MASTER'S THESIS | LUND UNIVERSITY 2016

Summarizing Product Reviews Using Dynamic Relation Extraction

Oskar Handmark, Mikael Gråborg

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2016-40

Summarizing Product Reviews Using Dynamic Relation Extraction

(Combining Dependency Parse Trees with Statistical Prediction Models to Identify Feature Opinions)

Oskar Handmark
dat11oha@student.lu.se

Mikael Gråborg
dic11mgr@cs.lth.se

October 18, 2016

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

The accumulated review data for a single product on Amazon.com could potentially take several weeks to examine manually. Computationally extracting the essence of a document is a substantial task, which has been explored previously through many different approaches. We explore how statistical prediction can be used to perform dynamic relation extraction. Using patterns in the syntactic structure of a sentence, each word is classified as either product feature or descriptor, and then linked together by association. The classifiers are trained with a manually annotated training set and features from dependency parse trees produced by the Stanford CoreNLP library.

In this thesis we compare the most widely used machine learning algorithms to find the one most suitable for our scenario. We ultimately found that the classification step was most successful with SVM, reaching an FScore of 80 percent for the relation extraction classification step. The results of the predictions are presented in a graphical interface displaying the relations. An end-to-end evaluation was also conducted, where our system achieved a relaxed recall of 53.35%.

Keywords: review analysis, relation extraction, nlp, data mining, machine learning

Acknowledgements

First and foremost, we would like to thank Pierre Nugues for his open mind, persistant efforts and invaluable advice. Thanks to him, we have been given the opportunity to explore new endeavors, think on our feet and develop as engineers. We would also like to thank all the great people at Tactel, specifically Olof Råborg and Jonas Neldeborn for trusting us when we came up with this project. Finally, we would like to acknowledge the teams behind Stanford CoreNLP and scikit-learn, thank you for creating these wonderful libraries.

Contents

Chapter 1
Introduction

1.1 Problem Definition

Tactel is a consultancy company that aims to bring intelligent systems to its customers. As a part of taking the next step in data analysis, we have been tasked to create a smart system for product recommendation. We will investigate the possibility to accomplish this by analysing Amazon reviews and applying several Natural Language Processing (NLP) and Machine Learning (ML) tools.

The research is primarily focused on relation extraction with the aim of extracting reviewers' opinions about product features. E.g "the design" or "battery life" of a certain product. A small UI will be created to demonstrate the work.

The problem can be formulated as an relation extraction task where we try bind a product feature (PF) to a descriptor (DESC). The task is visualized in Table 1.1 .

Input	Extracted Product Feature(s)	Extracted Description(s)
The battery is great	battery	great
The auxiliary battery is short-lived	auxiliary battery	short-lived
I was super impressed by the sound	sound	super impressive
The design is crisp and colorful	design	crisp, colorful
The base and treble is terrible	base, treble	terrible

Table 1.1: The goal of the relation extraction task.

1.2 Purpose

The objective of this thesis is to provide a system that can perform review summarization using relation extraction. The main focus is to explore ways to dynamically extract

product features and their respective descriptors. The dynamic part refers to the process of not needing any manual tools, such as word lists or domain knowledge to perform the extraction. Furthermore, we research different ways to effectively:

- Modify text and perform preprocessing.
- Tokenize text.
- Do semi-automatic annotation to train a classifier.
- Use the proper machine learning algorithm for each purpose.
- Identify relevant machine learning features through feature engineering.
- Perform chunking and linking.
- Associate words with a sentiment.
- Aggregate results.

Combining dependency parsing with ML, another purpose is to improve upon earlier work in the field of Natural Language Processing and develop the field even further within our specific niche.

1.3 Related Work

The use of machine learning to perform statistical prediction is a broadly researched subject which can be encountered in multiple fields. In language processing, it is widely used to measure the accuracy of an algorithmic task. The foundation of our research will mainly focus on using relation extraction to identify product features and their respective descriptors in reviews. This has been attempted in various ways before by, among others, Popescu and Etzioni who used an unsupervised system called OPINE [22], which is built on top of the KnowItAll Web information-extraction system by Etzioni et al [12].

More closely, our endeavor is related to the work of Minqing Hu and Bing Liu [15]. They explicitly aim to identify product features and the corresponding opinions to present a summary for every product. Somewhat similarly to our feature set, Hu and Liu uses Part-of-Speech tags and the Apriori [1] algorithm to identify product features. This is then used as a foundation to identify opinions and thus implicitly also which sentiment the users have towards that product feature. Hu et al. uses WordNet to determine sentiment, while our system relies on its successor; SentiWordNet.[26]

Sentiment analysis or opinion mining, is an active research field that uses more or less the same core strategies for scoring. Each token is given a score (optionally passed through some function) and the sentence or paragraph is scored by averaging or summarizing the total score (depending on the scoring method). Previous research by Dave et al. [10] shows moderate success in classifying sentiment in product features using machine learning methods, and Pang et al. [21] sentiment in entire review corpora.

1.4 Research Questions

Research will be conducted to mainly verify the feasibility of extracting, chunking and linking product features and descriptors as well as conducting a sentiment analysis on the descriptors. Implementing such a system to behave intelligently requires knowledge about multiple aspects of the problem. Hence, our research questions can be summarized as:

- Can we alleviate discrepancies in the analysis by performing intelligent preprocessing?
- Which approaches and algorithms are suitable in our scenario?
- How can we perform intelligent product feature extraction and find relevant descriptors?
- How can product feature & descriptor ambiguation be reduced?

1.5 Limitations

The main focus of the thesis was to make use of NLP and ML for the intended domain. Thus, less focus was targeted towards business analysis and presentation. Although these parts are still present they were kept to a minimum to fit in the current time frame. We felt it important to provide some visual presentation of the results, and this is probably where we could have made the most improvements given a little more time.

