Utrecht University

Bachelor Thesis Artificial Intelligence

7.5 ECTS

On Sentiment Classification and Aspect Extraction in Online Product Reviews

June 2017

Author:
Mark Berger
4143930

Supervisor:
dr. Alexis Dimitriadis
2nd assessor:
dr. Ben Harvey

TABLE OF CONTENTS

INTRODUCTION--4
BACKGROUND OF THE PROBLEM --4
DETERMINING THE TASK --5
THE STRUCTURE OF THIS DOCUMENT--6
CHAPTER 1: BACKGROUND--7
1.1 ONLINE PRODUCT REVIEWS---7
1.2 RELATED WORK--8
CHAPTER 2: THE DATA -- 16
2.1 DOMAIN --- 16
2.2 INITIAL DATASET: SENTIMENT CLASSIFICATION --- 16
2.3 BIGGER DATASET: WORD EMBEDDINGS --- 17
CHAPTER 3: SENTIMENT CLASSIFICATION-- 19
3.1 DATA PREPROCESSING-- 19
3.2 ALGORITHM PERFORMANCES AND SELECTION --- 21
3.2.1 Multinomial Logistic Regression --- 22
3.2.2 Multinomial Logistic Regression results--- 23
3.2.3 Support Vector Machine --- 24
3.3 CONFIGURATION OF LINEARSVC--- 26
3.4 LINEARSVC RESULTS--- 27
CHAPTER 4: ASPECT EXTRACTION -- 30
4.1 EXPLICIT ASPECT TERMS-- 30
4.2 PART-OF-SPEECH PATTERNS AND DEPENDENCY PARSING ------------------------------ 32
4.3 WORD EMBEDDING: WORD2VEC + DEPENDENCY PARSING----------------------------- 34
4.3.1 Word2vec + dependency parsing evaluation --- 43
4.4 WORD EMBEDDING: WORD2VEC ONLY --- 45
4.5 WORD2VEC ASPECT EXTRACTOR EVALUATION --- 46
CHAPTER 5: SUMMARIZER --- 50
5.1 COMBINING THE TWO SYSTEMS-- 50
CHAPTER 6: CRITICAL REFLECTION -- 56

6.1 SENTIMENT CALCULATION -- 56
6.2 ASPECT EXTRACTION -- 56
6.3 OVERALL REFLECTION --- 57
CHAPTER 7: CONCLUSION AND FUTURE WORK -- 58
7.1 CONCLUSION -- 58
7.2 FURTHER WORK -- 59
WORKS CITED --- 60
LIST OF ABBREVIATIONS -- 62

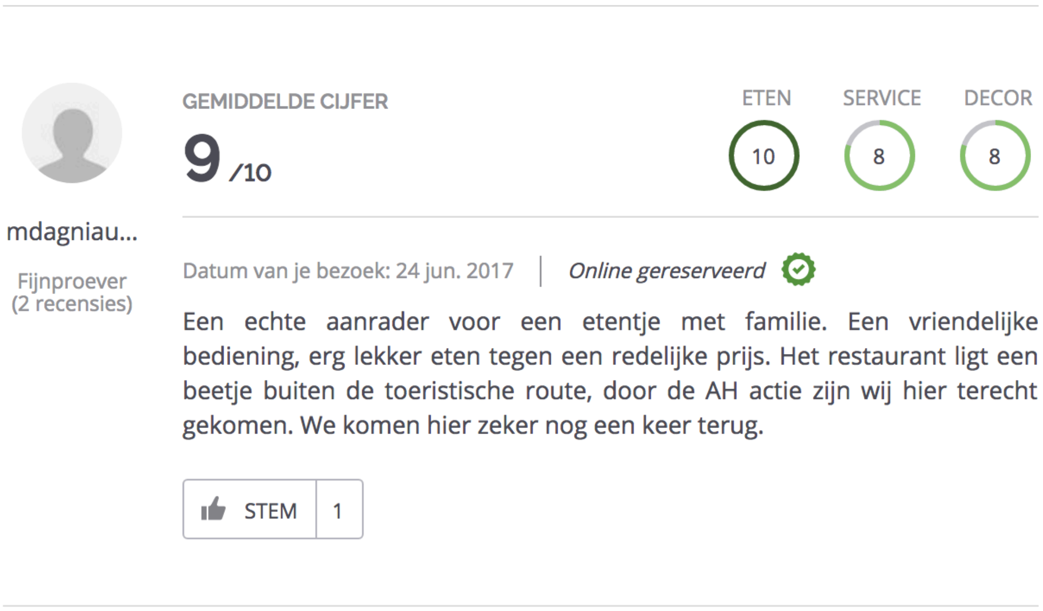
Introduction

Background of the problem

In the age of exponentially growing amount of information on the Web users are more often than not overwhelmed with the amount of content they have to get through before getting to the gist of what they are looking for. In particular, while deciding whether to buy a certain product on an online platform like Amazon, users would like to see what other buyers thought of that product. However, with thousands of reviews to read through, it becomes a time-consuming task. Most users end up reading between 2 and 5 reviews[1] and miss out on a lot of information. In order to save a reader's time and make a reviewing as a tool more effective, computerized processing that produces a review assessment based on the aspects of the product would be very welcome. This way, instead of just looking whether the review is positive overall, the user can see what aspect of the product is positive, neutral or negative. A good example of a system with reviews on multiple aspects is a Dutch restaurant finder site *www.iens.nl,* which enjoys great success. Each reviewer assesses different aspects of the restaurant like *décor*, *food* and *service* *(*see figure 1*)*.

To build such a summarization system, we first need to look at the sentiment classification and aspect extraction tasks on their own. Especially, we need to decide which algorithms are useful for dealing with those tasks on small online product reviews.

[1] https://www.brightlocal.com/learn/local-consumer-review-survey/#6


mdagniau...

Fijnproever
(2 recensies)

GEMIDDELDE CIJFER
9 /10

ETEN 10 SERVICE 8 DECOR 8

Datum van je bezoek: 24 jun. 2017 | *Online gereserveerd*

Een echte aanrader voor een etentje met familie. Een vriendelijke bediening, erg lekker eten tegen een redelijke prijs. Het restaurant ligt een beetje buiten de toeristische route, door de AH actie zijn wij hier terecht gekomen. We komen hier zeker nog een keer terug.

STEM 1

Figure 1: Example of an aspect based review on www.iens.nl

Determining the task

Research on automatic summarization of texts often focusses on either long documents like books, movie plots and in-depth articles (Barzilay, R., & Elhadad, M. 1999) or on multi-document summarization, where a summary is formed by extracting information from multiple documents on the same topic (Hu, M., & Liu, B. 2004, August) (Goldstein J. et al 2000, April). In this thesis I will look at the possibilities of a domain-specific, aspect-based sentiment summarizer of short reviews. In this thesis I will focus on exploring which algorithms perform well on small reviews and then look at a possible application of combining the sentiment classifier with the aspect extractor.
First, a sentiment classifier is chosen, fine-tuned, trained and evaluated on the available reviews. Once such a classifier is trained we can use it to classify single sentences to one of the three sentiment classes (positive, neutral and negative). After the sentiment classifier, the aspect extractor is set-up and also evaluated by looking at hand-tagged data. This aspect extractor would be able to extract predefined aspects from input strings of arbitrary size. Both systems are evaluated and discussed thoroughly, according to my main research goal.

Finally, the two systems are combined to form a pretty-printable summary of a single input review by breaking the review down in sentences, extracting the aspects from each one of them and determining the sentiment of the sentence, according to my secondary research goal.

Research goals

As discussed above, there are multiple possible applications for sentiment analysis and aspect extraction in the online reviews field. To successfully develop those applications, both the topics need to be researched.
My research goals follow up on the project goals and can be formulated as follows:

- Exploration of well-performing sentiment classification algorithms and aspect extraction model on small online product reviews.
- Exploring a possible way to combine those two systems in the field of small online product reviews

This thesis will try to address this goals by means of comprehensive, step by step explanation of the system I have created.

The structure of this document

In the first chapter of the thesis I will discuss the background of the problem and the existing works. Second chapter will lay out the datasets I used for my project. In the third chapter, the sentiment classification will be discussed. In the fourth chapter, I will talk about the aspect extraction. In the fifth chapter I will talk about combining both the sentiment classifier and the aspect extractor. Finally, the last chapter will be devoted to further research, improvements and evaluation of the research goals.

Chapter 1: Background

1.1 Online product reviews

Platforms like Amazon, bol.com and eBay are all online retailers where people are buying millions of products worldwide. Online shopping is a rapidly growing practice.[2] Because of that, the convenience of online shopping is becoming a growing concern of the online retailers.
A big part of such convenience is the ability to decide whether the product is worth buying by looking at the reviews left by other users (Lee, J., Park, D. H., & Han, I. 2011). Online shops can even benefit from those online customer reviews in a bigger way than the traditional stores by quoting the favorable reviews in their advertising campaigns, free of charge, seen as the reviews are just content that is generated by the users of the retailer's website willingly (Lee, J., Park, D. H., & Han, I. 2011).
Previous research also showed that reviews can also induce informational cascade (Huang, J. H., & Chen, Y. F. 2006). Informational cascade is a social phenomenon in which people tend to follow the decisions made by a mass of people before them while (partly) disregarding their own opinion or information (Hirshleifer, D. A. 1994). By seeing multiple positive reviews in a quick succession a customer can just decide to buy the product instead of waiting around for a long time without making a rational decision based on his own interest.
What also impacts the sales is the amount of perceived risk from the customers about the quality of the product, security of the transaction and the quality / speed of the delivery. It is still a threshold for some users to buy products online because of their distrust for the Internet, for online retailers or the inability to physically go to the store with a complaint (Greval et al. 2003). When the users see multiple short assessments of the product that the delivery process went without problems and was quick their perceived risk may go down and their willingness to purchase the product would go up.

[2] https://www.wsj.com/articles/survey-shows-rapid-growth-in-online-shopping-1465358582

Lastly, the reviews from peer customers induce more empathy than the advertisements from the retailer itself, as those reviews are considered similar to word-to-mouth effects and therefore seem to contain greater relevance, trueness and credibility than the ‘always good and friendly’ marketing advertisements. By having users share their opinions in a very accessible way fosters the relationship among and between customers and sellers (Chiou, J. S., & Cheng, C. 2003) which, again, can increase the sales and customer loyalty.

It is clear that convenient access to a lot of (positive) reviews can substantially increase sales on an e-commerce platform and help build up the loyalty of the customers.

Therefore, influencing the decisions of the buyers by presenting the reviews in the best possible way is a vital task for the marketers of such an online e-commerce platform. Multiple solutions for this task can be developed. First, a short summary of the text can be created to show the main aspects discussed in the review and the sentiment of the writer towards these aspects. Another application would be to create a search system that allows users to search relevant reviews by aspect keywords. That would again increase the convenience for the users searching for just one specific aspect of the product.

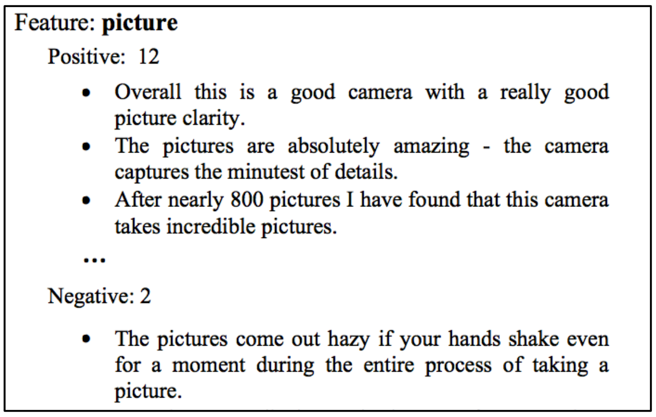
For both those applications, multiple steps need to be taken in order to achieve it. In particular, we need to be able to calculate the sentiment and to extract the aspects from the text.

1.2 Related work

Numerous works have been written about both sentiment analysis of reviews, aspect extraction and a combination of both. One of the works I relied heavily on in the beginning of my project was *Mining and Summarizing Customer Reviews* by Hu and Liu (2004*)* where they pioneered the generation of *feature-based summaries* of customer reviews. The techniques they used were divided in three parts: 1) identifying the features the users commented on, 2) determining the sentiment of the sentence the feature is in and 3) producing the summary using that information.

This is a fairly old work in this field and bases itself on some primitive concepts of sentiment analysis. However, their aspect extraction system is interesting to discuss here, as finding a well-performing aspect extraction system that builds upon different ideas is one of my main project goals.
For the identification of product features Hu and Liu first POS-tag every sentence in their dataset to extract only identified nouns and noun phrases as they believe (mostly) only those sentence parts contain product features in it. Also some pre-processing is done like the removal of stopwords and stemming.
Afterwards, explicit frequent features like *“this screen is very bright”* where *screen* is a product feature are extracted by using association mining, a technique that searches for frequently used combination of words. The next step is to identify opinion words: words that carry sentimental value relative to the product. Hu and Liu only extract adjectives as such words as they believe that most sentiment is carried by adjectives.
The way the actual opinion about the product feature is extracted is by looking at each sentence, checking whether the sentence contains a frequent feature and assign the nearest adjective to the feature as its *effective opinion.*
The approach used by Hu and Liu to determine the sentiment, or *semantic orientation*, of the opinion adjectives is to use a small seed of known positive and negative words and then compare all extracted opinion adjectives to the seed list using WordNet. If the opinion adjective ends up being a synonym of a word in a seed list, the sentiment of the opinion adjective is set to the sentiment of the word in the seed list. If it ends up to be an antonym, the opposite sentiment is assigned. Each word, when put in one or another category, is added to the seed list until the seed list is as big as the adjectives list.
Afterwards, the authors determine the sentiment of the whole sentence by a fairly simple algorithm that just adds or subtracts a one from the total sentiment score when it encounters a positive respectively negative adjective while considering the negation words like *‘not’* as well.

