How to use the CSO Classifier in other domains

Angelo A. Salatino, Francesco Osborne
Knowledge Media Institute, The Open University, MK7 6AA, Milton Keynes, UK
`{firstname.lastname}@open.ac.uk`

Abstract. Being able to characterise research papers according to their topics enables a multitude of high-level applications such as i) categorise proceedings in digital libraries, ii) semantically enhance the metadata of scientific publications, iii) generate recommendations, iv) produce smart analytics, v) detect research trends, and others. In our recent work, we designed and developed an unsupervised approach to automatically classify research papers according to an ontology of research areas in the field of Computer Science. This approach uses well-known technologies from the field of Natural Language Processing which makes it easily generalisable. In this article, we will show how we can customise the CSO Classifier and apply it to other fields of Science.

Keywords: Scholarly Data, Digital Libraries, Bibliographic Data, Ontology, Text Mining, Topic Detection, Word Embeddings, Science of Science.

1 Introduction

Being able to characterise research papers according to their topics enables a multitude of high-level applications such as i) categorising proceedings in digital libraries [1], ii) enhancing semantically the metadata of scientific publications, iii) generating recommendations [2], iv) producing smart analytics [3], v) detecting research trends [4, 5], and others.

In our recent work, we designed and developed the CSO Classifier[1] [6], an unsupervised approach to automatically classify research papers according to an ontology of research areas in the field of Computer Science. This approach uses well-known technologies from the field of Natural Language Processing which makes it easily generalisable. In this article, we will show how we can customise the CSO Classifier and apply it to other fields of Science. In particular, we will briefly introduce the CSO Classifier in Section 2, then in Section 3 we will describe the main field-dependent components, such the Computer Science Ontology, and the pre-trained word2vec model, and finally, in Section 4 we will show how to integrate knowledge from other fields in the classifier.

2 CSO Classifier

The CSO Classifier is an application that takes as input the text from abstract, title, and keywords of a research paper and outputs a list of relevant concepts from the Computer Science Ontology (CSO). It consists of two main components: (i) the syntactic module and (ii) the semantic module. Figure 1 depicts its architecture. The syntactic module

[1] GitHub repository for the CSO Classifier — https://github.com/angelosalatino/cso-classifier

parses the input documents and identifies CSO concepts that are explicitly referred to in the document. The semantic module uses part-of-speech tagging to identify promising terms and then it exploits word embeddings to infer semantically related topics. Finally, in a further post-processing module, the CSO Classifier combines the results of these two modules and enhances them by including relevant super-areas.

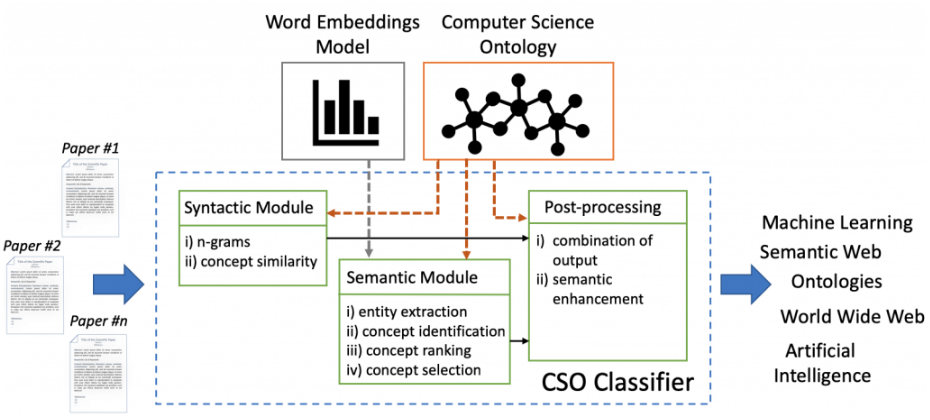

Figure 1: Framework of CSO Classifier (from [6]).

More information about CSO Classifier and how it was developed can be found in [6].

