Using Word Embeddings for Ontology-Driven Aspect-Based Sentiment Analysis

Sophie de Kok
Erasmus University Rotterdam
Rotterdam, the Netherlands
sophiedekok@gmail.com

Flavius Frasincar
Erasmus University Rotterdam
Rotterdam, the Netherlands
frasincar@ese.eur.nl

ABSTRACT

Nowadays, the Web is the main platform to gather information. The growing amount of freely available unstructured data has increased the interest in sentiment analysis, where the goal is to extract opinions from text. In this paper we focus on review-level aspect-based sentiment analysis, where we predict the sentiment of a certain aspect in a review. We propose a two-stage sentiment analysis algorithm. In the first stage a domain ontology is utilized to predict the sentiment. If the domain ontology stage is inconclusive, a back-up stage based on an SVM bag-of-words model is employed. Furthermore, the use of word embeddings to improve the domain ontology coverage in the first stage by finding semantically similar words is investigated. We find that the two-stage approach significantly outperforms two baseline methods and achieves competitive results for the SemEval-2016 data. Furthermore, by not employing the back-up stage, we still perform significantly better than the baselines. Lastly, we find that employing word embeddings improves the accuracy when the domain ontology size is relatively small.

CCS CONCEPTS

• **Information systems → Sentiment analysis**; *Information extraction*; Web mining;

KEYWORDS

aspect-based sentiment analysis, review-level sentiment analysis, domain ontology, word embeddings

ACM Reference Format:
Sophie de Kok and Flavius Frasincar. 2020. Using Word Embeddings for Ontology-Driven Aspect-Based Sentiment Analysis. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3341105.3373848

1 INTRODUCTION

Nowadays, the Web is the main platform to gather information [17]. Since the Web 2.0 era, people are invited to share their thoughts and opinions online [15]. This results in a growing number of online reviews available for different products. This user-generated content helps other users make decisions. Furthermore, these reviews also help businesses to improve. The growing amount of freely available unstructured data has increased the interest in sentiment analysis. In sentiment analysis or opinion mining, the goal is to extract an opinion from a (review) text [9].

There are different levels of sentiment analysis [9]. We focus on aspect-based sentiment analysis, where the goal is to predict the sentiment of a certain aspect. An advantage of aspect-based sentiment analysis is that we can extract multiple opinions from one review. Thus, someone can be positive about one aspect of a product and be negative about another aspect. For example, in the single sentence review "*The servers were friendly, but the food was not tasty.*", we have two different aspects, namely *service* and *food*. The polarity of the first aspect is *positive* and the polarity of the second aspect is *negative*. Therefore, aspect-based sentiment analysis can be used for a more detailed analysis than, e.g., document-level sentiment analysis as it utilizes more information of a review [18].

Furthermore, it is found that using a knowledge base improves the accuracy of sentiment analysis [19]. We also make use of a knowledge base in our model by creating a domain ontology consisting of domain concepts and their associated axioms. The axioms define relations between the concepts and help to extract implicitly stated information from reviews. For example, when in a review it is mentioned that the waiters are friendly, we can infer that the service was good. Moreover, a knowledge base is not dependent on how much training data there is available. This can be of value in situations where annotated data might be limited.

When there is not enough data available to model language correctly this is called data sparsity [2]. In this case we address possible data sparsity when using the domain ontology by making use of word embeddings. Word embeddings are NLP techniques to represent words as vectors of numbers, where semantically similar words ought to have similar vectors. As the lexicalized domain ontology is created manually, there might not be enough data to properly use it, as it might not be large enough and thus have limited lexical and semantic coverage. By employing word embeddings, more words that are not present in the lexicalized domain ontology, can be used to improve the number of words found in the text matching the employed ontology.

There are many different methods researched for aspect-based sentiment analysis. Most methods are based on using either a knowledge base like sentiment scores, machine learning, or a combination of both [18]. We propose a two-stage algorithm for review-level aspect-based sentiment analysis. First, we use a knowledge base in the form of an ontology to predict the sentiment of the aspects. If the ontology phase is inconclusive, we fall back on a bag-of-words

SAC '20, March 30-April 3, 2020, Brno, Czech Republic
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373848

model in combination with machine learning. We compare the accuracy of this two-stage ontology-driven aspect-based sentiment analysis algorithm with baseline algorithms.

Lastly, as our domain ontology is manually created, it is limited in lexical and semantic coverage. Therefore, we possibly miss relationships between words and the ontology might be too small to be properly used in the first phase. We research the effect of employing word embeddings to improve the first stage of the two-stage algorithm. For this we experiment with different ontologies varying in size. Furthermore, we study different word embedding models to compute the word vectors. The word vectors are computed when a word is not present in the lexicalized domain ontology. For such a word we compute its neighbouring words by finding vectors that are similar to the computed word vector. By checking if the nearest neighbours of the word are present in the ontology, we can capture more words. This increases the number of relevant review words that appear in the ontology and thus the first stage of the algorithm might improve as we have more hits in the ontology, and thus a better domain coverage.

2 RELATED WORK

The authors of [18] give an overview of aspect-based sentiment analysis. Both aspect detection as well as aspect-based sentiment analysis are discussed. For aspect-based sentiment analysis the authors categorize the algorithms based on the methods used. They distinguish three different types of algorithms: only using a knowledge base - like sentiment scores, only using machine learning algorithms, or a hybrid method that combines both. We will make use of a hybrid method that combines a knowledge base in the form of an ontology with a machine learning algorithm as back-up as the domain ontology and the machine learning algorithm can augment each other. The ontology is able to recognize concepts and relations that the machine learning algorithm might not capture and vice versa.

The authors of [1] also make use of an ontology to perform sentiment analysis. The domain specific ontology is created using ConceptNet [10] and WordNet [5]. First, the important product features or aspects in a review are detected by a domain ontology. Then, the opinion words that belong to these aspects are obtained with the Stanford dependency parser [11]. Using a polarity lexicon the authors then determine the sentiment scores and aggregate these to predict the polarity of an entity. The method using an ontology has a better accuracy than the considered baseline. Like [1], we make use of an ontology for performing sentiment analysis. Furthermore, we also employ a dependency parser to find words that belong to aspects by calculating dependency based word windows. However, instead of using a polarity lexicon to determine sentiment scores we use the ontology to calculate ontology scores in terms of the hits.

