair, F. Jacob and M. Zhao, "Towards a job title classification system", WSDM'14 - Workshop on Web-Scale Classification: Classifying Big Data from the Web, 2014.

[18] T. Joachims, "Text categorization with support vector machines: learning with many relevant features", In Proceedings of ECML, 1998.

[19] R. E. Fan et al., "Liblinear: a library for large linear classification," Journal of Machine Learning Research, vol. 9, pp. 1871-1874, 2008.

[20] J. C. Platt, "Probablistic outputs for support vector machines and comparisons to regularized likelihood methods," In Advances in Large Margin Classifiers, pp. 61-74, 1999.

[21] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," Inf. Process. Manage, vol. 45(4), pp. 427-437, 2009.

[22] P. Refaeilzadeh, L. Tang and H. Liu, "Cross-validation", Encyclopedia of Database Systems, pp. 532-538, 2009.

[23] C. Aggarwal and C. Zhai, A survey of text-classification algorithms, a chapter in Mining Text Data, Springer, 2012.

[24] G. Tsoumakas and I. Katakis, "Multi-label classification: an overview," International Journal of Data Warehousing and Mining, vols. 3(3), pp. 1-13, 2007.

[25] W. Cochran, Sampling Techniques, 3rd Edition, John Wiley, 1977.