



Free University of Bozen-Bolzano
Faculty of Computer Science
Bachelor in Computer Science and Engineering

Capturing Semantic Similarity in Querying User Reviews

Bachelor Dissertation of
Giulia Baldini

Supervisor:
Prof. Dr. Mouna Kacimi El Hassani

Second Supervisor:
Prof. Dr. Werner Nutt

July 2018

Acknowledgment

A big thank you to mum and dad, whose dream always was to see me in this moment. You supported me and stood by my side in every decision I made, and I could never be more thankful or feel more love. To Elisa, who has been like a mum to me. You taught me how to be strong and had a home for me when I felt like I did not have one. To her beautiful daughter Camilla, my oldest friend and confidant. To my amazing Jonas, who has been the most important part of my life in the last year. To the very long list of friends I have in Bolzano, who patiently kept listening to my nonsense for the past three years. To England, to Spain, to Germany. To everyone I met in the last three years, who has shared at least a bit of this path with me.

GIULIA BALDINI
Bolzano
July 2018

Abstract

Travel websites give users the opportunity to review hotels by writing comments. However, there is no standardised way to do so, making it hard for other users to retrieve the information they want in a structured way. Moreover, comments are written by visitors coming from very different cultural and social backgrounds. As a result, two reviews describing the same hotel in the same manner will use very different words. In this work we unite semantic differences in words by using query expansion. In that context we extend the system ToKnow, developed in an internal project at Unibz, which has the aim to collect and index hotel reviews such that they can be queried easily. Here user opinions are mostly described with adjectives, which define an asset of the hotel with many different shades. Our aim is to make all the similar shades accessible to a querying user. We explore the tools that allow to investigate semantic similarity and extend them by counting in context. Furthermore, we propose new ranking methods based on static scales and learned scales. Finally we evaluate the effectiveness of our method on a user-generated ground truth using statistics and evaluation measures such as precision and recall.

I siti di viaggi offrono agli utenti la possibilità di recensire hotel scrivendo commenti. Tuttavia, non esiste nessun modo standardizzato per farlo, il che rende difficile recuperare le informazioni desiderate in modo strutturato. Inoltre, i commenti dimostrano una grande variabilità, in quanto scritti da visitatori provenienti da contesti culturali e sociali molto diversi. Di conseguenza, due recensioni che descrivono lo stesso hotel nello stesso modo utilizzeranno parole molto diverse. In questo lavoro uniamo le differenze semantiche delle parole utilizzando la metodologia dell'espansione di query. In questo contesto estendiamo il sistema ToKnow, sviluppato in un progetto interno di Unibz, che ha lo scopo di raccogliere e indicizzare le recensioni degli hotel in modo che possano essere facilmente consultate. Qui le opinioni degli utenti sono per lo più descritte con aggettivi, che definiscono una risorsa dell'hotel con molte sfumature diverse. Il nostro obiettivo è quello di rendere tutte le tonalità simili accessibili all'utente che effettua la ricerca. Esploriamo gli strumenti che permettono di indagare la somiglianza semantica e li estendiamo tenendo conto del loro contesto. Inoltre, proponiamo nuovi metodi di classificazione basati su *static scales* e *learned scales*. Infine valutiamo l'efficacia dei nostri metodi su una *ground truth* generata tramite un questionario, usando statistiche e misure di valutazione come precisione e recupero.

Reise-Websites bieten ihren Nutzern die Möglichkeit, Hotels durch Kommentare zu bewerten. Dafür gibt es jedoch keinen standardisierten Weg, sodass es für andere Nutzer schwierig ist, die gewünschten Informationen in strukturierter Form abzurufen. Darüber hinaus werden Kommentare von Personen mit stark verschiedenen kulturellen und sozialen Hintergründen verfasst. Deswegen können zwei Bewertungen, die dasselbe Hotel auf die gleiche Art und Weise beschreiben, sehr unterschiedliche Wörter verwenden. In dieser Arbeit vereinen wir semantische Unterschiede von Wörtern, indem wir Suchanfragen erweitern. In diesem Zusammenhang entwickeln wir das System ToKnow weiter, welches in einem internen Projekt der Universität Bozen entstanden ist. Dieses hat zum Ziel, Hotelbewertungen zu sammeln und zu indexieren, sodass sie leicht abgefragt werden können. Meist drücken die Nutzer ihre Meinung mit Adjektiven aus, die einen Aspekt des Hotels in vielen verschiedenen Ausprägungen beschreiben können. Unser Ziel ist es, einem anfragenden Benutzer alle ähnlich ausgeprägten Meinungen leicht zugänglich zu machen. Wir erforschen dazu die Werkzeuge, die das Untersuchen von semantischer Ähnlichkeit ermöglichen, und erweitern diese durch Zählen im Kontext. Darüber hinaus schlagen wir neue Rankingmethoden vor, die sowohl auf statischen Skalen als auch auf erlernten Skalen basieren. Abschließend evaluieren wir die Wirksamkeit unserer Methoden mithilfe von Daten aus einer Benutzerstudie und statistischen Evaluationsmaßen wie der Genauigkeit und Sensitivität.

Contents

List of Figures	vi
List of Tables	vii
1 Introduction	1
2 Related Work	3
2.1 WordNet	3
2.2 SentiWordNet	4
2.3 Word2Vec	5
2.4 SpaCy	6
3 Problem Description	7
3.1 ToKnow	7
3.2 The Problem	7
4 Approaches	9
4.1 Identifying Features and Adjectives	9
4.2 Finding Synonyms and Similarities	9
4.2.1 Lexical Approaches	10
4.2.2 Word-Vector Approaches	11
4.3 Expanding and Ranking Adjectives According to Intensities	13
4.3.1 Ranking with Polarities	13
4.3.2 Ranking with Word Vectors	14
4.4 Integrating Synonyms into Queries	15
4.4.1 Expanding with Synonyms	16
4.4.2 Expanding with Scales	17
5 Evaluation	19
5.1 Looking at the Numbers	19
5.2 Building the Ground Truth	21
5.3 Comparing the Results	22
5.4 Lesson Learned	25
6 Conclusion and Future Work	27
Bibliography	29

List of Figures

1.1	Results given by the ToKnow system while looking for <i>good breakfast and fantastic location</i>	1
2.1	Results of the search for the word <i>nice</i> on WordNet [17].	3
2.2	The structure of the WordNet lexical database.	4
2.3	Search of the similarity between <i>thief</i> and <i>robber</i> in WordNet [16].	5
2.4	Positions in which the vectors might be with respect to each other.	6
3.1	Dependencies from CoreNLP [8].	8
3.2	Example of synonym and similar adjective for the triple <i>staff is trustworthy</i>	8
3.3	Example of not synonym and similar adjective for the triple <i>breakfast is broad</i>	8
4.1	Representation of the sum of the vectors “good” and “breakfast” in two dimensions.	12
4.2	Visualisation in two dimensions of vector sum and difference to retrieve “queen”.	15
4.3	Illustration of how expansion is carried out on scales. The dotted arrows show the directions of the expansion considering if the adjective has a positive or a negative polarity.	17
5.1	Distribution of synonyms for the WordNet approaches.	20
5.2	Distribution of synonyms for the SpaCy approaches.	20
5.3	A comparison of the approaches with the most synonyms from the two categories for the feature <i>breakfast</i>	21
5.4	Result of users’ preference over the “tasty” scale.	23

List of Tables

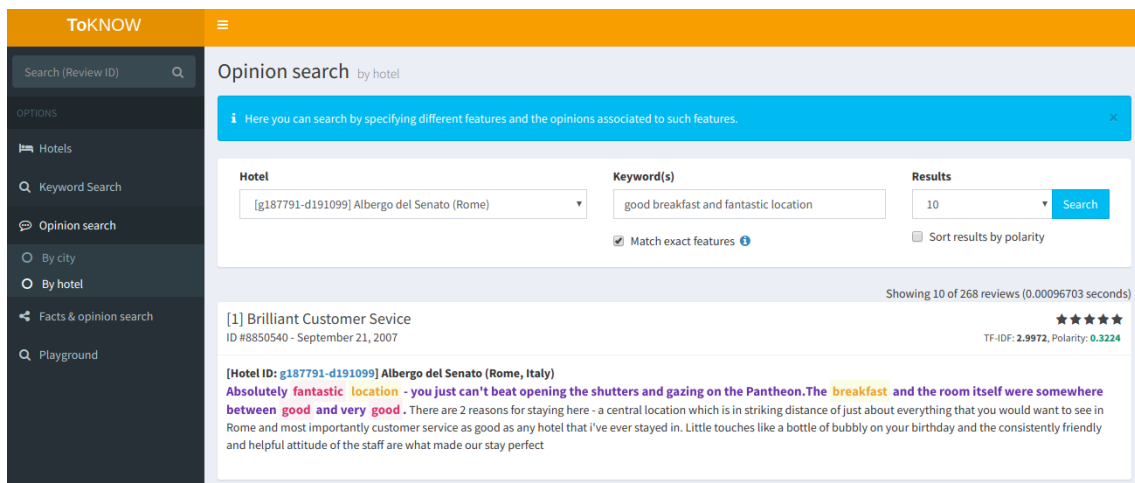
5.1	Average of synonyms for the WordNet methods.	21
5.2	Average of synonyms for the SpaCy methods.	21
5.3	Weighted average of synonyms for the WordNet methods.	21
5.4	Weighted average of synonyms for the SpaCy methods.	21
5.5	Pairs considered synonyms by the users using different thresholds for similarity. In the first line, setting only $average \geq threshold$. In the second line, $average \geq threshold$ and $variance \leq 1.5$	23
5.6	Pairs considered synonyms by the users using the WordNet methods using 3.5 as a threshold.	24
5.7	Pairs considered synonyms by the users using the SpaCy methods using 3.5 as a threshold and 0.7 as cosine similarity threshold.	24
5.8	Values of Precision, Recall and F-score for different thresholds for the methods with the highest number of synonyms (SpaCy2Feat and SpaCyDocs).	25

We are in the era of online businesses, where it is not possible to see or touch a product before buying it. Thus, consensus is based on reviews. They can express appreciation, enthusiasm or criticism towards some particular business. Particularly in the hotel domain, it has become very common to look at reviews before choosing one hotel instead of another.

Platforms based on reviews, such as TripAdvisor¹, offer the opportunity to review a business both with stars or actual text paragraphs describing user perception of the stay. Unfortunately, many users are not consistent at giving stars; in fact, there might be reviews with exclusively positive words but only three stars out of five. In this context, Poggio et al. [14] created the system ToKnow, that does not evaluate hotels by number of stars, but by calculating the positiveness of reviews and reacts according to search terms given by users.