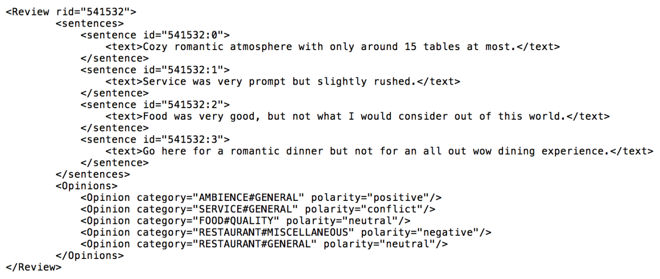
Most approaches that employ word embeddings for sentiment analysis use the word vectors as features for a machine learning classifier. The authors of [21] make use of word embeddings to perform sentiment analysis on tweets. They create sentiment-specific word embedding features and combine these with handcrafted features to use with a support vector machine. Their word vectors are created using a neural network that is trained on a corpus of 10 million tweets. This paper participated in the competition of SemEval-2014. Of the 45 participating teams they achieved the second best result.

The authors of [6] and [8] also use word embeddings as features of their classifier to perform sentiment analysis. [6] employs the CBOW architecture of the `word2vec` algorithm to make domain specific semantic word clusters. They then create features that indicate if a certain word cluster is present in the current sentence. [6] predicts the polarity for each aspect on sentence level employing a Maximum Entropy classifier. Then, they aggregate these predictions for each aspect category in a review and obtain review-level aspect category sentiment predictions. [8] uses the pre-trained GoogleNews word vectors as features in their Logistic Regression classifier. Including these features improved their results. In this paper we also employ word vector to perform sentiment analysis. However, we do not use the created word vectors as features of a machine learning algorithm, but employ them for our ontology to improve the first phase of the algorithm. We use the pre-trained GoogleNews word vectors and we also create our own restaurant specific word vectors using the `word2vec` framework.

The authors of [22] employ word embeddings to improve the performance of semantic textual similarity. Many NLP methods depend on co-occurrences of words to evaluate the similarity between texts. However, this does not always need to be the case. As an example the authors give the following two sentences: *A storm will spread snow over Shanghai* and *The earthquakes have shaken parts of Oklahoma.* These sentences contain similar information, however they do not have a single word in common. The authors combine pre-existent features of similarity measures with word embedding features. They define two different word embedding features. The first feature, micro, uses the similarity between words. In the aforementioned sentences, 'storm' is similar to 'earthquake', 'spread' to 'shaken' and Shanghai and Oklahoma are both places. They then take the difference between each similar word pair, e.g., 'earthquake' and 'storm', and sum these to get a semantic similarity value. For their second feature, macro, the authors find the centroid of each sentence by determining a point that has the same distance to all the word vectors of the sentence. Then, they define the semantic similarity of two sentences as the distance between corresponding centroids. They find that using both word embeddings features improves their accuracy. Like [22], we also employ word embeddings to find similar words to improve our results, but this in relation to a domain ontology.

3 DATA

The data used to train and evaluate the sentiment analysis model is obtained from SemEval-2016 [16]. The test data is used to evaluate our models and compare them with each other and with the participants of SemEval-2016. The training data has two purposes. First, it is used to train the back-up model. Second, part of the training data is also used as an evaluation set while developing our models. This way our model is optimized independently of the test data. A `notion` is defined as an aspect category paired with a review in which it is mentioned. The textual unit of a `notion` contains the text of the review. The training data exists of 1435 `notions` (335 different reviews) and the test data contains 404 `notions` (90 reviews).



```
<Review rid="541532">
        <sentences>
            <sentence id="541532:0">
                <text>Cozy romantic atmosphere with only around 15 tables at most.</text>
            </sentence>
            <sentence id="541532:1">
                <text>Service was very prompt but slightly rushed.</text>
            </sentence>
            <sentence id="541532:2">
                <text>Food was very good, but not what I would consider out of this world.</text>
            </sentence>
            <sentence id="541532:3">
                <text>Go here for a romantic dinner but not for an all out wow dining experience.</text>
            </sentence>
        </sentences>
        <Opinions>
            <Opinion category="AMBIENCE#GENERAL" polarity="positive"/>
            <Opinion category="SERVICE#GENERAL" polarity="conflict"/>
            <Opinion category="FOOD#QUALITY" polarity="neutral"/>
            <Opinion category="RESTAURANT#MISCELLANEOUS" polarity="negative"/>
            <Opinion category="RESTAURANT#GENERAL" polarity="neutral"/>
        </Opinions>
</Review>
```

Figure 1: Annotated data fragment

In Fig. 1, a fragment of the annotated data is shown. This review consists of four sentences and mentions five aspect categories. The data is pre-processed using the Stanford CoreNLP tool [11] before being used in the algorithm. The data is tokenized, Part-Of-Speech (POS) tags are assigned, and all words are lemmatized. Lastly, the data is processed by a dependency parser which finds all the relations between words.

For each `notion` the algorithm predicts the sentiment label of the corresponding aspect category in the review. As people can mention a certain aspect multiple times in a review, a sentiment label that summarizes all sentiment mentioned regarding this aspect category is assigned. We distinguish four different sentiment labels: positive, negative, neutral, and conflict. The conflict label is assigned to a `notion` when conflicting sentiments are expressed. For example, *"The lasagna was delicious, but the risotto was not good"*, has the sentiment label 'conflict' assigned for the aspect FOOD#QUALITY.

Fig. 2 and 3 show how the aspect categories and the sentiment is distributed. Each review always has the aspect category RESTAURANT#GENERAL, which represents the overall sentiment of the review. The aspect category FOOD#QUALITY is also almost always mentioned in a review. Furthermore, the sentiment labels are imbalanced as approximately 70% of the `notions` have a positive sentiment. This indicates that if we always predict positive we already achieve an accuracy of about 70%. Lastly, there are only a few `notions` that have the sentiment labels 'conflict' or 'neutral' assigned.