3 External sources

The CSO Classifier takes advantage of two main external sources: i) the Computer Science Ontology (CSO) and ii) a pre-trained word embedding (word2vec) model. The Computer Science Ontology is currently used across all the modules within the CSO Classifier. Indeed, both syntactic and semantic modules use it as a controlled list of topics. The post-processing module uses the superTopicOf relationship to include broader topics that have been inferred from the previous two modules. This allows us to obtain a wider context for the analysed research papers and enable further high-level analytics.

The pre-trained word2vec model, instead, is solely used by the semantic module to support the semantic inference of CSO concepts starting from the identified entities within the metadata. We devote the following subsections to describe more in detail those sources.

3.1 The Computer Science Ontology

The Computer Science Ontology is a large-scale ontology of research areas that was automatically generated using the Klink-2 algorithm [7] on a dataset of 16 million publications, mainly in the field of Computer Science. Differently, from other solutions available in the state of the art, CSO includes a much larger number of research topics, enabling a granular characterisation of the content of research papers, and it can be easily updated by running Klink-2 on recent corpora of publications.

The current version of CSO [8] includes 14K semantic topics and 162K relationships. The main root is Computer Science; however, the ontology includes also a few secondary roots, such as Linguistics, Geometry, Semantics, and others.

The CSO data model[2] is an extension of SKOS[3]. It includes eight semantic relationships (in bold are the relationships that the classifier uses):

- **superTopicOf**, which indicates that a topic is a super-area of another one (e.g., Semantic Web is a super-area of Linked Data).
- **relatedEquivalent**, which indicates that two topics can be treated as equivalent for the purpose of exploring research data (e.g., Ontology Matching and Ontology Mapping).
- **preferentialEquivalent**, this relation is used to state the main label for topics belonging to a cluster of relatedEquivalent. For instance, the topics ontology and ontologies will both have their preferentialEquivalent set to ontology.
- **rdfs:label**, this relation is used to provide a human-readable version of a resource's name.
- contributesTo, which indicates that the research output of one topic contributes to another.
- rdf:type, this relation is used to state that a resource is an instance of a class. For example, a resource in our ontology is an instance of a topic.
- owl:sameAs, which lists entities from other knowledge graphs from the Linked Open Data Cloud (DBpedia, Freebase, Wikidata, YAGO, and Cyc) that refer to the same concepts.
- schema:relatedLink, which links CSO concepts to related web pages that either describes the research topics (Wikipedia articles) or provide additional information about the research domains (Microsoft Academic).

The Computer Science Ontology is available through the CSO Portal[4], a web application that enables users to download, explore, and provide granular feedback on CSO at different levels. Users can use the portal to rate topics and relationships, suggest missing relationships, and visualise sections of the ontology. More information about CSO and how it was developed can be found [8].

3.2 Word Embedding model (Word2vec)

We applied the word2vec approach to a collection of text from the Microsoft Academic Graph (MAG)[5] for generating word embeddings. MAG is a scientific knowledge base and a heterogeneous graph containing scientific publication records, citation relationships, authors, institutions, journals, conferences, and fields of study. It is the largest dataset of scholarly data publicly available, and, as of December 2018, it contains more than 210 million publications.

We first downloaded titles, and abstracts of 4,654,062 English papers in the field of Computer Science. Then we pre-processed the data by replacing spaces with underscores in all n-grams matching the CSO topic labels (e.g., "digital libraries" became "digital_libraries") and for frequent bigrams and trigrams (e.g.,

---

[2] CSO Data Model - https://cso.kmi.open.ac.uk/schema/cso
[3] SKOS Simple Knowledge Organization System - http://www.w3.org/2004/02/skos
[4] Computer Science Ontology Portal - https://cso.kmi.open.ac.uk
[5] Microsoft Academic Graph - https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/

"highest_accuracies", "highly_cited_journals"). These frequent n-grams were identified by analysing combinations of words that co-occur together, as suggested in [9] and using the parameters shown in Table 1. Indeed, while it is possible to obtain the vector of an n-gram by averaging the embedding vectors of all its words, the resulting representation usually is not as good as the one obtained by considering the n-gram as a single word during the training phase.