It is often assumed that a larger training set yields better results, and that a smaller training set is more prone to overfitting. We use a relatively small training set, and one major limitation is related to the required resources to annotate a large training set with high quality. Given more resources and time, we probably would have outsourced this part of the project to e.g Mechanical Turk[17] or similar. If the data would be of high quality and correctly annotated, this would likely reduce the effect of outliers and erroneous data. In the same sense, high quality review data from difference sources would probably also benefit our system. The Amazon data does contain a large amount of text, but if the training set was extended to include other data sources about the same products, it is possible that the end-to-end result would be improved.

As always when dealing with large sets of data and complex computations, hardware is a limitation. In turn this is correlated with the time limitation since more sophisticated hardware often enable computations to be done in less time.

To further narrow the scope we decided to take the flexible approach and not be dependent on any manually written product feature list. It is possible that an alternative approach with a list would produce higher accuracy and recall, but reducing the flexibility of finding new, to the writer unknown, product features. Naturally, this approach would only work with a very extensive product feature list, or for a narrow product category with unclear certainty on new categories.

When trying out different classifiers we limited ourselves to use the machine learning library Scikit-learn [24]. At the time of writing the thesis, neural networks was not offered

as a classifier-option. If made available, it would have been logical to assess the performance in our system. But with the current situation, we made the decision not to include another machine learning library, just for the sake of neural networks.

1.6 Contributions

We wrote this report in complete collaboration. Very few chapters were written individually.

Chapter 2
Theory

2.1 Natural Language processing

2.1.1 Relation Extraction

A general definition of Relation Extraction and Information Extraction is the processes of extracting structured information from unstructured documents or text. Although the terms are similar, information extraction generally differs from relation extraction due to it being able to extract information about many different relations, whereas relation extraction usually aims to extract a single relationship. Extracting tuples, instead of triplets. Therefore, relation extraction, rather than information extraction, will be the term used to describe our process.[2] [4]

2.1.2 Depedency grammar

Dependency grammar describes various theories for syntactic structuring of a sentence. In dependency grammar there generally exist directed links between lexical items that makes up a sentence such as words and dots etc. The syntactic sentence structure is often determined by relations between a head word and dependant words which modify the head word.

In order to create a structured graph of links between the lexical items, a dependency parser is used. There exist multiple dependency parsers and they generally analyze a combination of word classes and grammatical structure to create a dependency structure. In Figure 2.1 and example of a dependency parse tree is shown. It was generated by the Stanford CoreNLP's dependency parser [7]. The structure is generally comprised of three main components:

- Head - dependant relations.

- Arc labels.
- Part of speech labels.

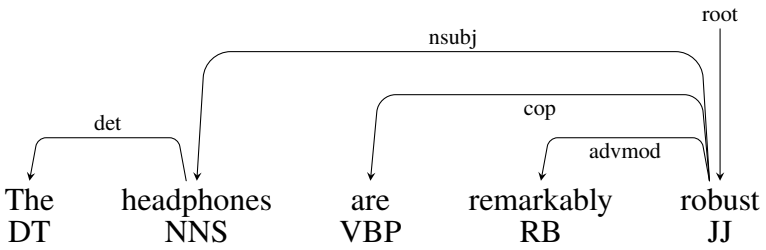

Figure 2.1: A dependency parse tree.

2.1.3 Phrase structure grammar

A constituency parser structures a text by breaking it down into sub phrases. In a constituency parse tree (visualized in Figure 2.2, all the non-leaf nodes are phrase groups and the leaves are the lexical items. The edges between nodes are unlabelled. This is different from a dependency parse tree, where the edges have labels indication the relationship between the lexical items.

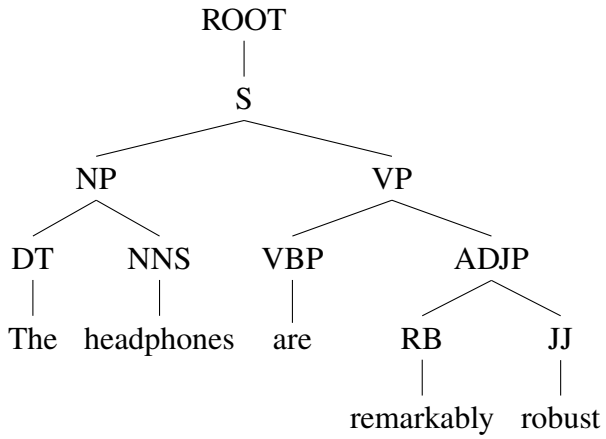

Figure 2.2: A constituency parse tree.

2.1.4 SentiWordNet

SentiWordNet[26] is a polarity dictionary that has positive and negative scores for a large number of nouns, verbs, adverbs and adjectives. It is an extension to WordNet[28] and is currently in version 3.0, but was first made available to the public as SentiWordNet 1.0 in 2006[11]. In order to understand how SentiWordNet was created, we must first briefly discuss WordNet.

WordNet

The development of WordNet started in 1985 at the Cognitive Science Laboratory of Princeton University. WordNet is a database where words are grouped into sets of cognitive synonyms (synsets), i.e the representation is consistent with the way our brain processes the meaning of words. WordNet's power comes from its net-like structure, since there are links between the synsets which forms a hierarchy that can lead you back to one distinct root[19]. This is used in NLP applications to perform word sense disambiguation and similarity computation. The data set consists of English nouns, verbs, adverbs and adjectives and is composed by 155287 unique words.

Improvements

WordNet is powerful, but it doesn't differentiate between positive or negative words, or how strong sentiment each synset expresses. SentiWordNet solves this problem, and also adds the finesse of objectivity scoring. SentiWordNet was created in two steps: the semi-supervised learning step and the random-walk step.[3]

The Semi-supervised learning step

In order to classify the synsets as negative or positive, a manually labelled seed set was created. Then, using the net-like structure of WordNet, these seed sets were expanded to a larger set by traversing connected synsets. The synsets were labelled as positive or negative (binary) and the expansion uses rules to either preserve the label, or flip it. For example by following the "also-see"-relation or "direct antonymy"'-relation. This automatic expansion annotation is quite efficient and results in a larger training set, one for positive synsets, and one for negative.