When both the sentiment of the sentences and the product features are extracted, a summary is then compiled by putting all the positive and negative opinion



Feature: **picture**

Positive: 12

- Overall this is a good camera with a really good picture clarity.
- The pictures are absolutely amazing - the camera captures the minutest of details.
- After nearly 800 pictures I have found that this camera takes incredible pictures.

…

Negative: 2

- The pictures come out hazy if your hands shake even for a moment during the entire process of taking a picture.

Figure 2: Example summary in Hu and Liu, 2004

sentences in its category and the counts are computed. Finally, the sentences are presented per product feature (figure 2).

The whole process is schematically displayed in figure 3.

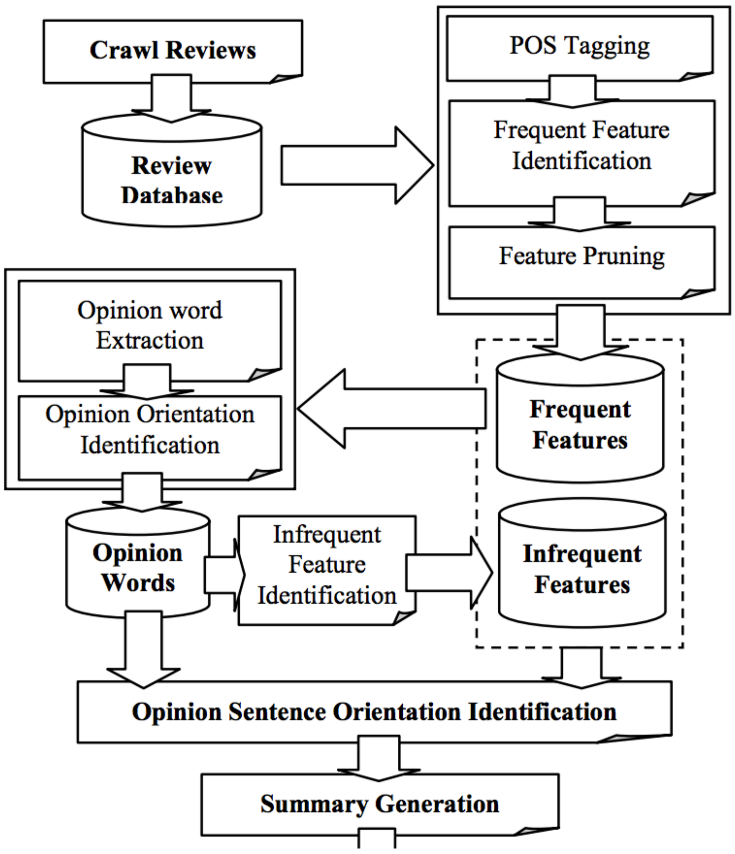


Figure 3: Schematic representation of the system used in Hu and Liu, 2004

Now to the evaluation of Hu and Liu's algorithm. For the evaluation, the authors manually read all the reviews and assigned the product features that they detected in the review to that review and compared the manually tagged sentences with the ones tagged by the algorithm. They also measured the performance of the sentiment (*semantic orientation*) prediction. The results of both evaluations are presented in tables 1 and 2.

Product name	No. of manual features	Frequent features (association mining)		Compactness pruning		P-support pruning		Infrequent feature identification	
		Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
Digital camera1	79	0.671	0.552	0.658	0.634	0.658	0.825	0.822	0.747
Digital camera2	96	0.594	0.594	0.594	0.679	0.594	0.781	0.792	0.710
Cellular phone	67	0.731	0.563	0.716	0.676	0.716	0.828	0.761	0.718
Mp3 player	57	0.652	0.573	0.652	0.683	0.652	0.754	0.818	0.692
DVD player	49	0.754	0.531	0.754	0.634	0.754	0.765	0.797	0.743
Average	**69**	**0.68**	**0.56**	**0.67**	**0.66**	**0.67**	**0.79**	**0.80**	**0.72**

Table 1: Recall and precision at each step of feature generation in Hu and Liu 2004

Product name	Opinion sentence extraction		Sentence orientation accuracy
	Recall	Precision	
Digital camera1	0.719	0.643	0.927
Digital camera2	0.634	0.554	0.946
Cellular phone	0.675	0.815	0.764
Mp3 player	0.784	0.589	0.842
DVD player	0.653	0.607	0.730
Average	**0.693**	**0.642**	**0.842**

Table 2: Results of opinion sentence extraction and sentence sentiment prediction in Hu and Liu 2004

All in all, a relevant work for my project as it covers an interesting way of *product features* extraction.

Poria et al. 2014 used sentence dependency trees to detect both explicit and implicit aspects.

Dependency parsing is a technique that is based on extracting the dependency relations between the words in a sentence where phrasal constituents and phrase-structure rules do not play a crucial role (Jurafsky, D., & Martin, J. H. 2017, 3rd ed. draft, chapter 14). In those parsers, the two parts of a relation are **head** and **dependent.** There is a fixed set of grammatical (dependency) relations between the words and each pair of words is dependent on one another by one of those relations. Figure 4 is a representation of such a dependency tree with the grammatical relations written in blue.

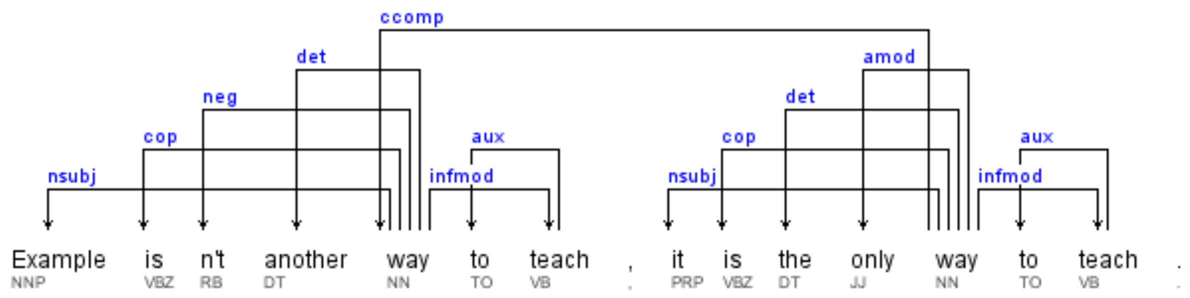


Figure 4: An example of a dependency tree (https://github.com/awaisathar/dependensee, accessed 21th of June, 2017)

In their work, Poria et al. introduce the term *implicit aspect clue* (IAC) that refers to words like *'expensive'* and *'sleek'* which are implicit indicators of the '*Price'* and '*Appearance'* aspects of the product. They introduce a rule-based approach to tackle the phenomenon of *desirable fact* which is: *"communicating fact that by commonsense is good or bad, which indirectly implies polarity" (*Poria et al. 2014*).*
Desirable fact makes it harder for the explicit aspect extractors because some sentences do not contain any explicit aspect sentiment indicators but do carry an aspect sentiment in them, like it is the case in
"I can keep putting stuff in this backpack!" which implies the positive sentiment towards the size of the bag but does not contain any explicit opinion words.
The authors used an existing implicit aspect corpus developed by Cruz-Garcia et al. 2014 to extract the initial seed of IACs in each of the nine categories they defined.

Afterwards, WordNet and SenticNet were used to enrich the aspect categories by synonyms and antonyms of the words in those categories.
As mentioned above, the novelty of work by Poria et al. 2014 consisted mainly of the dependency parsing of the sentences. By handcrafting multiple dependency-rules and dependency-parsing the sentences the authors extract word-(combination)s that are indicative of an aspect in a sentence. An example would be their *subject noun rule:*

Trigger: when the active token is found to be the syntactic subject of a token. Behavior: if an active token h is in a subject noun relationship with a word t then: if t has any adverbial or adjective modifier and the modifier exists in SenticNet, then t is extracted as an aspect. (Poria et al. 2014)

As for the results, the algorithm used by Poria et al. 2014 performs well on the data that Hu and Liu used in their work. The results are presented in table 3. The numbers are considerably higher than the numbers in Hu and Liu.

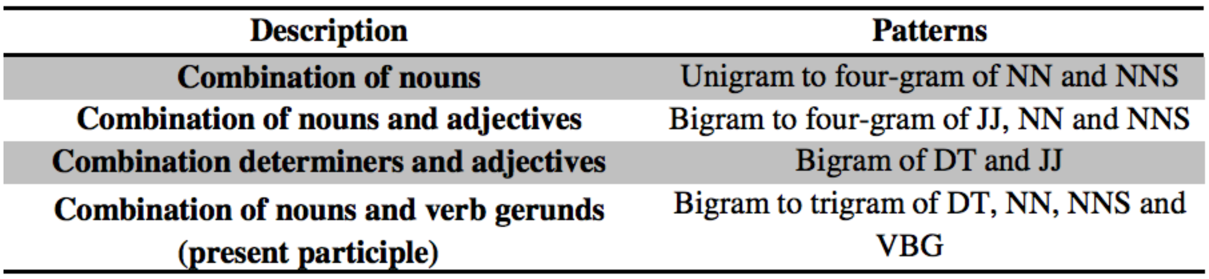
Pavlopoulos and Androutsopoulos 2014 were the ones to introduce a 'hot', new technique for use in aspect extraction for sentiment analysis: *word embeddings.* They combined word2vec implementation by Mikolov et al. 2013 with the implementation by Hu and Liu. This was a very important step for my own project because word2vec eventually became the backbone of my aspect extractor. I will dive into word2vec in general and my implementation in chapter four.
Other important notion that the authors made is that a lot of previous aspect term extraction research has been focused on the extraction of multi-word aspect terms while single-word aspect terms are very relevant as well. Multi-word aspect terms are words like '*hard disk'* while *'expensive'* is a single-word aspect term. Having both multi-word- and single-word aspect terms provides more information for the extraction system. I used that knowledge in my sentiment classification step, which also benefits from that observation.

Dataset	*Precision*	*Recall*
DVD-player	89.25%	91.25%
Canon G3	90.15%	92.25%
Jukebox	92.25%	94.15%
Nikon Coolpix	82.15%	86.15%
Nokia-6610	93.25%	93.32%

Table 3: Evaluation results from Poria et al., 2014

In yet another work, Bagheri, Saraee and de Jong 2013 are proposing their approach to the unsupervised model for detecting aspects in reviews. Their argument for an unsupervised system is that a huge amount of data available is unlabeled and that data is available in different domains and different languages. To develop real world applications, we have to be able to do aspect extraction unsupervised.

The authors work relies heavily on POS-tagging for the extraction of aspect terms. They use the assumption that aspects are nouns and noun phrases and so they have come up with the POS patterns in figure 5 for their aspect extraction.



Description	**Patterns**
Combination of nouns	Unigram to four-gram of NN and NNS
Combination of nouns and adjectives	Bigram to four-gram of JJ, NN and NNS
Combination determiners and adjectives	Bigram of DT and JJ
Combination of nouns and verb gerunds (present participle)	Bigram to trigram of DT, NN, NNS and VBG

Figure 5: POS-patterns used in Bagheri, Saraee and de Jong 2013

Afterwards, they use a new, self-made metric called *A-score* to enrich their initial seed list of aspects and use that bigger seed list as a final aspect list after some pruning. The most important part for me was the emphasis on POS-tags in this paper which could be useful in combination with the dependency parser. More on that in chapter four.

In general, even though it looks like a lot of work has been done on the subject of aspect extraction in reviews, most of the works are quite similar in their essence. Most researchers try to come up with a good unsupervised algorithm rather than a supervised one. Most of the papers also extract aspects from documents within multiple domains and evaluate those extractions while my evaluation will be per aspect in just one domain, namely *food reviews*. I also found that sentiment orientation classification in aspect extraction is not discussed extensively and is often let out as something that is ‘out of scope’ of the aspect extraction and is just a given. Some papers, like the one by *Bagheri, Saraee and de Jong* states that their aspect extraction algorithm might be a great asset for the eventual sentiment analysis but does not venture into the actual analysis. When the sentiment is discussed, like in Hu and Liu*,* a fairly simple algorithm is used and while it is a reasonably performing algorithm, I wanted to look into something more machine learning oriented.

The next chapter will be about the data I used for my research.

Chapter 2: The data

2.1 Domain

The domain I chose is food reviews on Amazon. The reason I chose that domain is mainly its availability. After searching for convenient datasets of a large amount of reviews I had to look for a couple of criteria. First, the data had to contain full review texts, preferably without too much noise in it. Noise in such reviews is mostly excessive html tags but also excessive whitespace and other structural information from the website that is not related to the content of the review.
Secondly, the data had to contain some kind of scores from the users to train and evaluate the sentiment classifier. Those scores must be categorical or convertible to categorical to feed it to a machine learning algorithm conveniently.

2.2 Initial dataset: sentiment classification

Amazon Food Reviews dataset from Kaggle[3] was a good match to my needs. It contained 568,454 food reviews with a score from 1 to 5 for each review. This score data is easily converted to some other scale, like the one I used. Initially, I converted the scores to fall just in three categories: 1 for 'positive', 0 for 'neutral' and -1 for 'negative. The final conversion in my summarizer was: score 1-3 in the 'negative' class, 4 in the 'neutral' class and 5 in the 'positive' class. For motivation for this choice see chapter 3.
The dataset also contained a short summary of each review which works like a title of the review on Amazon. I did not end up using it but it might be interesting for further research, as it could work as an additional feature.
It is important to note that the dataset was imbalanced. Figure 6 shows the initial distribution of reviews in the dataset. In fact, a statistical analysis of Amazon

[3] https://www.kaggle.com/snap/amazon-fine-food-reviews

reviews[4] and my own observations show that users leave many more positive reviews than negative ones. So this imbalance is inherent to the domain. It is however an unwanted feature of the data so I deal with it by balancing out the training set and introducing class weights to the machine learning algorithms. The way I approached the balancing is by taking 70.000 reviews from each sentiment class and putting them in one training set of 210.000 texts. The test set I took was however imbalanced, because we do want to keep the real world distribution when we are testing. More about it in the next chapter.