Finally, we trained the word2vec model using the parameters provided in Table 2. The parameters were set to these values after testing several combinations.

Table 1: Parameters used during the collocation words analysis and for the word2vec model.

	min-count	threshold
collocations (grams)	5	10

Table 2: Parameters used for training the word2vec model.

word2vec	method *skipgram*	emb. size 128	window size 10
	negative 5	max iter. 5	min-count cutoff 10

After training the model, we obtained a **gensim.models.keyedvectors.Word2VecKeyedVectors** object[6] weighing **366MB**.

The size of the model hindered the performance of the classifier in two ways. Firstly, it required several seconds to be loaded into memory. This was partially fixed by serialising the model file (using python pickle[7], see version v2.0 of CSO Classifier, ~4.5 times faster). Secondly, when processing a document, the classifier needs to retrieve the top 10 similar words for all tokens, and compare them with CSO topics. In general, accessing the model to retrieve the most similar words does not require a large amount of time, and indeed the Gensim[8] library is already optimised to perform such operation in the most efficient way. However, this process becomes quite expensive — in terms of time — when processing research papers, since the classifier performs multiple accesses to the model. This multiple access to the model actually requires several seconds to be completed, therefore becoming a bottleneck for the classification process.

To this end, we decided to create a cached model (**token-to-cso-combined.json**) which is a dictionary that directly connects all token available within the vocabulary of the model with the CSO topics. This strategy allows us to quickly retrieve all CSO topics that can be inferred by a particular token. In the next section, we show more in detail the structure of this cache.

[6] Download pre-trained model from here— https://cso.kmi.open.ac.uk/download/model.p
[7] Python object serialization — https://docs.python.org/3/library/pickle.html#module-pickle
[8] Gensim Library — https://pypi.org/project/gensim/

3.2.1 token-to-cso-combined file

As already mentioned, this is a JSON file that contains a large dictionary, which functions as a look-up table between the different possible tokens used when training the word2vec model and its semantically similar CSO concepts.

To generate this dictionary/file, we collected all the different words available within the vocabulary of the model. Then iterating on each word, we retrieved its top 10 similar words from the model, and we computed their Levenshtein similarity against all CSO topics. If the similarity was above 0.7, we created a record which stored all CSO topics triggered by the initial word.

In particular, for each word in our model, we created a key entry within our cached dictionary, as the entry "digital_libraries" shown in the example below. The value of such entry is a list of matched CSO concepts. Each item of this list contains: i) one of the top similar words according to the model (*wet*), ii) the label of the CSO topic most similar to wet (*topic*), iii) the Levenshtein similarity between the topic and the similar word wet (*sim_t*), and iv) the cosine similarity between the vector representation of the key concept and the top similar word (*sim_w*). This large dictionary is then exported into a JSON file: *token-to-cso-combined.json*.

```
"digital_libraries": [
        {
            "topic": "digital_libraries", # CSO Topic
            "sim_t": 1.0, # Lev. Sim.ty between CSO Topic and Word Emb. Token
            "wet": "digital_libraries",    # Word Embedding Token
            "sim_w": 1 # Cosine Sim.ty between key and Word Embedding Token
        },
        {
            "topic": "digital_library",
            "sim_t": 1.0,
            "wet": "digital_library",
            "sim_w": 0.893683910369873
        },
        {
            "topic": "digital_collections",
            "sim_t": 1.0,
            "wet": "digital_collections",
            "sim_w": 0.8221904039382935
        },
        {
            "topic": "institutional_repositories",
            "sim_t": 1.0,
            "wet": "institutional_repositories",
            "sim_w": 0.789232611656189
        }
    ]
```

4 Use the CSO Classifier in other domains

In order to use the CSO Classifier in other domains of Science, it is necessary to replace the two external sources mentioned in the previous section. In particular, there is a need for a comprehensive ontology or taxonomy of research areas, within the new domain, which will work as a controlled list of research topics. In addition, it is important to train a new word2vec model that fits the language model and the semantic of the terms in this particular domain. In the next subsections, we will show how to integrate knowledge from other fields of Science within the CSO Classifier.