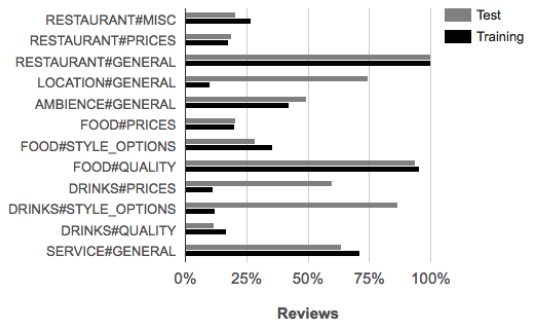


Figure 2: Relative frequencies for each aspect category label

To train the word embedding models, data is obtained from external sources to have enough data for representative word vectors. We use two datasets to create different word embedding models.

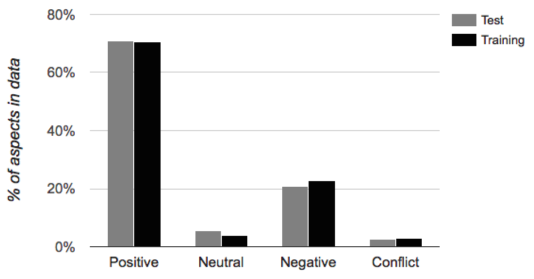


Figure 3: Relative frequencies for each sentiment label

The first word embedding model uses the pre-trained GoogleNews word vectors from the original `word2vec` project. This set is trained on news articles containing approximately 100 billion words from the GoogleNews dataset resulting in 3 million different word vectors.

The other model consists of vectors that we train ourselves. The input corpus consists of reviews obtained from the Yelp Dataset Challenge [23] Round 9. The dataset contains 4.1 million reviews. As not all of the reviews are restaurant reviews, the dataset is filtered using the business IDs such that only restaurants are in the dataset. Furthermore, all non-English reviews are removed using the language-detection package [14]. There are 1779 different restaurant businesses present in the filtered dataset totalling to 95437 English restaurant reviews with approximately 11 million words. The dataset consists only of restaurant reviews to ensure that the word vectors created are best suited for the restaurant domain. For example, for the word 'turkey' we want 'chicken' and 'meat' to be returned as related words instead of 'Erdogan' and 'Ankara'. The reviews are first tokenized, before being used as input in the `word2vec` algorithm from the deeplearning4j package [20].

4 FRAMEWORK

Now we will discuss the framework employed for this paper. First, we discuss the structure of the domain ontology. Then, we elaborate on the use of word embeddings. Finally, we present the two-stage algorithm that we propose.

4.1 Ontology

The domain ontology is divided into multiple classes grouped by types. The main classes represent the sentiment, the aspects, and the words that denote sentiment. The first main class contains the subclasses *Positive*, *Negative*, and *Neutral*. The second class called *AspectIndicator*, contains expressions that indicate if the aspect is implied in the text. For example, when the word 'waiter' is present in the review we can deduce that it is being talked about the aspect service. All expressions in this class are linked to their corresponding aspect categories. Thus, *Waiter* is connected to the aspect category SERVICE#GENERAL by the axiom $Waiter \sqsubseteq \exists aspect.\{\text{'SERVICE\#GENERAL'}\}$. Furthermore, the lexical representation of a word is linked to the corresponding concept in the ontology. For example, the lexical form 'waiter' is linked to the ontology concept *Waiter* by the axiom $Waiter \sqsupseteq \exists lex.\{\text{'waiter'}\}$.

The last class *SentimentWord* is split in different types that represent different categories of expressions. The first category contains words that are always positive or negative and are independent of an aspect. E.g., 'good' belongs to this category as this word is always positive and not dependent on an aspect. The second type contains sentiment words that only belong to a single aspect. For example, 'helpful' only belongs to the aspect SERVICE#GENERAL and is always positive. The third type has words that belong to more than one aspect, but not to all, and are either always positive or always negative independent of the aspect. An example is 'delicious'. This word is always positive and belongs to both DRINKS#QUALITY and FOOD#QUALITY. The fourth and last class has the remainder of the sentiment words, which are words that are positive or negative dependent on the context. For example, 'cold' in combination with fries has a negative sentiment, but in combination with beer is positive. We make use of axioms to represent these relations.

In Fig. 4 a graphical representation of the ontology class *SentimentWord* is depicted. For example, *Cold* represents the type-4 word 'cold' that can be both positive and negative. Furthermore, the asterisk indicates that *Cold* is related to all aspect categories of Drink and also is related to all aspect categories of Food. An aspect category is for example DRINKS#QUALITY.

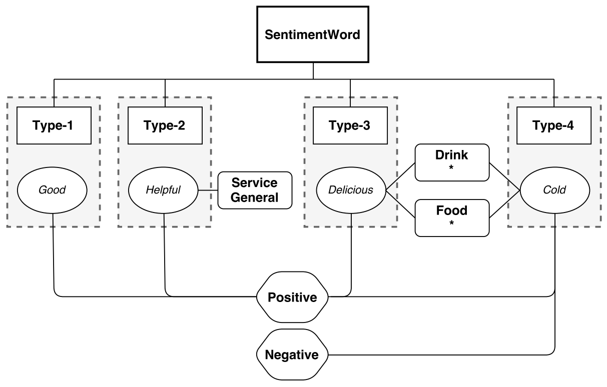


Figure 4: SentimentWords class in the ontology

In Fig. 5 part of the class *AspectIndicator* is represented. The asterisk represents the other aspect categories. The concept *ColdDrink* is here part of all the aspect categories that are related to Drink. Furthermore, *ColdDrink* in combination with *Cold* is a subclass of Positive. The concept *ColdDrink* represents for example the word 'beer'.

The main ontology consists of 95 AspectIndicators and 141 SentimentWords. To test the influence of word embeddings we create several smaller sub-ontologies. The first sub-ontology consists of 90% of the original elements, the second one of 80% etc., where the ratio of AspectIndicators to SentimentWords is kept the same. Furthermore, the most specific concepts like *Breathtaking* are removed first resulting in the smallest ontology only containing fundamental concepts like *Drink* and *Good*. In total there are ten ontologies differing in size.