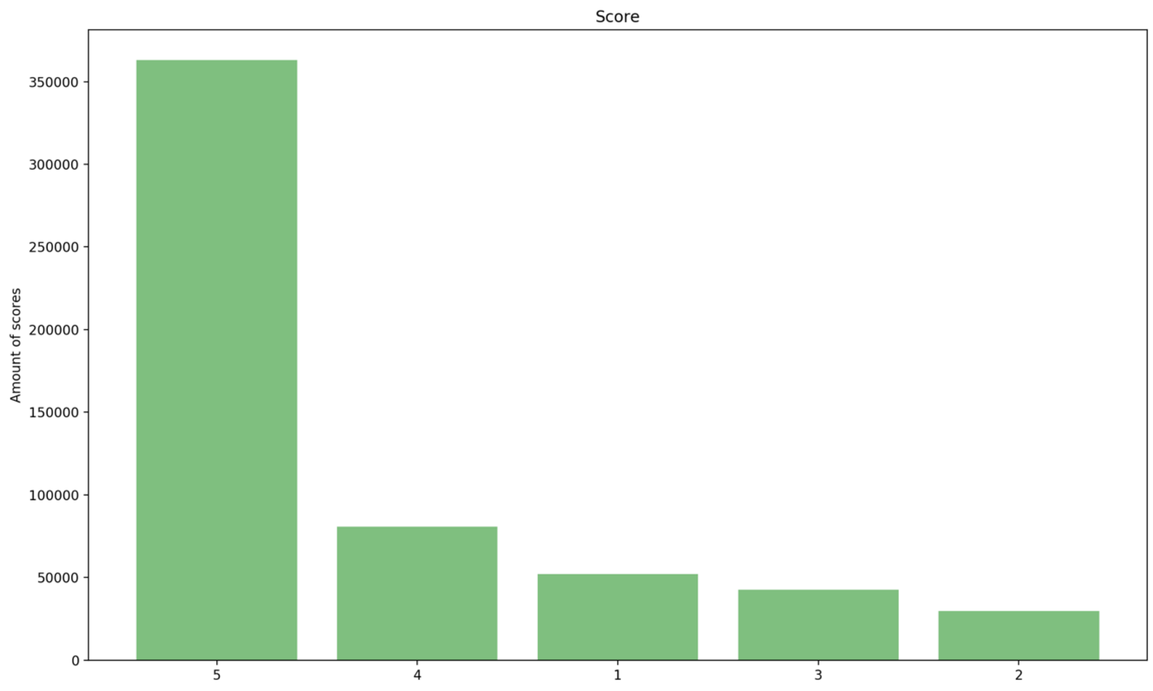


Figure 6: Initial distribution of the scores in the Amazon food reviews dataset from Kaggle

2.3 Bigger dataset: word embeddings

Further down the road, when both my sentiment classifier and my aspect extractor were up and running, I searched for more data to maybe optimize both models. I

[4] Max Woolf, ‘A Statistical Analysis of 1.2 Million Amazon Reviews’, http://minimaxir.com/2014/06/reviewing-reviews/, accessed 22nd of June, 2017

found a collection[5] [6] [7] of Amazon review datasets which consists of millions of reviews within multiple domains. I requested the food dataset, which contained another 1,297,156 Amazon food reviews and a dataset of 4,253,926 reviews in the kitchen & home category.

A word2vec model on the domain data could only benefit from extra entries. I ended up using the extra data to train my final word2vec model.

For the word2vec aspect extraction model I also have briefly experimented with the pre-trained model[8] by Google on their News dataset which was trained on about 1 billion words. It consisted of 300-dimensional vectors of around 3 million words and phrases. The model was slow to load and was a huge file on the hard drive and with a subpar performance I quickly stepped away from it. It is, however, an interesting idea to look for the best trade-off between large amounts of not domain specific data and a smaller amount of domain-specific data to see what fits better on the domain in question. The evaluations of each of the word2vec models I used are shown in chapter four.

---

[5] He, R., & McAuley, J. (2016, April). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In Proceedings of the 25th International Conference on World Wide Web (pp. 507-517). International World Wide Web Conferences Steering Committee.

[6] McAuley, J. Amazon product data, http://jmcauley.ucsd.edu/data/amazon/, accessed on 11th of June 2017

[7] McAuley, J., Targett, C., Shi, Q., & Van Den Hengel, A. (2015, August). Image-based recommendations on styles and substitutes. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 43-52). ACM.

[8] https://code.google.com/archive/p/word2vec/, accessed on 12th of June, 2017

Chapter 3: Sentiment classification

In sentiment classification, most approaches are either corpus-based or lexicon-based. A lexicon-based approach makes use of a set of sentiment-labeled words to gradually calculate the sentiment of a document. Such approach often performs quite well and is useful for incorporating lexical modifiers like amplification (*'very tasty'*) and negation (*'don't like it'*). A good example of such a classifier is *SO-CAL* described by Taboada et al. 2011 where the authors achieve accuracy scores of around 75% on multiple datasets while also being able to implement interesting syntactic patterns that can aid the classifier even more. However, for such classifier a good *polarity dataset* is needed and the syntactic rules have to be developed. Taboada et al. 2011 (pp. 268-269) mention the supervised machine learning classifiers in the following passage: "*although such classifiers perform very well in the domain that they are trained on, their performance drops precipitously (almost to chance) when the same classifier is used in a different domain (Aue and Gamon 2005)*".
But I am working in a single particular domain so the supervised, corpus-based sentiment classification would be a better solution for me. Moreover, once a model is trained, the corpus-based classifier tends to be much faster than the lexicon-based one because a lexicon-based classifier has to essentially compute everything word-by-word and do a lexicon lookup every time.

3.1 Data preprocessing

As I mentioned in the previous chapter, the data I got from Kaggle was quite imbalanced.
Before doing something about this imbalance, the data first had to be categorized in lesser number of classes. For sentiment, the use of a neutral class is advised (Koppel, M., & Schler, J. 2006). That seems like a good choice for the analysis on reviews as well, as some reviews may only contain some factual information or

contain mixed sentiment such that binary classification is not justified. So, the three classes in this classification problem were negative, neutral and positive.
The scores in the dataset ranged from 1 to 5, 1 indicating the most negative score and 5 indicating the most positive score. For the purpose of classification, I have initially binned those scores into three categories the following way (Table 4):

Score	*Label*
1 and 2	-1
3	0
4 and 5	1

Table 4: Initial class distribution after binning

The results of this categorization will be discussed in 3.2 but it was clear that the neutral class was the algorithm's bottleneck. Following the observation that people tend to value negative reviews more than positive reviews and that there are in general many more positive reviews I decided to move the 3's to the negative class and move 4's down to the neutral class. That seemed to work better and was my default configuration till the end.
Other important step in data preprocessing is tokenizing. I have stuck to the default tokenizer from sklearn tf-idf vectorizer[9] but I have experimented with some custom tokenization as discussed in great detail by Christopher Potts[10]. Tokenization can be a great asset to the classifier but, as Christopher Potts mentioned in his tutorial, with a large amount of data the need for a careful tokenizing is smaller.
Minimum amount of preprocessing was done for the sentiment classification step in comparison with the aspect extraction step.
I also chose not to remove stopwords as it decreased my performance by a few percent (see section 3.2). The text was lowercased, the punctuation was removed and the html entities were removed as they introduced noise to the data.

[9] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html, accessed on 15th of May, 2017
[10] Christopher Potts, Sentiment Symposium Tutorial, http://sentiment.christopherpotts.net/tokenizing.html, accessed on 2nd of June, 2017

3.2 Algorithm performances and selection

Pang, Lee et al. 2002 delivered a very influential work on machinal sentiment analysis. They analyzed IMDb movie reviews while trying to not make use of any knowledge-based methods. Their goal was to see *"whether it suffices to treat sentiment classification simply as a special case of topic-based categorization (with the two "topics" being positive sentiment and negative sentiment)" (Pang, Lee et al. 2002, page 3, section 5).*
The authors evaluated three algorithms: Naive Bayes classification, maximum entropy classification (multinomial logistic regression), and support vector machines. The features they used were straightforward:

$$\vec{d} := (n_1(d), n_2(d), \dots, n_m(d))$$

where $\vec{d}$ is a document vector representation, $n_i(d)$ is the number of times feature f_i occurs in document d and $\{f_1, \dots, f_m\}$ a predefined set of m features (words/word combinations) that can appear in a document.
Authors, however, mention that the presence of the feature rather than its frequency delivered much better performance. I decided to use tf-idf vectors of reviews as my features. Tf-idf vectorizing is used to determine which words are indicative of certain 'topics' by calculating the inverse proportion of the frequency of that word compared to the total frequency of that word (Ramos, J. 2003, December).
I have briefly tested the Naive Bayes classifier but I found it not that interesting for my research for its relative simplicity and inferior performance on my dataset. The two algorithms I used and will discuss in this chapter are multinomial logistic regression (Maximum Entropy) and a support vector machine.
This section will be divided in subsections for each algorithm I found promising and different results using different data and setups.

3.2.1 Multinomial Logistic Regression

The first algorithm that performed rather well was one of the oldest and most trusted models in statistics. Multinomial logistic regression algorithms perform well on data that, given a set of features that are not necessarily statistically independent of each other, is classifiable into more than two categorical dependent variables. The statistical independence of features is important for my task as we don't know whether terms appear independent of each other. On the contrary it is likely that the terms are not independent of each other. That is also the reason for inferior performance of Naïve Bayes. It is also shown by the hidden semantics in word embeddings. More on that in chapter four.

For a sentiment classification problem, it is intuitive to think that when we do not see features that are indicative of a certain class then the sentiment could be any class with the same probability. This principle is exactly what a maximum entropy classifier incorporates: it considers the data to have maximum entropy (maximum uniformness) when no constraints are put on the data. When such constraints appear in the form of external knowledge, for example our tf-idf features, the algorithm tries to model the data in such way that it becomes non-uniform enough to conform to the constraints (Nigam, K., Lafferty, J., & McCallum, A. 1999, August). In all other situations it prefers maximum entropy. Figure 7 shows the principle of entropy in my food review sentiment classification task. The green boxes are clearly positive, red boxes are clearly negative but the ones in the middle are not defined and do not have clear features. So the algorithm puts them somewhere in the middle, while trying to maintain maximum entropy.

Importance of the neutral class is explained in Koppel, M., & Schler, J. 2006: not every document expresses sentiment or it just expresses objective facts or expresses mixed or conflicted sentiment. Remember that there are many more positive reviews and the *negativity bias* (Wu, P. F. 2013) in Amazon reviews led me to think that when people give 4 stars to a product, they are not all that happy and it is more likely that only 5 stars are in fact positive reviews.

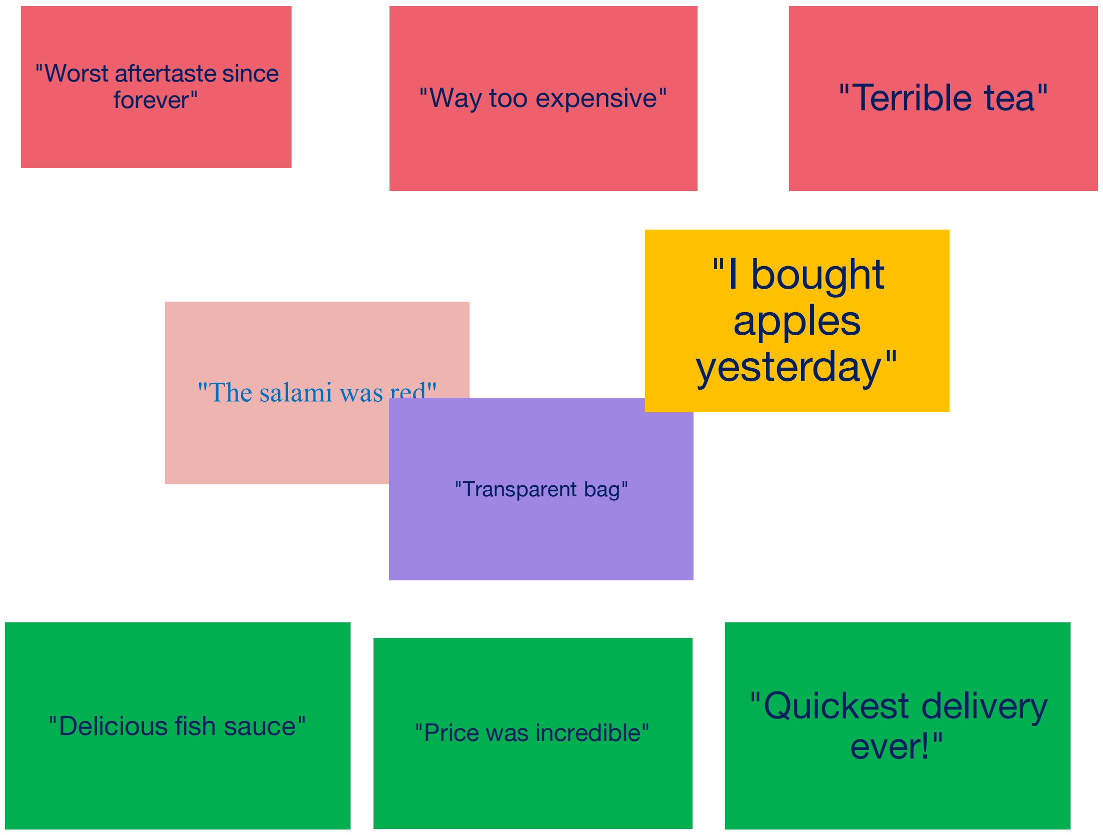


Figure 7: Abstract entropy visualization in food reviews. Box size is not indicative and the sentences are equally representative of the sentiment categories.

That led me to try to put 4's in the neutral class. I also have decided to manually balance out the train set by using the amount of data advantage I had. I selected 70.000 entries from each class and put them in the trainset. The final cross-validated result of logistic regression before I moved into different algorithm is displayed in table 5.

