Term Set Expansion based on Multi-Context Term Embeddings: an End-to-end Workflow

Jonathan Mamou,[1] **Oren Pereg,**[1] **Moshe Wasserblat,**[1] **Ido Dagan,**[2] **Yoav Goldberg,**[2]
Alon Eirew,[1] **Yael Green,**[1] **Shira Guskin,**[1] **Peter Izsak,**[1] **Daniel Korat**[1]
[1]Intel AI Lab, Israel
[2]Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
[1]firstname.lastname@intel.com
[2]{dagan,yogo}@cs.biu.ac.il

Abstract

We present SetExpander, a corpus-based system for expanding a seed set of terms into a more complete set of terms that belong to the same semantic class. SetExpander implements an iterative end-to end workflow for term set expansion. It enables users to easily select a seed set of terms, expand it, view the expanded set, validate it, re-expand the validated set and store it, thus simplifying the extraction of domain-specific fine-grained semantic classes. SetExpander has been used for solving real-life use cases including integration in an automated recruitment system and an issues and defects resolution system.[1]

1 Introduction

Term set expansion is the task of expanding a given partial set of terms into a more complete set of terms that belong to the same semantic class. For example, given the partial set of personal assistant application terms like ‘Siri’ and ‘Cortana’ as seed, the expanded set is expected to include additional personal assistant application terms such as ‘Amazon Echo’ and ‘Google Now’. Many NLP-based information extraction applications, such as relation extraction or document matching, require the extraction of terms belonging to fine-grained semantic classes as a basic building block. A practical approach to extracting such terms is to apply a term set expansion system. The input seed set for such systems may contain as few as two to ten terms which is practical to obtain.

SetExpander uses a corpus-based approach based on the *distributional similarity hypothesis* (Harris, 1954), stating that semantically similar words appear in similar contexts. Linear bag-of-words context is widely used to compute semantic similarity. However, it typically captures more *topical* and less *functional* similarity, while for the purpose of set expansion, we need to capture more functional and less topical similarity.[2] For example, given a seed term like the programming language ’Python’, we would like the expanded set to include other programming languages with similar characteristics, but we would not like it to include terms like ‘bytecode’ or ‘high-level programming language’ despite these terms being semantically related to ‘Python’ in linear bag-of-words contexts.

Moreover, for the purpose of set expansion, a seed set contains more than one term and the terms of the expanded set are expected to be as functionally similar to *all* the terms of the seed set as possible. For example, ‘orange’ is functionally similar to ‘red’ (color) and to ‘apple’ (fruit), but if the seed set contains both ‘orange’ and ‘yellow’ then only ‘red’ should be part of the expanded set. However, we do not want to capture only the term sense; we also wish to capture the granularity within a category. For example, ‘orange’ is functionally similar to both ‘apple’ and ‘lemon’; however, if the seed set contains ‘orange’ and ‘banana’ (fruits), the expanded set is expected to contain both ‘apple’ and ‘lemon’; but if the seed set is ‘orange’ and ‘grapefruit’ (citrus fruits), then the expanded set is expected to contain ‘lemon’ but not ‘apple’.

[1]A video demo of SetExpander is available at https://drive.google.com/open?id=1e545bB87Autsch36DjnJHmq3HWfSd1Rv (some images were blurred for privacy reasons).

[2]We use the terminology introduced by (Turney, 2012): the *topic* of a term is characterized by the nouns that occur in its neighborhood while the *function* of a term is characterized by the syntactic context that relates it to the verbs that occur in its neighborhood.

While term set expansion has received attention from both industry and academia, there are only a handful of available implementations. Google Sets was one of the earliest applications for term set expansion. It used methods like latent semantic indexing to pre-compute lists of similar words (now discontinued). Word Grab Bag[3] builds lists dynamically using word2vec embeddings based on bag-of-word contexts, but its algorithm is not publicly described. State-of-the-art research techniques are based on computing semantic similarity between seed terms and candidate terms in a given corpus and then constructing the expanded set from the most similar terms (Sarmento et al., 2007; Shen et al., 2017).