With the help of ToKnow, a user that considers the quality of the breakfast and of the location to be the discriminant for choosing a hotel can search for “good breakfast and fantastic location”. The system will return a list of ranked hotels according to the query, then for each hotel it is possible to access its list of reviews, which are ordered by relevance and positiveness (Figure 1.1). However, there are many ways to express concepts such as “good breakfast”; one person might say “nice”, another one “lovely”. These differences in speech depend on age, country, mother tongue and many other factors. Our work aims to improve the existing ToKnow system allowing related-adjective search. In this way, the example user from before with the same search terms will also obtain all the reviews mentioning “nice breakfast”, “great breakfast”, “lovely breakfast”, “excellent location”, “perfect location”, etc.

¹www.tripadvisor.com



The screenshot displays the ToKnow web application interface. On the left is a dark sidebar with navigation options: 'Hotels', 'Keyword Search', 'Opinion search' (selected), 'By city', 'By hotel', 'Facts & opinion search', and 'Playground'. The main content area is titled 'Opinion search by hotel' and features a search form with the following fields: 'Hotel' (set to '[g187791-d191099] Albergo del Senato (Rome)'), 'Keyword(s)' (set to 'good breakfast and fantastic location'), and 'Results' (set to 10). A 'Search' button is visible. Below the form, a notification bar states 'Showing 10 of 268 reviews (0.00096703 seconds)'. The first result is for '[1] Brilliant Customer Service' with ID #8850540, dated September 21, 2007, and a 5-star rating. The review text is: '[Hotel ID: g187791-d191099] Albergo del Senato (Rome, Italy) Absolutely fantastic location - you just can't beat opening the shutters and gazing on the Pantheon. The breakfast and the room itself were somewhere between good and very good. There are 2 reasons for staying here - a central location which is in striking distance of just about everything that you would want to see in Rome and most importantly customer service as good as any hotel that I've ever stayed in. Little touches like a bottle of bubbly on your birthday and the consistently friendly and helpful attitude of the staff are what made our stay perfect'.

FIGURE 1.1: Results given by the ToKnow system while looking for *good breakfast and fantastic location*.

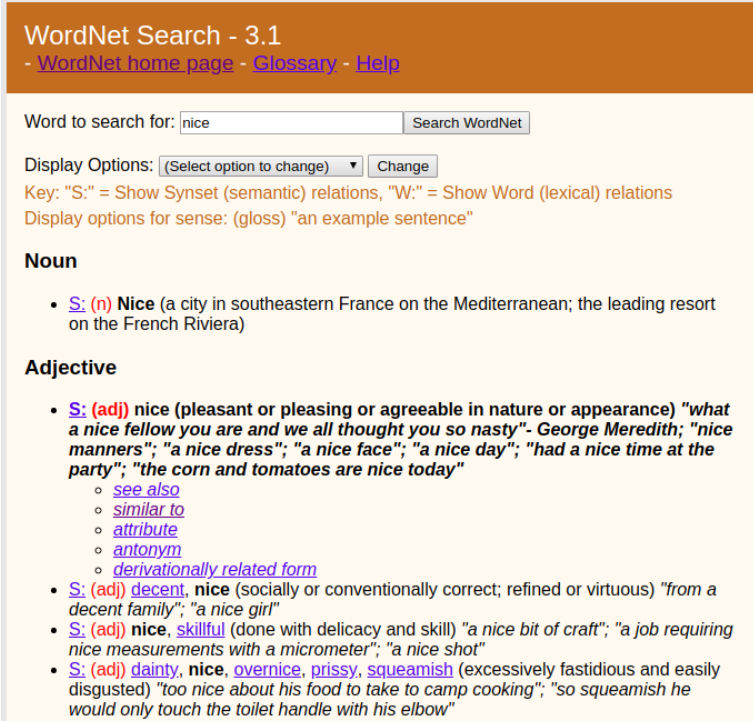
This thesis handles the problem of finding similarity between adjectives by heuristically trying different approaches. We will discuss lexical approaches ([subsection 4.2.1](#)) and vector oriented approaches ([subsection 4.2.2](#)). In these second ones, we will also consider the context of the feature to which an adjective refers, such as “breakfast”. We will also show our proposals to expand queries in [section 4.3](#). In [chapter 5](#) we will conduct extensive quantitative and qualitative tests on the retrieved data.

Our work can be located in the the *Natural Language Processing (NLP)* area. In this section we discuss all the technologies that aim to find similarity between words.

2.1 WordNet

WordNet [12, 5] is a lexical database where each English word can be defined with many *synonym sets (synsets)* in the WordNet vocabulary, which are used to capture the meaning of that word in different contexts. Looking at the words contained in a synset, it is possible to find word synonyms. For example we can have a look at the results given for the word “nice” (Figure 2.1). It can be a noun (and be a city in France), or can be an adjective and mean pleasant or socially or conventionally correct.

Figure 2.2 shows the standard structure for an adjective in this database. Every word has many synsets, which correspond to the different semantic meanings that a word can have. Each synset is a set of lemmas with related meanings. A lemma is the canonical form of a word, such as “thief”



The screenshot shows the WordNet Search interface. At the top, it says "WordNet Search - 3.1" with links to "WordNet home page", "Glossary", and "Help". Below this is a search bar with "nice" entered and a "Search WordNet" button. There are also "Display Options" and "Change" buttons. A key explains that "S:" shows synsets and "W:" shows word relations. The search results are organized into two sections: "Noun" and "Adjective".

Noun

- **S: (n) Nice** (a city in southeastern France on the Mediterranean; the leading resort on the French Riviera)

Adjective

- **S: (adj) nice** (pleasant or pleasing or agreeable in nature or appearance) "*what a nice fellow you are and we all thought you so nasty*"- George Meredith; "*nice manners*"; "*a nice dress*"; "*a nice face*"; "*a nice day*"; "*had a nice time at the party*"; "*the corn and tomatoes are nice today*"
 - [see also](#)
 - [similar to](#)
 - [attribute](#)
 - [antonym](#)
 - [derivationally related form](#)
- **S: (adj) decent, nice** (socially or conventionally correct; refined or virtuous) "*from a decent family*"; "*a nice girl*"
- **S: (adj) nice, skillful** (done with delicacy and skill) "*a nice bit of craft*"; "*a job requiring nice measurements with a micrometer*"; "*a nice shot*"
- **S: (adj) dainty, nice, overnice, prissy, squeamish** (excessively fastidious and easily disgusted) "*too nice about his food to take to camp cooking*"; "*so squeamish he would only touch the toilet handle with his elbow*"

FIGURE 2.1: Results of the search for the word *nice* on WordNet [17].

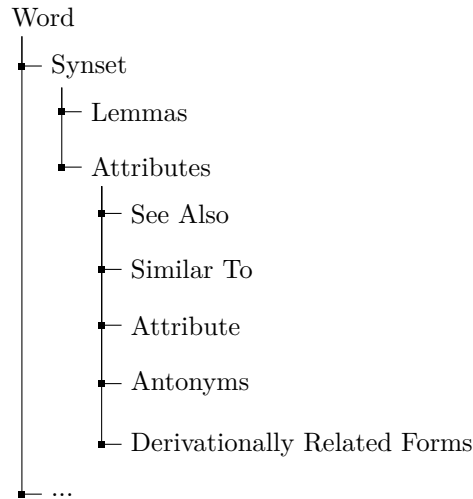


FIGURE 2.2: The structure of the WordNet lexical database.

is the lemma of “thieves”. Lemmas belonging to the same synset can be seen as actual synonyms. Looking at our example in [Figure 2.1](#), “decent” and “nice” are lemmas for the second synset. A synset is named after its most common lemma. Since more than one synset might have the same lemma, a *sense* is added to the name. The sense is an integer identifying the frequency of use of a synset, where lower numbers indicate a high frequency. The most common sense is numbered one. Synsets also have attributes; each of these might exist or not depending on the synset:

- The “see also” attribute refers to synsets which do not contain the original word as lemma but have the same meaning. For example, “good” has “best” as one of its “see also” synsets.
- The “similar to” attribute contains synsets that are related to the synset. For example, “satisfactory” is a “similar to” synset of “good”.
- The “attribute” refers to the quality that the synset describes. For example, for “big” it is “size”.
- The “antonym” attribute contains all the words that are antonyms of that specific synset.
- The “derivationally related forms” are terms that have the same root forms and are semantically related. For example, “largeness” is a “derivationally related form” of “large”.

WordNet has a similarity function for verbs and nouns. In fact, those have a hierarchical structure which allows WordNet to calculate the shortest path between two words. This path is calculated with different measures, as it can be seen in [Figure 2.3](#).

Unfortunately, adjectives are not organised in this structure, and thus the results of these similarity functions yield nothing. The only two measures that can be retrieved are the *Lesk* [7] and the *Hirst and St-Onge (HSO)* [6] measures. The Lesk measure finds how many overlaps there are between the WordNet definitions of the two words. The HSO measure identifies links between the WordNet synsets of the two words. However, the computational cost for both these measures is quite high and they rely heavily on the synsets definitions.

It is also possible to convert adjectives to nouns using “derivationally related forms”, and then use the similarity functions mentioned before. However, many synsets do not have these forms, so in practice it is not possible to retrieve many similar adjectives.

2.2 SentiWordNet

SentiWordNet is a lexical resource for opinion mining [3, 1]. It builds on WordNet by assigning three sentiment scores to each synset: positivity, negativity, objectivity. All three scores are $\in [0, 1]$ and sum up to 1, but only positivity and negativity are defined; objectivity is calculated as $1 - \text{positivity} - \text{negativity}$. The scores describe how objective, positive, and negative the terms contained in the synset are.

WS4J Demo
 WS4J (WordNet Similarity for Java) measures semantic similarity/relatedness between words.

Type in texts below, or use:

1.	Input mode	<input checked="" type="radio"/> Word <input type="radio"/> Sentence
2.	Word 1	<input type="text" value="thief"/>
3.	Word 2	<input type="text" value="robber"/>
4.	Submit	<input type="button" value="Calculate Semantic Similarity"/>

Summary

wup(thief#n#1 , robber#n#1) = 0.9630
jcn(thief#n#1 , robber#n#1) = 0.5139
lch(thief#n#1 , robber#n#1) = 2.9957
lin(thief#n#1 , robber#n#1) = 0.8941
res(thief#n#1 , robber#n#1) = 8.2104
path(thief#n#1 , robber#n#1) = 0.5000
lesk(thief#n#1 , robber#n#1) = 466
hso(thief#n#1 , robber#n#1) = 4

FIGURE 2.3: Search of the similarity between *thief* and *robber* in WordNet [16].