3.2.2 Multinomial Logistic Regression results

Below is the result of a 10-fold cross validation run on the following configuration and data:

Data:

Train: Balanced set of 210.000 reviews, 70.000 from each of the classes;

Test: 20.000 reviews, unbalanced

Three classes with 4's in the neutral set:

Scores 1-3 are 'negative', -1

Score 4 is 'neutral', 0

Score 5 is 'positive', 1

	Precision	**Recall**	**F1-score**
-1	**0.84**	**0.88**	**0.86**
0	**0.71**	**0.54**	**0.62**
1	**0.90**	**0.94**	**0.92**

Table 5: 10-fold cross validation of Max Entropy algorithm

Overall good performance except on the neutral class. It does makes sense, as neutral reviews are mostly either not sentimental or have conflicted sentiment which is harder to classify.

3.2.3 Support Vector Machine

Support Vector Machine (SVM) is a *large-margin* classifier that relies on a (small) set of *support vectors* to classify an input element. Essentially, an SVM tries to find (multiple) hyperplane(s) (a line in 2d space) that divide the input into *n* classes while maintaining the biggest margins from the hyperplane to those classes. Figure 8 from the fundamental work by Cortes and Vapnik 1995 shows this principle.

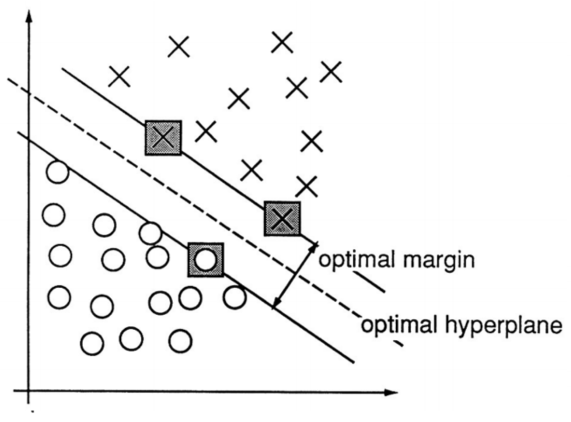


Figure 8: A separation by an SVM in 2-dimensional space. The three support vectors are showed in gray (Cortes and Vapnik 1995)

An SVM is built upon the *structural risk minimization* principle in which the classifier needs to make the best fit as to minimize misclassification on the training data but also performs well on unseen data, so it does not overfit. SVM accomplishes that with its margin maximization hyperplane idea.

There are a number of big advantages of SVMs for NLP tasks, especially data-driven ones. Due to their dimensionality independence for generalization and a small amount of hyperparameters to tune they tend to perform very well on highly dimensional data. And seeing that I work with tf-idf vectors of hundreds of thousands of texts, the number of features can become large.

Another advantage is that you can use non-linear kernels with an SVM. A kernel in machine learning is a data transformation technique that allows us to work with data that is not linearly separable by feeding the data points to a cleverly chosen function. By doing that one can perform classification on non-linear data. I experimented with different kernels but found out that the performances were sub-par to the linear kernel and took way longer. The reason for that is that for high-dimensional data, that data is likely linearly-separable (Joachims, T. 1998). So, for the rest of this paper I will talk about linear SVM when I mention SVM.

Yet another plus of an SVM is that, because of its ability to handle large feature spaces, the need for dimensionality reduction is taken away. That is favorable for text classification problem, as there are usually only very few irrelevant features in the problem. Joachims 1998 stated that *"a good classifier should combine many features (learn a "dense" concept) and that feature selection is likely to hurt performance due to a loss of information"* which makes support vector machine a perfect candidate for this need. For that reason, I chose not to perform any dimensionality reduction in SVMs.

Using tf-idf vectors yields a lot of sparsity in the features because each document contains only a very small amount of words from the vocabulary. SVMs are well suited for problems with sparse data "*because they scale linearly with the number of non-missing values"* (Li, X. et al. 2015, June).

In my implementation of SVM I have used sklearn's LinearSVC class[11]. While vanilla SVM is made to be a binary classifier, a multi-class implementation works just as well by, in my case, using *one-vs-rest* classification. In this approach a separate classifier is built for each class where all the samples from that class are labeled as being positive and all the rest is negative.

3.3 Configuration of LinearSVC

C is a parameter that deals with a situation that is showed in figure 9. The star in lower right corner is misclassified but is obviously an outlier to the data. Because SVMs initially value right classification higher than margin maximization the machine would not be *'content'* with the result. However, by specifying smaller values of C parameter, we tell our SVM to penalize misclassifications less and prioritize larger-margin separating hyperplane. Besides C the only parameter I've tuned were the class weights to counteract the class imbalances in the test data. The class weights are a workaround for the regularization of the classes in the data. Internally, class weights work the following way:

$$classes = \{-1, 0, 1\}$$

$$class\ weights = \{-1: 0.12, 0: 0.08, 1: 0.7\}$$

$$C\ parameters\ for\ each\ class = \{C_{-1}, C_0, C_1\}$$

$$C\ parameters\ for\ each\ class\ after\ class\ weights\ correction$$
$$= \{C_{-1} * 0.12,\ C_0 * 0.08,\ C_1 *\ 0.7\}$$

This way, when a class is given smaller weight, the regularization parameter C becomes smaller as well, which ensures that the algorithm prefers large-margin separation above correct classification. When the class is underrepresented that

[11] http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html, accessed on 20th of May, 2017

helps classification as it generalizes the minority class better and prevents general overfitting on the majority class.

And this is exactly one of the strengths of support vector machines: there are hardly any hyperparameters to tune, the algorithm takes care of everything on itself.

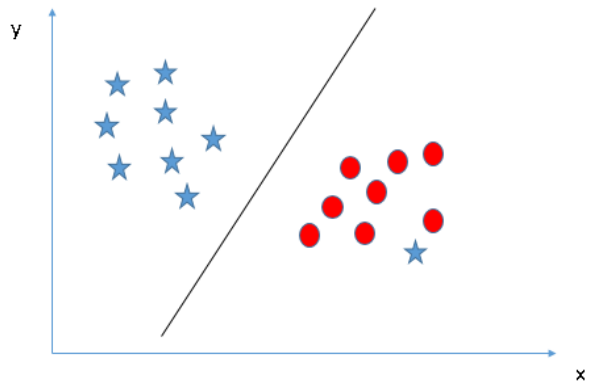


Figure 9: Example of an outlier regularization by an SVM

3.4 LinearSVC results

Below are the results from the 10-fold cross validated SVM model I used.

Data:

Train: Balanced set of 210.000 reviews, 70.000 from each of the classes;

Test: 20.000 reviews, unbalanced

Three classes with 4's in the neutral set:

Scores 1-3 are 'negative', -1

Score 4 is 'neutral', 0

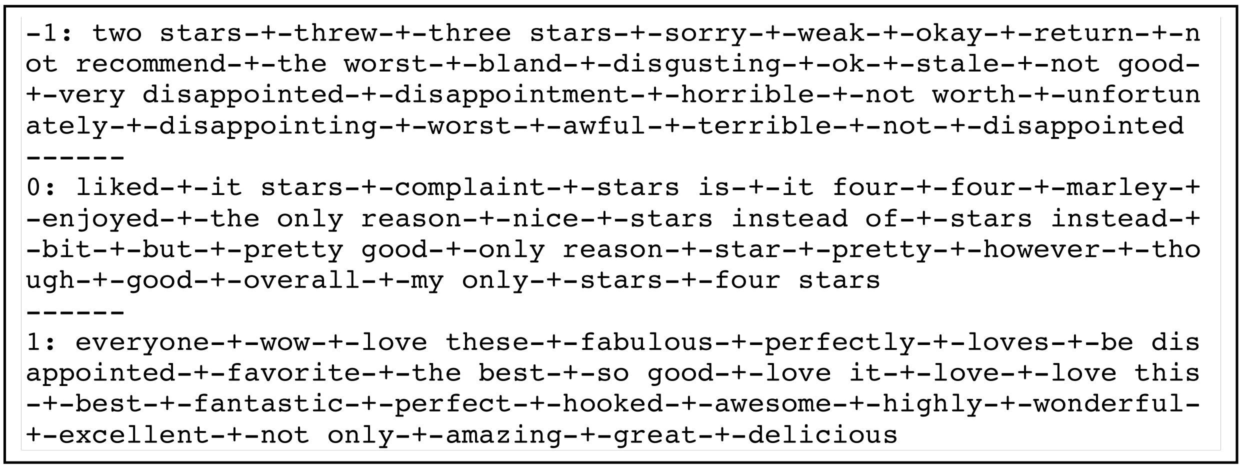
Score 5 is 'positive', 1

	Precision	**Recall**	**F1-score**
-1	**0.88**	**0.90**	**0.89**
0	**0.79**	**0.56**	**0.66**
1	**0.91**	**0.96**	**0.93**

Table 6: 10-fold cross validation of the LinearSVC

Once again inferior performance on the neutral class and good performance on the negative and positive classes.
This was my **best performing classifier** so I decided to also print top contributing features per class.

The features with the highest coefficient values per class were as follows:



```
-1: two stars-+-threw-+-three stars-+-sorry-+-weak-+-okay-+-return-+-n
ot recommend-+-the worst-+-bland-+-disgusting-+-ok-+-stale-+-not good-
+-very disappointed-+-disappointment-+-horrible-+-not worth-+-unfortun
ately-+-disappointing-+-worst-+-awful-+-terrible-+-not-+-disappointed
------
0: liked-+-it stars-+-complaint-+-stars is-+-it four-+-four-+-marley-+
-enjoyed-+-the only reason-+-nice-+-stars instead of-+-stars instead-+
-bit-+-but-+-pretty good-+-only reason-+-star-+-pretty-+-however-+-tho
ugh-+-good-+-overall-+-my only-+-stars-+-four stars
------
1: everyone-+-wow-+-love these-+-fabulous-+-perfectly-+-loves-+-be dis
appointed-+-favorite-+-the best-+-so good-+-love it-+-love-+-love this
-+-best-+-fantastic-+-perfect-+-hooked-+-awesome-+-highly-+-wonderful-
+-excellent-+-not only-+-amazing-+-great-+-delicious
```

Figure 10: The features with the highest coefficient in the best performing SVM

To compare the performances of the algorithms to some baseline I have also ran a dummy classifier with a *most_frequent* setting to have a dummy baseline to compare to my algorithm. This dummy classifier just always predicts the most frequent label in the training set which was 'positive' in my case. It is of course not necessary to run such a classifier as I could just look at the distribution of data but this classifier runs almost instantly and how the advantage of a pretty printable classification report.
The results of the dummy classifier that always predicts 'positive' are presented in table 7.

	Precision	**Recall**	**F1-score**	**Accuracy**
1	**0.63**	**1.00**	**0.77**	**0.62595**

Table 7: Results of the dummy classifier

All in all, the SVM showed the best performance on my data. So, for the sentiment classification part this particular trained model was used from now on. As the first part of the project was now taken care of, the next chapter will cover the aspect extraction part.

Chapter 4: Aspect extraction

Aspect extraction is traditionally divided into two subfields: explicit and implicit extraction.

4.1 Explicit aspect terms

Poria et al. 2014 used a small corpus of aspect words as a seed that they expanded by using WordNet. While it is a popular approach, I find that WordNet excels at finding semantically related lemmas for different senses of the words, it is not the most efficient and fast way to expand the lexicon. In WordNet, a lemma must be specified to search through the corpus. The result of one query gives just a handful of words. It is possible to iteratively look for more words by querying the first list of words. I decided to try a *quick and dirty* approach, as I wanted to get a big list of related words.
So to achieve that I decided to use thesaurus.com function to find synonyms and antonyms of a word. It also provides a possibility to get “related” words, however I found those lists not suitable for my task as they included a lot of terms that are semantically related to other meanings of my terms. Therefore, I limited myself to the synonyms and antonyms and used web scraping to iteratively collect related terms per aspect.
My initial aspect categories were:

{Price, Quality of food, Delivery}

This way I managed to get fairly extensive term lists for each of the aspects. Some caveats came along the way: initial seed for the *Price* category consisted of terms such as '*deal*', '*overpriced*', '*budget*', '*affordable*', '*affordability*', '*bang-for-the-buck*', '*money*', '*cheaper*', '*pay*', '*costs*', '*mid-priced*', '*cost-effective*'. However, those words brought a lot of noise in the final list of terms, even though those terms are legitimate by themselves. So I initialized the *Price* seed with just two words, ‘*expensive’* and ‘*cheap’* and gradually added new terms that did not introduce a lot of noise. Another observation is that some terms found by the thesaurus scraper might as well fit into another category, good example of that is the word ‘*cheap’*. That word, in its main sense, is indicative of the price of the product. However, it can also be used to describe the quality of the product. During the aggregation of terms, I ignored that fact.

After some hand-filtering of the aggregated terms, I ended up with three sets of aspect terms (Table 13). The terms themselves are available to see in the appendix.

Aspect category	Amount of terms
Price	235
Delivery	415
Food quality	214

Table 13: Amount of terms per aspect category

Note that the term sets contained not only adjectives but also nouns and verbs. The reason for that is that I want to extract all aspect-related terms from the phrases, so if a phrase like “I despise that cake” is present in the review, I would like to extract that as a quality aspect of food (even though it does not convey the exact quality it still carries a strong sentimental value towards food’s quality).

However, even before I set out to test aspect extraction using this explicit aspect terms I found that it would not work well. The reason for that is that the explicit set of aspect terms contained a lot of noise and too general words. For example, the *Price* set contained words like *amazing, lovely* and *an arm and a leg*. Those terms are either too general and can indicate other aspects just as well or are too situational. That is a pitfall of using aspect term aggregation via synonyms and antonyms scraping and also a possible argument for WordNet. WordNet links not just word forms but specific senses of those words. This way words that are found near each other in the network are semantically disambiguated, which is not the case in a simple thesaurus. So, a word like ‘cheap’ can have multiple senses relevant for both ‘Price’ and ‘Quality’ and WordNet disambiguates that.

