Job Type Extraction for Service Businesses

Cheng Li
Google, Mountain View, CA, USA
chgli@google.com

Yaping Qi
Google, Mountain View, CA, USA
qiyaping@google.com

Hayk Zakaryan
Google, Mountain View, CA, USA
haykzakaryan@google.com

Mingyang Zhang
Google, Mountain View, CA, USA
mingyang@google.com

Michael Bendersky
Google, Mountain View, CA, USA
bemike@google.com

Yonghua Wu
Google, Mountain View, CA, USA
yonghuawu@google.com

Marc Najork
Google, Mountain View, CA, USA
najork@google.com

ABSTRACT

Google My Business (GMB) is a platform that hosts business profiles, which will be displayed when a user issues a relevant query on Google Search or Google Maps. GMB businesses provide a wide variety of services, from home cleaning and repair, to legal consultation. However, the exact details of the service provided (a.k.a. *job types*), are often missing in business profiles. This places the burden of finding these details on the users. To alleviate this burden, we built a pipeline to automatically extract the job types from business websites. We share the various challenges we faced while developing this pipeline, and how we effectively addressed these challenges by (1) utilizing structured content to tackle the cold start problem for dataset collection; (2) exploiting context information to improve model performance without hurting scalability; and (3) formulating the extraction problem as a retrieval task to improve both generalizability, efficiency, and coverage. The pipeline has been deployed for over a year and is scalable enough to be periodically refreshed. The extracted job types are serving users of Google Search and Google Maps, with significant improvements in both precision and coverage.

CCS CONCEPTS

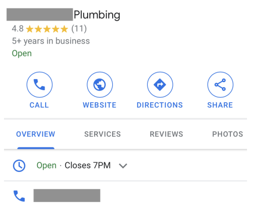
• **Applied computing** → **Document management and text processing**.

KEYWORDS

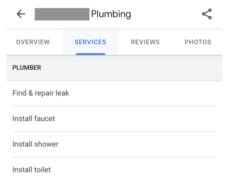
job type extraction, service

ACM Reference Format:
Cheng Li, Yaping Qi, Hayk Zakaryan, Mingyang Zhang, Michael Bendersky, Yonghua Wu, and Marc Najork. 2023. Job Type Extraction for Service Businesses. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3543873.3584636

WWW '23 Companion, April 30–May 04, 2023, Austin, TX, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9419-2/23/04.
https://doi.org/10.1145/3543873.3584636



(a) A business that provides plumbing services.



(b) The *SERVICES* tab shows a list of job types.

Figure 1: A business place (on a mobile device).

1 INTRODUCTION

Google My Business (GMB) has been hosting a large number of business profiles. The profiles provide various details about the business, including opening hours, phone numbers, and user reviews. An example of a business profile is shown in Figure 1 (a). These profiles will be displayed on Google Search or Maps in response to a relevant query.

GMB businesses provide a wide array of services, from home cleaning and plumbing, to finance and education. Users searching for service businesses are often looking for a particular *job type*, e.g., *water heater installation*. While some owners manually list the provided job types, this information is missing in most business profiles. Thus, previously users would have to either directly call the business or scan through their website, both of which are time-consuming.

Thus, to reduce user effort, we developed and deployed a pipeline to automatically extract the job types from business websites. For example, if a web page owned by a *plumbing* business states: "*we provide toilet installation and faucet repair service*", our pipeline outputs ***toilet installation*** and ***faucet repair*** as the job types for this business.

We share **challenges and solutions** during the development of this extraction pipeline. First, in contrast to most academic papers that work on ready-to-use datasets, we need to collect our training dataset from scratch, leading to cold start issues. As human annotation is costly, randomly sampling pages is ineffective for producing sufficient positive examples of pages containing job types. To overcome this problem, we utilize items appearing in structured fields, which have a higher chance of containing job types. The identified job types are then treated as seeds to deal with free-text pages.

Second, scalability is critical. To keep the extracted job types up-to-date, we run the pipeline every few days over *billions* of web pages. It is impractical to treat job types as entities and directly run off-the-shelf named entity recognition (NER) models over the entire text of billions of web pages. In addition, these models output a label per word, while we only need to identify the mentioned job types, and the labels of other words can be safely ignored. We developed a tailored model that does not need to take as input the entire page, which is highly scalable.

Third, we aim at high precision and reasonable coverage to bring the best user experience. To improve precision without hurting scalability, we designed various information contexts that are lightweight, easy to use, informative, and can be used in ways other than model input. To enhance coverage, we formulate the extraction task as a retrieval problem – the seed job types from structured fields are treated as queries to retrieve free-text pages that might contain job types. This formulation also helps us address the scalability challenge.