Relative to prior work, the contribution of this paper is twofold. First, it describes an iterative end-to-end workflow that enables users to select an input corpus, train multiple embedding models and combine them; after which the user can easily select a seed set of terms, expand it, view the expanded set, validate it, iteratively re-expand the validated set and store it. Second, it describes the SetExpander system which is based on a novel corpus-based set expansion algorithm developed in-house; this algorithm combines multi-context term embeddings to capture different aspects of semantic similarity and to make the system more robust across different domains. The SetExpander algorithm is briefly described in Section 2. Our system has been used for solving several real-life use cases. One of them is an automated recruitment system that matches job descriptions with job-applicant resumes. Another use case involves enhancing a software development process by detecting and reducing the amount of duplicated defects in a validation system. Section 4 includes a detailed description of both use cases. The system is distributed as open source software under the Apache license as part of NLP Architect by Intel AI Lab [4]

2 Term Set Expansion Algorithm Overview

Our approach is based on representing any term of a training corpus using word embeddings in order to estimate the similarity between the seed terms and any candidate term.

Noun phrases provide good approximation for candidate terms and are extracted in our system using a noun phrase chunker.[5] Term variations, such as aliases, acronyms and synonyms, which refer to the same entity, are grouped together. We use a heuristic algorithm that is based on text normalization, abbreviation web resources, edit distance and word2vec similarity. For example, *New York, New-York, NY, New York City* and *NYC* are grouped together forming a single term group. Then, we use term groups as input units for embedding training; it enables obtaining more contextual information compared to using individual terms, thus enhancing the robustness of the embedding model.

Our basic algorithm version follows the standard unsupervised set expansion scheme. Terms are represented by their linear bag-of-words window context embeddings using the word2vec toolkit.[6] At expansion time, given a seed of terms, the most similar terms are returned where similarity is estimated by the cosine similarity between the centroid of the seed terms and each candidate term. While word2vec typically uses a linear bag-of-words window context around the focus word, the literature describes other possible context types (Table 1). We found that indeed in different domains, better similarities are found using different context types. The different contexts thus complement each other by capturing different types of semantic relations. Typically, explicit list contexts work well for the automated recruitment system use case, while unary patterns contexts work well for the issues and defects resolution use case (Section 4). To make the system more robust, we extended the basic algorithm to combine multi-context embeddings. Terms are represented with arbitrary context embeddings trained using the generic word2vecf toolkit.[7] Taking the similarity scores between the seed terms and the candidate terms according to each of the different contexts as features, a Multilayer Perceptron (MLP) binary classifier predicts whether a candidate term should be part of the expanded set, where training and development term lists are used for the MLP training. The MLP classifier is implemented on top of Neon,[8] the Intel Nervana Deep Learning Framework. The performance of the algorithm was first evaluated by MAP@n

[3]`www.wordgrabbag.com`

[4]`http://nlp_architect.nervanasys.com/term_set_expansion.html`

[5]`http://nlp_architect.nervanasys.com/chunker.html`

[6]`http://code.google.com/archive/p/word2vec`

[7]`http://bitbucket.org/yoavgo/word2vecf`

[8]`http://github.com/NervanaSystems/neon`

(Mean Average Precision at n). MAP@10, MAP@20 and MAP@50 on an English Wikipedia dataset [9] are respectively 0.83, 0.74 and 0.63.

Context Type	**Example sentence**	**Focus term**	**Context units**
Linear bag-of-words (Mikolov et al., 2013)	*Siri uses voice queries and a natural language user interface.*	*Siri*	*uses, voice queries, natural language user interface*
Explicit lists (Sarmento et al., 2007)	*Experience in Image processing, Signal processing, Computer Vision*	*Image processing*	*Signal processing, Computer Vision*
Syntactic dependency (Levy and Goldberg, 2014)	*Turing studied as an undergraduate ... at King's College, Cambridge.*	*studied*	*(Turing/nsubj), (undergraduate/prep_as), (King's College/prep_at)*
Symmetric patterns (Schwartz et al., 2015)	*Apple and Orange juice drink*	*Apple*	*Orange*
Unary patterns (Rong et al., 2016)	*In the U.S. state of Alaska ...*	*Alaska*	*U.S. state of __*

Table 1: Examples of extracted contexts per context type.