4.1 Ontology or taxonomy of research topics

Alternative ontologies or taxonomies of research topics, from other domains, will replace the Computer Science Ontology. In general, a simple — and structureless — list of topics can be a good starting point to replace the CSO. However, we have evidence that a taxonomical structure of topics can definitely improve the results of the classifier. As previously mentioned, CSO consists of several semantic relationships and only four of them are actually used by the CSO Classifier:

- klink:broaderGeneric (which is the legacy relationship for *superTopicOf*)
- klink:relatedEquivalent (which is the legacy relationship for *relatedEquivalent*)
- klink:primaryLabel (which is the legacy relationship for *preferentialEquivalent*)
- rdfs:label

The function *load_cso()* in the misc.py file parses the CSV file containing the CSO ontology and for each row (triple: {subject, predicate, object}) it fills the content of the final cso object, if the predicate is among the previously mentioned relationships.

The Python object loaded in memory is structured as follows:

```
cso = {
        'topics': topics, # the list topics
        'broaders': broaders, # the list of broader topics for a given topic
        'narrowers': narrowers, # inverse of broaders, the list of narrower topics
for a given topic
        'same_as': same_as, # all the siblings for a given topic
        'primary_labels': primary_labels, # all the primary labels of topics, if
they belong to clusters
        'topics_wu': topics_wu, # topic with underscores
        'primary_labels_wu': primary_labels_wu # primary labels with underscores
    }
```

On their turn, all these sub-entities, such as topics, broaders, narrowers, and others, are also dictionaries.

The sub-object **topics** contains a list of topics and each key is connected to a dummy flag always set to true. This is mainly used to check if a topic exists in the ontology.

The sub-object **broaders** connects all topics with its super-topics. Each key is a topic in the ontology, which has at least one super-topic, and the value contains a list of all its super-topics.

The sub-object **narrowers** connects all topics with its sub-topics. Each key is a topic in the ontology, which has at least one sub-topic, and the value contains a list of all its sub-topics.

The sub-object **same_as** connects all topics with its related equivalent topics. Each key is a topic in the ontology, which has at least one equivalent topic, and the value contains a list of all its equivalent topics. This object is logically complete. For instance, if a topic A is relatedEquivalent to B — and therefore B relatedEquivalent to A, — there will be both keys A and B respectively having values B and A, as shown in the example below.

```
{
    ...,
    "same_as": {
        "A": [B],
        "B": [A],
        ...
    },
    ...
}
```

The sub-object **primary_labels** connects related equivalent topics with its preferred label, which is unique for the whole cluster, please refer to the preferentialEquivalent relationship above. Each key is a topic in the ontology, which belongs to a related equivalent cluster of topics, and the value contains the preferred label.

The sub-object **topics_wu** is similar to the sub-object **topics**. The only difference is that the key is a topic in the ontology with its tokens glued by an underscore, and the value contains the actual label of the topics.

The sub-object **primary_label_wu** is similar to the sub-object **primary_labels**. Each key is a topic that belongs to a cluster of related equivalent topics, but its tokens are glued with an underscore. The value contains the preferred label with its tokens also glued by an underscore.