Two binary classifiers are then trained by using the training sets, which consists of the glosses of the synsets. SentiWord 1.0 used a "bag of words" model by just using the glosses, but in SentiWord 3.0, glosses were linked to the Princeton WordNet Gloss Corpus, where each gloss is a sequence of WordNet synsets, forming a "bag of synsets" model. These two classifiers are used to classify synsets as positive or not positive, and negative or not negative. If a synset $\{Pos, Neg\}$ is classified as $\{1, 1\}$ or $\{0, 0\}$ it is considered to be objective. In total, this builds a ternary classifier that labels $\{Pos, Neg, Obj\}$.

The random-walk step

The random walk algorithm is similar to Google's Pagerank algorithm. It is a graph based method that gives more weight to nodes with a large amount of incoming links. The algorithm is run until convergence on WordNet as a graph. The graph consists of links between glosses in synsets to other synsets. If two synsets share context, it is very likely they share the same sentiment. For example, if the synset *definiens)* occurs in the gloss of another synset *defiendum* they tend to have the same polarity. Using this comparison, a number of negative and positive edges are set between each node. Thus, a node (synset) is classified as either positive or negative on a scale $\{0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0\}$ by iteratively flowing throw the graph.

SentiWordNet 3.0 structure

SentiWordNet 3.0 structure visualized in Table 2.1. The Part of speech (POS) tag of the words (limited to nouns, verbs, adverbs and adjectives) allows the same word to appear multiple times, but with different meanings. For example the words *slow (verb)* and *slow (adjective)* belong to different synsets. The objectivity score can be calculated as

$$Obj = 1 - (PosScore + NegScore).$$

Table 2.1: SentiWordNet 3.0 structure. The WordNet ID has been omitted for clarity. The SynsetTerms column reports the terms, with sense number, belonging to the synset

POS	PosScore	NegScore	SynsetTerms	Gloss
a	0	0.625	wrecked#1	destroyed in an accident; "a wrecked ship"
a	0.25	0	preserved#2	kept intact or in a particular condition
a	0	0.625	conserved#1	protected from harm or loss
a	0.75	0	well-kept#1 maintained#1 kept_up#1	kept in good condition
a	0.5	0	preservable#1	capable of being preserved

2.1.5 Stanford CoreNLP

Stanford CoreNLP is a annotation pipeline framework developed and maintained at Stanford University by the Stanford NLP Group. We use this as our main library to perform the required text processing. It is a suite of core NLP tools that allows you access to a wide range of functionality, such as tokenization, part of speech tagging, named entity recognition (NER), co-referencing, dependency parse trees and more. There are multiple core libraries for text processing, such as NLTK [25] and SpaCy [27] to name a few. This chapter is dedicated towards explaining the functionality in CoreNLP that we use the most; dependency parse trees.

When annotating, CoreNLP provides both the dependency and constituent representation of a sentence, as described in 2.1.2 and 2.1.3. The main difference between Stanford CoreNLP and the other predominant parsers, such as NLTK, Spacy, etc. is which element is being selected as root, and thus the main entry point in the sentence. In CoreNLP, the root of the the sentence is often selected as the adjective of the sentence, instead of the verb. As in figure 2.3 displayed below.

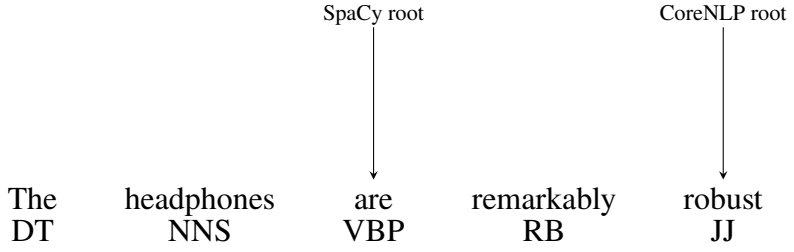

Figure 2.3: Contrast between two parsers when selecting root element.

CoreNLP uses a transition-based parser based on the arc-standard system. The arc-standard consists of a stack, a buffer and a set of dependency arcs, which will storing the resulting parse tree. For example:

$s = [ROOT]$
$b = [$"The","$battery$","is", $good$"$]$
$A = \phi$

The words are then successively popped from the buffer and considered for three types of transitions until the buffer is empty and the stack only contains the ROOT. The three types of transitions are as follows, with s_i being the top ith element on the stack:

1. LEFT-ARC: adds an arc $s_1 \rightarrow s_2$, with the corresponding label to arc set and removes s_2 from the stack. Assuming stack has at least two elements.

2. RIGHT-ARC: adds an arc $s_2 \rightarrow s_1$ with the corresponding label to arc set and removes s_1 from the stack. Assuming stack has at least two elements.

3. SHIFT: moves first element from the buffer to the stack. Assuming buffer has at least one element.

The transition and label selection at each state is done with a neural network classifier. The classifier is trained using an oracle, with an annotated training set containing sentences and their correct parse. The sentences are split into many training examples by the oracle with different transition selection sequences for reaching a correct final parse state. The training examples are in turn used to train the neural network with adaptive gradient descent (AdaGrad) with hidden unit dropout. [7][20]

2.2 Machine Learning

2.2.1 Machine Learning Features

A feature is a characteristic that might be useful when the model is performing predictions. Although an abstract definition, it's much easier to understand given the problem context. For example, given the task of classifying fruits, possible features could be the fruit's weight, form and color. Hence, the classifier will predict all items being round, green and weighing around 150g as apples, while round, orange items weighing around 200 g will be predicted to be oranges.