4.2 Part-of-speech patterns and dependency parsing

Observation that most aspect terms are incorporated in nouns and noun phrases is one that multiple works I covered in the previous chapter converge on. For that reason, my second approach for the aspect extraction problem was to extract certain POS-patterns from the text. However, POS-tagging can usually only extract single words with good precision. The aspects are often composed of multiple words. What makes it harder for a standard POS-tagger, is that word combinations that make up an aspect do not strictly appear near each other in a sentence. Take a look at a review sentence like:

*"One of the bags had a hole in it and the gummi was rock hard in that **bag** showing that it was **damaged** before shipping."*

One of the aspects in this sentence is a ***damaged bag.*** Those two words are not adjacent however and will not be extracted by simply asking the POS-tagger to give you ADJ + NOUN combinations. Therefore, a more sophisticated system should be used.
Dependency parsing is a good solution for this issue. In this implementation, I used the Stanford Dependency Parser implementation in NLTK[12]. I already gave a short introduction on this technique in the first chapter but the core idea is that words are linked to each other by binary relations called dependencies. A dependency relation consists of the head of the relation and the dependent which inherits some of the characteristics of the head. The dependency relations defined in Stanford Dependency Parser can be found in the appendix.
For the aspect extraction it is useful to not only consider relevant dependency relations but also extract only relations with nouns in it, as nouns incorporate the most aspect semantics.
My dependency parser gave output in the form of tuples with three elements, where the first element represented the head of the relation, the second element was the

[12] http://www.nltk.org/_modules/nltk/parse/stanford.html

dependency relation and the third one was the dependent of the relation. It looked like this:

((u'fox', u'NN'), u'amod', (u'brown', u'JJ'))

For the aspect extraction I used the following combination of dependency rules and POS-patterns for my aspect extraction:

POS-patterns	head or dependent is: NN, NNS, NNP, PRP
Dependency relations	'amod', 'compound', 'advmod', 'nmod', 'neg', 'num', 'nsubj', 'nmod:npmod'

Table 14: POS-patterns and dependency relations used for aspect extractions

The results were mixed. Even though the parser extracted the patterns well and delivered meaningful results, some aspects were not extracted simply because they did not match the patterns. Such was the case for the following sentences:

"Candy was delivered quickly. My only complaint would be that it seems old as it is tougher to chew. Have ordered and had this product many times and find it delicious but not usually so tough to chew."

The parser did not see anything in the first sentence because it does not have any pattern and dependency relation that matches that sentence. The relations extracted by the parser were:

```
((u'be', u'VB'), u'dep', (u'delivered', u'VBN'))
((u'delivered', u'VBN'), u'nsubjpass', (u'candy', u'NN'))
((u'delivered', u'VBN'), u'auxpass', (u'was', u'VBD'))
((u'delivered', u'VBN'), u'advmod', (u'quickly', u'RB'))
```

When a matching relation was found, POS-patterns didn't match and when a noun was found, the relation was "wrong". And that while we clearly want to extract aspect *Delivery* from the sentence.

But besides occasional problems like the one above the dependency parser did a good job extracting pairs of words that closely resembled possible aspects from the reviews. But with just pairs of words I still didn't know to which aspect those words belong. This seemed like a problem that needed a semantic approach.

4.3 Word embedding: word2vec + dependency parsing

One of the 'hottest' topics in the NLP world anno 2017 are word embeddings. Initial research on *distributed representations of words* was done by Bengio et al. 2003. There it was proposed as a way to combat the *curse of dimensionality*. The idea behind the approach is that it reduces the dimensionality of word representations by only considering words that are *semantically related.* This semantic relatedness is based on words' distributional properties in big amount of textual data. The core principle used in this idea is the one of *distributional hypothesis.* Back in 1954, Harris made a careful discovery that words that appear in the same context have similar meanings.
Word embedding is a *predictive method* which leverages the distributional hypothesis. Given a set of neighbors, the *context,* it tries to predict the next word using dense and small embedding vectors.
One of the biggest breakthroughs in word embedding applicability came with the paper and the subsequent toolkit word2vec developed by Mikolov et al. 2013. In their work the authors showed that learned word vectors can be used in vector calculations to express very intuitive patterns such as:

1. ***vec("Madrid") - vec("Spain") + vec("France") ≈ ("<u>Paris</u>")***
2. ***"Berlin" is to "Germany" as "Bangkok" is to "<u>Thailand</u>"***
3. **but also "Which word doesn't fit?"**
   "sweet sour bitter ~~expensive~~"

This resembles the way humans think about concepts and so is very convenient to use in semantically-driven tasks.

The most used models in word2vec are the Skip-gram model developed by Mikolov et al. and the continuous bag-of-words model (CBOW). Skip-gram predicts source context-words from the target words, so it can predict the words ‘dog loudly’ from the word ‘barks’. I general, skip-gram is a variation on n-gram where the words do not necessarily are positioned near each other but are *k* distance units away from each other. CBOW however, does it the other way around, it predicts target words from the source context-words. The training of such a model is happening inside a two-layer neural network. The following figure comes from the TensorFlow[13] explanation of the word2vec algorithm and explains the way the neural net works using the CBOW model.

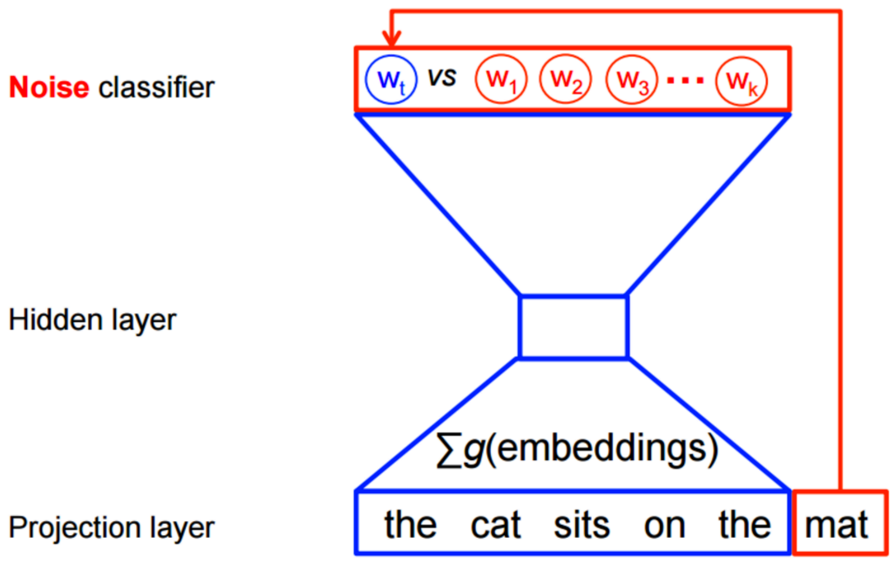


Figure 10: The use of the efficient negative sampling in the CBOW model

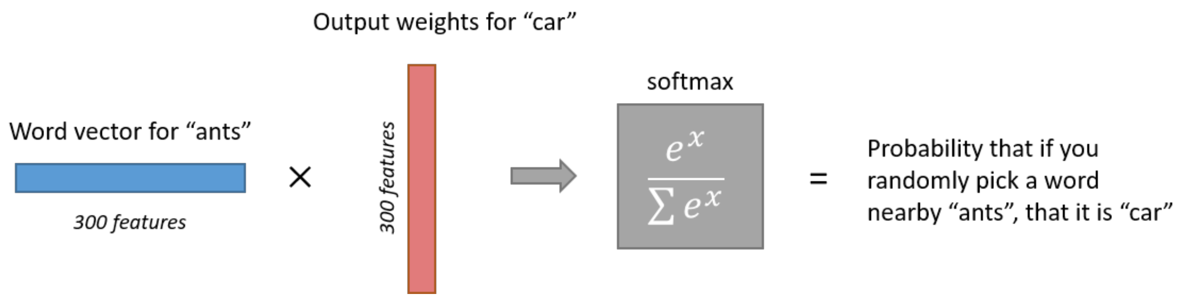
The way this model works is to try to maximize the likelihood of the next (target) word w_t given a set of previous (history) words h. We want to express the scores of w_t in regard to h as probabilities and those probabilities need to sum up to 1. So we need some kind of normalization to achieve that. *Softmax function* does that: it transforms an N-dimensional vector of any real values to an N-dimensional vector of real values in the range [0, 1] that add up to 1. An intuitive way to think of this

[13] https://www.tensorflow.org/tutorials/word2vec

approach is to see the hidden layer as a lookup table. Each word in the vocabulary is encoded as a *one-hot* vector: it is all zeros and only has a 1 in the position corresponding to the word. So our hidden layer is essentially a weight matrix with *n* rows (corresponding to the vocabulary of the size *n*) and *k* columns (corresponding to the number of hidden neurons, number of words the model compares the input word to, a hyperparameter of the model). If an input is a one-hot vector and we multiply that with the weight matrix we get exactly the row in the matrix which corresponds to the word:

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

This way we get a word weight vector for a single word with the size of the number of the hidden neurons. But, as mentioned before, we need this vector to sum up to 1. We use *softmax* for that. The hidden layer output is put into the softmax function to get such a normalized vector. The whole system looks like this, simplified[14]:



The model would then be trained by maximizing the log-likelihood on the training set. The log-likelihood is used because it is more convenient for the calculation of the maximum likelihood as we will end up with the sum of the likelihoods instead of the product. However, this model is still very inefficient as it will need to calculate the likelihoods of words for every word in history at every training step, as seen in:

$$\begin{aligned} J_{\text{ML}} &= \log P(w_t|h) \\ &= \text{score}(w_t, h) - \log\left(\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}\right). \end{aligned}$$

[14] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

So, *Mikolov's* team solved this problem by using a technique called *negative sampling.* It is too elaborate to introduce it here with the proper math but essentially the model now compares the word w_t with a set of noise words instead of the whole vocabulary. The problem of predicting the correct word is reduced to a binary classification task, where the model tries to distinguish real data from noise samples (figure 10). In short, for every word and its context we generate *k* noise words from some noise distribution. Afterwards we label the correct words as positive given their context and all the noise words as negative. Logistic regression is then used to minimize the log-likelihood of our training examples against the noise. The neural net used here is the same that is explained above. For in-depth explanation of the negative sampling method I refer to Mikolov et al. The most important part to understand for now is that given a word *w*, the model has a vector in it with *n* dimensions that is in close proximity to other words that are similar to *w* in the vector space.

Now for my implementation of word2vec. I used the distribution that is incorporated in *gensim,* a Python package. This word2vec implementation takes a sequence (or list) of sentences where each sentence is a list of words. So some preprocessing and tokenization is necessary.

I used gensim's internal preprocessing library on my data. The preprocessing included the stripping of punctuation, removing stopwords, stripping non alphanumerical characters, stripping all the html tags and lowercasing the texts. The reason for removing the stopwords is that if we look at the way word2vec works it doesn't make sense to include words that are often positioned inside the skip-gram of the target word but carry no semantic meaning. The same goes for the stripping of the excessive characters and tags. Lowercasing is a way to normalize the data to just encompass the semantic values. As for the parameters, my configuration was as following:

model = gensim.models.Word2Vec(processed, min_count=10, size=500, workers=8)

where *processed* is a list of preprocessed texts; *min_count* is the minimal amount of appearances of a word in the dataset; *size* is the dimensionality of the feature vectors, the de-facto size of the layers in the neural network and *workers* is the number of threads used for parallel computation. I used CBOW as my model for its speed but will reflect on the potentially better use of skip-gram in the *Future works and reflections* chapter, as skip-gram might perform better for infrequent words*.*

As mentioned, I tried different combinations of data to train my model. Results from all the different models will be presented in the section 4.5. For now, I will focus on a single model, namely the one that used the original review data (560k+) and a subset of extra data. Extra 'Food' dataset was used fully and I took 500k random entries from the 'Kitchen and home' set as well, for a grand total of 2.365.610 reviews. Training on this whole dataset took 487 seconds, which is under 10 minutes. I think that this speed is mostly achieved by a combination of multithreading and the use of CBOW model instead of the skip-grams.

Once done, I could perform queries like ***model.similarity('tasty', 'delicious')*** and ***model.most_similar(positive=['wonderful'], negative=['terrible'], topn=10).***

Coming back to the problem at hand, for aspect extraction I used a combination of this trained word2vec model and the dependency parser. To do that, I first have written a small function that, given a word *w*, a list of candidate words and a word2vec model, calculated the most similar candidate word for *w* together with its similarity score*.* This way, I could make queries like

most_similar_from_list('expensive', ['delivery', 'price', 'taste', 'packaging'], model)
> ['price', 0.43795922192373477]