3 System Workflow and Application

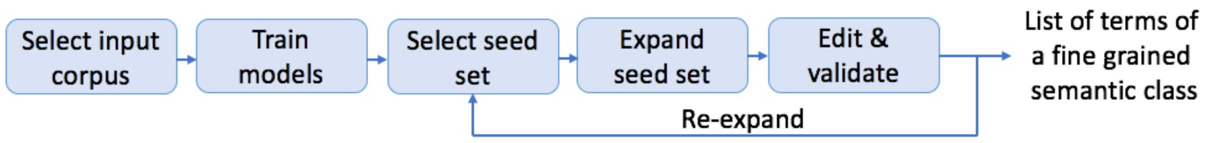


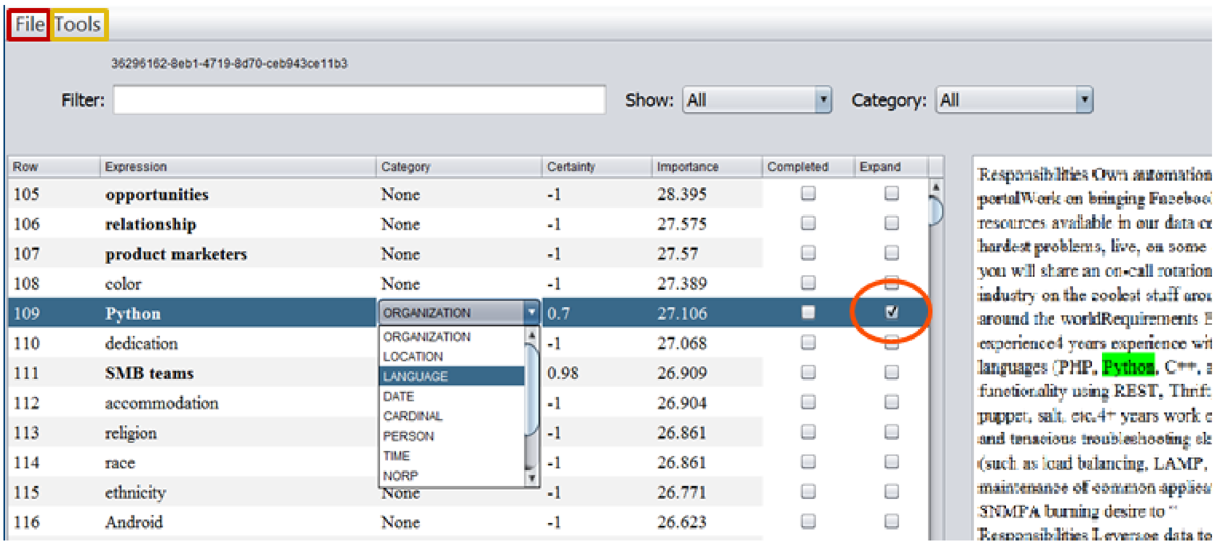
Figure 1: SetExpander end-to-end workflow.

This section describes the iterative end-to-end workflow of SetExpander as depicted in Figure 1. Each step of the flow is performed by the user using the system's user interface (Figures 2 and 3). The first two steps of the flow are to **select an input corpus** and to **train models**. The "train models" step extracts term groups from the corpus and trains the combined term groups embedding models (Section 2). Next, the user is able to **select a seed set** for expansion. Figure 2 shows the seed set selection and expansion user interface. Each row in the displayed table corresponds to a different term group. The term group names are displayed under the 'Expression' column. The 'Filter' text box is used for searching for specific term groups. Upon selecting (clicking) a term group, the context view on the right hand side displays text snippets from the input corpus that include terms that are part of the selected term group (highlighted in green in Figure 2). The user can create a seed set assembled from specific term groups by checking their 'Expand' checkbox (see the red circle in Figure 2). The user can select or set a name for the semantic category of the seed set (see drop down list in Figure 2). Once the seed set is assembled, the user can **expand the seed set** by selecting the Expand option in the tools menu (not shown).

Figure 3 shows the output of the expansion process. The expanded term groups are highlighted in green. The Certainty score represents the relatedness of each expanded term group to the seed set. This score is determined by the MLP classifier (Section 2). The Certainty scores of term groups that were manually selected as part of the seed set, are set to 1. The user can perform **re-expansion** by creating a new seed set based on the expanded terms and the original seed set terms. The user is also able to validate

[9]Dataset is described at `http://nlp_architect.nervanasys.com/term_set_expansion.html`.

each expanded item by checking the “Completed” checkbox. The validated list can then be **saved** and later used as a fine-grained taxonomy input to external information-extraction systems.

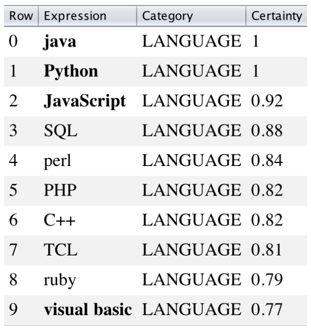


Filter: Show: All Category: All

Row	Expression	Category	Certainty	Importance	Completed	Expand
105	**opportunities**	None	-1	28.395	☐	☐
106	**relationship**	None	-1	27.575	☐	☐
107	**product marketers**	None	-1	27.57	☐	☐
108	color	None	-1	27.389	☐	☐
109	**Python**	ORGANIZATION	0.7	27.106	☐	☑
110	dedication		-1	27.068	☐	☐
111	**SMB teams**		0.98	26.909	☐	☐
112	accommodation		-1	26.904	☐	☐
113	religion		-1	26.861	☐	☐
114	race		-1	26.861	☐	☐
115	ethnicity	None	-1	26.771	☐	☐
116	Android	None	-1	26.623	☐	☐