2.2.2 Support Vector Machines

Support vector machines is a supervised learning algorithm that learns a hyperplane to classify data into classes. It can be used for both classification and regression, but is mostly used in classification problems. This extends to very high dimensions, but for just two classes, the hyperplane can simply be a line of the form $y = mx+b$. The model creates such a hyperplane by maximizing what's called the *margin*. Support vectors are computed from

data points close to the boundaries between two classes. From these vectors, two parallel hyperplanes are chosen to separate the data, and the distance between is called the margin. The optimal hyperplane lies in the middle of the two parallel hyperplanes. SVM performs classification by checking which side of the optimal hyperplane a data point falls in to. SVM also uses a kernel function to map data points to higher dimensions. This is done without affecting the hyperparameters. Using this kernel strategy is incredibly powerful and is one of the reasons why SVM often performs so well even in high dimensions. SVM is very memory efficient since it uses a subset of training points for creating the support vectors. The drawback is that the training time is long for large data sets. [13]

2.2.3 Nearest Neighbors Classification

Nearest neighbors is a broad subject that is the base of many learning methods. It works both unsupervised or supervised and even if the methods vary from algorithm to algorithm, the basic principle is to find the closest (in distance) set of points (from the training set) to the the new, unknown data point. In a supervised nearest neighbors algorithm, the class is predicted from which neighbors it is assigned to. The number of neighbors is often named as K, and different algorithms focus on this metric to perform classification (such as K-nearest neighbor learning) whereas other algorithms use radius-based methods. This simply means that a specific radius limits how many neighbors should be assigned to a new data point. What's interesting about nearest neighbors methods is that they do not create a dedicated, internal model. Instead, whenever it's tasked to perform prediction, a majority vote is held by the nearest neighbors of the new point to compute the classification. Since it makes no assumptions of the training data distribution, it is well suited to deal with irregular decision boundary situations. [9]

2.2.4 Decision Tree

Decision Trees are constructed through dissecting the annotated training set with every feature as a node. The tree is not necessarily binary and thus both features and classes can hold discrete values. The tree is constructed node first and the subsequent tree nodes are selected by their corresponding *information gain*. The information gain defines how well a feature separates the annotated training set. In other words; if the feature information is closely related to the result it will have a high information gain. In the classic example of predicting if the day is optimal for playing tennis; the training data might show that e.g the data for the feature "Forecast" being "Sunny" almost always correlates to "Yes" and "Rainy" almost always correlates to "No" more than say the data of feature "Wind", then that node will be placed closer to the root node. In turn this will result in a more compact thus more optimal tree. Smaller trees are usually favored, especially in the case of ID3, where deep trees have tendencies to overfit the training sets. [18]

2.2.5 RandomForest

To solve the previously mentioned problem with overfitting in Decision Trees, Random Forest (RF) extends the method with modified tree bagging. Instead of creating an extensive tree for the entire training data, RF differs by randomly selecting, with replacement,

several subsets from the annotated set and forms many smaller trees with it. To enhance accuracy and enable generalization error the procedure of tree construction also differs from regular decision trees. Instead of selecting the best node from all features, the best split from a random sample of features is chosen. A prediction can then be made by averaging the results from the trees, or by selecting the majority vote. This reduces variance and usually doesn't increase bias. Making it a better algorithm than decision trees on complex models. [6]

2.2.6 AdaBoost

AdaBoost defined as a boosting method to, by combination and iteration, improve the result of other, weak, learning algorithms. A weak learning algorithm classifies with little more prediction accuracy than chance. The classifier used in this project, as it is the default base estimator by Scikit-learn, is the DecisionTreeClassifier. The algorithm is "boost iterated" on a single data set, meaning that different weights are applied to the training samples in each iteration. The weights are increased on the samples previously classified falsely and thus increasing the likelihood of the samples being predicted correctly in the next round.

As concluded by Robert Schapire [23] it is difficult to comprehend the true advantages and disadvantages of this method, though he concludes that noise can be a noticeable problem, though through enough training data, generalization error can be reduced to close to zero, which is ideal.

2.2.7 Logistic Regression

In contrast to its name, Logistic Regression is linear model for classification. Other names for Logistic Regression is Logit Regression, Maximum-entropy classification or Log-Linear classification. Given a set of training data, Logistic Regression uses a logistic function to classify the probabilities for different classes. This means that it can perform classification with probabilistic outcomes. The logistic function stems from the basic *standard logistic function* which has the form:

$$f(x) = \frac{1}{1 + e^{-x}}$$

For example, if you have 5 years of data from different scientific experiments with labels describing wether the experiments were successful or not, Logistic Regression can be trained with this data, and perform classification predictions on a scale from 0 to 1.0. When you input your current experiment parameters the model might predict 0.87. This means that the experiment has a 87% to be successful. In ordinary regression the model tries to minimize the the sum of squared errors. Logistic Regression however, tries to maximize the likelihood of recognizing the sample values. For classification problems, each class receives a probability which describes to likelihood of that sample belonging to that class. However, in order to produce stable results a very large training set is required. [14]

Chapter 3

Method

3.1 About the Review Data

We got access to the Stanford SNAP data, which is a collection of Amazon reviews (in english) from 1996-2014. It also includes meta data about the products the reviews are for, such us as the price of the product. There are a few different subsets for download, where each subset is a collection of reviews belonging to products in a specific category. We chose electronics because we already have domain knowledge about products regarding that area. For each category, there are dense "K-core" subsets. This means that the reviews have been reduced to just include reviews such that each item has at least K reviews, and each user has reviewed at least K reviews. We use 5-core in our thesis. Since we're interested in aggregating our results, having products with just 1 single review doesn't contribute much to the end result. Also, requiring that each user has written at least 5 reviews has the effect of getting rid of a lot of fake reviews. In total, 1.7 million reviews for products in the electronics category were downloaded and used for our task.

As for the reviews themselves, the number of reviews per product varies a lot, from 5 up to a couple thousands. The length of the reviews varies as well, since some people like to keep themselves short, or simply didn't have the time or effort to write a long review. The biggest challenge in when it comes to the review data is that the text itself is comprised of paragraphs with poor grammatical structure. It is important to bring this up, because it affects every part of a system, especially statistical parsers, that are trained on properly written text. This complicated things for us, and in order to alleviate some of the damage caused by this, we performed some preprocessing of the text, which we will talk about in the next chapter.

3.2 Preprocessing

In order to perform consistent computations without having to parse a text file every time, we extracted the review texts and meta data and put them in our MongoDB database. The Stanford Snap data comes in a very convenient JSON format, which we restructured to fit our needs, and saved to our database. This also allowed us to analyze and understand the data set in a simpler way by querying the information using the Mongo shell. Using this convenient data structure, we could experiment and figure out what we needed to correct in the reviews in order to achieve better results. The preprocessing steps will be described in this chapter.