Generalization. The lessons we shared in developing the large-scale extraction pipeline from scratch can generalize to other information extraction or machine learning tasks. They have direct applications to domain-specific extraction tasks, exemplified by expertise finding, legal and medical information extraction. Three most important lessons are: (1) utilizing the data properties such as structured content could alleviate the cold start problem of data annotation; (2) formulating the task as a retrieval problem could help researchers and practitioners deal with a large dataset; (3) the context information could improve the model quality without sacrificing its scalability.

The pipeline described in this paper has been successfully deployed for over a year. It is efficient enough to be run over billions of pages every few days. The extracted job types are serving users issuing service related queries on Google Search or Maps. They are either displayed directly as in Figure 1 (b)[1], served as explanations of showing a business to a search query, or used as a signal to rank business places.

2 RELATED WORK

We are not aware of any work directly focusing on job type extraction for businesses. Thus we survey papers on information extraction tasks that are most closely related to our work.

Named entity recognition (NER). NER aims to identify references to certain types of objects [16]. Various deep learning based encoders have been explored, including convolutional neural networks (CNN) [20], recurrent neural networks (RNN) [4], and transformers [10]. To make predictions, a tag decoder is utilized to produce a tag per word. It can be a multi-layer perceptron and a softmax layer [19], CRFs [4], and RNNs [17, 18].

Mention detection. Mention detection is a binary classification task of identifying mentions of particular entities in the text given a set of candidate mentions. They are often incorporated as one component when training coreference systems end-to-end. Lee et al. [8] proposed an LSTM based coreference system that jointly learns mention detection. It is improved by adding a biaffine attention [21], ELMO embeddings [5, 9], and BERT [1].

Information extraction from web pages. Many extractors for web pages rely on the Document Object Model (DOM) of web pages [6]. Training data are used to learn a rule-based extractor based on DOM features. To reduce the size of training data, distant supervision is proposed based on knowledge bases [12], shared XPath [2], or visual features [13]. Various deep neural encoders have been explored to encode web pages, including CNNs [3], hierarchical RNNs [11] and Graph Neural Networks (GNNs) [14].

Differences to our work. Unlike the standard NER task, where the model is trained to output one tag per word, our task only needs to identify the text spans that correspond to a job type. On the other hand, models for mention detection consider all noun phrases in text as candidate mentions [7]. In addition, most methods require joint training with the coreference task. Therefore, directly applying NER or mention detection models to our task would be too costly and impractical.

Most web extractors rely on DOM elements, which cannot work on any web page. The existing web page encoders consider the entire page, which is not efficient enough for our task.

3 JOB TYPE CLASSIFICATION AND EXTRACTION

3.1 Pipeline Overview

We start from a collection of billions of web pages. To only retain service related pages, we rely on homepages shown in business profiles. We only keep a page if: (1) it is a homepage; (2) it is one click away from the homepage; **or** (3) it is in the same domain as the homepage. Note that (2) and (3) introduce **noise** into the dataset, which will be handled in the next few steps.

The retained pages are candidates for job type extraction. Without sufficient seed job types to start with, we create seeds based on the observation that many businesses advertise their job types in a structured format (i.e., tables or lists). This strategy has greatly mitigated the **cold start** problem. We extract items from the structured content and classify whether they are job types. We call this step **job type classification**, where we develop a model that encodes various context information to classify job types without feeding the entire page to the model, addressing the challenge of maintaining high **precision** and **scalability**. The details are in Section 3.2.

To handle any web page, in Section 3.3 we formulate **job type extraction** as a retrieval problem – seed job types from job type classification are used as queries to retrieve relevant pages. Job types are then extracted from these pages. This handles challenges of both **coverage** and **scalability**. The extracted job types are stored in a knowledge base and consumed by downstream applications such as web search.

3.2 Job Type Classification

3.2.1 The General Procedure. We aim to train a classifier to predict whether a given candidate job type displayed on a web page is an

[1]Note that the service tab in Figure 1 (b) has been introduced before the deployment of our pipeline. Business owners can also manually edit it as indicated on the Help page. See https://support.google.com/business/answer/9455399.

actual job type; i.e., whether this candidate job type is a type of service provided by the business. A negative example would be the phrase *bed bug control* mentioned on a page providing instructions on dealing with bed bugs, but not the service itself.

The first step is to collect candidate job types for labeling. We tried to randomly sample some pages for human annotation, but the discovery rate of job types is fairly low. We also tried to use predefined or existing job types provided by business owners on their profiles to match web pages. However, such a set is small and results in a very low coverage empirically.