Having this function, I could then calculate which aspects are present in each relevant dependency tuple in a review by taking the maximum aspect score (given by word2vec) of that tuple. This was incorporated in two main functions, *which_aspect* and *get_aspects.* For the sake of its importance I will include those two functions below. For the rest of my (relevant) code see Appendix.

```
""" This function accepts a word and a word2vec model as input arguments.
It returns multiple strings and booleans:
nextsim = next similar word to the input word, according to word2vec model.
Used to find an aspect even when the input word cannot be matched with an
aspect
aspectwoord = the actual aspect word corresponding to the input word
score = the confidence score of the aspect word being related to the input
word
teklein = boolean that indicates whether the confidence score is too small
isnotin = boolean that indicates whether the word could not be found in the
vocabulary
aspectsimilar = aspect of the most similar word (nextsim)
The function first tries to find the most suiting aspect term for the input
word from the list of aspects. Then, if it finds such an aspect, it throws
'packaging', 'delivery' and 'service' aspects in one broad category. Then,
it checks whether the confidence score is too low and looks for an aspect
on the most similar word in a similar fashion as explained."""

def which_aspect(word, model2):
    score = 0
    aspectwoord = ""
    aspectsimilar = ""
    nextsim = ""
    teklein = False
    isnotin = False
     try:
        aspect_candi = most_similar_from_list(word, ASPECTS, model2)
        aspectwoord = aspect_candi[0]
        score = aspect_candi[1]
        if aspectwoord == 'packaging' or aspectwoord == 'delivery' or
aspectwoord == 'service':
            aspectwoord = 'service'

        if aspect_candi[1] < 0.01:
            teklein = True
            similar = model2.most_similar(word, topn=10)
            for i in range(len(similar)):
                nextsim = similar[i][0]
                aspect_similar = most_similar_from_list(nextsim, ASPECTS,
model2)
                aspectsimilar = aspect_similar[0]
                scoresimilar = aspect_similar[1]
                if scoresimilar > 0:
                    if aspectwoord == 'packaging' or aspectwoord ==
'delivery' or aspectwoord == 'service':
                        aspectwoord = 'service'
                    score = scoresimilar
                    break
        else:
            teklein = False

    except:
        isnotin = True

    return nextsim, aspectwoord, score, teklein, isnotin, aspectsimilar
```

```
'''
This function accepts a text and a word2vec model. It returns a set of all
seen aspects in the text.
The 'weights' of the aspects can be manually adjusted by means of trial and
error or domain knowledge, like when we know that aspect 'health' is harder
to detect because of the more general words that are often associated with
the aspect and are not present often in the dataset.
'''
def get_aspects(text, w2vmodel):
    most_prominent = []
    max_prom = ""
    most_prominent1 = ""
    most_prominent2 = ""
    most_prominent_tuple = ()
    all_aspects = []
    for p in parse_list([text]):
        firstword = p[0][0]
        secondword = p[2][0]
        first = which_aspect(firstword, w2vmodel)
        second = which_aspect(secondword, w2vmodel)
        sim1 = first[0]
        sim2 = second[0]
        aspct1 = first[1]
        aspct2 = second[1]
        score1 = first[2]
        score2 = second[2]
        teklein1 = first[3]
        teklein2 = second[3]
        isnotin1 = first[4]
        isnotin2 = second[4]
        aspctsim1 = first[5]
        aspctsim2 = second[5]
        if aspct1 == 'service' or aspctsim1 == 'service':
            score1 = score1*0.70
        if aspct2 == 'service' or aspctsim2 == 'service':
            score2 = score2*0.70
        if aspct1 == 'health' or aspctsim1 == 'health':
            score1 = score1*1.70
        if aspct2 == 'health' or aspctsim2 == 'health':
            score2 = score2*1.70
        if firstword == 'size' or firstword == 'product' or firstword ==
'products' or firstword == 'buy':
            score1 = score1*0.70
        if secondword == 'size' or secondword == 'product' or secondword ==
'products' or secondword == 'buy':
            score2 = score2*0.70

""" Here we look at the absolute difference between the scores of the first
word in the dependency pair and the second one and put them to the
extracted aspect set accordingly. If the scores do not differ and are
significant, both words are put in the set. """

        if (abs(score1-score2) <= 0.01) and (score1 > 0.30 or score2 > 30):
            if aspctsim1 == "" and aspctsim2 == "":
                most_prominent1 = (aspct1, score1)
                most_prominent2 = (aspct2, score2)
            elif aspctsim1 != "" and aspctsim2 == "":
                most_prominent1 = (aspctsim1, score1)
                most_prominent2 = (aspct2, score2)
            elif aspctsim1 == "" and aspctsim2 != "":
                most_prominent1 = (aspct1, score1)
                most_prominent2 = (aspctsim2, score2)
            elif aspctsim1 != "" and aspctsim2 != "":
                most_prominent1 = (aspctsim1, score1)
                most_prominent2 = (aspctsim2, score2)
    else:
    """ Otherwise, only the highest scoring one is put in the set."""
                max_ = max(score1, score2)
                if max_ > 0.30:
                   if max_ == score1 and aspctsim1 == "":
                       most_prominent1 = (aspct1, score1)
                   elif max_ == score1 and aspctsim1 != "":
                       most_prominent1 = (aspctsim1, score1)
                   elif max_ == score2 and aspctsim2 == "":
                       most_prominent1 = (aspct2, score2)
                   else:
                       most_prominent1 = (aspctsim2, score2)
          else:
                continue
        most_prominent.append(most_prominent1)
        if most_prominent2 != "":
            most_prominent.append(most_prominent2)
        all_aspects = [x[0] for x in most_prominent]
    return list(set(all_aspects))
```

For both functions, a lot of handcrafted regularizations can be made if certain patterns are seen. For example, the ‘price’ aspect gets assigned to words like ‘sell’, ‘box’, ‘buy’ and ‘store’. However, in practice, a lot of those contexts are associated with other aspect like ‘delivery’. So some regularizations, like discarding some ‘deceiving’ words or manually bumping up the score for an aspect like ‘health’ is a way to fine-tune the algorithm to the knowledge we have about the domain. Those two functions made it fairly easy to do. Another feature I implemented was a kind of propagational search through semantically related words if a target word could not be assigned to an aspect (got a score below some really low threshold). This way, I ensured that all relevant words that are extracted by the dependency parser got put in an aspect with a reasonable certainty. In practice, the propagation never exceeded two steps and it significantly helped right aspect extraction. All in all, *which_aspect* function prepared everything for the final aspect extraction in *get_aspects.*

I decided to bring aspects ‘packaging’, ‘delivery’ and ‘service’ under one aspect ‘service’. The reason for that being that a lot of opinions on those subjects are connected to the service aspect of the product but are difficult to bring under a certain, niche, aspect. So that makes the final aspect list being {'taste', 'health', 'price', 'service', 'delivery', 'packaging'}

Detailed sample output from the system without manual balancing of the aspect weights looked like this:

INPUT:

review3 = "*Candy was delivered quickly. My only complaint would be that it seems old as it is tougher to chew. Have ordered and had this product many times and find it delicious but not usually so tough to chew.*"

```
processed = ' '.join([tok for tok in preprocess_string(review3.lower(), [re
move_stopwords, strip_punctuation, strip_tags]) if len(tok) >2])
show_aspects(processed, big_model)
asps = get_aspects(processed, big_model)
print(asps)
```

OUTPUT:

```
candy delivered quickly complaint old tougher chew ordered product times de
licious usually tough chew

-+-+-+-

((u'chew', u'VBP'), u'nsubj', (u'candy', u'NN'))
Aspect of chew is: taste 0.101363752045
Aspect of candy is: service 0.125595597839
Most prominent aspect is service with a score of 0.125596

-+-+-+-

((u'chew', u'VBP'), u'nsubj', (u'complaint', u'NN'))
Aspect of chew is: taste 0.101363752045
Aspect of complaint is: service 0.146401411858
Most prominent aspect is service with a score of 0.146401

-+-+-+-

((u'complaint', u'NN'), u'amod', (u'old', u'JJ'))
Aspect of complaint is: service 0.146401411858
old can't be put in one of defined aspects
olds is the most similar word to old that has a relation to an aspect and i
t's aspect is: service 0.00335930430473
Most prominent aspect is service with a score of 0.146401

-+-+-+-

((u'delicious', u'JJ'), u'nsubj', (u'times', u'NNS'))
Aspect of delicious is: taste 0.209588576926
Aspect of times is: service 0.0614434350653
Most prominent aspect is taste with a score of 0.209589

-+-+-+-

((u'times', u'NNS'), u'compound', (u'product', u'NN'))
Aspect of times is: service 0.0614434350653
Aspect of product is: service 0.35765115948
Most prominent aspect is service with a score of 0.357651

-+-+-+-
```

```
[('service', 0.12559559783907934), ('service', 0.14640141185761685), ('serv
ice', 0.14640141185761685), ('taste', 0.20958857692639116), ('service', 0.3
5765115948033566)]
All aspects of the sentence are: ['service', 'service', 'service', 'taste',
 'service']
```

As seen here, aspect taste is not identified with big confidences while it should be in some cases like '*delicious*', while 'product' is identified as 'service' with a high confidence. Balancing the aspects and fine-tuning the addition threshold improves this extraction substantially.

4.3.1 Word2vec + dependency parsing evaluation

And now for the evaluation of this combined system. I had no aspect labeled data in my domain so I decided to do it myself. Around 700 sentences were chosen randomly from the dataset by way of random shuffling the whole reviews list and taking the first 700 sentences out of it. This had an advantage in comparison to just selecting a random index and taking 700 sentences from there on that it accomplished the same goal of taking 700 random sentences but now they were also randomly shuffled which removes possible internal ordering of the reviews. Namely, it can be the case that depending on the way the data was collected (web scraping, API calls or just an Amazon dump) the reviews in the set are ordered by the appearance on the site or food category in one way or another. If it is the case than it could interfere with the evaluation because the reviews could be similar and so capture less of the variance in the dataset. I did not check if this internal ordering is the case in the data but either way it would be counteracted this way. A sample from my hand tagged data is found in the Appendix.
To evaluate the extraction algorithm by using conventional classification metrics the output should be converted to a form that is convenient for those metrics. In my case, each hand tagged sentence had a zero or a one for each aspect depending on whether the aspect is present in the sentence or not. This way each sentence had a vector of the form

$$['Taste', 'Health', 'Price', 'Service'] \rightarrow [1, 0, 0, 1]$$

My aspect extractor just gave the list of detected aspects as output so I converted this list to a vector by assigning 1 or a 0 to the index in a vector where the aspect belongs. So if the algorithm's output was ['taste', 'price'] given a certain sentence, this output was transformed to [1, 0, 1, 0].
This way I had everything I need to perform precision and recall measures on the data.
Below are the results from using sklearn's classification report feature and the accuracy.

Data: new + old + kitchen

	avg. precision	avg. recall
Taste	0.88	0.68
Health	0.80	0.81
Price	0.90	0.89
Service	0.69	0.67
Avg.	**0.818**	**0.763**

4.4 Word embedding: word2vec only

With performances being reasonable, the biggest concern was still the speed of the aspect extraction. It took my aspect extractor around half an hour to create aspect vectors of the 700 hand tagged sentences. I can't name the exact reason for the slow performance but I think that the bottleneck in this approach is the slow implementation of the dependency parser: the Stanford Parser needs external Java jars to be used in Python.

So I decided to try to drop the dependency parsing and use word2vec method on every word in the sentence, instead of just noun phrases. In the end, dependency parsing is a way to smartly target only relevant words for the aspect extraction which could save us some time. However, this is not the case right now, as the dependency parser implementation is clearly the time / complexity bottleneck here. So the algorithm for word2vec now looked like this, much shorter and much more simple.

```
""" The function below accepts text and a word2vec model. It first pre-proc
esses the text by lowercasing it, removing stopwords[15], stripping html tags
, removing each token shorter than two letters and finally splitting the te
xt into tokens. It then iterates through the tokens list and calculates asp
ect and score for each token. The weights are trial and error or domain kno
wledge (if the word is package it is very important for its aspect) and the
 exact numbers can't be explained by some logic. """

def get_aspects_no_parsing(text, w2vmodel):
    toks = [tok for tok in preprocess_string(text.lower(),
         [remove_stopwords, strip_punctuation, strip_tags]) if len(tok) >2]
    most = []
    for tok in toks:
        w_a = which_aspect(tok, w2vmodel)
        aspect = w_a[1]
        score = w_a[2]
        if aspect == 'service':
            score = score*0.70
        if aspect == 'health':
            score = score*1.70
        if tok == 'size' or tok == 'product' or tok == 'products' or tok ==
 'buy':
            score = score*0.70
        if tok == 'delicious':
            score = score*2
        if tok == 'packages' or tok =='package':
            score = score*2
        if aspect == 'taste':
            score = score*1.3
        if score > 0.3:
            most.append(aspect)
```

[15] https://github.com/RaRe-Technologies/gensim/blob/develop/gensim/parsing/preprocessing.py

```
    return list(set(most))
```

With the removal of dependency parsing, the 700 vectors were created in 18 seconds. That is **100 times faster** than with the dependency parser. And what was unexpected is that the performance actually got better as well! The explanation for that might be that my POS-patterns and dependency relations did not capture the whole width of the aspect indicators.

4.5 Word2vec aspect extractor evaluation

With having word2vec only implementation I could perform the same evaluation metrics as before. Below are the results from it using different data combinations for the training of the models.

Types of data:

Vanilla = 'old' data, the initial 560k+ food reviews from Amazon

Old + new + kitchen = the data I used in the previous sections. Basically everything with a subset from kitchen.

Google model = pretrained Google model data on Google news (100 billion words)[16]

Only new food = extra food reviews from Amazon only

Kitchen + new food = extra food reviews and 500k kitchen reviews

Only kitchen = only 500k kitchen reviews

Old + new = 'old' and new food reviews combined, no kitchen

Vanilla

	precision	recall
Taste	0.85	0.81
Health	0.83	0.83
Price	0.88	0.82
Service	0.73	0.69
Avg.	**0.822**	**0.788**

[16] https://code.google.com/archive/p/word2vec/

Old + new + kitchen

	precision	recall
Taste	0.86	0.79
Health	0.83	0.83
Price	0.92	0.91
Service	0.70	0.69
Avg.	**0.828**	**0.805**

Google model

	precision	recall
Taste	0.85	0.86
Health	0.74	0.68
Price	0.85	0.75
Service	0.67	0.66
Avg.	**0.778**	**0.738**

Only new food

	precision	recall
Taste	0.85	0.80
Health	0.83	0.83
Price	0.88	0.81
Service	0.71	0.70
Avg.	**0.818**	**0.785**

Kitchen + new food

	precision	recall
Taste	0.86	0.80
Health	0.84	0.84
Price	0.88	0.81
Service	0.71	0.69
Avg.	**0.823**	**0.785**

Only kitchen

	precision	recall
Taste	0.87	0.78
Health	0.80	0.70
Price	0.90	0.89
Service	0.70	0.70
Avg.	**0.818**	**0.768**

Old + new

	avg. precision	avg. recall
Taste	0.86	0.80
Health	0.83	0.83
Price	0.88	0.81
Service	0.71	0.69

Avg. **0.82** **0.783**

Old + new + kitchen seem to win it by the slightest so I end up using it as a model for my summarizer. Interesting part is that the 'only kitchen' data configuration did not perform bad at all. The reason for that might be that people tend to use the same kind of words in the kitchen reviews because the most products are still related to food in one or another way.
Even more appealing of this new, dependency free, approach is that when we compare the results on the same data (old + new + kitchen), we get these two tables:

With dependency parsing

	avg. precision	avg. recall
Taste	0.88	0.68
Health	0.80	0.81
Price	0.90	0.89
Service	0.69	0.67

Avg. **0.818** **0.763**

Without dependency parsing

	avg. precision	avg. recall
Taste	0.86	0.79
Health	0.83	0.83
Price	0.92	0.91
Service	0.70	0.69

Avg. **0.828** **0.805**

The variant without the parser gives better results and is way faster and more memory efficient than the variant with the parser.
All in all, with reasonable results from both the sentiment classifier and the aspect extractor I could begin combining those two together for a summarizer.