This tool adds a second power to WordNet; it is now possible to order words according to their polarity (that is, according to these scores). We will show a way to order reviews based on this technology.

2.3 Word2Vec

In language modeling, word vectors represent words or phrases in a multidimensional space. Word2Vec [11] gives the possibility to create word-vector models with sentences as an input, which influence heavily the positions of the vectors in the space. Since it assigns a vector to each word, they all start from the same point, which by definition is 0^d where d is the number of dimensions. This system uses the *cosine similarity* to measure the similarity between two vectors. The positioning of the words in space and therefore their similarity is based on the context in which they appear. For example, in the sentences “the breakfast at the hotel was good” and “the breakfast at the hotel was bad”, the words good and bad are used in the same way, which means that they will appear as similar. The concept of cosine similarity comes from the *dot product* or *scalar product*. The scalar product between two vectors \vec{a} and \vec{b} is defined as:

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \cdot \|\vec{b}\| \cdot \cos(\theta) \quad (2.1)$$

and thus their cosine is:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|} \quad (2.2)$$

The positions of the vectors with respect to each other can be grouped in three cases (Figure 2.4):

- Two vectors are related when they point in a similar direction, as in Figure 2.4a. In this case, $\cos(\theta)$ is very close to 1. We can consider two related vectors synonyms.
- Two vectors are unrelated if the angle between them is 90° (Figure 2.4b). In such case $\cos(\theta)$ will be very close to 0. These vectors have very different meaning or context.

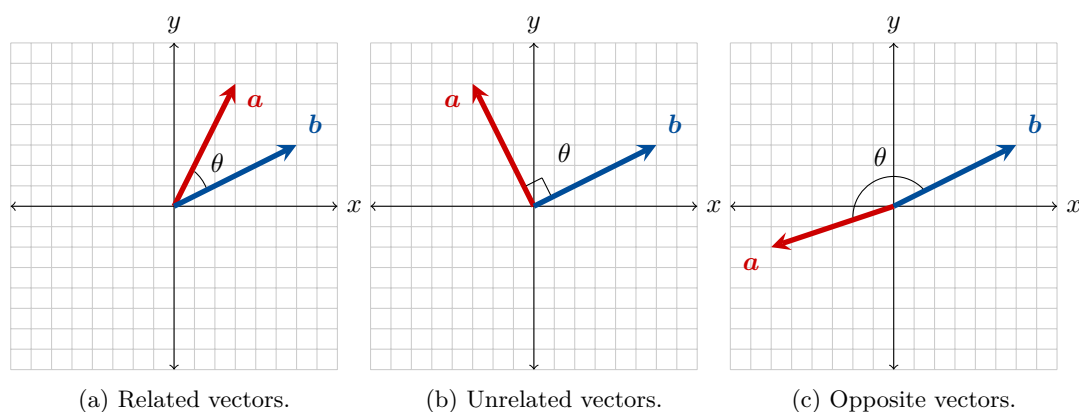


FIGURE 2.4: Positions in which the vectors might be with respect to each other.

- Two vectors are opposites if the angle between them is close to 180° and so $\cos(\theta)$ is close to -1 (Figure 2.4c). In this case the vectors might have something in common, but in an opposite fashion.

After this analysis we can conclude that the Cosine Similarity $\cos(\theta) \in [-1, 1]$.

2.4 SpaCy

SpaCy¹ is based on Word2Vec and provides many useful tools in the NLP area. By default, it uses 300-dimensional vectors trained on the Common Crawl² corpus using the GloVe algorithm [13]. Like Word2Vec, it also measures similarity by calculating the cosine between word vectors.

Our work can be seen as a similarity quest too; in fact, we will use each of these systems to find similarity or synonyms between adjectives in the context of features.

¹<https://spacy.io/>

²<https://commoncrawl.org/>

Here we give a more technical overview of the ToKnow system, to then present a deeper and more precise description of our problem, which will be done in the second subsection of this chapter.

3.1 ToKnow

As previously mentioned, ToKnow is the system we base our research on. The idea is to index hotel reviews using triples, where triples are three-word tuples describing a *feature*, which is some particular asset of a hotel. Examples of triples for [Figure 1.1](#) are “breakfast is good” and “location is fantastic”.

The input given by users is transformed into triples and used for retrieving the most interesting hotels in whose reviews the triple is somehow mentioned.

ToKnow exploits the StanfordNLP tool [\[9\]](#) to extract dependency trees of sentences and then extracts triples thanks to custom-defined rules. These rules consist in finding relations between features and the surrounding parts of a sentence.

Each triple is considered either a *factual triple* or an *opinionated triple*. The first kind is used to describe the reality of a hotel, such as “hotel in rome”, “room with view” and “hotel near train station”. The second kind, instead, expresses the writer’s opinion, such as “breakfast is good” and “room is nice”.

With the aim of finding similarity, factual triples are not very interesting. In fact, they mostly take into account nouns, which do not have many similar words because there is only a certain number of ways in which one can express the word “hotel” in natural languages. On the contrary, in opinionated triples 94% of the third terms are adjectives and in the case in which they are not, the triples are mostly of the type “condition is luggage” or “pizza is place”, which are not semantically interesting. Adjectives have more shades and tones, which makes them very suitable for finding similarity.

Opinionated triples are originated by a very common form in natural languages: *adjective phrases*. They can appear in two forms; it can either be a noun plus a copula or an adjective followed by a noun.

In [Figure 3.1a](#), we can see that “is” is a third person regular present (VBZ), “breakfast” is a noun, singular (NN) and “good” is an adjective (JJ). The *cop* arrow indicates that good is the copula of the verb, and that breakfast is its nominal subject (*nsubj*).

In [Figure 3.1b](#) good is the adjective and breakfast is the noun; good acts as an adjectival modifier (*amod*) of the noun and modifies the meaning of breakfast adding a more positive value to it.

In ToKnow, both forms are translated into the opinionated triple “breakfast is good”.

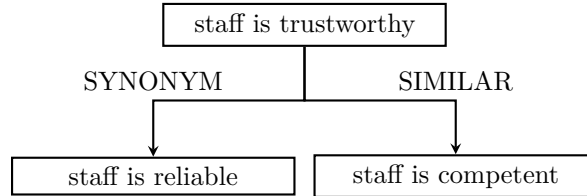
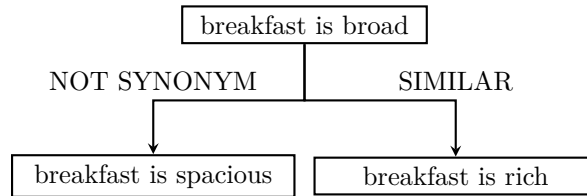
Each of these triples is now saved with a pointer to the reviews and the hotels to which it belongs; in this way it is possible to retrieve the reviews when a triple is queried.

3.2 The Problem

We find interesting the need of capturing semantic sense instead of mere keywords, and thus the main objective of this work is to find synonyms or similarity and to expand keyword searches. We



FIGURE 3.1: Dependencies from CoreNLP [8].

FIGURE 3.2: Example of synonym and similar adjective for the triple *staff is trustworthy*.FIGURE 3.3: Example of not synonym and similar adjective for the triple *breakfast is broad*.

consider two words *synonyms*, if they are related according to the English dictionary. However, in our particular area of application that is user reviews, we might also be interested in finding *similarity*, defined as the relation between some words given a specific context, which is a feature in our case. Given the data of the ToKnow system, for each triple (f, a) of the form “feature is adjective”, where f is the feature and a is the adjective, we have to find another adjective $a1$ of a triple $(f, a1)$ such that either $synonym(a, a1)$ or $similar(a, a1)$ returns true.

Figure 3.2 shows how for the adjective “trustworthy” alone, we can consider the adjective “reliable” alone a synonym. In this particular case, we can easily see how both adjectives express the same meaning when related to the feature “staff”. However, there are cases in which a synonym of an adjective might not really be related to the feature (Figure 3.3). Going back to Figure 3.2, it can be seen how “trustworthy” and “competent” are not strictly synonyms, but they can be considered as similar when the feature “staff” is taken into account. Since determining the context in which an adjective is used is very hard, we propose to use both of these approaches.

After finding synonyms, we aim to *expand queries*. This means that, starting from a triple t queried by the user, we generate some triples t_1, t_2, \dots, t_n where the third term is a synonym of the third term of t . The system will then be asked to return the hotels whose reviews mention these triples (i.e. t, t_1, t_2, \dots, t_n). This concept is called *query expansion*. To produce these triples, we present lexical approaches (subsection 4.2.1), where we try to find synonym using WordNet, and word-vector approaches (subsection 4.2.2), which will count in the context of the feature. We will also propose some ranking methods after the triple expansion.

In our work we use the TripAdvisor reviews of 2015 for the city of Rome as the underlying dataset. This only acts as a proof of concept, since it could be applied to any city. The same goes for ToKnow: even though we base our research on it, our work could be used for any other system. Each of the proposed methods is written in *Python 2.7* and the data is stored in *JSON* files, as it is done in ToKnow.

In this chapter we present the approaches we considered to find semantic similarity between adjectives. We start mentioning how we cleaned the input data. Then, we will go through the lexical approaches, that is, the ones that use the lexical database WordNet. In the following section we describe the word-vector methods in which we use SpaCy. Finally, we present some techniques that allow our approaches to be integrated into the ToKnow system.

4.1 Identifying Features and Adjectives

In the context of opinionated triples, at the beginning the data had a total of 71,487 features and 28,1505 triples, which results in an average of 4 triples per feature. This very low number is due to the fact that many features are misspelled, and thus, they are not recognised as the same word. In addition to this issue, we also wanted to speed up the synonym finding process, so we decided to perform *triple cleansing*.

We started by checking if the features were words in WordNet, so that we could leave out the spelling errors, the presence of symbols and non-existing words. After this first action, the features were down to 4,817. After that, we checked if all opinions were adjectives. After this last check, there were 595 features left, with an average of 111 triples per feature.

After this process, all the triples left are of the form “feature is adjective” or “feature is not adjective” and so are stored in the ToKnow index.

4.2 Finding Synonyms and Similarities

Here we present all the approaches we used to find synonyms and similar words. Since our aim is to find either synonyms or similarities, we use these words indistinctly. We start by describing first the ones generated with WordNet and then we move on to the ones captured with SpaCy. For each of these, we are going to give a name with which we will label them throughout the entire thesis.