To address this problem, we rely on the observation that many businesses present their job types in a structured form, i.e., tables and lists. The items extracted from such content are treated as candidate job types.

The second step is to train a classifier based on the labeled data set. With a limited number of human labels, we employ the BERT model [1], which has already encoded a large amount of external knowledge via pretraining. However, language models like BERT usually take pure text as input, whereas we incorporate additional feature types: (1) The job type to be predicted. (2) The last segment of the URL tokens. Based on our observation, many job types are mentioned on a page whose link ends with patterns such as *service.html*. So we extract the last segment of a URL, tokenize it by treating all punctuation marks as separators. (3) Meta information of the associated business of this page, i.e., its category and country.

We found that a simple strategy to deal with multiple features works pretty well – we convert all features to text, prepend the feature name to each feature value to give the model a hint at the feature types, and separate different features by the special token "[SEP]". An example input $\mathbf{x}$ might be: *[CLS] job to predict: tree landscaping [SEP] website: service html [SEP] country: US [SEP] category: landscaper.*

This input $\mathbf{x}$ is fed to a pre-trained BERT model for fine-tuning on the binary classification task of predicting whether the candidate job type is an actual job type (1) or not (0): $\hat{y} = \sigma(\mathbf{W} \cdot \mathbf{h}^{[\mathrm{CLS}]})$, where $\mathbf{h}^{[\mathrm{CLS}]}$ is the contextualized embedding of the CLS token. $\mathbf{W}$ is the fine-tuned parameter matrix and σ is the sigmoid function. The model is trained to minimize the cross entropy loss.

3.2.2 *Improvement Based on the Context.*

The BERT model produces reasonable results, however its performance does not meet our high precision bar. To further improve the performance, we utilize the context information, which is defined as other job types extracted from the same structured field. E.g., if a list includes *legal services* together with items like *education* and *publications*, this page is less likely to be a service page, but a person's homepage describing their resume. We utilize the context information in three ways.

Encoding context jobs. We encode context job types as additional features in the input, which are all the other items from the same structured field. An example input would be: *[CLS] job to predict: tree landscaping [SEP] website: service html [SEP] category: landscaper [SEP] other job: landscape design [SEP] other job: landscape installation.*

Weak supervision. The candidate job types in the same structured field are used for weak supervision. For an unlabeled job type from the same structured field, we create a training example with a weak label whose value is the same as that of the labeled one, but with a lower weight.

Majority vote. As a post-process, we adjust the predictions by a majority vote. If a large proportion of job types from the same list are predicted as negative, we flip the predicted positives to negatives.

3.2.3 *Classification Evaluation.*

We have 77,302 human labeled examples in total, which covers 57,130 unique job types. The examples are partitioned into train/validation/test set with the ratio 0.8/0.1/0.1. We evaluate the baseline BERT and variants of our proposed models on the test set.

BERT. The standard BERT without considering any context information.

ENCODE. We encode context job types as input.

ENCODE$_{\text{WeakSup}}$. On top of ENCODE, we do weak supervision.

ENCODE$_{\text{WeakSup-Vote}}$. We additionally do majority vote.

Table 1: Results for job type classification. We report relative improvement over BERT. *, †, ‡ indicate statistically significant improvement over BERT, ENCODE and ENCODE$_{\text{WeakSup}}$ respectively at the level of 0.05.

	Precision	Recall	ROC-AUC
ENCODE	+2.49%*	+1.70%*	+3.61%*
ENCODE$_{\text{WeakSup}}$	+2.58%*	+1.83%*	+3.88%*
ENCODE$_{\text{WeakSup-Vote}}$	+3.32%*†‡	+2.08%*	+4.57%*†‡

Note: we can only disclose the relative improvement but not the absolute numbers. The absolute precision of all the proposed variants are above 90% and are sufficient for real-world applications.

Based on the results shown in Table 1, ENCODE gives a large boost in performance, meaning that encoding context job types can help the model make a better decision by giving a context of where the candidate appears. ENCODE$_{\text{WeakSup}}$ only marginally improves over ENCODE, probably because the two methods provide similar information. ENCODE$_{\text{WeakSup-Vote}}$ can further improve the performance significantly by employing the majority vote as a post-process. This implies that a hard filter is still beneficial even if the model has considered the context in a soft manner.

3.3 Job Type Extraction

3.3.1 *Addressing the Challenges.*

The task of job type extraction aims to extract job types from any web page, regardless of whether it is free-text only or contains structured fields.