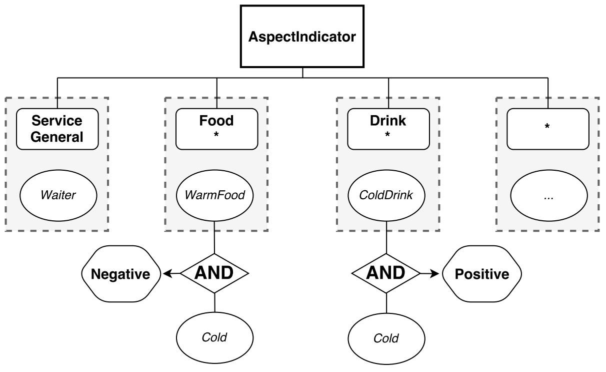


Figure 5: AspectIndicator class in the ontology

4.2 Word Embedding

We employ the `word2vec` framework to train word embedding models [12]. The framework implements two different models and different training methods. The first model, the Continuous Bag-Of-Words (CBOW) method, takes the surrounding words, the context of a word, as input and tries to predict the word. The other model, the skip-gram method, takes the word itself as input and predicts the context of the word. A graphical representation of both methods is depicted in Fig. 6, where t is the position of the current word and k the size of the context window. The CBOW model is faster than the skip-gram model, but the skip-gram model performs better for infrequent words. As CBOW averages the vectors of the context words to predict the center word it is worse at predicting uncommon words. Assume we have the following two sentences: "The food was delicious" and "The food was devine". CBOW uses the context to predict the word of interest. Now given the context [the, food, was], where we want to predict the last word, the model is much more likely to predict 'delicious' because CBOW predicts the most probable word.

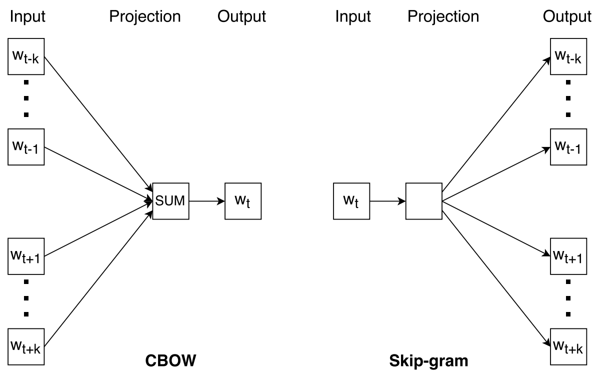


Figure 6: Continuous Bag-of-Words and skip-gram methods

Furthermore, the framework implements two training algorithms. The first algorithm is the hierarchical softmax algorithm. This algorithm is better for infrequent words. The second training algorithm is the negative sampling algorithm. This algorithm performs better for frequent words and also works better with low dimensional vectors [13]. The framework has many different parameters. We only consider the main parameters here. First, the dimensionality parameter determines the number of dimensions of the word vectors, where usually a higher dimensionality is better. However, more dimensions results in a longer computation time. Next, the `word2vec` context window size parameter determines how many words are part of the context of a word based on the proximity to the word. Furthermore, the minimum word frequency parameter determines how often a word has to be in the corpus to be considered. Finally, the sub-sampling frequency makes frequent words less important by removing frequent words with a probability of $p = \frac{f-t}{f} - \sqrt{\frac{t}{f}}$, where f is the frequency of the word and t is the sub-sampling parameter. The accuracy and the speed can be improved by sub-sampling the frequent words of a large data set as the word vectors of frequent terms like 'the' do not change significantly after a few million iterations [13].

The first word embedding model uses Google's pre-trained `word2vec` vectors with vector dimension 300. The model is trained using the Continuous Bag of Words (CBOW) architecture, where the mean of the context vectors is used to find the vector of the center word. The context window size parameter is set to 5, the minimum word frequency equals 10 and the sub-sampling of frequent words threshold is set at 10^{-5}. Finally, negative sampling is used with the noise word parameter equal to 3.

The other word embedding model is a `word2vec` model created using the deeplearning4j package [20]. As input we use the cleaned, tokenized Yelp restaurant review data as described in the data section. We compare two word embedding models. The first model is a skip-gram model with the default parameters: context window size of 5, a minimum word frequency of 5, a learning rate of 0.025 and minimum learning rate of 10^{-4}. The vectors have a dimension of 100. The other Yelp model is a CBOW model also configured with the same default parameters. Both models are trained using the hierarchical softmax algorithm, as the Yelp dataset is relatively small and we want to appropriately model also the infrequent words.

4.3 Algorithm

Our approach consists of a two-stage algorithm. In Fig. 7 an activity diagram of the approach is depicted. First, we obtain the ontology words of the textual unit of a `notion`. If a word is not present in the ontology we make use of word embeddings to find if there is a match in the ontology. The next step is to calculate the ontology scores. If there are hits in the ontology, we check if the outcome of the ontology scores is conclusive, thus if only one sentiment is assigned. If the outcome is inconclusive, we use the back-up algorithm to predict the sentiment. Below we explain how the hits are counted and when an outcome is inconclusive.

4.3.1 Ontology Words. The first step in the algorithm is getting the ontology words. It starts by checking if a word is present in the ontology. When a word is not present in the ontology, it calculates the corresponding word vector. Then, it finds all neighbours of the word. Neighbours are words that are related to the original word found by the different word embedding models. If the similarity between the neighbour and the original word exceeds an optimized

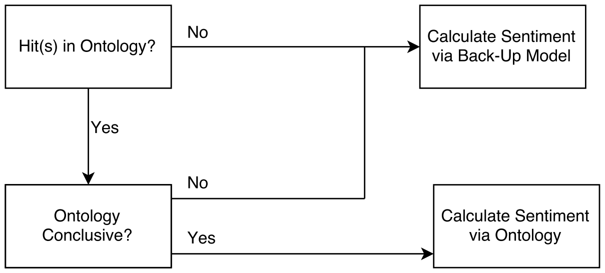

Hit(s) in Ontology?
No
Calculate Sentiment via Back-Up Model
Yes
No
Ontology Conclusive?
Yes
Calculate Sentiment via Ontology

Figure 7: Activity Diagram

similarity threshold t, the algorithm checks if the neighbour is present in the ontology. If the neighbour is present, it then adds the new word to the set of ontology words. The pseudocode for this method is presented in Algorithm 1. To save computation time, we first find the neighbours of all non-ontology words and pre-compute the similarity before running the algorithm.