In general, we use a pretty heuristic approach; we show all our ideas and then we evaluate them against a ground truth, that in our case is the result of a survey. We define *similarity triples* the triples resulted from one of our similarity functions. In fact, these functions do not return the synonyms of an adjective, but well-formed triples of the form “feature is synonym_of_adjective”.

In general, our approach takes as input all the triples contained in the dataset and the queried triple, which can be divided into its feature and its adjective. We mention once again that all the triples are of the form “feature is adjective”. The triples are stored in a dictionary data structure, where each key is a feature and its value is a list of all the triples having that feature as first term. The first step is to retrieve these triples for the feature concerned. Then, in each of the methods, which we describe later, we use a different *synonym_function* to retrieve the synonyms. The input of this function varies on the method, but in general the feature is considered only by the methods that take context into account. It returns a list of triples of the form “feature is synonym_of_adjective”, which is then intersected with the triples existing in the data for that feature. It would have been

possible to implement this intersection inside of the *synonym_function*, but we decided to keep these functions free from the data, so that they could be re-used for other purposes too. However, for one method this step is done inside of the *synonym_function* because this method is very slow at retrieving and giving it a smaller dataset helps speeding up the computations. This procedure is explained in [Algorithm 1](#). The inputs between brackets in the *synonym_function* show the parameters that are taken into account only by some methods, and will be explained later.

Algorithm 1 Given the existing triples in the data, the feature and the adjective of the queried triple, it returns a list of triples of the form “feature is synonym_of_adjective” if they are present in the data and retrieved by the *synonym_function*.

```

1: function RETURN_SYNONYMS(triples, feature, adj)
2:   triples_feature = triples [feature]
3:   similarity_triples = synonym_function(adj, (feature), (triples_feature), (threshold))
4:   return similarity_triples  $\cap$  triples_feature

```

An interesting fact about the triples we are considering is that they are not only positive, but also negative. We consider “feature is not adjective” and “feature is antonym(adjective)” as synonyms, since they express the same feeling, as in “room is quiet” and “room is not noisy”. Thus, we will not discuss a different way to treat these triples and will use the most effective method together with WordNet’s antonym function to retrieve their similarity triples.

4.2.1 Lexical Approaches

The lexical approach involves the use of a lexical database, WordNet. We choose to use this tool because its synsets represents synonyms, which are the base of our research. Going back to [Figure 2.2](#), we are going to concentrate on the “see also”, “similar to” and “antonym” attributes. Since we are looking for similarity between adjectives, we do not consider the “attribute” and the “derivationally related forms” useful.

Since WordNet acts as a dictionary, it is not possible to discover to which synset a particular triple belongs. Thus, in all our approaches with this technology we consider all the possible synsets and, as mentioned before, the context is then given by the existing data.

We now show different approaches in which we capture different levels of synsets and attributes. Let us consider:

$$lemmas_syns(S_i) = lemmas(S_i \cup see_also(S_i)) \quad (4.1)$$

$$lemmas_syns_simil(S_i) = lemmas(S_i \cup see_also(S_i) \cup similar_to(S_i)) \quad (4.2)$$

The approaches we analyse are:

1. UnSyn: union of all the lemmas of the synsets and of their “see also”.

$$\bigcup_{i=1}^{n_synsets} lemmas_syns(S_i) \quad (4.3)$$

2. UnSynSim: union of all lemmas of the synset, their “see also” and their “similar to”.

$$\bigcup_{i=1}^{n_synsets} lemmas_syns_simil(S_i) \quad (4.4)$$

3. UnSynSyn: find the lemmas of the synsets of all the lemmas of [Equation 4.1](#) and then unite them with [Equation 4.1](#) itself.

$$\bigcup_{i=1}^{n_synsets} lemmas(synsets(lemmas_syns(S_i))) \cup lemmas_syns(S_i) \quad (4.5)$$

4. UnSynSynSim: equivalent to the third one, but considering [Equation 4.2](#).

$$\bigcup_{i=1}^{n_synsets} lemmas(synsets(lemmas_syns_simil(S_i))) \cup lemmas_syns_simil(S_i) \quad (4.6)$$

We could consider larger and larger sets, but we did not think it was meaningful because the more we consider synsets of higher degree, the more we distance ourselves from the original word.

4.2.2 Word-Vector Approaches

We use SpaCy for our word-vector approaches because it is a tool for NLP based on the latest research, implemented to be used in real products and that already contains pre-trained models for word vectors. We are going to use its largest model to find similarity between adjectives.

The SpaCy vocabulary contains all the words retrieved by the text it is trained on, and since the vocabulary used in reviews is pretty straight-forward, we can assume that $vocabulary_reviews \subseteq vocabulary_spacy$.

The algorithm for the first of our approaches, SpaCyPlain, is described in the following lines. Given the adjective of the triple and a threshold for the cosine similarity, we first find the vector associated with the adjective using SpaCy. Then, we find the list of antonyms for all synsets with the help of WordNet. Afterwards we go through the SpaCy vocabulary and for each *word* we check: (1) if the *word* is an adjective (with WordNet), (2) if it is not in the list of antonyms and (3) if the cosine similarity between the vector of the current *word* and the vector of the adjective has a score above a certain threshold. If all apply, the *word* is converted to a triple of the form “feature is *word*” and added to the found synonyms. The necessity of finding the antonyms and pruning them from the final result arises because word vectors only take into account context and not polarities. Therefore, if a sentence appears with both “good” and “bad”, both will appear in the results.

Logically we would consider two vectors similar if $0.5 \leq \cos(\theta) < 1$. To find the best threshold among those values we evaluated them against the ground truth ([section 5.3](#)) and found that the most suitable one is 0.7. This value will be used for all the SpaCy approaches.

Algorithm 2 Given the adjective of the triple and a threshold for the cosine similarity, it returns a list of triples of the form “feature is synonym_of_adjective” if the cosine similarity between the adjective and its possible synonyms is greater or equal to the threshold.

```

1: function SYNONYM_FUNCTION_SPACY(adj, threshold)
2:   similarity_triples = []
3:   vector_adj = vector(adj)
4:   antonyms = antonyms(adj)
5:   for each word ∈ spacy_vocabulary do
6:     vector_word = vector(word)
7:     if adjective(word) and word ∉ antonyms then
8:       if similarity(vector_adj, vector_word) ≥ threshold then
9:         similarity_triples += to_triple(word)
10:  return similarity_triples

```

The second approach, namely SpaCy1Feat, is a slight modification of the previously mentioned one. From this approach on we try to take context into account. The intuition is that if we want our target adjective to be in a certain context, we can try to use a vector that takes us closer to that specific context.

Considering our standard triple “feature is adjective”, we add a new input to [Algorithm 2](#), that is the sum of the vector of the feature and of the vector of the adjective:

$$vector(SpaCy1Feat) = vector(feature) + vector(adjective) \quad (4.7)$$

We will always simplify the view of our vectors to two dimensions. Visually, the sum is obtained by the *parallelogram law*. We can draw the vectors such that their initial points coincide and then complete the parallelogram. The resulting diagonal between the initial point and its opposite is the sum of the two vectors ([Figure 4.1](#)).

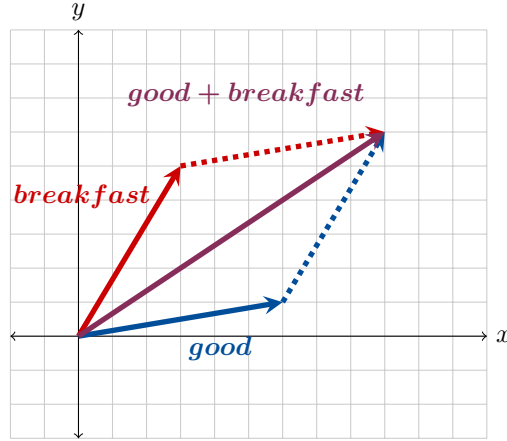


FIGURE 4.1: Representation of the sum of the vectors “good” and “breakfast” in two dimensions.

Mathematically, it is calculated by adding the correspondent components of the vectors. Since all vectors have the same size of 300, considering the vectors $A = (a_1, a_2, \dots, a_{300})$ and $B = (b_1, b_2, \dots, b_{300})$ the equation is as follows:

$$\vec{A} + \vec{B} = (a_1 + b_1, a_2 + b_2, \dots, a_{300} + b_{300}) \quad (4.8)$$

However, this would work for any vector size.

Going back to [Algorithm 2](#), this approach uses the sum of vectors as a comparison for similarity instead of using the vector of the *word*.

The SpaCy2Feat approach applies the principle just explained to both the input vector and the adjectives in the data. Therefore, the sum of vectors of feature and adjective is computed for the input triple and also for the same feature and the current *word* taken into account. In [Algorithm 3](#) it is possible to see that the algorithm works in the same way as before, but now it computes the similarity between the two sums of vectors and this might take us even closer to the context in which we want the target adjective to appear.

Algorithm 3 Given the triple’s adjective and feature and a threshold for the cosine similarity, it computes the sum of vectors for adjective and feature and compares them with the sum of the current *word* in the SpaCy vocabulary and the feature. It returns a list of triples of the form “feature is synonym_of adjective” if they are similar according to the threshold.

```

1: function SYNONYM_FUNCTION_SPACY_SUMS(adj, feature, threshold)
2:   similarity_triples = []
3:   vector_adj = vector(adj)
4:   vector_feature = vector(feature)
5:   sum_adj_feature = vector_adj + vector_feature
6:   antonyms = antonyms(adj)
7:   for each word ∈ spacy_vocabulary do
8:     vector_word = vector(word)
9:     if adjective(word) and word ∉ antonyms then
10:       sum_word_feature = vector_word + vector_feature
11:       if similarity(sum_adj_feature, sum_word_feature) ≥ threshold then
12:         similarity_triples += to_triple(word)
13:   return similarity_triples

```

In the context approaches, another idea is to take the intersection between the closest adjectives to our original adjective and the ones closest to the feature. We call this method SpaCyInter. The result set is then:

$$\text{synonyms} = \text{synonym_function_SpaCy}(\text{adjective}) \cap \text{synonym_function_SpaCy}(\text{feature}) \quad (4.9)$$

The *synonym_function_SpaCy* used in this case is the same of [Algorithm 2](#), but in *synonym_function_SpaCy(feature)* it uses the feature instead of the adjective for the comparison.