To this end, we need to address two challenges. First, ground-truth positive examples from free-text pages are rare. Sampling free-text pages randomly could leave us with hardly any pages that contain job types. Second, we want to maintain reasonable coverage while being efficient. This means that we cannot afford off-the-shelf NER models, which scan through the text of the entire page and output labels for each word. Furthermore, we want to build a generic system – it should not rely on DOM trees of pages to create extraction rules that only work for a small set of pages.

To address the two challenges, we cast job type extraction as a retrieval problem. Specifically, we utilize the job types predicted

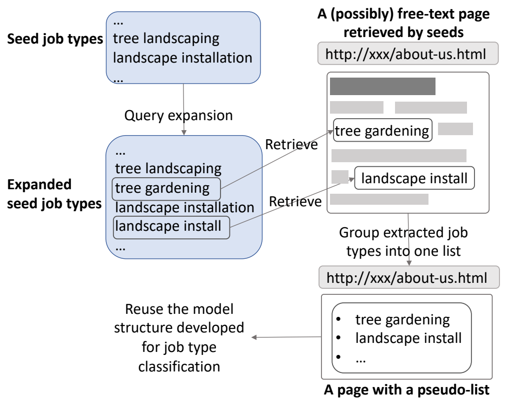

Seed job types
...
tree landscaping
landscape installation
...
Query expansion
Expanded seed job types
tree gardening
landscape install
Retrieve
Retrieve
A (possibly) free-text page retrieved by seeds
http://xxx/about-us.html
Group extracted job types into one list
Reuse the model structure developed for job type classification
A page with a pseudo-list

Figure 2: The procedure of job type extraction.

as positive in the stage of job type classification. We treat them as seeds and use them as search queries to retrieve pages that contain these seeds. Casting it as a retrieval problem has another advantage – we can improve the coverage by query expansion. To this end, we replace each word in a job type with one of its synonyms, producing a new seed job type. Synonyms can be obtained from external lexical databases like WordNet [15]. All possible combinations of original words and synonyms are considered to produce multiple seed job types. By sampling a seed job type and one of its retrieved pages for human labeling, we have greatly improved the possibility of sampling a positive example, solving our first challenge.

The second challenge of model efficiency can be addressed by converting job type extraction to classification. Specifically, one page could be retrieved by multiple seed job types. By treating all the seed job types from one page as context job types in a list, we could reuse the model structure designed for job type classification. In this way, we have considered the information from the entire page by feeding other extracted candidate job types as input without relying on the model to process the entire page text. The process is illustrated in Figure 2.

However, the results are not as promising as we expected. We found a major problem through error analysis.

3.3.2 A New Type of Context. We found that many pages mention job type names for other purposes like giving life tips. For example, a web page that teaches readers to deal with bed bugs might contain a sentence like *a solution is to call* ***home cleaning services*** *if you find bed bugs in your home. They usually provide services like* ***bed bug control***. Though this page mentions multiple job type names, the page is not provided by a home cleaning business.

To alleviate this problem, we exploit an additional type of context information – the surrounding words. Using the example mentioned above, the words to the left of *home cleaning services* is *a solution is to call*, which is not a common pattern when a business introduces its service content. To encode such context, we include up to $k = 5$ words both to the left and to the right of the candidate job type. The resultant input might look like: *[CLS] job to predict: a solution is to call home cleaning services if you find bed bugs [SEP] website: bed bugs html [SEP] category: cleaning [SEP] other job: they usually provide services like bed bug control.*

Using only the surrounding words still maintains scalability **without** feeding the entire page to the model.

Table 2: Results for job type extraction based on relative improvement over JobModel. * indicate statistically significant improvement over JobModel at the level of 0.05.

	Precision	Recall	ROC-AUC
JobModel$_{\text{Surround}}$	+0.31%	+5.79%*†	+8.59%*†

Note: for competitive reasons, we can only disclose the relative improvement but not the absolute numbers. The absolute precision of all the proposed variants is above 90%, sufficient for real-world applications.

Table 3: Relative coverage improvement over Structure.

	# Job types	# Businesses
ExactRetrieval	+2087.0%	+1561.0%
QueryExpand	+4392.9%	+1824.6%

3.4 Extraction Evaluation

We have 69,190 human labeled examples in total, which are partitioned into train/validation/test set with the ratio 0.8/0.1/0.1. We compare two variants of our models on the test set.

The JobModel model shares the same structure as the one for job classification, but is trained on the labeled data for job type extraction. The JobModel$_{\text{Surround}}$ model takes the surrounding words of candidate job types as an additional input.