Algorithm 1: Ontology Words Algorithm

```
Get the ontology words from notion $n$.
function getOntologyWords(ont, n)
    Set $ontologyWords$
    foreach $word \in textualUnit_n$ do
        if $inOntology(word) = true$ then
            $ontologyWords \longleftarrow ontologyWords \cup word$
        else
            Set $neighbours \longleftarrow getNeighbours(word)$
            foreach $neighbourWord \in neighbours$ do
                if $inOntology(neighbourWord) = true$ &
                  $similarity(word, neighbourWord) \geq t$ then
                    $ontologyWords \longleftarrow$
                      $ontologyWords \cup neighbourWord$
                end
            end
        end
    end
return $ontologyWords$
```

4.3.2 Ontology Scores. The next step in the algorithm is calculating the ontology scores. Fig. 8 shows how the ontology scores per `notion` are determined when there are hits in the ontology. There are four types of sentiment expressions as explained before. For each type $i = 1, 2, 3, 4$, we return p_i, the number of positive sentiment hits in the ontology, n_i, the number of hits in the ontology with a negative sentiment and t_i, the number of hits in the ontology with a neutral sentiment. Aggregating the number of hits with their corresponding weights w_i, results in three different scores P, N, and T which are the ontology scores. Furthermore, as we want to optimize the effect of negative ontology concepts with respect to the positive concepts, we multiply the negative score N with the negativity parameter ϑ. There is not a neutrality parameter as the algorithm first checks if the predicted sentiment is positive or negative. Only when this is not the case, it checks if the neutral score T is larger than zero.

Each type of sentiment word is handled differently as follows:

Type-1. As we are working with reviews, we need to take into account that if an expression with, e.g., a general positive sentiment is found at the very beginning of the review, it is unlikely that this

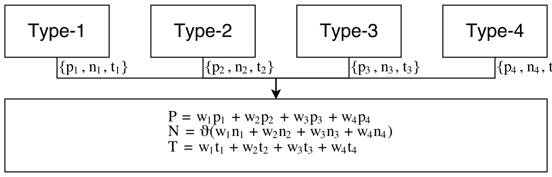

Type-1
Type-2
Type-3
Type-4
{p₁, n₁, t₁}
{p₂, n₂, t₂}
{p₃, n₃, t₃}
{p₄, n₄, t₄}
P = w₁p₁ + w₂p₂ + w₃p₃ + w₄p₄
N = ϑ(w₁n₁ + w₂n₂ + w₃n₃ + w₄n₄)
T = w₁t₁ + w₂t₂ + w₃t₃ + w₄t₄

Figure 8: Ontology Scores

expression affects aspects that are only mentioned at the end of the review. For example, assume a review contains 10 sentences. In the first sentence the word 'good' is present and only in the last sentence the aspect SERVICE#GENERAL is mentioned. It is then unlikely that the word 'good' found in the first sentence, affects the aspect mentioned only in the last sentence. Therefore, for type-1 expressions we make use of a word window to determine whether an expression is relevant for the current aspect. This dependency-based word window is established using the relations of the type-1 sentiment word with the other words in the review. Next, the algorithm finds all the aspect indicators, which are elements of the ontology class *AspectIndicator*, present in the word window. If any of these indicators is linked to the current aspect, then it counts the type-1 expression as a hit for the current `notion`. For example, assume that we currently have a `notion` *n*, which consists of the review in Example 4.1 and is paired with the aspect SERVICE#GENERAL. As 'nice' is a type-1 expression, we find its word window [are, the, ., and, nice, cheap, food, waiters, so, is] (windows size 1 and taking all the dependent parents and children). Next, we loop over the words in this word window. As 'waiter' is an aspect indicator in our ontology and is connected to the current aspect category SERVICE#GENERAL, we count 'nice' as a positive type-1 hit for this `notion`.

Example 4.1. "The food is so cheap and the waiters are nice."

Type-2. When the algorithm finds a type-2 expression in the text, it checks if the aspect linked to the type-2 sentiment word matches the aspect of the current `notion`. If the aspect categories match, the word is counted as a type-2 hit. Assume that the current `notion` consists of the review in Example 4.2 paired with the aspect AMBIENCE#GENERAL. As 'peaceful' is a type-2 expression with a positive sentiment linked to the aspect AMBIENCE#GENERAL, it counts as a positive type-2 hit for the current `notion`. The link between a sentiment word and its aspect is represented by the following axiom: $Peaceful \sqsubseteq \exists\, aspect.\{\text{'AMBIENCE\#GENERAL'}\}$.

Example 4.2. "This is such a lovely, peaceful place to eat outside."

Type-3. A type-3 expression is handled in the same way as a type-2 expression. When we find a type-3 sentiment word, we check if one of the linked aspects correspond to the aspect category of the current `notion`. If this is the case, we have a type-3 hit. For example, assume that the current `notion` has aspect category FOOD#QUALITY. As the word 'delicious' in Example 4.3 is linked to the aspects FOOD#QUALITY and DRINK#QUALITY, this is counted as a positive type-3 hit.

Example 4.3. "The service was excellent and the food was delicious."

Type-4. Type-4 expressions are special as they are only counted when an axiom in the ontology is triggered. When a type-4 word is found, the word window surrounding this sentiment word is calculated. Next, the algorithm finds the aspect indicators that are present and check for each of the indicators if there is an axiom with the corresponding type-4 expression. For example, the word 'cold' has two different sentiments in the two examples below.

Example 4.4. "Instead ordered an ice cold beer which to me works with indian."