In our last method we want to consider each triple as an entity so that we can measure the similarity between entities. In SpaCy it can be achieved using documents. In the SpaCy API a document is a *sequence of words, punctuation symbols, whitespaces etc.* Documents have a similarity method that estimates the semantic similarity (cosine similarity) of the words contained in them. We achieve this with the SpaCyDocs method. In [Algorithm 4](#) we show how the similarity of documents is computed. We create a document for the original triple and for each triple that exists in the data. The “to_sentence” method transforms a triple of the type “breakfast is good” in “good breakfast”. Then, the similarity between the documents is compared with a threshold. As you can see, this method loops over the already existing triples in the data and not the SpaCy vocabulary like the previous ones. The reason is that this method is very slow because of the document creation step, and cutting down the number of iterations enhances its performance.

Algorithm 4 Given a triple, the existing triples in the data and a threshold for the cosine similarity, it returns a list of triples of the form “feature is synonym_of_adjective” if the documents produced from the original triple and the ones generated with the existing triples are similar according to the threshold.

```

1: function SYNONYM_FUNCTION_SPACY_DOCS(triple, triples_for_feature, threshold)
2:   similarity_triples = []
3:   sentence1 = to_sentence(triple)
4:   doc1 = Document(sentence1)
5:   for each current_triple ∈ triples_for_feature do
6:     sentence2 = to_sentence(current_triple)
7:     doc2 = Document(sentence2)
8:     if similarity(doc1, doc2) ≥ threshold then
9:       similarity_triples += to_triple(word)
10:  return similarity_triples

```

4.3 Expanding and Ranking Adjectives According to Intensities

In the previous section we consider all synonyms to be equal. Here, we show an expansion approach that does not consider them equal, but instead uses scales. A scale is a sequence of adjectives that could be applied to the same feature. Ideally, considering a scale “good, decent, excellent” we would like to order these adjectives such that the positiveness of the previous one is always less than the positiveness of the following one. This produces “decent, good, excellent”. If we introduce negative words and keep the same rules, we could get something like “abominable, bad, decent, good, excellent”.

To achieve this, we use two different approaches: SentiWordNet and Word2Vec. We choose SentiWordNet because it provides a ordering scheme based on sentiment. Since the adjectives we consider always take an opinion into account, we think that this approach is reasonable. The choice of Word2Vec relies on the fact that it allows analogy queries and it is possible to use it to get something comparable, such as “Rome is to Italy like Paris is to...?”.

4.3.1 Ranking with Polarities

As mentioned before, SentiWordNet assigns three scores to each synset. However, we cannot know which of the synsets is the one we intend to use for query expansion. In this context, Poggio et al. [14] managed to create an algorithm that calculates the weighted polarity for the synsets of a word. According to this scheme, the ones with negative polarity are indicated with a negative number, while the ones with positive polarity with a positive one. To achieve this, he first computes the *synset_polarity* = *positive_score* − *negative_score*, and then for each word the average of their synset polarities weighted by their sense number (i.e. the frequency of that synset) such that the more frequent synsets weigh more.

Now we can order them by their weighted average score of polarities, which yields a scale from the

most negative to the most positive adjective.

To generate some adjective scales we did the following:

- Choose an adjective *adj* for which we want the scale and a scale size n
- Generate $\frac{n}{2} - 1$ similar terms with the help of Word2Vec, avoiding antonyms.
- Find one antonym for *adj* with WordNet.
- Generate $\frac{n}{2} - 1$ similar terms for the antonym with the help of Word2Vec, avoiding the words of the previous set.
- Order the scale according to the previously described method.

In this case we use Word2Vec and its *most_similar* function to generate the *neighbourhood* of the adjective and the *neighbourhood* of its antonym. The *most_similar* function takes as input one word and returns all the closest words (that are adjectives) to the word vector of that word in descending order. Thus, the closest one is the first one appearing in the list. In practice we consider lists of 10 adjectives meaningful for our research, so we look for five positive ones and five negatives ones.

However, Word2Vec has to take a model as an input. Very famous is the *Google News word-vector model*. It includes word vectors for a vocabulary of 3 million words and phrases that are trained on roughly 100 billion words from a Google News dataset. We also train our own model on the reviews of our dataset for Rome.

Since we have both models, we generate scales for both of them, and then merge them: in this way we can have a scale with a more averaged position of the adjectives and more words. In fact, the merged scale has as position the average of the positions of that word in the two scales.

For example, we select the adjective “adequate” and a scale size of 10. We start first with the *Google News word-vector model*, and we find four adjectives in the neighbourhood of “adequate”:

sufficient, insufficient, ample, necessary

Then, we find with WordNet one antonym for “adequate”, that is “inadequate”, and we find its neighbourhood:

deficient, ineffective, unsatisfactory, inefficient

Now we have 10 adjectives: eight from the neighbourhoods, “adequate” and “inadequate”.

We now combine them by looking at their SentiWordNet scores and obtain:

insufficient → sufficient → deficient → inadequate → unsatisfactory → ineffective → inefficient
→ adequate → ample → necessary

We do again the same for the reviews model, and we obtain the following scale:

insufficient → sufficient → deficient → inadequate → ineffective → dysfunctional → adequate →
serviceable → functional → satisfactory

We can now merge the two scales calculating the average of their position and obtain the following scale:

insufficient → sufficient → deficient → inadequate → unsatisfactory → ineffective →
dysfunctional → inefficient → adequate → serviceable → ample → functional → satisfactory →
necessary

4.3.2 Ranking with Word Vectors

As far as we are concerned, the real power of word vectors is predicting a relation between two words given an input relation of two other words. A very common example is “Man is to king like woman is to..?” In this case most of the word-vector models returns “queen”. In the vector world, this can be visualised as $king - man + woman = queen$, which involves projection of vectors and then finding the closest word to the *expected point* where the word should be (Figure 4.2).

We call the following approach *scale learning*. Assume that we use the scale “abominable, bad, decent, good, excellent” and we want to find an equivalent scale for “friendly”. We select “good”

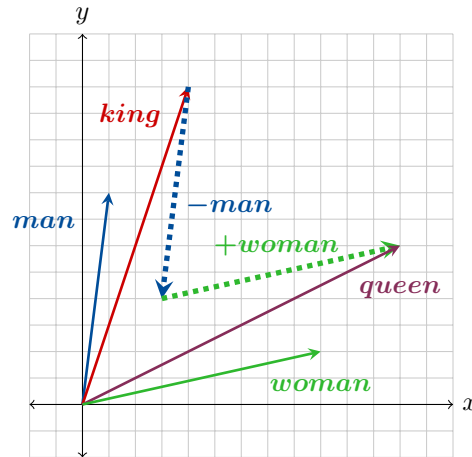


FIGURE 4.2: Visualisation in two dimensions of vector sum and difference to retrieve “queen”.

as seed word and we ask the system “good is to excellent like friendly is to...?” and it returns a word that has the same relation as excellent has to good. If we do this for each term in our reference scale, we obtain a new scale that refers to “friendly” in the same way “good” refers to the reference scale.

To create a reference scale we used the SentiWordNet approach with the adjective “good” and its antonym “bad”. From this scale we generate the others. Here follows our reference scale:

horrific → dreadful → terrible → bad → reasonable → decent → good → nice → superb → excellent

Also in this case we use both the *Google News word-vector model* and the reviews models and then merge the result. Looking at the example “adequate” from before, we use the above reference scale to generate the Google News scale:

horrifying → inadequate → inadequate → inadequate → sufficient → sufficient → adequate → sufficient → sufficient → sufficient

Then, we generate the reviews scale:

atrocious → inadequate → atrocious → objectionable → modest → sufficient → adequate → serviceable → exquisite → sufficient

And finally we merge them:

horrifying → atrocious → inadequate → objectionable → modest → adequate → sufficient → serviceable → exquisite

In the scale learning approach the merging of scales helps us dispose of duplicates. In these scales we use “good” as seed word, so each comparison is always of the type “good is to horrific like adequate is to...?” and the term returned is always inserted at the position that *horrific* has in the reference scale. This is repeated for all terms in the scale.

4.4 Integrating Synonyms into Queries

The purpose described from the start is to relieve people from formulating complicated queries, and let the system instead do so. To integrate our methods into the system we have two proposals:

- Use the best of our methods from [section 4.2](#) to extend the original query as if the synonyms were just other occurrences of the queried terms.
- Create a scale of adjectives starting from the queried one with one of the previous approaches.

Due to time constraints we cannot present any practical results for these approaches. However, both of them rely heavily on the quality of the retrieved synonyms. Therefore, the good results for the synonym retrieval ([section 5.3](#)) indicate, that the results for our query expansion approaches should also be favorable

4.4.1 Expanding with Synonyms

Assume that *synonym_function*(“location is stunning”) returns “location is beautiful”, “location is marvelous”, “location is good” and “location is not ugly”. If we retrieve the reviews to which these triples belong, we still would not have a way to order them. Ideally, we would like to have all occurrences of the synonyms of an adjective substituted by the adjective itself, because we consider all of them as equal. The ToKnow system uses the *Term Frequency-Inverse Document Frequency* (*TF-IDF*) score for each possible combination of term and document, where in our case each term is a triple and each document is a review. This score is a measurement of how important a term is in a document, which we adjust in such way that the resulting new score *TF'-IDF'* score measures the importance of a triple *and* all its similarity triples in the document. The original *TF-IDF* is the product of the weighted term frequency and the inverse document frequency:

$$TF-IDF(t, d) = TF(t, d) \cdot IDF(t) \quad (4.10)$$

Assume that $tf(t, d)$ is the number of occurrences of term t in document d , then for $tf(t, d) > 0$ we define the weighted term frequency $TF(t, d) = 1 + \log tf(t, d)$, and $TF(t, d) = 0$ otherwise. Therefore higher values of $TF(t, d)$ imply a higher frequency of t in d . The value $IDF(t)$, on the other hand, measures how important a term is in relation to the entire corpus of documents, giving lower scores to terms that occur in many documents. Let N be the total number of documents and $df(t)$ the number of documents that contain term t , then we get:

$$IDF(t) = \log \left(\frac{N}{df(t)} \right) \quad (4.11)$$

In both $TF(t, d)$ and $IDF(t)$ a logarithmic scale is used, such that only large differences in frequencies lead to significant changes in scores. Also, we want to mention that, in the special case that a term occurs in all documents (i.e. $df(t) = N$), the value $IDF(t)$ and therefore $TF-IDF(t, d)$ is 0. Now we use the scores to define our new *TF'-IDF'* score. Our *TF'-IDF'* calculates the score for a virtual set of documents where we replaced all the synonyms with the original adjective, without actually substituting, but computing it over the existing data. Taking the previous example of “location is stunning”, we virtually substitute “location is beautiful”, “location is marvelous”, “location is good” and “location is not ugly” with “location is stunning” itself. We now make two reasonable assumptions: We assume, that there are no triples that occur in all documents (i.e. there is no term with $IDF(t) = 0$, that is, no term that occurs in all documents). Also, let $S = \{s_1, \dots, s_n\}$ be the set of synonym terms of t , we do not expect two triples that are synonyms to appear in the same review (i.e. there cannot be $\sum_{s \in S} df(s) > N$). We assume that this is reasonable because overlaps in reviews are rare. Suppose we have substituted all the instances of t with its synonyms, now $TF(t)$ is:

$$TF'(t, d) = TF(s_1, d) + TF(s_2, d) + \dots + TF(s_n, d) \quad (4.12)$$

Since we have the document frequencies stored, we can now calculate $IDF'(t)$ by leaving the definition of $IDF(t)$ unchanged except for summing the document frequencies of all synonym terms. Considering the assumption above (i.e. there are no overlaps), this is exactly the value that we want to compute; and even if there are few, this would still be a good approximation:

$$IDF'(t) = \log \left(\frac{N}{df(s_1) + df(s_2) + \dots + df(s_n)} \right) \quad (4.13)$$