As shown in Table 2, JobModel$_{\text{Surround}}$ performs significantly better than JobModel, which suggests that the surrounding words could indeed explain the intent of the seed job type mentions. This successfully improves the semantic understanding without processing the entire text of each page, keeping our models efficient.

3.4.1 Coverage improvement. We are also interested in how casting the problem as a retrieval task could contribute to coverage improvement. We compare extracted results from three sources:

Structure. The job types are extracted from structured fields by job type classification.

ExactRetrieval. On top of Structure, we add job types extracted by retrieving web pages using job types from Structure as queries. No query expansion is applied.

QueryExpand. Similar to ExactRetrieval, but we do synonym expansion on queries, as described in Section 3.3.1.

As shown in Table 3, formulating the problem as a retrieval task can bring in 20 times more job types and 15 times more businesses with job types. Through query expansion, we can improve the coverage further. Therefore, the retrieval formulation has greatly improved coverage without relying on NER models to scan through an intractable amount of text.

4 CONCLUSION

We describe our pipeline for extraction of service job types from web pages. We share practical experiences of handling challenges of building this pipeline, including the cold start problem of dataset collection with limited human annotation resources, and achieving high precision and reasonable coverage without hurting scalability. These challenges prevent us from direct application of existing models developed for information extraction. We have developed various strategies to combat the challenges, including (1) bootstrapping structured content to deal with the data collection cold start problem; (2) designing various information contexts for better decision making; (3) casting the extraction task as a retrieval problem to improve coverage while maintaining high scalability. Our pipeline is executed periodically to keep the extracted content up-to-date. It is currently deployed in production, and the output job types are surfaced to millions of Google Search and Maps users.

REFERENCES

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[2] Anna Lisa Gentile, Ziqi Zhang, and Fabio Ciravegna. 2015. Early steps towards web scale information extraction with lodie. *AI Magazine* 36, 1 (2015), 55–64.

[3] Tomas Gogar, Ondrej Hubacek, and Jan Sedivy. 2016. Deep neural networks for web page information extraction. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer, 154–163.

[4] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint arXiv:1508.01991* (2015).

[5] Ben Kantor and Amir Globerson. 2019. Coreference resolution with entity equalization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 673–677.

[6] Nicholas Kushmerick. 1997. *Wrapper induction for information extraction.* University of Washington.

[7] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational linguistics* 39, 4 (2013), 885–916.

[8] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. *arXiv preprint arXiv:1707.07045* (2017).

[9] Kenton Lee, Luheng He, and Luke Zettlemoyer. 2018. Higher-order coreference resolution with coarse-to-fine inference. *arXiv preprint arXiv:1804.05392* (2018).

[10] Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. 2019. A unified MRC framework for named entity recognition. *arXiv preprint arXiv:1910.11476* (2019).

[11] Shengpeng Liu, Ying Li, and Binbin Fan. 2018. Hierarchical RNN for few-shot information extraction learning. In *International Conference of Pioneering Computer Scientists, Engineers and Educators*. Springer, 227–239.

[12] Colin Lockard, Xin Luna Dong, Arash Einolghozati, and Prashant Shiralkar. 2018. Ceres: Distantly supervised relation extraction from the semi-structured web. *arXiv preprint arXiv:1804.04635* (2018).

[13] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 3047–3056.

[14] Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. 2020. ZeroShotCeres: Zero-shot relation extraction from semi-structured webpages. *arXiv preprint arXiv:2005.07105* (2020).

[15] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[16] David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30, 1 (2007), 3–26.

[17] Thien Huu Nguyen, Avirup Sil, Georgiana Dinu, and Radu Florian. 2016. Toward mention detection robustness with recurrent neural networks. *arXiv preprint arXiv:1602.07749* (2016).

[18] Yanyao Shen, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. 2017. Deep active learning for named entity recognition. *arXiv preprint arXiv:1707.05928* (2017).

[19] Emma Strubell, Patrick Verga, David Belanger, and Andrew McCallum. 2017. Fast and accurate entity recognition with iterated dilated convolutions. *arXiv preprint arXiv:1702.02098* (2017).

[20] Lin Yao, Hong Liu, Yi Liu, Xinxin Li, and Muhammad Waqas Anwar. 2015. Biomedical named entity recognition based on deep neutral network. *Int. J. Hybrid Inf. Technol* 8, 8 (2015), 279–288.

[21] Rui Zhang, Cicero Nogueira dos Santos, Michihiro Yasunaga, Bing Xiang, and Dragomir Radev. 2018. Neural coreference resolution with deep biaffine attention by joint mention detection and mention clustering. *arXiv preprint arXiv:1805.04893* (2018).