Example 4.5. "The pizza was delivered cold and the cheese wasn't even fully melted!"

In Example 4.4, 'cold' refers to the beer, which is positive. However, in the Example 4.5 it refers to the pizza, which carries a negative sentiment. In the ontology, we have two different axioms defined for this example:

$$ColdDrink \sqcap Cold \sqsubseteq Positive \tag{1}$$
$$WarmFood \sqcap Cold \sqsubseteq Negative \tag{2}$$

Axiom 1 states that the intersection of class *Cold* with the class *ColdDrink* is a subclass of *Positive*. Moreover, 'beer' belongs to the class *ColdDrink* and 'pizza' to *WarmFood*. This if formally defined as follows:

$$Cold \sqsupseteq \exists\, lex.\{\text{'cold'}\} \tag{3}$$
$$ColdDrink \sqsupseteq \exists\, lex.\{\text{'beer'}\} \tag{4}$$
$$WarmFood \sqsupseteq \exists\, lex.\{\text{'pizza'}\} \tag{5}$$

Thus, if 'cold' is found in combination with 'beer' like in Example 4, it counts as a positive type-4 hit. However, when 'cold' is found in combination with the word 'pizza' this is counted as a negative type-4 hit.

Furthermore, for each type of expression we check for negation. If one of the two preceding words of the sentiment word in the sentence is a negating word like 'not' or 'never', we flip the polarity of the hit around. Thus, if we find a positive sentiment word with a negating word we count it as a negative hit. We look at two preceding words as the authors of [7] found that looking at two words following a negation word improved their sentiment classification performance the most.

For type-1 and type-4 expressions the algorithm employs word windows. To determine which words will be part of a certain word window, we take the structure of the review into account using the Stanford CoreNLP package [11]. In particular, we use the Stanford dependency parser to find the words that belong to the word window. The size of the word window equals $1 + \omega$ as the word window always includes the center expression. If the expression is multi-worded like 'nothing special', both words are part of the center expression. In Algorithm 2, the pseudocode of how the word windows are determined is given.

In the last stage of the two-part algorithm, the algorithm predicts the sentiment of the `notions`. It uses the ontology scores calculated previously to make a prediction. If there are hits in the ontology it checks if the outcome is conclusive, thus if only one sentiment is assigned. We compare the ontology scores for positive and negative, where one score has to be at least ε larger than the other to account

Algorithm 2: Word Window Algorithm

```
Returns a set of all words in the word window surrounding the
  $centerExpression$. The functions $getParents(word)$ and
  $getChildren(word)$ return all the words that are respectively parents
  and children to the input $word$ as annotated by a dependency parser.
function getWordWindow(centerExpression)
    Set $window \longleftarrow centerExpression$
    for $i \leftarrow 1$ to $\omega + 1$ do
        foreach $word \in window$ do
            $window \longleftarrow window \cup getParents(word)$
            $window \longleftarrow window \cup getChildren(word)$
        end
    end
return $window$
```

for bias in review writing and to be less affected by a single word with a different sentiment. Next, the algorithm checks if there are any neutral hits. When there are no hits at all in the ontology or the outcome from the ontology is inconclusive, we employ the back-up model to predict the sentiment. The pseudocode of this method can be found below in Algorithm 3.

Algorithm 3: Sentiment Computation Algorithm

```
Calculate the predicted polarity for $n$, where $n$ is a notion. Furthermore, we
  denote the predicted polarity of the notion as $p_n$.
Data: $n$, a notion; $ont$, the ontology
Result: $p_n$, the polarity of notion $n$
begin
    $ontologyWords \longleftarrow getOntologyWords(ont, n)$
    $ontologyScore \longleftarrow$
      $getOntologyScore(ont, n, ontologyWords)$
    if $ontologyScore[positive] \geq ontologyScore[negative] + \varepsilon$
      then
        $p_n \longleftarrow$ positive
    else if
      $ontologyScore[positive] \leq ontologyScore[negative] - \varepsilon$
      then
        $p_n \longleftarrow$ negative
    else if $ontologyScore[neutral] \neq 0$ then
        $p_n \longleftarrow$ neutral
    else
        $p_n \longleftarrow BackUpPrediction(n)$
    return $p_n$
end
```

We have eight different parameters in the first phase of the two-stage algorithm. The main seven parameters are the four different ontology scores weights w_1, w_2, w_3, w_4, the difference parameter ε, the negativity parameter ϑ, and the word window size ω. These seven parameters are optimized together using a grid search for cross-validation on training data. The ontology score weights are optimized over a range of 0 to 2.5 with a step of 0.1. The difference parameter and the negativity parameter are both found in a range of 0 to 3.5 with a step of 0.25. The word window size is found in a range of 0 to 3 with a step of 1. The last parameter is the similarity threshold t which determines whether word vectors are sufficiently similar to be considered a related word. This threshold is optimized separately for each word embedding model using values from 0.5 till 0.9 with a step of 0.05.

As back-up algorithm for the two-stage method we distinguish between two algorithms. The first algorithm is the default algorithm which always predicts the majority class, namely positive. Such an algorithm does not need training. The second algorithm is a bag-of-words method, namely a linear multi-class Support Vector Machine (SVM) [4], which is trained using the training data for all available labels: positive, negative, neutral, and conflict. We used a linear kernal as it has been shown to work well for text classification tasks where there are relatively many input features compared to the number of training instances [4]. As SVM features we use a bag-of-words, that is all the words present in the review, the current aspect category of the `notion`, and the number of sentences in the review. The SVM complexity parameter c is optimized using a grid search from 10^{-6} to 10^{3}, with the exponent being increased by 1.

The main measure we use to evaluate our models is the accuracy, which equals the F_1 score in our setup. This score is defined as follows:

$$F_1 = \frac{2TP}{2TP + FP + FN}, \tag{6}$$

where FP is the number of false positives, FN the number of false negatives, and TP the number of true positives. A true positive is defined as a correctly predicted sentiment label. When the sentiment label of a `notion` is predicted incorrectly we define this as both a false negative as well as a false positive.