We can now compute $TF'-IDF'(t, d)$ as $TF'(t, d) \cdot IDF'(t)$. If an user formulates a combined query q , such as “beautiful location and large room”, it is translated into the triples “location is beautiful” and “room is large”. Then, given a document d , we can calculate the *TF'-IDF'* score for each of the triples. The sum of all scores is the total score for the query-document combination:

$$TF'-IDF'_{(q,d)} = \sum_{t \in q} TF'-IDF'(t, d) \quad (4.14)$$

However, we need some algorithm to rank the sums of *TF'-IDF'* for each document (i.e. review). We compute it, as done before in ToKnow, with the *Threshold Algorithm* [4]. All the reviews are now ordered by the frequency of the searched terms, but not by intensity. In the next sections we propose an approach that expands using the scales described before.

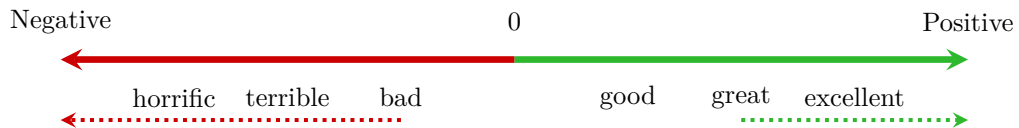


FIGURE 4.3: Illustration of how expansion is carried out on scales. The dotted arrows show the directions of the expansion considering if the adjective has a positive or a negative polarity.

4.4.2 Expanding with Scales

We imagine that a user that queries the system for “great breakfast” would like to have only positive comments about this feature. Likewise, if an user queries for “bad breakfast” (for instance, with the intent of checking whether the hotel they want to book has any bad reviews on such feature) they would like to only have reviews mentioning the bad aspects of this feature. Our idea is expressed by [Figure 4.3](#). So, we use SentiWordNet to recognise if an adjective used in a query has a positive or a negative sentiment. In the first case, we expand only with the ones above the term, making the query “breakfast is great” produce the “breakfast is great” and “breakfast is excellent”. On the contrary, we expand only with the ones below the term, making the query “breakfast is bad” produce “breakfast is bad”, “breakfast is terrible” and “breakfast is horrific”. This concept applies for both the SentiWordNet and the Word2Vec scale.

We discuss in the next lines this approach with the SentiWordNet scale. We first check if sentiment of the adjective is positive or negative. Afterwards, the approach is very similar to the one used in [subsection 4.3.1](#), but instead of selecting a neighbourhood from all the possible words in Word2Vec, we select only the terms that actually appear in our data for such feature. For an adjective with positive sentiment, we generate only the positive scale with adjectives that have a synset score higher or equal to the starting adjective. For an adjective with negative sentiment, we generate only the negative scale with adjectives that have a synset score lower or equal to the starting adjective.

Formulating a query with the Word2Vec scale have different results. Using the reference scale mentioned in [subsection 4.3.2](#), if the queried adjective is included in this scale, this is used and we would fall back in to the previously explained approach. If not, a new scale is created starting from the seed word *decent*, which we picked is reasonably in the middle. The scale is generated with the approach described in [subsection 4.3.2](#), but instead of taking any adjective in the space, it takes the adjective for such feature that exists in the data and is closest to the *expected point*. Only the more positive or more negative part of the scale for the queried adjective are considered. To actually rank the reviews containing the triples generated from one or the other scale, we first do the same as in [subsection 4.4.1](#) to compute the $TF'-IDF'$ scores for the query adjective and all the adjectives that are going to be used for the expansion. In this case, we consider all these adjectives as equivalent. Then we multiply the relevance score for each review ($TF'-IDF'$) by the rank of its used adjective in the scale. This rank is the position of the adjective in the scale and is normalised to be between 0 and 1. Now we can rank the reviews according to this value.

To evaluate our approaches we used two different types of assessment. The first one is based on statistics and shows the quantity of synonyms that can be retrieved using our methods. The second one is a qualitative evaluation, that judges our methods against a ground truth. In the first section, we describe the statistical evaluation. In the second one, we show how we built our ground truth. In the third one, we present the qualitative assessment.

5.1 Looking at the Numbers

To have an idea of how many synonyms we get in general, not considering whether they are actually right or wrong, we calculate their distribution for each of the previously mentioned approaches. As a quick summary, here follow the names:

1. UnSyn
2. UnSynSim
3. UnSynSyn
4. UnSynSynSim
5. SpaCyPlain
6. SpaCy1Feat
7. SpaCy2Feat
8. SpaCyInter
9. SpaCyDocs

To have an initial evaluation of the number of synonyms retrieved for each approach we asked ourselves the question *how often do we get how many?*.

Figure 5.1 shows how many synonyms we get for the lexical approach. Even though the maximum number of synonyms retrieved by the UnSynSynSim approach is 143, we cut the plot at 60 synonyms, because after that only few adjectives have higher numbers.

Many of the adjectives have up to 30 synonyms in at least one of the approaches, but still many more have five or less synonyms. For the UnSynSynSim method, the adjectives that capture five or less synonyms are three times more than the ones that capture more.

Similarly, in **Figure 5.2** it is possible to see the number of synonyms for the word-vector approaches. In the same way we cut the plot at 300, even though the SpaCyDocs approach retrieved up to 1160 synonyms and the SpaCy2Feat approach retrieved up to 1030 for some features. Here, all approaches but two have very often 20 synonyms. The other two, instead, have less but distributed more homogeneously.

Let us now have a look at the distribution of synonyms for one of the most common features: breakfast. In **Figure 5.3** it is possible to see the UnSynSynSim, SpaCy2Feat and SpaCyDocs

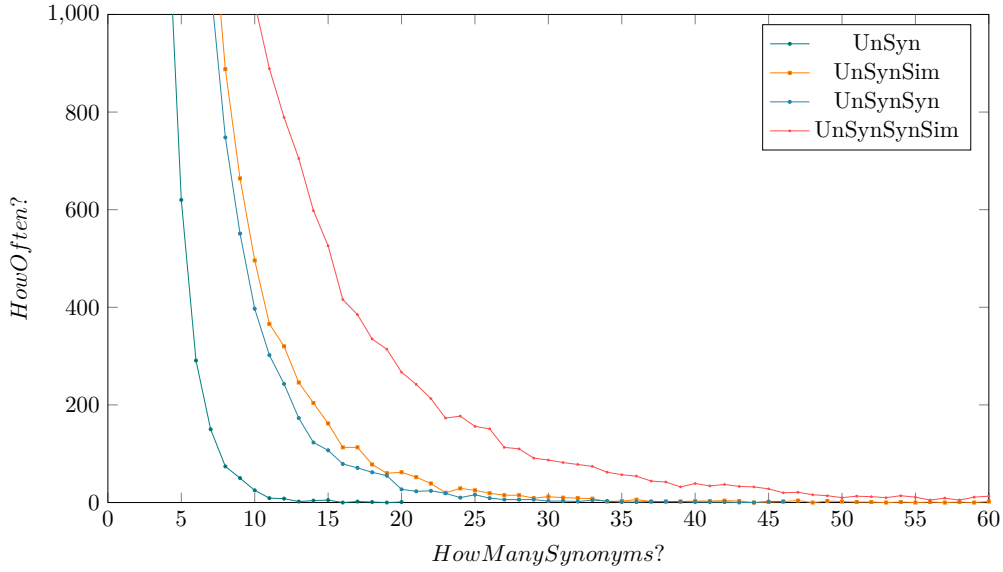


FIGURE 5.1: Distribution of synonyms for the WordNet approaches.

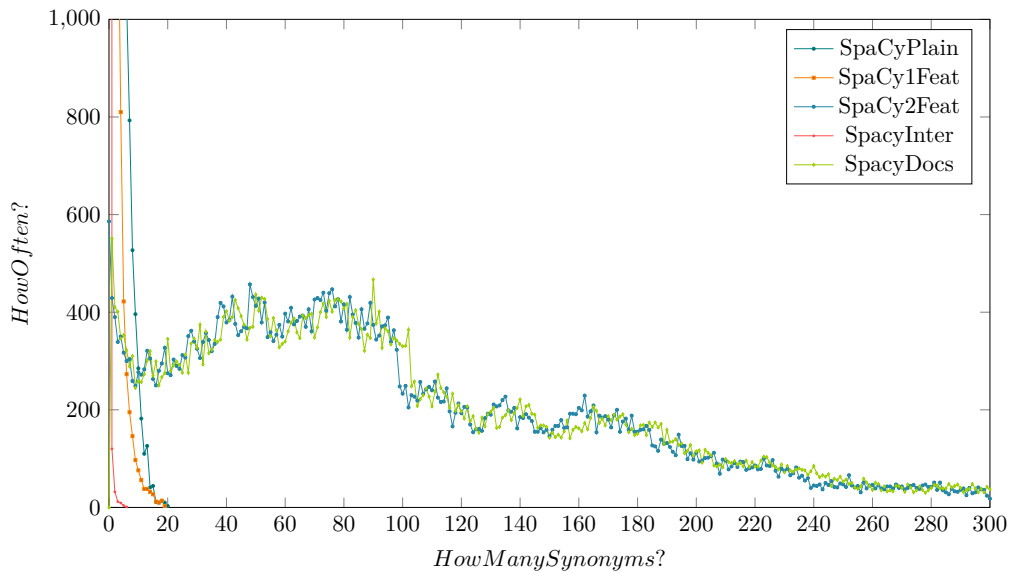


FIGURE 5.2: Distribution of synonyms for the SpaCy approaches.

approaches. We choose to show these ones because they are the ones with the most significant results over the whole set of adjectives.