Responsibilities Own automation
portalWork on bringing Faceboo
resources available in our data c
hardest problems, live, on some
you will share an on-call rotation
industry on the coolest staff aro
around the worldRequirements E
experienced4 years experience wi
languages (PHP, Python, C++, a
functionality using REST, Thrift
puppet, salt, etc.4+ years work e
and tenacious troubleshooting sk
(such as load balancing, LAMP,
maintenance of common applica
SNMPA burning desire to "
Responsibilities Leverage data to

Figure 2: SetExpander user interface for seed selection and expansion.



Row	Expression	Category	Certainty
0	**java**	LANGUAGE	1
1	**Python**	LANGUAGE	1
2	**JavaScript**	LANGUAGE	0.92
3	SQL	LANGUAGE	0.88
4	perl	LANGUAGE	0.84
5	PHP	LANGUAGE	0.82
6	C++	LANGUAGE	0.82
7	TCL	LANGUAGE	0.81
8	ruby	LANGUAGE	0.79
9	**visual basic**	LANGUAGE	0.77

Figure 3: SetExpander user interface for expansion results output. Seed terms are ‘java’ and ‘python’.

4 Field Use Cases

This section describes two use cases in which SetExpander has been successfully used.

4.1 Automated Recruitment System

Human matching of applicant resumes to open positions in organizations is time-consuming and costly. Automated recruitment systems enable recruiters to speed up and refine this process. The recruiter provides an open position description and then the system scans the organizations resume repository searching for the best matches. One of the main features that affect the matching is the skills list, for example, a good match between an applicant and an open position regarding specific programming skills or experience using specific tools is significant for the overall matching. However, manual generation and maintenance of comprehensive and updated skills lists is tedious and difficult to scale. SetExpander was integrated into such a recruitment system. Recruiters used the system’s user interface (Figures 2 & 3) to generate fine-grained skills lists based on small seed sets for eighteen engineering job position categories. We evaluated the recruitment system use case for different skill classes. The system achieved a precision of 94.5%, 98.0% and 70.5% at the top 100 applicants, for the job position categories of Software Machine Learning Engineer, Firmware Engineer and ADAS Senior Software Engineer, respectively.

4.2 Issues and Defects Resolution

Quick identification of duplicate defects is critical for efficient software development. The aim of automated issues and defects resolution systems is to find duplicates in large repositories of millions of software defects used by dozens of development teams. This task is challenging because the same defect may have different title names and different textual descriptions. The legacy solution relied on manually constructed lists of tens of thousands of terms, which were built over several weeks. Our term set expansion application was integrated into such a system and was used for generating domain specific semantic categories such as product names, process names, technical terms, etc. The integrated system enhanced the duplicate defects detection precision by more than 10% and sped-up the term list generation process from several weeks to hours.

5 Conclusion

We presented SetExpander, a corpus-based system for set expansion which enables users to select a seed set of terms, expand it, validate it, re-expand the validated set and store it. The expanded sets can then be used as a domain specific semantic classes for downstream applications. Our system was used in several real-world use cases, among them, an automated recruitment system and an issues and defects resolution system.

Acknowledgements

This work was supported in part by an Intel ICRI-CI grant. The authors are grateful to Sapir Tsabari from Intel AI Lab for her help in the dataset preparation.

References

Zellig S Harris. 1954. Distributional structure. *Word*, 10(2-3):146–162.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 302–308.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Xin Rong, Zhe Chen, Qiaozhu Mei, and Eytan Adar. 2016. Egoset: Exploiting word ego-networks and user-generated ontology for multifaceted set expansion. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 645–654. ACM.

Luis Sarmento, Valentin Jijkuon, Maarten de Rijke, and Eugenio Oliveira. 2007. More like these: growing entity classes from seeds. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 959–962. ACM.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 258–267.

Jiaming Shen, Zeqiu Wu, Dongming Lei, Jingbo Shang, Xiang Ren, and Jiawei Han. 2017. Setexpan: Corpus-based set expansion via context feature selection and rank ensemble. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 288–304. Springer.

Peter D Turney. 2012. Domain and function: A dual-space model of semantic relations and compositions. *Journal of Artificial Intelligence Research*, 44:533–585.