Furthermore, we also compare our main models with baseline models. As baseline models we use both back-up algorithms. The first baseline model, default, always predicts the majority class positive. The second baseline model is the bag-of-words SVM model as explained previously.

We also perform two-tailed t-tests to compare the average F_1 scores obtained from ten repetitions of 10-fold cross-validation performed with the training data. In 10-fold cross-validation the data is split into 10 equally sized sets. Then, for each iteration a different set is used as test set, while the remaining 9 sets are used to train the model.

5 EVALUATION

In this section we present the results. First, we show the results of our two-stage algorithm without word embeddings. Next, we present the results of the models with word embeddings.

5.1 Two-Stage Algorithm

In Table 1, we compare our two-stage aspect-based sentiment analysis model, $Ont + BoW$, with the two baseline models, $default$ and BoW. The $default$ model always predicts the majority class, in our case positive. The BoW model is the bag-of-words SVM model with optimized complexity parameter $c = 0.1$. Model $Ont + BoW$, is the two-stage aspect based model with optimized parameters as stated in Table 2.

Furthermore, we also present the results of the Ont model, which is the same as our $Ont + BoW$ model, but predicts the majority class positive as back-up instead of employing the BoW model when the ontology is inconclusive. All of the above models do not employ word embeddings.

In the first columns, we present the average results of ten runs of 10-fold cross-validation only using the training data. Furthermore, we give the p-values of the two-sided t-tests where we compare the average F_1 scores. In the last couple of columns the results of a single run using the test data is shown. For this single run we use the training data to train the models and the test data to

Table 1: Performance of the two-stage aspect-based sentiment classification model

	avg. F_1	st.dev.	p-values of two-sided t-tests default	BoW	Ont	Ont+ BoW	in-sample F_1	out-of-sample F_1
default	0.7049	0.0711	-	-	-	-	0.7052	0.7079
BoW	0.7804	0.0564	<0.0001	-	-	-	0.8718	0.7847
Ont	0.7956	0.0459	<0.0001	<0.0001	-	-	0.7958	0.8045
Ont+BoW	0.8091	0.0429	<0.0001	<0.0001	<0.0001	-	0.8530	0.8168

evaluate the models. The in-sample F_1 uses the training data and the out-of-sample F_1 uses the test data.

The *Ont + BoW* model outperforms both baseline models significantly for the 10-fold cross validation employing the training data and for the single run with the test data. The *Ont + BoW* model has increased the test accuracy of the *BoW* model by over 3 percentage points. Furthermore, the *Ont* model also significantly outperforms both baseline models. The *Ont* model has an accuracy approximately 2 percentage points higher than the *BoW* model for the test data. The difference between the *Ont + BoW* model and the *Ont* model for the 10-fold cross-validation is approximately 1.5 percentage points and for the test data approximately 1.2 percentage points.

Looking at the parameter values in Table 2, we note that the negativity parameter ϑ equals 3.0. This indicates that negative expressions count three times as heavy as positive sentiment words. As approximately 70% of the sentiment labels in our data is positive, concepts that express negative sentiment are important. Furthermore, we note that the different weight values indicate that type-4 sentiment words are most important. Type-4 sentiment words are expressions that are context dependent. These words are defined very precisely, because they have to be formalized by an axiom. This indicates that the sentiment word is most probably correctly interpreted and thus carries the sentiment specified in the ontology. Therefore, it is logical that type-4 sentiment words are crucial.

Compared to the results of the SemEval-2016 submissions of task 5 subtask 2, we are ranked second. The accuracy of the *Ont + BoW* model of this paper is 0.25 percentage point lower than the best model and outperforms the third ranked model by 0.24 percentage point.

Table 2: Model Parameters

Parameter	w_1	w_2	w_3	w_4	ε	ϑ	ω
Value	1.0	0.75	0.75	1.5	1.5	3.0	1

5.2 Word Embeddings

In Table 3, we present the results of the two models created using the Yelp restaurant review data. The skip-gram model with similarity threshold $t = 0.75$ performs better than the Continuous Bag-Of-Words (CBOW) model with similarity threshold $t = 0.85$. This can be explained by the fact that generally the skip-gram model works better on smaller data sets and represents uncommon words well. Both models have 25894 word vector representations. We continue with the skip-gram model as our Yelp word embedding model.

Table 3: Performance of Yelp Word Embedding models

Word Embedding Model	Average F_1 of 10 ontologies
skip-gram	0.7965
CBOW	0.7904

In Fig. 9 we compare three different models with each other for different ontology sizes. The percentages on the horizontal axis are the percentages of concepts used from the original ontology. The values on the vertical axis are the F_1 scores obtained by performing a single run using the test data. The dotted line represents the *Ont+BoW* model without employing word embeddings. The striped line is the *Google* model, which consists of the *Ont + BoW* model with the GoogleNews word embeddings and a similarity threshold t of 0.80. The last model called *Yelp* is our *Ont + BoW* model in combination with the Yelp reviews and the skip-gram `word2vec` architecture. The *Yelp* model has a similarity threshold of 0.75. The similarity thresholds t are optimized separate from the other model parameters.

As you can see, the accuracy decreases when the ontology gets smaller. Up to 40% the *Google* and the *Ont + BoW* model have the same accuracy. However, for even smaller ontologies, the *Google* model does outperform the *Ont + BoW* model. Furthermore, the *Yelp* model never performs better than our *Ont + BoW* model and mostly has a lower F_1 score than the *Ont + BoW* model. The *Google* model is the best performing model in our experiment.

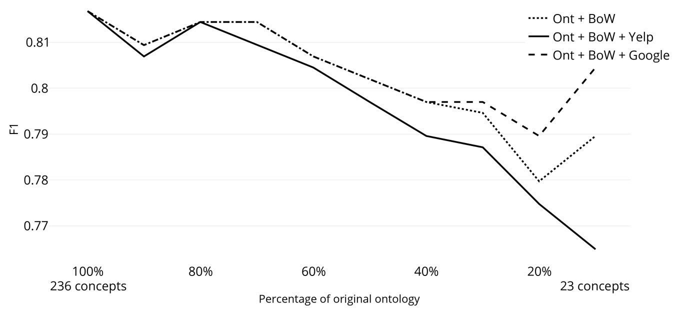


Figure 9: Effect of word embeddings for different ontology sizes

6 CONCLUSION

We employed a two-stage ontology-driven algorithm for aspect-based review-level sentiment analysis. The two-stage algorithm significantly outperforms both considered baseline algorithms. The bag-of-words SVM baseline has an accuracy that is over 3 percentage points lower than the accuracy of our final model.