The WordNet approach retrieves up to 50 synonyms, and the *SpaCy2Feat* collects many adjectives but not very often. On the contrary, *SpaCyDocs* collects many between 0 and 250. The maximum number for the two word-vector methods is around 650.

As a summary, [Table 5.1](#) and [Table 5.2](#) contain the averages for the different methods. The results confirm what we saw in the plots: WordNet retrieves many time times a low number of synonyms, while SpaCy finds many more.

This is also shown by the weighted average. [Table 5.3](#) and [Table 5.4](#) answer to the question *what is the expected number of synonyms?* Here we compute the average weighted by the frequency of the original adjective. The results are not very different from before and they confirm what has been previously said.

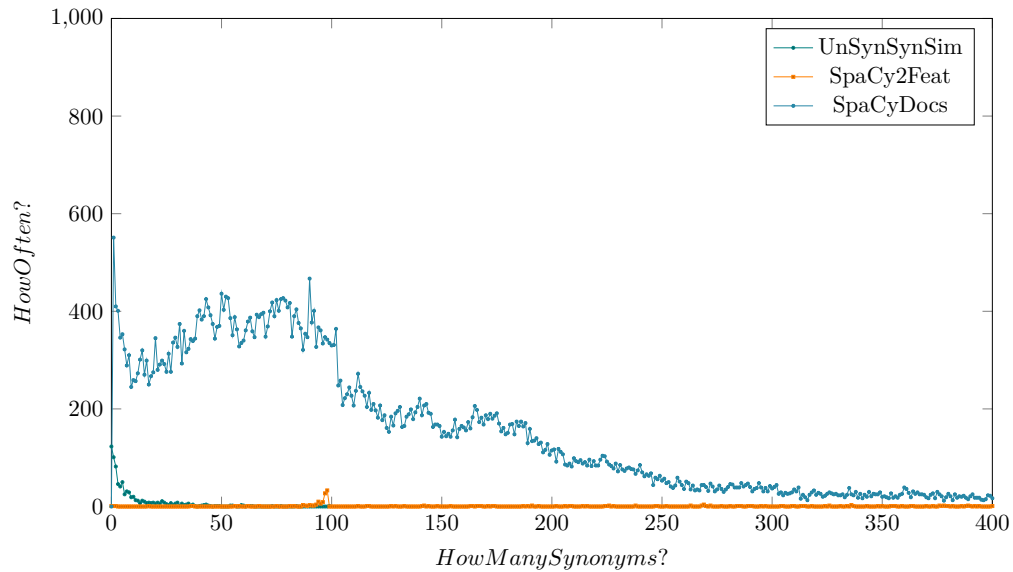


FIGURE 5.3: A comparison of the approaches with the most synonyms from the two categories for the feature *breakfast*.

5.2 Building the Ground Truth

The intuition for building the ground truth is to ask some authority if two words are synonyms or not, so that we can calculate measures such as precision and recall. However, such authority does not exist, especially when considering context. A second idea was that we ourselves could be the judges, but we did not think that our perception alone was a good enough motivation. Thus, we decided to create a survey and gather participants, so that the average of their opinions could be our judge.

Our first aim was to assess the quality of the similarity triples retrieved. Thus, we thought about how many questions could be useful, without asking too many to a single participant. We reasoned that 100 questions, if very simple, could be a good compromise. In fact, by choosing the five most common features, we could question the participants over 20 pairs of adjectives each.

Table 5.1: Average of synonyms for the WordNet methods.

Method	UnSyn	UnSynSim	UnSynSyn	UnSynSynSim
Average	0.51	1.47	1.12	2.77

Table 5.2: Average of synonyms for the SpaCy methods.

Method	SpaCyPlain	SpaCy1Feat	SpaCy2Feat	SpaCyInter	SpaCyDocs
Average	0.90	0.37	68.54	0.003	71.06

Table 5.3: Weighted average of synonyms for the WordNet methods.

Method	UnSyn	UnSynSim	UnSynSyn	UnSynSynSim
Weighted Average	0.85	2.46	1.78	4.40

Table 5.4: Weighted average of synonyms for the SpaCy methods.

Method	SpaCyPlain	SpaCy1Feat	SpaCy2Feat	SpaCyInter	SpaCyDocs
Weighted Average	1.63	0.55	81.06	0.009	84.57

The process of choosing the pairs worked as follows in the next lines. We consider *frequent adjectives* for a feature F all adjectives that belong to the first half of the list of *adjectives* $\in F$ in descending order of frequency. We consider *infrequent adjectives* the ones belonging to the second half of the list. Both sets are interesting for our research; in fact we would like to expand from a frequent term to another frequent term, so that we can gather more reviews and more opinions, but also from a frequent term to an infrequent term, so that we can retrieve reviews we would not have searched for otherwise.

- For each feature, 10 pairs of adjectives were chosen randomly.
- For each feature, 15 pairs of related frequent adjectives were chosen by us.
- For each feature, 15 pairs of related infrequent adjectives were chosen by us.

This procedure was repeated two times. After that, we had two documents for a total of 400 pairs of adjectives.

In addition to the frequent and infrequent pairs, we thought it was a good idea to add some randomness. However, not many of them were interesting and the ones, which were interesting, were translated into the frequent/infrequent scheme.

Ideally we wanted half of the pairs to be labeled as related and half of them as unrelated, so that we could test both the precision and the recall of our system. Thus, we passed the set through our methods. Afterwards we chose the most interesting 100 out of 400, dividing them first into *retrieved and interesting* and *not retrieved but interesting*. The choice was made for trying to keep them half retrieved, half not retrieved, half frequent, half infrequent.

During this process a question emerged: *is similarity symmetric?* We could not answer the question ourselves, especially because some adjectives can include some others, but the contrary might not be true. An example is that a clean room is a good room, but a good room is not always a clean room. So we decided to have two different versions of the survey: one with the chosen pairs and one with their symmetric version.

All questions were of the form *Do you think that competent staff is similar to exceptional staff?* and the participants were asked to rate the similarity in a Likert scale from one to five. We chose this type of answer because its result can be assessed with different thresholds.

A second interesting assessment was about the performance of word-vector on scaling adjectives. The participants were asked which scale out of two was the one describing one particular adjective best. One scale was created by SentiWordNet with the approach described in [subsection 4.3.1](#), the other one with Word2Vec with the approach described in [subsection 4.3.2](#). However, we had to resize the scales to make them of the same size, so that participants could compare them better. This was done by cutting the larger scale by selecting the closer adjectives to the queried one.

Since we had two versions of the survey, we decided to use different *seed words* (we picked *decent* and *good*) for the second approach, but the same adjectives.

In the end, the survey was composed of three parts:

- A section regarding information about the participant (age and sex).
- A second part with ten questions on the comparison between the SentiWordNet and the Word2Vec scales.
- 100 questions for adjective comparison.

5.3 Comparing the Results

For our survey we managed to collect 40 participants: 21 for the former questionnaire, 19 for the latter.

Regarding the comparison of the scales, the participants did not find one scale significantly better than the other. In fact, the result showed that the scales are hardly of any difference. In [Figure 5.4](#) you can see the results for this type of question for the adjective *tasty*. The scale of adjectives for SentiWordNet is:

tasteless → puerile → watery → tasty → scrumptious → yummy → delectable

The scale for Word2Vec in questionnaire 1 is:

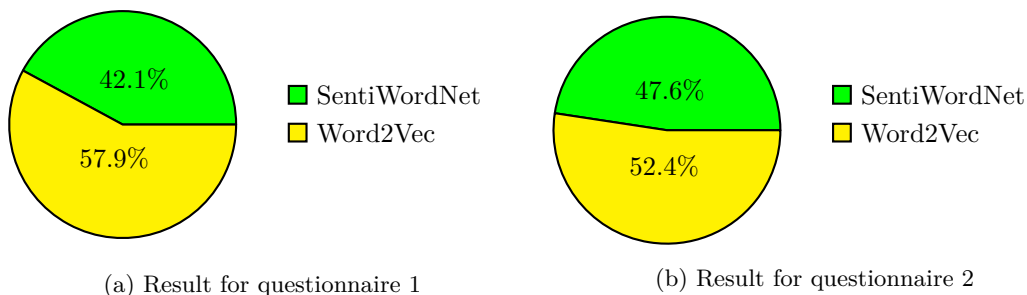


FIGURE 5.4: Result of users' preference over the "tasty" scale.

nightmarish → horrifying → vile → appetizing → tasty → yummy → delicious

The scale for Word2Vec in questionnaire 2 is:

nightmarish → gruesome → tasteless → appetizing → delicious → tasty → delectable

In this case, the participants slightly preferred the Word2Vec approach. However, the average for SentiWordNet in questionnaire one is 52% and in questionnaire two is 48%, which gives 50% when averaging the two questionnaires.

This results shows how, given a reference scale, Word2Vec can mimic the relations between the words given by SentiWordNet, so that humans cannot have a strong opinion on which one is better.

The answers given by the participants for the synonym questions are on a scale from 1 to 5. To access them, we compute the average for each question, and we check first how many out of the 200 given pairs are considered synonyms using different thresholds. To make it more precise, we set as constraint that $average \geq threshold$ and $variance \leq 1.5$. We choose to take into account the variance because some pairs might have an average of 4, but some people might still have rated them as 1s. We think that the value 1.5 can reduce this problem without setting too strict constraints. The problem of the variance shows how people evaluate the similarity between two adjectives differently, even if a context is given, and shows why this research is important. The results are summed up in Table 5.5. Out of those, we chose to use a threshold of 3.5 because this number is above average but not too strict.

As explained by Manning et al. [10], in order to access the *effectiveness* (i.e. the quality of the results) of our system we need some measures. These measures are called *precision*, *recall* and *F-score*. To compute these measures, we need first some definitions:

- True Positive: an instance retrieved by the system and considered correct by the ground truth.
- False Positive: an instance retrieved by the system and considered wrong by the ground truth.
- True Negatives: an instance not retrieved by the system and considered wrong by the ground truth.
- False Negatives: an instance not retrieved by the system and considered correct by the ground truth.

Table 5.5: Pairs considered synonyms by the users using different thresholds for similarity. In the first line, setting only $average \geq threshold$. In the second line, $average \geq threshold$ and $variance \leq 1.5$.