3.2.1 Correcting reviews

Unfortunately, reviews written by humans are seldom linguistically correct. These faults range from spelling to grammar and can cause the tokenization of text to produce erroneous results. In order to help the tokenizer, we performed some sanitation and correction of the reviews before feeding it to the tokenizer. In order to reduce the complexity of dependency parse trees we broke up sentences to smaller sentences whenever possible. Also, we found a few discrepancies that caused the tokenization to produce weird results. Table 3.1 shows a list of problems and their respective solutions.

Table 3.1: Corrections performed

Problem	Example	Solution
No space after dot causes the tokenizer to treat the two surrounding words as one.	The sound is great.I like them	Insert a space after dot where this is the case.
Multiple dash characters does not produce a new sentence	Don't get a unit that simply insulates the sound — you might as well use earplugs.	Replace two or more dashes with a dot.
"..." does not produce a new sentence	My wife thought it was heavy... when she picked it up.	Replace multiple dots with a single dot.

3.2.2 Spell Correction

Reviews written by humans are seldom linguistically correct, especially on the Internet. Thus, we implemented a simple heuristic to spell correction, in order to avoid missing some product features and descriptors just because the were misspelled. We used PyEnchant, which is based on the Enchant library. The algorithm is based on edit distance and statistics and we use it to auto correct some misspelled words. During the preprocessing

step, we analyze each word with the spell checker, checks if it flags it as a spelling error, and use the first (most relevant) suggested word. We do not allow the spell checker to change words that contain numbers or uppercase letters, since these words could potentially be product names or specs, such as "iPhone" or "20M".

3.3 Relation Extraction

Our type of relation extraction is a difficult task, primarily because the semantic structure of sentences vary depending on the multiple possible ways to phrase sentences containing descriptors for product features. For example, the sentence "The sound is impressive" and "I was impressed by the sound" has major dependency parse tree differences. Additionally, what makes relation extraction of review texts produced by humans is the fact that a many of the reviews has poor structure or grammar, which complicates dependency parsing. To provide an overlook, these are the steps we took to perform our tuple extraction:

- Annotation - Construction of a training set using semi-automatic annotation.
- Feature selection - Using NLP tools to induce features.
- Chunking - A separate classification step to chunk for example groups of descriptors
- Linking - Manual rules to provide correct links between product features and descriptors, so that descriptors doesn't get attached to a product featured that was not linguistically intended in the original text.
- Testing - Evaluation of the extraction process using cross validation and a separate test set.

3.3.1 Annotation

As the method involves a supervised machine learning it implicitly also involves a manually annotated training set and with it an implicit or explicit test set. The end result and accuracy of the classifiers are much dependent on the training set's quality and size. To achieve the optimal size, it is therefore necessary to perform a measurement of the learning curve. While keeping the explicit test set static the training set was dynamically changed with more annotations, measuring both the learning curve from the explicit test set and cross validating the training set.

Requiring a fairly big test and training set the annotation was done semi-automatically, to simplify the processes for the annotator. Thus, a set of manual rules was used together with the data pre-processing before presenting the words to be annotated to the annotator through a minimal user interface in the terminal. Through the terminal the user could enter if the rules had annotated the words correctly or not. If the words were classified incorrectly the UI enabled the annotator to type the indexes of all product features followed by the indexes of all the descriptors. The system then annotated a text file automatically in a format readable by our parser. With the word-to-be-classified occupying the first column, the second column was thusly filled with the word's first label with 'DESC', 'PF' or

'-'. This allowed training of our main classifier. The second classifier which was implemented to predict product feature/descriptor groups, also known as the chunker, needed additional information. Manual additions was thus inserted directly into the annotated text file, adding a third column to the file with the labels 'b.PF', 'i.PF', 'b.DESC' and 'i.DESC'. An extraction of the annotation set can be seen in Figure 3.2.

Table 3.2: Extraction of annotation file

Word	PF/DESC	Group
The	-	-
sound	PF	b.PF
is	-	-
pretty	DESC	b.DESC
good	DESC	i.DESC
.	-	-
These	PF	b.PF
are	-	-
perfect	DESC	b.DESC
with	-	-
really	DESC	b.DESC
good	DESC	i.DESC
sound	PF	b.PF
.	-	-
.	.	.
.	.	.

Review data from the retail site Amazon.com was used to assemble the sentences to be annotated. The reviews were read from a JSON text file aggregated and formatted by McAuley et al. for their quest in "Inferring networks of substitutable and complementary products"[16]. The data was collected from May 1996 - July 2014. The data sets available, referred to as SNAP sets, is organized by category. To limit the scope of the thesis, the 'Electronics' set was used. This set was also stripped of any duplicates and only contains items with 5 reviews or more. Consisting of 1.7 million reviews accounting for a total of 1.478 Gigabyte.

To train our classifier, only sentences serving our objective of mining sentiment from product features were selected for inspection, and thus removing the noise from our training set. Meaning only sentences containing pre-defined product features were selected, and with no or very little ambiguity concerning the product feature and the sentiment. Finally the sentences was also corrected as per the pre-processing.

Realizing the possibility of potentially confusing the classifier with inconsistent annotations a set of of manual checks was written as seen in the list below, specified as explicit as possible. The primary purpose of the check list was to help the human annotator and was thus not included in the internal annotation tool.

Many iterations were made to perfect the training set and the check list. In the end, the decision was made to focus on being consistent with labelling sentences uniformly regard-

ing their sentences structure, while assuming grammatically correct English, realizing that everything can not be filtered in the first classification.

Annotation check list

1. Select every possible product feature (PF)
   (a) Typically a noun
   (b) Can be an actual PF or something referring to the product itself e.g product name, 'it', 'they', 'these'
   (c) Has to be described
   (d) Never a determiner, e.g 'the'
   (e) Never 'I'
2. Select every possible descriptor (DESC)
   (a) Typically an adjective
   (b) Can be a determiner, e.g 'the' if neighboring to other DESCs
   (c) Can be a verb if other than lemma of be
   (d) Can be preposition 'in', 'for', 'than' etc, if directly assisting a DESC and not a subordinate clause

The semi-automatic annotation also served a secondary purpose. While annotating, the manual rules were updated as problems arose with the current guidelines. Thus raising the accuracy of the automatic part of the annotation and requiring less manual intervention, but more importantly it simultaneously provided insight about how to design the machine learning features.