Furthermore, we investigate the effect of not using any training data for our model. The model thus only depends on the ontology and does not use the bag-of-words SVM model as back-up. We find that even when we do not train a back-up model we still significantly outperform the baseline methods. The model without the back-up algorithm has a 1.2% decrease in accuracy compared to the model with the bag-of-words back-up.

Lastly, we examined the effect of using word embeddings for the two-stage algorithm. Two different word embedding models are created. The first model employs the pre-trained GoogleNews word vectors. The other model uses Yelp restaurant reviews to train word vectors. We find that, when used in conjunction with the main ontology, the word embedding models do not improve the accuracy. The Yelp word embedding model even decreased the accuracy of the predictions. A reason for this might be that we did not have enough Yelp restaurant data to create representative word vectors. However, we find that when the ontology is sufficiently small, in our case containing 30% of the main ontology concepts, the Google word embedding model does improve the accuracy.

As in around 40% of the `notions` the back-up algorithm is used to predict the sentiment, the first stage of the algorithm could be improved. For future work, one could look at either expanding the ontology or improving the algorithm that uses the ontology hits to predict the sentiment. Especially, the sentiment labels 'neutral' and 'conflict' are rarely predicted. One could try to research if a pattern among these `notions` is present in order to better predict these polarities. For example, [6] believe that the overall sentiment of a certain aspect gravitates toward the polarity of the last mention in the review of that aspect.

The domain ontology is the main component of our algorithm. We find that the accuracy decreases with the size of the ontology. This indicates that an extensive ontology is needed to get good results. However, creating an ontology is a very time consuming task. For future work, one could look into automating the creation of the ontology [3].

REFERENCES

[1] Basant Agarwal, Namita Mittal, Pooja Bansal, and Sonal Garg. 2015. Sentiment Analysis Using Common-Sense and Context Information. *Computational Intelligence and Neuroscience* 2015 (2015), 30.

[2] Ben Allison, David Guthrie, and Louise Guthrie. 2006. Another look at the data sparsity problem. In *9th International Conference on Text, Speech and Dialogue (TSD 2006)*. Springer, 327–334.

[3] Paul Buitelaar, Philipp Cimiano, and Bernardo Magnini. 2005. *Ontology learning from text: methods, evaluation and applications*. Vol. 123. IOS press.

[4] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology* 2 (2011), 27:1–27:27. Issue 3. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[5] C. Fellbaum. 1998. *WordNet: An electronic lexical database*. MIT Press.

[6] T Hercig, T Brychcın, L Svoboda, and M Konkol. 2016. UWB at SemEval-2016 Task 5: Aspect based sentiment analysis. In *10th International Workshop on Semantic Evaluation (SemEval 2016)*. Association for Computational Linguistics, 354–361.

[7] Alexander Hogenboom, Paul Van Iterson, Bas Heerschop, Flavius Frasincar, and Uzay Kaymak. 2011. Determining negation scope and strength in sentiment analysis. In *IEEE International Conference on Systems, Man, and Cybernetics 2011 (SMC 2011)*. IEEE, 2589–2594.

[8] Mengxiao Jiang, Zhihua Zhang, and Man Lan. 2016. ECNU at SemEval-2016 Task 5: Extracting effective features from relevant fragments in sentence for aspect-based sentiment analysis in reviews. In *10th International Workshop on Semantic Evaluation (SemEval 2016)*. Association for Computational Linguistics, 361–366.

[9] Bing Liu. 2012. *Sentiment Analysis and Opinion Mining, Synthesis Lectures on Human Language Technologies*. Vol. 16. Morgan & Claypool.

[10] Hugo Liu and Push Singh. 2004. ConceptNet — A Practical Commonsense Reasoning Tool-Kit. *BT Technology Journal* 22, 4 (2004), 211–226.

[11] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, 55–60.

[12] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013). https://arxiv.org/abs/1301.3781.

[13] 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 26 (NIPS 2013)*. Curran Associates, Inc., 3111–3119.

[14] Shuyo Nakatani. 2010. Language Detection Library for Java. https://github.com/shuyo/language-detection.

[15] Tim O'Reilly. 2005. What is Web 2.0. http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html.

[16] Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. 2016. SemEval-2016 Task 5: Aspect based sentiment analysis. In *10th International Workshop on Semantic Evaluation (SemEval 2016)*. Association for Computational Linguistics, 27–35.

[17] Reuters Staff. 2009. Internet most popular information source. http://www.reuters.com/article/us-media-internet-life-idUSTRE55G4XA20090617.

[18] Kim Schouten and Flavius Frasincar. 2016. Survey on aspect-level sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering* 28, 3 (2016), 813–830.

[19] Kim Schouten, Flavius Frasincar, and Franciska de Jong. 2017. Ontology-Enhanced Aspect-Based Sentiment Analysis. In *17th International Conference on Web Engineering (ICWE 2017) (Lecture Notes in Computer Science)*, Vol. 10360. Springer, 302–320.

[20] Skymind. 2017. Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0. http://deeplearning4j.org.

[21] Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. 2014. Coooolll: A deep learning system for Twitter sentiment classification. In *8th International Workshop on Semantic Evaluation (SemEval 2014)*. Association for Computational Linguistics and Dublin City University, 208–212.

[22] Junfeng Tian and Man Lan. 2016. ECNU at SemEval-2016 Task 1: Leveraging word embedding from macro and micro views to boost performance for semantic textual similarity. In *10th International Workshop on Semantic Evaluation (SemEval 2016)*. Association for Computational Linguistics, 621–627.

[23] Yelp. 2017. Yelp Dataset Challenge. https://www.yelp.com/dataset/challenge.