Threshold	2.5	3.0	3.5	4.0	4.5
N. Synonyms (no variance)	174/200	140/200	90/200	52/200	12/200
N. Synonyms (with variance)	140/200	116/200	77/200	49/200	12/200

Ideally, we would have only true positives and true negatives. The precision is defined as the fraction of relevant instances among the retrieved instances (Equation 5.1). A document is relevant if a user perceives the contained information of value.

$$\text{Precision} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalsePositives}} \quad (5.1)$$

The recall is defined as the fraction between the number of correct positive results and the number of all relevant samples (Equation 5.2).

$$\text{Recall} = \frac{\text{TruePositives}}{\text{TruePositives} + \text{FalseNegatives}} \quad (5.2)$$

The F-score is the harmonic mean of precision and recall and can show very neatly how a method performs.

$$F - \text{score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.3)$$

For all these measures, their best value is one and their worst value is zero.

We now present a table showing the results of our similarity methods, taking the variance into account and not doing so. The results with the variance constraint are in brackets. In Table 5.6 we can see the results for the WordNet methods and in Table 5.7 the results for the SpaCy methods. This shows that the methods that in general collect more synonyms are also the ones performing better on the ground truth. Others have a quite high precision, but do not retrieve many synonyms in general. This can be deducted by the fact that the recall is quite low: it means that the number of false negatives is quite high. The SpaCyInter method does not retrieve any synonyms, which means that the intersection of the two sets of synonyms of feature and adjective is not a good approach. Comparing the UnSynSyn and the UnSynSynSim approaches, we notice that the first one does not retrieve many and is also not precise, unlike the second one, although it is just an extension of the first one. We can deduct that expanding with the “similar to” attributes yields more synonyms, while doing so with the synsets creates a too wide distance between the retrieved words and the original adjective. It is easy to see that the SpaCy2Feat and the SpaCyDocs methods are the ones that perform best, which means that word vectors produce better results when taking into account the context of the feature

Table 5.6: Pairs considered synonyms by the users using the WordNet methods using 3.5 as a threshold.

Method	UnSyn	UnSynSim	UnSynSyn	UnSynSynSim
Precision	0.5 (0.5)	0.89 (0.89)	0.25 (0.25)	0.69 (0.69)
Recall	0.02 (0.03)	0.18 (0.21)	0.02 (0.03)	0.2 (0.23)
F-Score	0.04 (0.05)	0.3 (0.34)	0.04 (0.05)	0.31 (0.35)

Table 5.7: Pairs considered synonyms by the users using the SpaCy methods using 3.5 as a threshold and 0.7 as cosine similarity threshold.

Method	SpaCyPlain	SpaCy1Feat	SpaCy2Feat	SpaCyInter	SpaCyDocs
Precision	0.89 (0.83)	0.83 (0.83)	0.49 (0.42)	0 (0)	0.48 (0.41)
Recall	0.18 (0.19)	0.11 (0.13)	0.96 (0.96)	0 (0)	0.93 (0.94)
F-Score	0.3 (0.32)	0.2 (0.22)	0.64 (0.58)	0 (0)	0.63 (0.57)

From the beginning we assumed that the best threshold was 0.7 and now we are going to give a reason by looking at a table containing precision, recall and F-score for the two SpaCy approaches with the highest values (Table 5.8). We are going to show only the results considering the variance, since the distribution is the same in both cases.

In the table is possible to see how the number of synonyms retrieved by the two approaches for thresholds 0.5 and 0.6 are the same, which means that for this set of adjectives there are many with $0.5 \leq \cos(\theta) \leq 0.6$. For 0.7, we get a bit more precision and a bit less recall: the number of synonyms retrieved is slightly less than before, more are categorised as false negatives and less as false positives. For threshold 0.8 the ratio changes very much for both approaches. In fact, the number of relevant instances, which were captured before with lower thresholds, are now false negatives. For 0.9 not many are retrieved; we can assume that is quite unlikely for word vectors to have such a high cosine similarity. We choose 0.7 as threshold because it is not too strict but still is reasonable to assume that a cosine of 0.7 yields similar results. Furthermore, it is the one for which we obtain the best results.

The only still open assessment is symmetry. We handed out two different versions of the synonyms so that we could test if humans think that similarity is symmetric. To test for symmetry we checked if the averages given by the participants were over 3.5 or below 3.5 in both questionnaires. The result is that 86% of pairs are symmetric when not considering the variance, 83% when considering it, thus, we can confirm that naturally humans find similarity symmetric.

As a last remark, we computed the correlation between the personal data we have retrieved in the survey (i.e. age, sex) and the answers regarding scales and synonyms, but no significative correlation was found.

5.4 Lesson Learned

Our research is based on the quest of semantic similarity in the context of user reviews. This problem arises because people have different ways to express themselves and interpret what they read. Our aim is to allow semantic-related search for adjectives. We addressed this using tools to develop methods that find synonyms for adjectives. The first idea was to use a lexical database, WordNet, for this purpose. According to our evaluation, the approach that yields the most results is the one that expands the most with synsets and “similar to”. The second tool we use is SpaCy, a system that has pre-trained word-vector models. We used its largest model to find similarity using word vectors. Here, the approaches that performed better were the ones taking context into account.

To order these synonyms, we propose two different schemes: one treats all the synonyms of an adjective as equal, the other uses scales to order them. We introduce two different types of scales. One created with SentiWordNet, with which we statically order the adjectives according to their polarity, and another using Word2Vec, that is based on analogies and is created from a reference scale. Here we summarise our findings:

- Lexical approaches perform well at finding synonyms, even if they do not find many, but cannot take context into account.
- Word vectors have the power to take context into account, even though they might miss some synonyms and cannot discover antonyms.

Table 5.8: Values of Precision, Recall and F-score for different thresholds for the methods with the highest number of synonyms (SpaCy2Feat and SpaCyDocs).

Threshold	SpaCy2Feat			SpaCyDocs		
	Precision	Recall	F-score	Precision	Recall	F-score
0.5	0.39	0.99	0.56	0.39	0.99	0.56
0.6	0.39	0.99	0.56	0.39	0.99	0.56
0.7	0.42	0.96	0.58	0.41	0.94	0.57
0.8	0.51	0.66	0.58	0.49	0.64	0.55
0.9	0.75	0.23	0.36	0.75	0.23	0.36

- The performance of the word-vector approaches depend heavily on the trained data, the threshold values and the ground truth for similarity. These values should be experimentally defined depending on the application domain. In our experience, vectors trained on larger corpus perform better and are more precise.
- The idea of learning scales has the potential to improve ranking of subjective content. We found out that lexical approaches and Word2Vec are comparable for scale learning. But we can see that Word2Vec allows to learn any scale without the need of complex supervised learning such as the one used by SentiWordNet. Moreover, scales can be dynamically adapted according to the vocabulary used by Word2Vec while SentiWordNet provides static scales due to the static polarities assigned to words.

In this thesis we presented different approaches for finding semantic similarity. We introduced methods to retrieve synonyms with the help of a lexical database and of word vectors. Moreover, we have shown efficient ordering schemes. This was done by either looking up a static value for a sentiment in SentiWordNet or doing analogy reasoning with Word2Vec. Our results have shown that some of our approaches capture reasonably enough synonyms out of the ground truth, and that given an ordering scheme, Word2Vec can mimic its relations. Furthermore, we can conclude that the lexical approaches based on WordNet retrieve fewer adjectives, but they are very precise. On the contrary, SpaCy is less precise but has a overall better F-score.

Considering that some of the approaches examined in this thesis perform well, we assume that a combination of these might yield good results. However, in the context of similarity, many other tools can be used to improve the performance of the methods developed in this thesis. *Adagram (Adaptive Skip Gram)* [2] is a word vector based tool that allows disambiguation of terms, such that one can first disambiguate, and then use the vectors related only to that sense. In the context of disambiguation we mentioned *Lesk* [7], which could be used together with WordNet to first find the sense and to consider only the synsets and the “similar to” synsets of those. Regarding sentiment analysis, Schouten et al. [15] propose a ontology-based method to find sentiment using machine learning methods, which could be used an ordering scheme for our approach.

Bibliography

- [1] S. Baccianella, A. Esuli, and F. Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *in Proc. of LREC*, 2010. URL <http://nmis.isti.cnr.it/sebastiani/Publications/LREC06.pdf>.
- [2] S. Bartunov, D. Kondrashkin, A. Osokin, and D. Vetrov. Breaking sticks and ambiguities with adaptive skip-gram. In A. Gretton and C. C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 130–138, Cadiz, Spain, 09–11 May 2016. PMLR. URL <http://proceedings.mlr.press/v51/bartunov16.html>.
- [3] A. Esuli and F. Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC'06)*, pages 417–422, 2006. URL <http://nmis.isti.cnr.it/sebastiani/Publications/LREC10.pdf>.
- [4] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 102–113, New York, NY, USA, 2001. ACM. ISBN 1-58113-361-8. doi: 10.1145/375551.375567. URL <http://doi.acm.org/10.1145/375551.375567>.
- [5] C. Fellbaum. *WordNet: An Electronic Lexical Database*. A Bradford Book, 1998. URL <http://mitpress.mit.edu/books/wordnet>.
- [6] G. Hirst and D. St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms, 1997.
- [7] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, SIGDOC '86, pages 24–26, New York, NY, USA, 1986. ACM. ISBN 0-89791-224-1. doi: 10.1145/318723.318728. URL <http://doi.acm.org/10.1145/318723.318728>.
- [8] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. Core nlp, 2014. URL <http://www.corenlp.run/>.
- [9] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. *ACL (System Demonstrations)*, pages 55–60, 2014.
- [10] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521865719, 9780521865715.
- [11] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL <http://arxiv.org/abs/1301.3781>.
- [12] G. A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995. URL <http://nlp.cs.swarthmore.edu/~richardw/papers/miller1995-wordnet.pdf>.
- [13] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL <http://www.aclweb.org/anthology/D14-1162>.

- [14] R. A. E. Poggio. Exploiting user-generated content for hotel ranking. Master's thesis, Free University of Bozen-Bolzano, Italy, 2017.
- [15] K. Schouten and F. Frasincar. Ontology-driven sentiment analysis of product and service aspects. In *ESWC*, volume 10843 of *Lecture Notes in Computer Science*, pages 608–623. Springer, 2018.
- [16] H. Shima. Wordnet similarity for java, 2013. URL <http://ws4jdemo.appspot.com/>.
- [17] P. University. About wordnet, 2010. URL <http://wordnetweb.princeton.edu/perl/webwn>.