Here follows a reduced example of the final cso object:

```
{
    "topics": { # each key (topic) is connected to a dummy flag (always true).
This is used to quickly check the existence of a topic
        "world wide web": true,
        "wordnet": true,
        "word sense disambiguation": true,
        ...
    },
    "broaders": { # each key (topic) is connected to the list of all its super
topics
        "artificial intelligence": [
            "computer science"
        ],
        "neural network": [
            "machine-learning",
            "machine learning",
            "machine learning techniques",
            "machine learning methods",
            "machine learning algorithms",
            "machine learnings"
        ],
        ...
    },
    "narrowers": { # each key (topic) is connected to the list of all its sub
topics
        "computer science": [
            "artificial intelligence",
            "robotics",
            "computer vision",
            "hardware",
            "computer operating systems",
            "computer networks",
            "bioinformatics",
            "software engineering",
            "information technology",
            "data mining",
            "information retrieval",
            "computer programming",
            "human computer interaction",
            ...
        ],
        ...
    },
    "same_as": { # each key (topic) is connected to the list of all its similar
topics
        "semantic web": [
            "semantic web applications",
            "semantic web technologies",
            "semantic technologies",
            "semantic web technology",
            "semantic technology"
        ],
        "human robot interaction": [
            "human-robot interaction",
            "human robot interactions"
        ],
        ...
    },
    "primary_labels": { # each key (topic) is connected to its preferred label
        "context- awareness": "context-awareness",
        "context awareness": "context-awareness",
        "context-awareness": "context-awareness",
        "architecture designs": "architecture designs",
        "software architecture design": "architecture designs",
        "architecture design": "architecture designs",
        ...
    },
    "topics_wu": { # each key (topic with underscore) is connected to its topic
original topic
        "zero_forcing": "zero forcing",
        "zero_moment_point": "zero moment point",
        "z-scan": "z-scan",
        "yag": "yag",
        "xpath": "xpath",
        ...
    },
    "primary_labels_wu": { # each key (primary label with underscore) is connected
to its original label

        "context-_awareness": "context-awareness",
        "context_awareness": "context-awareness",
        "context-awareness": "context-awareness",
        "architecture_designs": "architecture_designs",
        "software_architecture_design": "architecture_designs",
        "architecture_design": "architecture_designs",
        ...
    }
}
```

This *cso* object is then employed as it is throughout the whole classification process. Therefore, when using another ontology of research topics it would be important to populate this object maintaining the described structure.

4.2 Word2vec model

The newly trained word2vec model will replace the model that the CSO Classifier is currently using. This will need to be trained on a corpus of research papers that fits the new domain of application. Our word embedding model was trained using titles and abstracts from Microsoft Academic Graph in the field of Computer Science. Considering that MAG covers also several other areas of Science, it can be easily re-used to generate the corpus for training the new model. However, depending on the field there can be further available sources that can be used to train the word2vec model, such as PubMed for the field of medicine. One might also think to train a model on the whole MAG corpus, which includes over 300M publications covering the different fields, that would be general enough to suit also other fields. However, we have no evidence on how this one-size-fits-all model would actually perform compared to an ad-hoc one, customised and trained for the individual field.

Regarding the size of the corpus, we are not aware of the minimum amount of text needed to train a good word2vec model. This would be subject to further evaluation.

Here follows a sample of Python code for training the word2vec model using the Gensim library:

```
from gensim.models import word2vec

#######################################################
# READ SENTENCES
sentences = read_sentences(file="corpus.txt")
#######################################################

#######################################################
#PARAMETHERS
model_name='cso-classifier'
SIZE=128
WINDOW=10
MIN_COUNT=10
#######################################################

print("\n-------------------------------------\nCreating model...")
model = word2vec.Word2Vec(sentences, size=SIZE, window=WINDOW, min_count=MIN_COUNT
, max_vocab_size=None, trim_rule=None, sg=1)
print("\n-------------------------------------\nSaving model...")
model.wv.save_word2vec_format(model_name + "[" + str(SIZE) + "-" + str(WINDOW) +
"]_sg.bin", binary=True)
```

4.3 Cached model

The cached model, as already mentioned, is a light-weight representation of the word2vec model. Compared to the trained model, it can be quickly loaded into memory and it almost instantly allows to retrieve the relevant CSO topics associated with a particular word. In order to build this cached model, it is necessary to complete the training phase of the word2vec model first.