Chapter 5: Summarizer

5.1 Combining the two systems

My main research goal was to explore the different sentiment analysis algorithms and aspect extraction possibilities. However, it is interesting to see whether the two systems can be combined to form one possibly useful application. In this chapter I will show an example of such an application in the form of a summarizer. However, I did not have a way to evaluate this system so this is purely a possible way to think of combining the two systems.

As mentioned before, convenience for the user is very important for the online retailers. So a concise and insightful summary of each review that shows exactly which aspects of the product are discusses can be helpful.
My approach for the summarizer was to split a review into sentences, calculate the sentiment of each sentence using my trained SVM model and extract tuples of ***(aspect, words that triggered that aspect)***. So, for example in the following review:

multiple_sentiment_review = 'Haribo has always made the best gummy bears, with the perfect texture and flavor. However, the aftertaste was really nasty.'

The text was split into two sentences and I extracted the following with a help of *combined* function which just ran the sentence through both my models:

```
(array([1]), [('taste', 'texture_flavor'), ('taste', 'perfect_textur
e')])
(array([-1]), [('taste', 'however_aftertaste')])
```

with 1 or -1 indicating either positive or negative sentiment. The tuples indicate that *texture, flavor* and *aftertaste* all contributed to the extraction of the 'Taste' aspect in

their sentences. The word *however* suggests that aftertaste is not a good addition to the product. We would of course also like to extract *nasty_aftertaste.* That is not a simple task for this system but is a fairly trivial one for the dependency parser. That might be a good application of the parser still, if we would prefer that above performance speed. But that is a topic for further research chapter.
The next step was to extract these indicative terms per aspect in a single review to twelve lists of words (one corresponding to each aspect / sentiment combination, 4x3) in the following way:

```
aspects_terms_per_sentiments = defaultdict(list)
sentiments_per_sentence = []

    for s in re.split('! |\. |\?', text):

        sentiment = ""
        info = combined(s, vectorizer_model, sentiment_model, w2vmodel)
        aspects = info[1]
        polarity = info[0][0]
        if polarity == 0:
            sentiment = 'neutral'
        elif polarity == 1:
            sentiment = 'positive'
        elif polarity == -1:
            sentiment = 'negative'

        sentiments_per_sentence.append(sentiment)

       taste_positive_words = [combi[1] for combi in aspects if combi[0]==
 'taste' and sentiment == 'positive' and combi[1] != '']

        taste_neutral_words = [combi[1] for combi in aspects if combi[0] ==
 'taste' and sentiment == 'neutral' and combi[1] != '']

       taste_negative_words = [combi[1] for combi in aspects if combi[0] ==
 'taste' and sentiment == 'negative' and combi[1] != '']
```

I did this four times to cover all aspect/sentiment pairs. Afterwards, the resulting lists were put in a dictionary. Example of that with the last aspect and the return of the whole extraction. All other aspects had exactly the same treatment:

```
        aspects_terms_per_sentiments['service_positives'].extend(service_po
sitive_words)

        aspects_terms_per_sentiments['service_neutrals'].extend(service_neu
tral_words)

        aspects_terms_per_sentiments['service_negatives'].extend(service_ne
gative_words)


    return dict(aspects_terms_per_sentiments), sentiments_per_sentence
```

The end result of the function looked like this

INPUT:

```
                              review to analyze          models for sentiment analysis    w2v model
extract_terms_per_aspect (multiple_sentiment_review, vectorizer_model, sentiment_model, model)
```

OUTPUT:

```
({'price_positives': [], 'price_negatives': [], 'health_negatives': [], 'taste_neutrals': [], 'ta
ste_negatives': ['however_aftertaste'], 'price_neutrals': [], 'health_positives': [], 'servic
e_neutrals': [], 'service_negatives': [], 'service_positives': [], 'health_neutrals': [], 'taste
_positives': ['texture_flavor', 'perfect_texture']}, ['positive', 'negative'])
```

This is a kind of an overview of what is going on in this single review. We see that there are words in the review that are negative on the aspect 'Taste' and there are words that a positive on the aspect 'Taste'. Further, no aspects are discussed.
Next step is just making a way of 'pretty printing' the summaries using the information we got from *extract_terms_per_aspect.* I will include the code for the making of a pretty summary in my Appendix as it is too long to show it here. In the next section a few examples are shown.

5.2 Sample summaries

Given a review, the summarizer first splits the text into sentences and then preprocesses it the way explained in chapter four on word embeddings. It then runs it through the *make_pretty_summary* function which creates a readable summary. Below are some examples of that.
In figure 16, the aspect discussed the most is clearly 'Taste'. Those words also get picked up by the algorithm, creating an overview of what is good in this product. The aspect price is also extracted, indicating the product is cheaper than somewhere else.
We would also like the algorithm to extract the availability bit about the larger bag in the first sentence of the review which does not get extracted. Especially because the word 'bag' is present in the sentence. This indicates that 'bag' doesn't match quite as much with packaging as we want. Luckily, we can fix that easily by manually adjusting its weight. In this case it is not a malicious thing, as generally we

always want the word ‘bag’ to be put in the packaging aspect. And yes, if we do just that and run this code:

```
if tok == 'packages' or tok =='package' or tok == 'bag':
            score = score*2
```

We get the result in figure 17.

```
----------------- Review Summary -----------------

The input text is: "These are the absolute best chocolates you can find for
 the price, and cheaper in the large size bag than what you can find in the
 store. The dark chocolate is the best melt-in-my-mouth chocolate. I just s
uck on it and taste the goodness until it dissolves. The caramel is the per
fect mix of caramel with chocolate. And the milk chocolate is perfect for t
hose who don’t like dark chocolate and want some solid chocolate. You can't
 go wrong with any of the flavors."

The overall sentiment of the text is positive

Below there is a summary of this product based on four aspects with the wor
ds associated with that aspect per sentiment.

Taste:
    Positive words: dark_chocolate, mouth_chocolate, suck_taste, caramel_ch
ocolate, mix_caramel, caramel_perfect, like_dark, dark_chocolate, don’t_lik
e, solid_chocolate, milk_chocolate, wrong_flavors
    There are no neutral words about the taste of this product!
    There are no negative words about the taste of this product!

Health:
    Health aspect is not mentioned or is not (significantly) detected by th
e algorithm!

Price:
    Positive words: price_cheaper, chocolates_price
    There are no neutral words about the price of this product!
    There are no negative words about the price of this product!

Service / delivery / packaging / availability:
    Service / delivery / packaging / availability aspects are not mentioned
 or are not (significantly) detected by the algorithm!

---------------------------------------------------
```

Figure 16: No packaging aspect extracted

```
---------------- Review Summary -----------------

The input text is: "These are the absolute best chocolates you can find for
 the price, and cheaper in the large size bag than what you can find in the
 store. The dark chocolate is the best melt-in-my-mouth chocolate. I just s
uck on it and taste the goodness until it dissolves. The caramel is the per
fect mix of caramel with chocolate. And the milk chocolate is perfect for t
hose who don’t like dark chocolate and want some solid chocolate. You can't
 go wrong with any of the flavors."

The overall sentiment of the text is positive

Below there is a summary of this product based on four aspects with the wor
ds associated with that aspect per sentiment.

Taste:
    Positive words: dark_chocolate, mouth_chocolate, suck_taste, caramel_ch
ocolate, mix_caramel, caramel_perfect, like_dark, dark_chocolate, don’t_lik
e, solid_chocolate, milk_chocolate, wrong_flavors
    There are no neutral words about the taste of this product!
    There are no negative words about the taste of this product!

Health:
    Health aspect is not mentioned or is not (significantly) detected by th
e algorithm!

Price:
    Positive words: price_cheaper, chocolates_price
    There are no neutral words about the price of this product!
    There are no negative words about the price of this product!

Service / delivery / packaging / availability:
    Positive words: size_bag
    There are no neutral words about the service/packaging/delivery/availab
ility of this product!
    There are no negative words about the service/packaging/delivery/availa
bility of this product!

---------------------------------------------------
```

Figure 12: Packaging aspect extracted after manual adjustment

One final review that captures the aspects is showcased in figure 18.
So much for the example summaries. This application could be an asset but, in the end, the keywords generated by the aspect extractor may be of better use in a search system for reviews per aspect. Such a search application remains to be a topic for future work. As I do not have a way to evaluate the summarizer I can’t state whether it is of use to clients or retailers.
In the next chapter I will go through some of the critical points towards my approach and possible improvements.

```
----------------- Review Summary -----------------

The input text is: "These are by far my favorite chips, they are extremely
crunchy (similar to other Kettle style chips), but it's the extreme Vinegar
 flavor that differentiates these from lesser fried potato snacks.  I've tr
ied several other brands of Salt and Vinegar and none of them come close. T
hey are also not that and greasy which makes it a better product for those
who want to lose weight! It came in a good plastic bag that was filled to t
he top. The shipping was quick!"

The overall sentiment of the text is positive

Below there is a summary of this product based on four aspects with the wor
ds associated with that aspect per sentiment.

Taste:
    Positive words: extremely_crunchy, vinegar_flavor
    There are no neutral words about the taste of this product!
    There are no negative words about the taste of this product!

Health:
    Positive words: lose_weight
    There are no neutral words about the healthiness of this product!
    There are no negative words about the healthiness of this product!

Price:
    Price aspect is not mentioned or is not (significantly) detected by the
 algorithm!

Service / delivery / packaging / availability:
    Positive words: shipping_quick
    Neutral words: plastic_bag
    There are no negative words about the service/packaging/delivery/availa
bility of this product!

---------------------------------------------------
```

Figure 18: A capture of the essential words by the summarizer

Chapter 6: Critical reflection

6.1 Sentiment calculation

I have not used stemming as a preprocessing step while multiple papers claim its effectiveness. The reason for not using it was plain simply forgetting about including it in the preprocessing step. I thought about it only when writing this document and it was a bit too late to properly redo all the work with the stemmer in place. Certainly something for the future works.
Another critical remark is that I did not invest time into researching why the extra data did not improve the results of my SVM. There are some possible explanations, like that the added data is not that linearly separable and I would need to increase the regularization parameter on the go to increase classification precision. Or maybe even go with a kernel altogether. I do not think that is the problem, however, as linearity usually 'increases' with the increase of data. Or I could be overfitting on the extra data altogether and so diminishing my test results. Anyway, I did not research into that properly and that would be indeed very interesting as I believe that when there is extra data we should exploit it in some way.

6.2 Aspect extraction

As for the aspect extraction, I noticed multiple implementation details while writing this document.
In my gensim word2vec implementation, I used the CBOW algorithm, which is faster than the skip-gram. However, the skip-gram model performs better on infrequent features and on a big dataset that could be interesting. Moreover, skip-gram model idea fits better for the aspect extraction idea: it tries to predict source words from the target word and that is essentially what we are doing with aspect extraction.
Secondly, the word2vec documentation states: '*dimensionality of the word vectors: usually more is better, but not always*'. I used 500 as my dimensionality but I could

have experimented with more, up to a thousand, because the actual training was remarkably fast.

6.3 Overall reflection

Overall I noticed that this project spawned a lot of documents, scripts, small result sheets and such. At a certain moment near the writing of this document that amount was too big really. Because of that I had to re-run some of my code because I couldn't find my notes anymore.
I think that this is partly because I tried too many different approaches and afterwards I think that it would be better if I would focus on a single, more restricted, topic. But nonetheless, I found this a very informative, interesting and challenging project to do.

Chapter 7: Conclusion and future work

7.1 Conclusion

In my research question I asked which sentiment analysis and aspect extraction techniques perform well on the field of online product reviews. I have evaluated two sentiment classifiers and came to the conclusion that a linear support vector machine is a good fit for the sentiment classification of short reviews. Afterwards, I looked into the aspect extraction and tried out both a system with explicit word pairs extraction (dependency parsing) and a system without the explicit word pair extraction. I found that word embeddings is a good technique for this task. It is fast, memory efficient because of the pretrained models and, more importantly, it captures the hidden semantic connections between the words, which other models struggle to do. Both the sentiment classifier and the aspect extractor were ready to be ‘plugged in’ any system that achieves to work with short online product reviews. Afterwards I have built a small possible application of the two models: a review summarizer. The system I have built showed some potential for a review summarizer but could be further improved to suit multiple needs. First, my system does not have a reliable evaluation metric for it to be called a summarizer. Once that is found, the system could be a first step towards a good summarization system for online reviews. Aspect based sentiment extraction, when done right, could be a great tool to make concise summaries of reviews based only on the relevant parts of those reviews.
Secondly, the aspect terms my system extracts could be used as a search tool for reviews that target certain aspects. Such a system does not exist on Amazon right now, for one, and could be of use for the customers.
The system I have built is adaptive. It is adaptive in the sense that it can be adapted to use domain knowledge, such as the manual weights on certain indicative words or reduced weights on overly ‘general’ aspects like ‘’. Any other data source can be supplied to train both sentiment and word2vec models on another domain. Also

improvements like I mentioned in the previous section can be implemented to further fine-tune the system.

7.2 Further work

Most of the further work would consist of the improvements I mentioned in the reflection section. This section will address mostly further work that can be of use for the summarizer application of my system.
One of the most feasible ways to improve the quality of the summarization step is to use dependency parsing after all. But not to extract the aspects at the word embeddings stage, as it does not add anything of value to the model as seen in chapter four, but rather use it as a way to generate sensible word pairs at the summary stage. As I explained, my summarizer reports pairs of consecutive words which are indicative of an aspect. However, just taking adjacent words does not always work. Often the adjective of a noun is not adjacent but stays somewhere else in the sentence. By using dependency parsing on the summarizer stage we could extract the dependency pairs with the aspect indicator as a head. This way we could use these word combinations instead of the pure adjacent words to the indicator.
Another improvement that would help the system at the sentiment stage is a better way to handle negations and amplifications. These parts are by far the best handled by lexicon-based classifiers that can use hand-crafted rules for sentiment score manipulation when a negator or an amplifier is encountered so that may be a possible further work as well.