3.3.2 Machine Learning Features

Identifying the most effective features to use for prediction is a complicated task. One must use domain knowledge and experiment with data points to figure out patterns and the most effective feature composition. This is called feature engineering and albeit often overlooked in research papers, it is a fundamental process in order to make machine learning algorithms work effectively and accurately. Choosing the right features often yield better results and a more flexible model when it is time to expand your machine learning task. An overview of the iterative feature engineering process can be seen Figure 3.1. In summation we:

- Investigated the domain and analyzed what patterns might exist
- Identified how we could represent that pattern as a feature
- Created algorithms which yielded a viable feature representation
- Checked how well the new features work with our model

- Experimented with different feature combinations to see what gave the best result
- Evaluated the recently created features and went back to further investigation

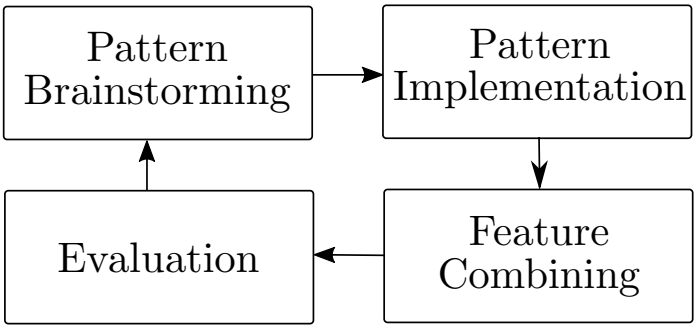

Figure 3.1: The feature engineering process

We started by overseeing the domain and all the raw data available. Since it's text-based, creating a phrase structure tree and a dependency parse tree, as seen in Figure 3.2, are the main components when we analyze possible feature candidates. We started with just one feature, the word's POS, see appendix: Section 6. We then included the word string itself, (e.g "Battery") but removed this feature due to the size of the training set. Since individual words occur scarcely, the result thus biases the prediction towards the word itself and away from the syntactic structure, and as follows binding the system to the training set itself, removing much of the dynamism. By analyzing the dependency parse tree further, extracting the linking of the sentence structure proved to be essential. This in combination with POS-tag proved to be very useful. Yielding a very dynamic extraction process, not minding which product or product category is under investigation. The procedure of extracting the link features are shown in algorithm 1.

We also experimented with a POS-tag window. This allows the classifier to more confidently predict groups of words that belong together.

We also experimented with features from the constituency parse tree but without any improvement.

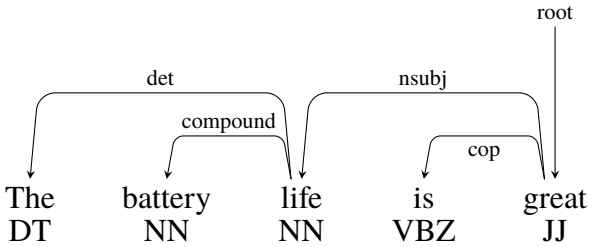

Figure 3.2: A dependency parse tree.

For the word "life" in the sentence "The battery life is great." the following vector in Table 3.3 would thusly be generated to be used by the main classifier:

input : Word; A tokenized word containing dependency parse tree information
Sentence; A list of words
output : FeatureList; A list of features for this word

Function *FindLinkFeatures (Word, Sentence)*
```
for each outgoing Edge from Word do
    F ← "out_" + Edge
    Add F to FeatureList
end
for each OtherWord in Sentence do
    for each outgoing Edge from OtherWord do
        if Edge's target is Word then
            F ← "in_" + Edge
            Add F to FeatureList
        end
    end
end
return FeatureList
```

Algorithm 1: Extracting outgoing and incoming links to use as features

pos :	NN
in_nsubj :	True
out_det :	True
out_compound :	True
pos_before_1 :	NN
pos_before_2 :	DT
pos_after_1 :	VBZ
pos_after_2 :	JJ

Table 3.3: An example how a feature vector could look

Note that the model will set all other features (that are implicitly present in the feature space due to other words) to false.

3.3.3 Training the model

Once we had an annotation set, we parsed the annotation file and separated the data into a list of the original words, and a list of labels. We used our feature extraction pipeline to derive features for each word into a list of feature vectors. This list of feature vectors and the list of labels were fed to a machine learning classifier for training. In order to see what produced the best results, we experimented with a few different machine learning algorithms, namely:

- Nearest Neighbors
- SVM
- Decision Tree

- Random Forest
- AdaBoost
- LogisticRegression

For each classifier we performed an extensive grid search to search for optimal estimator parameters. To evaluate each estimator, we performed 100 iterations of cross-validation with 10 random splits with about equal amounts of labels in each split (StratifiedKFold). The results (accuracy, recall and FScore) were calculated as mean values of the results from the 100 iterations. As we tried to improve the classifier, we continuously studied the confusion matrix to see what our main problems were in terms of false positives or negatives. We also plotted the learning curve for the model to see if it's trainable at all, and whether variance or biasing was a problem. Iteratively improving the model eventually resulted in a relatively accurate classifier that we could now use to confidently predict product features and descriptors in any given text.

3.3.4 Initial Product Feature and Description Extraction

With the model trained and everything in place, we can begin to extract relevant product features and descriptors from a review. We performed the relation extraction using the following steps:

1. pre-process the review text
2. Tokenize the text by feeding it to the Stanford CoreNLP Java server.
3. For each sentence, build feature vectors for each word using the feature extraction pipeline.
4. Classify each word as "-", "PF" or "DESC" using the trained model.