Here we provide the Python code we used to create such cached model:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  4 22:52:14 2019
@author: angelosalatino
```

```
"""

import Levenshtein.StringMatcher as ls
import json

########## This is for loading CSO and our model. !! Change the following to lines
to replace them with your ontology and word2vec model.
import misc as misc
cso, model = misc.load_ontology_and_model()
##########

min_similarity = 0.94 #
word_similarity = 0.7 # similarity of words in the model
top_amount_of_words = 10 # maximum number of words to select

output = {}
i=0
for wet, _ in model.vocab.items():

    i+=1
    if(i%1000 == 0):
        print(i, len(output))

    output[wet]=[]

    similar_words = []
    similar_words.append((wet,1)) #Appending the gram with max similarity

    similarities = model.most_similar(wet, topn=top_amount_of_words)
    similar_words.extend(similarities)

    for wet2, sim in similar_words:
        if sim >= word_similarity:
            topics = {}
            topics = [key for key, _ in cso['topics_wu'].items() if key.startswith
(wet2[:4])]
            for topic in topics:
                m = ls.StringMatcher(None, topic, wet2).ratio() #topic is from
cso, wet is from word embedding
                if m >= min_similarity:
                    try:
                        output[wet].append({"topic":topic,"sim_t":m,"wet":wet2,"si
m_w":sim})
                    except KeyError:
                        print(wet)

# now saving the cached model
with open('token-to-cso-combined.json', 'w') as outfile:
    json.dump(output, outfile, indent=4)
```

5 Conclusions

In this article, we showed how we can use the CSO Classifier for classifying research papers in other fields of Science. In particular, we showed that to migrate to other domains it is important to identify a new taxonomy or ontology of research areas that can characterise their topical structure, and a corpus of research papers that will be used to train a new word2vec model. These steps are necessary but might not be sufficient. Although we believe that the structure of CSO Classifier is highly general, which makes it easy to apply to other fields, there might be some variables which value will be subject to further evaluation.

If you plan to use the CSO Classifier, either in Computer Science or in other fields, please do get in touch with us. We would like to keep a record of potential users and provide further support.

References

1. Salatino, A.A., Osborne, F., Birukou, A., Motta, E.: Improving Editorial Workflow and Metadata Quality at Springer Nature. In: The Semantic Web – ISWC 2019. Springer Verlag (2019).
2. Thanapalasingam, T., Osborne, F., Birukou, A., Motta, E.: Ontology-Based Recommendation of Editorial Products. In: International Semantic Web Conference 2018. , Monterey, CA (USA) (2018).
3. Mccallum, a., Mann, G.S., Mimno, D.: Bibliometric impact measures leveraging topic analysis. Proc. 6th ACM/IEEE-CS Jt. Conf. Digit. Libr. (JCDL '06). (2006).
4. Chang, Y.W., Huang, M.H., Lin, C.W.: Evolution of research subjects in library and information science based on keyword, bibliographical coupling, and co-citation analyses. Scientometrics. 105, 2071–2087 (2015).
5. Salatino, A.A., Osborne, F., Motta, E.: AUGUR: Forecasting the Emergence of New Research Topics. In: Joint Conference on Digital Libraries 2018, Fort Worth, Texas. pp. 1–10 (2018).
6. Salatino, A.A., Osborne, F., Thanapalasingam, T., Motta, E.: The CSO Classifier: Ontology-Driven Detection of Research Topics in Scholarly Articles. In: TPDL 2019: 23rd International Conference on Theory and Practice of Digital Libraries. Springer.
7. Osborne, F., Motta, E.: Klink-2: Integrating Multiple Web Sources to Generate Semantic Topic Networks. In: The Semantic Web - ISWC 2015. pp. 408–424 (2015).
8. Salatino, A.A., Thanapalasingam, T., Mannocci, A., Birukou, A., Osborne, F., Motta, E.: The Computer Science Ontology: A Comprehensive Automatically-Generated Taxonomy of Research Areas. Data Intell. (2019).
9. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Distributed Representations of Words and Phrases and their Compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013).