Works Cited

1. Aue, A., & Gamon, M. (2005, September). Customizing sentiment classifiers to new domains: A case study. In *Proceedings of recent advances in natural language processing (RANLP)* (Vol. 1, No. 1-3, pp. 2-1).

2. Bagheri, A., Saraee, M., & de Jong, F. (2013, June). An unsupervised aspect detection model for sentiment analysis of reviews. In *International Conference on Application of Natural Language to Information Systems* (pp. 140-151). Springer Berlin Heidelberg.

3. Barzilay, R., & Elhadad, M. (1999). Using lexical chains for text summarization. *Advances in automatic text summarization*, 111-121.

4. Chiou, J. S., & Cheng, C. (2003). Should a company have message boards on its web sites?. Journal of Interactive Marketing, 17(3), 50-61.

5. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273-297.

6. Goldstein, J., Mittal, V., Carbonell, J., & Kantrowitz, M. (2000, April). Multi-document summarization by sentence extraction. In *Proceedings of the 2000 NAACL-ANLPWorkshop on Automatic summarization-Volume 4* (pp. 40-48). Association for Computational Linguistics.

7. Grewal, D., Munger, J. L., Iyer, G. R., & Levy, M. (2003). The influence of internet - retailing factors on price expectations. Psychology & Marketing, 20(6), 477-493.

8. Hirshleifer, D. A. (1994). The blind leading the blind: social influence, fads and informational cascades.

9. Hu, M., & Liu, B. (2004, August). Mining and summarizing customer reviews. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 168-177). ACM.

10. Huang, J. H., & Chen, Y. F. (2006). Herding in online product choice. Psychology & Marketing, 23(5), 413-428.

11. Ivan Omar Cruz-Garcia, Alexander Gelbukh, Grigori Sidorov. Implicit Aspect Indicator Extraction for Aspect based Opinion Mining. International Journal of Computational Linguistics and Applications, Vol. 5 No. 2, 2014, pp. 135–152.

12. McAuley, J., Targett, C., Shi, Q., & Van Den Hengel, A. (2015, August). Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 43-52). ACM.

13. Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, 137-142.

14. Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing* (Vol. 3). (Chapter 14).Pearson.

15. Kiritchenko, S., Zhu, X., & Mohammad, S. M. (2014). Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, *50*, 723-762.

16. Koppel, M., & Schler, J. (2006). The importance of neutral examples for learning sentiment. *Computational Intelligence*, *22*(2), 100-109.

17. Lee, J., Park, D. H., & Han, I. (2011). The different effects of online consumer reviews on consumers' purchase intentions depending on trust in online shopping malls: An advertising perspective. Internet research, 21(2), 187-206.

18. Li, X., Wang, H., Gu, B., & Ling, C. X. (2015, June). Data Sparseness in Linear SVM. In *IJCAI* (pp. 3628-3634).

19. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).

20. Nigam, K., Lafferty, J., & McCallum, A. (1999, August). Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering* (Vol. 1, pp. 61-67).

21. Pang, B., Lee, L., & Vaithyanathan, S. (2002, July). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10* (pp. 79-86). Association for Computational Linguistics.

22. Pavlopoulos, J., & Androutsopoulos, I. (2014). Aspect term extraction for sentiment analysis: New datasets, new evaluation measures and an improved unsupervised method. *Proceedings of LASMEACL*, 44-52.

23. Poria, S., Cambria, E., Ku, L. W., Gui, C., & Gelbukh, A. (2014, August). A rule-based approach to aspect extraction from product reviews. In Proceedings of the second workshop on natural language processing for social media (SocialNLP) (pp. 28-37).

24. He, R., & McAuley, J. (2016, April). Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In Proceedings of the 25th International Conference on World Wide Web (pp. 507-517). International World Wide Web Conferences Steering Committee.

25. Ramos, J. (2003, December). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*.

26. Taboada, M., Brooke, J., Tofiloski, M., Voll, K., & Stede, M. (2011). Lexicon-based methods for sentiment analysis. *Computational linguistics*, *37*(2), 267-307.

27. Wu, P. F. (2013). In search of negativity bias: An empirical study of perceived helpfulness of online reviews. *Psychology & Marketing*, *30*(11), 971-984.

Appendix

List of abbreviations

POS – part-of-speech

CBOW – continuous bag-of-words (model)

SVM – Support Vector Machine

word2vec – a model for word embedding, originally developed by Google

sklearn – a Python machine learning library

API - Application Programming Interface

gensim – a Python topic- and vector space modelling library

k – one thousand

TensorFlow – an open-source machine learning library by Google

NLP – Natural Language Processing

vec – vector

NN, JJ, NNP, PRP, NNS – part-of-speech tags corresponding to noun, adjective, singular proper noun, personal pronoun and plural noun, respectively

'amod', 'compound', 'advmod', 'nmod', 'neg', 'num', 'nsubj', 'nmod:npmod' – dependency relations corresponding to adjectival modifier, noun compound modifier, adverbial modifier, noun modifier, negation modifier, numeric modifier, nominal subject and noun phrase modifier, respectively

CV – cross-validation

WordNet – semantic lexical database for English[17]

SenticNet – a semantic knowledge base for English[18]

***n*-gram** – a sequence of *n* tokens

tf-idf - term frequency–inverse document frequency, measure of token importance

[17] http://wordnet.princeton.edu/

[18] http://sentic.net/

AUC ROC	- Area Under the Receiver Operating Characteristic curve, an evaluation metric
IMDb	– Internet Movie Database
Kaggle	– Data science platform
SO-CAL	– Semantic Orientation Calculator used by Taboada et al.
A-score	– metric used by Bagheri, A., Saraee, M., & de Jong, F.
IAC	– Implicit Aspect Clue, as used in Poria et al. 2014

Hand tagged sentences for the aspect extractor evaluation

Text;Taste;Health;Price;Delivery

Love love love this pasta, and it's whole wheat, so it's good for you. Haven't been so pleased since they found out chocolate was good for you...;1;0;0;0
I had read about these bars in various low carb blogs. Yesterday, I came across some at my local grocery store and decided to give it a try. I purchased the milk chocolate bar. I was pleasantly surprised. It was a bite of heaven. They are mildly sweet and have a creamy texture, as evidenced by the fat content. However, what pleases me the most is that they are sweetened without sugar or the use of malitol. I need to eat low carb due to medical reasons and I stray away from sugar alcohols due to gastric reasons!;1;1;0;0
"These are great and epitomize the reason I love Japanese candies. Each one is a tiny, light, crispy, chocolaty crunch. However I highly recommend <a href=""http://www.amazon.com/gp/product/B003N0QXCI"">Meiji Chocorooms, 3.13-Ounce Boxes (Pack of 10)</a> over these. Chocorooms are extremely similar to these except their crispier and have a higher chocolate to cookie ratio, I much prefer them.";1;0;0;0
"This is how jerky is supposed to taste. Extra salty bite with a good chew. If you can't make a nice ""brothy soup"" with your jerky, then you're probably eating the wrong jerky. Keep the soft sweet glazed gourmet crap outta the jerky industry because it's definitely killing it. Lol at the people who says jacks links is a superior product. Oh boy Oberto's thin style is the way to go.";1;0;0;0
I only like herbal tea especially raspberry. This one has a distincive flavor-it's great!!;1;0;0;0
Again this is good stuff but don't buy it here. At my wal-mart I got this exact same thing for $3.98!!!!!!!<br />$3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98 $3.98!!!!;1;0;1;0

"This is a great , mellow coffee. The orange flavoring is so subtle, yet distinct. The orange rinds with the beans make the difference and readily grind up nicely with the beans. People will comment on my excellent coffee and ask me what beans I use and when I inform them of the orange rinds, they remark ""this is excellent! Now that you mention it, I can taste the slight hint of orange."" I just bought another bag to give to my best friend for her birthday - it is that good!";1;0;0;0
they taste very good.and product was in good condition, but way to much packing penuts, i would prefer somthing biodegradabl or recyclabl;1;0;0;1
I recv'd this tea in great condition. The packing and timeliness were great. I love this tea. It takes me back to when I had our son in an Italian hospital near Pisa, Italy. They brought me Lipton Yellow Label Tea. That was many years ago and I never forgot how good it tasted or stopped looking to buy more once we left Europe. Hubby and I were talking about it and I thought of Amazon! It was right here. I'm enjoying it in the afternoon for a pick me up! Good stuff!;1;0;1;1
This coffee has a deep flavor with no bitterness. It stays fresh in the packaging, and with the subscription plan savings it's a great deal!;1;0;1;1
I have used this several times in my chicken soup and it is yummy! Nice new taste.;1;0;0;0
"My dogs did not show any interest in this. I even put peanut butter on it! They love their other Nylabone Bones, but did not care for the shape of this one. The only thing I could think of was that it was to large for them; 20 lb mini-Aussies. One of my dogs is a hard chewer; likes to chew amsot anything. I will stick with the regular bones.";1;0;0;0
"When looking for a ""green"" signature drink for a geeky wedding we came across the Finish This Drink recipe. It called for Kiwi syrup which we could not find locally in any store. Amazon to the rescue - this Monin kiwi syrup made a yummy drink!!";1;0;0;1
This is the most flavorful sea salt we have ever used. The salt will arrive slightly damp in the jar --at least the first four bottles we have opened have been damp -- but once again, no problem. Dump the jar of salt in a pie pan, and put it in the oven as the oven cools after use. Then the salt grinds in any table salt grinder with ease. It is very grey -- I suspect the source of the great flavor! It also makes great kosher dill pickles -- simply disolve the salt and let it sit for a day in a quart sealer, then pour off the clear saline solution. Fabulous taste -- no cloudy pickles!<br />This salt represents exceptional value. We use this salt for just about everything now.;1;0;0;1
My husband loves the green and white fusion tea!! He rates it excellent! I have not tried it as of yet, but of all he has tried this is the one for him.;1;0;0;0
"Got this yesterday, have been like a mad scientist since...adding it to almost everything and thinking about more. Last week got the newsletter from CSPI and PB2 chocolate and plain featured on the back. CSPI trustworthy, so I went for it. I normally get a jar of natural peanut butter, pour off the oil and then for several days put in paper towels to sop up the remaining oil. Then I have maybe a teaspoon at a time - rarely more than a full tablespoon in a day. We are in

Easter Candy Season and the spectre of chocolate peanut butter eggs is everywhere. Hard to be at peace with oneself when peanut butter - and especially combined with chocolate - is readily available. Now this PB2 stuff is not especially chocolatey - but it is a wonder and my next order [soon] will be to try the basic version. The contents of the one pound package will have to be transferred to an air tight container. Note recipe on the side of the bag: just plain weird for a ""health food product"" - a ""MexiNut"" dip involving an undiluted can of bean with bacon soup, canned tomatoes, velveeta, 1/2 cup of PB2 chocolate, and chilli powder. As the soup, the tomatoes and the velveeta are sodium nightmares and the soup and velveeta heavy with fat, loaded with addditives, why would the company promote such a disasterous use of its ""healthy"" product? Use your own good common sense and happy eating.";1;1;0;0
Very bitter! I couldn't taste the coffee at all. It basically tastes like baking cocoa. No sweetness, no richness, not coffee. I bought the 4 pack though, so I've begun mixing it half and half with Swiss Miss (in the can), and then sprinkling it with instant coffee after I've made a cup. Even making it with whole milk couldn't make this yummy by itself. If you like a rich chocolate taste with just a bit of coffee.... shop elsewhere.;1;0;0;0
Love that I could save gas money by not going to the grocery store to buy some food products! Thanks Amazon!;0;0;1;1
I started ordering this green tea after I got an extremely bad shipment of the Davidson's Gunpowder Green. I really like this, and it is a good price with the subscription. Sticking with this one and I will not go back to Davidson's. So much better. Uses less tea also, which further reduces cost.;1;0;1;1
"but ""Chips Ahoy! Chewy Gooey Megafudge"" is TOO MUCH chocolate. Seriously. Milk doesn't even help. In moments of weakness, I have been known to eat a whole package of chocolate chip cookies, with chocolate milk. I was barely able to eat two of these goo-balls. On one hand, maybe that's a good thing - it forced me to moderation. But the truth is, I really didn't like them that much.<br /><br />Oh sure, they are chocolate, and that's really never a bad thing! But these are just too much for my tastes. The description is accurate: they are chewy, they are gooey, they are fudge. Basically, they are the Chips Ahoy chewy fudge cookies, slit open and filled with gooey fudge in the middle.<br /><br />Not bad - just WAY too far over the top for my taste.";1;0;0;0
We haven't opened this yet as it is a Christmas gift, but we found the same drawer in a store for cheaper. I couldn't cancel the order though.;0;0;1;1
This wasabi mayonnaise comes in a squeezable bottle and is easy to use. It is a nice flavor addition to many foods, such as sandwiches, potato salad, and pretty much any food where regular mayo would normally be used. It is especially good on hamburgers and cheeseburgers. I started looking forward to eating a hamburger only because I can put wasabi mayonnaise on it.;1;0;0;0

This snack is the best. I love taking it to school and whenever I take it, all my friends gather around me. It is very crunchy and comes in various tastes like pizza, smoky, or bbq. I would recommend this as a fulfilling snack.;1;0;0;0
Good balance between the honey sweetness and the ginger/lemon tartness. Very natural taste, with no nasty aftertaste. Good hot or cold. Great after dinner drink.;1;0;0;0

Code

All of my relevant code for all the parts of my project is rendered nicely, with output, in Jupyter notebooks in the following GitHub repository. This can be viewed without a GitHub account.

https://github.com/mabergerx/Thesis_Mark

Data: https://we.tl/uDzCaJJUXA