This gives us a list of all product features and descriptors for a sentence. Moreover, there needs to be a distinction between words that are classified to the same class. For example, lets say that "Battery" and "Life" were classified as product features (PF). In order to link descriptors to these words, we need to figure out whether the descriptors relates to one of the words, or to both words as a group. Therefore, we perform further processing to be able to group words together, and to figure out which links should be created to preserve the intended targets for the descriptors. All in all, we call these steps chunking and linking, and we add these intermediate steps after performing the basic extraction. We will talk more about these in the next couple of chapters.

3.3.5 Chunking

In order to be able to group words together, we use a chunking classifier. Assume the first prediction step (described in the previous chapter) performs the following classification:

The :	-
battery :	PF
life :	PF
is :	-
super :	DESC
good :	DESC

The chunker's main task is to group "battery" and "life" together, as well as "super" and "good", forming two separate entities. We trained the chunker classifier using the same training set as for the first classifier, but with additional labels added following the *begin inside outside* notation BIO (see glossary). Moreover, we used the same features as the first classifier, with the addition of the predicted label. This is obviously a very prominent feature, which together with the link features proved to be very effective. We also use the POS-window features, but this time with a size of 6 instead of 4. Additionally, we used a window of size 6 for the predicted labels from the first classifier. This helps the model with prediction since it's very likely that the word to be predicted belongs to a group if it has previously been predicted as a product feature and it's neighbours are also in the same class.

The prediction is almost perfect (10-fold cross-validation, see results), which means that it could probably have been done in the same step as the first classification, using just one model. The reason we choose not to do this was because of the highly iterative approach we've chosen. In a more flexible, modular system such as this, we were able to identify problems more easily and implement improvements add a quicker rate.

3.3.6 Linking

With the words classified more specifically as chunks of product features and descriptors, they were linked together using a manual written procedure, consisting of mainly checking if there existed a syntactic link in the dependency parse tree between the two chunks. More explicitly, if any of out the outgoing links from any of the words in the product feature chunk pointed on any of the words in the description chunk, or vice versa, a link was said to be existing between the two. A tuple was thus created.

The linking procedure is not entirely unproblematic in the complexity of a written sentence, even provided that the previous classifications has been done correctly. To be able to catch as many true tuples as possible, we opened up for the possibility of a product feature chunk being linked to two descriptors in the same sentence, as well as vice versa. Demonstrated in the simple examples: "The battery is good and solid" and "The battery and sound is great".

3.4 Finalization

3.4.1 Aggregation

Several steps were taken to structure the data in a usable way after extracting the raw information together with adding additional meta data to the results. The final stage of the

system was thus to aggregate the data into something interpretable by the User Interface.

The tuples was thus expanded into a larger data structure containing meta data about the relation. The final data structure was product feature-first oriented, meaning that the data proceeds from the product feature chunk-string.

Apart from containing the chunks describing the product feature, the data structure also contains the sentence containing both chunks, as well as how often the product feature has been described with the exact same string in the extent of the entire corpus and which sentiment the descriptor carries. Additional meta data consists of a potential ASIN-string, how often the product feature has been mentioned in the corpus and its total registered sentiment.

3.4.2 Product Feature & Descriptor Disambiguation

To attain the most comprehensive and perspicuous overview of the results when amassing many reviews for single product, the extracted information was linked together. As the results were product feature-oriented, the main approach was to link together different variations of the product feature string. The product features are firstly sorted by the size of their string, this assures that the main product features to be displayed in the end will be the shortest of the variations. The first step of linking is then to reduce the product feature to its lemma-form. If the product feature is a single word, it's directly chained with its lemma-form. If the product feature consists of two or more words, e.g "battery life" it's first stripped of any appendage that are doubtlessly remains from the classification such as "these", "its" but does not helpfully describe a product feature. The product feature is then linked together with another product feature if any part of the remaining word's sequences matches with a previously inserted pf. Meaning that "battery life" will be linked with "battery", as will "extra battery life". If a conflict emerges when linking sequences, for example in the case of linking "front camera" with "front" or with "camera", the product feature is linked with the product feature with most relevance. When the two two product features are linked, the accumulated sentiment score and relevance points will be stored in the PF to be displayed. Thus, even if a product feature is mentioned in many different ways, the resulting product feature will percolate up, to be displayed further up in the results in an extended UI.

To also chain the descriptors inside the product features one could apply a similar method. However, being a lesser problem than product feature linking and which further introduces programmatic design questions to preserving the transparency in the overview of the results, we decided against implementing such a feature.

3.4.3 Scoring the sentiment with SentiWordNet

To use SentiWordNet in Python, the text file containing the scoring of the words had to be translated into a python dictionary. The SentiWordNet file was constructed as following

- *POS* the lexical class, e.g. verb, noun, adjective, etc.
- *ID*
- *PosScore*

- *NegScore*

- *SynsetTerms* all words relating to that score, indexed with a "Sense Number" how closely they relate. Meaning there might be a more accurate score for the word used in a different context.

To create the dictionary, the text file containing SentiWordNet was scanned line by line and each word put into the dictionary with their corresponding weighted average score. The weighted average score was calculated as shown in the algorithm below.

$$A(w) = \frac{\sum_{i=1}^{n} \frac{s(w_i)}{i}}{\sum_{i=1}^{n} \frac{1}{i}} \tag{3.1}$$

Where i is the sense number and $s(w_i)$ is the polarity score of w_i

To reach a conclusion about the product features, all descriptor-chunks were evaluated by adding together the sentiment score, stored in the newly created dictionary, corresponding to each individual word. The exception was when the chunk contained 'not', then the sentiment score was flipped for that chunk. All descriptors were then added together to reach a total score for the product feature, in the provided corpus.

3.4.4 User Interface

To present the end-to-end classification of our the project a user interface was built using a combination of the open sourced annotation tool *brat rapid annotation tool* (brat)[5] and the CSS-framework Bootstrap. To simplify the processes of setting up the user interface a previously made brat template was used, published as open source by the Stanford CoreNLP team on Github[8] and contributed to by several Github users. The template was modified to show relations between the selected chunks as well as indicative coloring showing if the description was of positive or negative sentiment exemplified in the Figure 3.3. To show a summarized view of a products features, a table is used to display the data effectively. As shown in the example in Figure 3.4

Figure 3.3: The User Interface for displaying relations

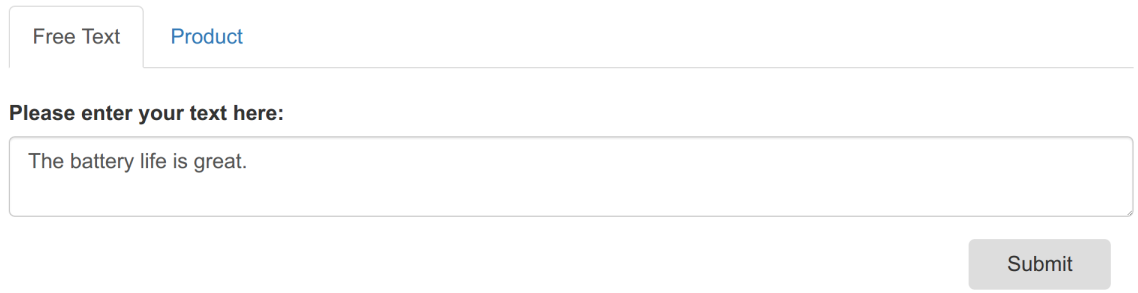

Figure 3.4: The User Interface summarizing a product

3.5 Evaluation

3.5.1 Evaluation

To answer the research questions fully, four sets of evaluations were made. To evaluate the best classifier the following evaluations were made:

1. Cross validation
2. End-to-end evaluation
3. Learning curve
4. Confusion matrix

As mentioned in Section 3.3.3, the different classifiers were trained and with grid searched under cross validation under the same conditions. With the parameters in place, the classifiers were evaluated both in respect to the training set with cross validation, but also with a end to end evaluation.

3.5.2 End-to-End Evaluation

As the system was built with the end user in mind, we found it important to evaluate how well it satisfies the purpose of extracting, and delivering, not only from a classifier's perspective but rather through the whole procedure, from pre-processing to the final aggregation.

To evaluate the end-to-end result we created a test-set of product reviews taken straight from Amazon the 9th of August 2016. The top six reviews, as classified by Amazon, were taken from five different categories, Adding up to a total number of 30 reviews to be tested. The reviews were annotated sentence by sentence and the humanly interpretable result was extracted manually. As the system never claims to solve the issue with translating implicit product features and descriptors the test-set will only be annotated with the words available in the sentence. For example, from the sentence "Its simply beautiful." the system is expected to try and and extract "It" as "simply beautiful", rather than realizing that beautiful probably is related to the design and more explicitly label "Design" as "simply beautiful".

The method of scoring the end-to-end evaluation is presented as follows. Going through every sentence in the test-set a tuple is created for every relation, the first field containing the PF-result for that relation and the other the DESC-result. The results are the fractions of the number of correctly extracted PF/DESC part over the anticipated number of PF/DESC part in that relation.

The scoring of the result is then displayed as:

- If all the PF and DESC parts are extracted, score as totally correcy
- If at least one word in the PF and one word in the DESC, score as partly correct
- Otherwise score the relation as missed.

This produces an interpreted recall result. Although the the overall accuracy of the system is also presented in its current state it is not as relevant, given the purpose of our implementation. This noise that would be reviewed in such an analysis is expected to be filtered away when classifying bigger amounts of data, and thus never reach the potential end-user.

3.5.3 Evaluating the Winning Classifier

Apart from Cross validation and end-to-end evaluation, a confusion matrix and learning curve was constructed for the winning classifier, to justify future improvements to the system.

The confusion matrix is constructed for both the main classifier and the chunking classifier. The confusion matrix is a 3x3 and 5x5 matrix, respectively, where each field represents the number of predicted entities labelled as i but classified as j.

The learning curve was constructed with the StratifiedKFold cross-validation generator, splitting the original training-set 10 times into training and test data-sets. The results from the subsets will be averaged and plotted, showing how the classifier training score and the cross-validation score relate.

Chapter 4

Results

4.1 Classifier Comparison

The relation extraction process described in chapter 3.3 uses two different classifiers. As such, we were able to optimize which features to use as well as which model and hyperparameters we should use for each classifier to achieve the best FScore. In Table 4.1 and 4.2 accuracy, recall, Fscore and variance are in percentages.The Fscore was computed using scikit-learn's 'weighted' option. The results below shows averaged cross-validation results (100 iterations, 10-fold) where the best two scores for each column are in bold.

Estimator	Hyperparameters	Acc	Recall	Fscore	Variance
AdaBoost	n_estimators = 50	75,97	75,96	75,84	7,09
Decision Tree	max_depth = 6	76,47	76,45	76,37	7,00
KNeighborsClassifier	n_neighbors = 10	79,28	79,27	79,36	6,79
Logistic Regression	C = 2.3, solver = "newton-cg"	**80,59**	**80,58**	80,45	**6,75**
Random Forest	n_estimators = 75	80,54	80,50	**80,50**	6,83
SVM	C=0.35, kernel = 'linear'	**80,78**	**80,78**	**80,72**	**6,62**

Table 4.1: Results for the first classifier. SVM outperforms the other classifiers.

Estimator	Hyperparameters	Acc	Recall	Fscore	Variance
AdaBoost	n_estimators = 50	91,38	91,38	89,92	8,99
KNeighborsClassifier	n_neighbors = 10	91,30	91,30	90,87	4,45
Logistic Regression	C = 2.3, solver = "newton-cg"	95,77	95,77	95,59	3,22
Random Forest	n_estimators = 75	96,60	96,56	96,50	2,96
SVM	C=0.35, kernel = 'linear'	**98,16**	**98,16**	**98,14**	**2,07**
Decision Tree	max_depth = 6	**98,26**	**98,25**	**98,24**	**2,08**

Table 4.2: Results for the chunking classifier. Interestingly, Decision Trees proved better than SVM for this task.

4.2 Best Classifier

In this section, we look at the best classifier for both experiments (main classification and chunking). This is meant to illustrate which problems the classifiers run into when running on our data set. Figure 4.1 and 4.2 shows the learning curve for the classifiers.

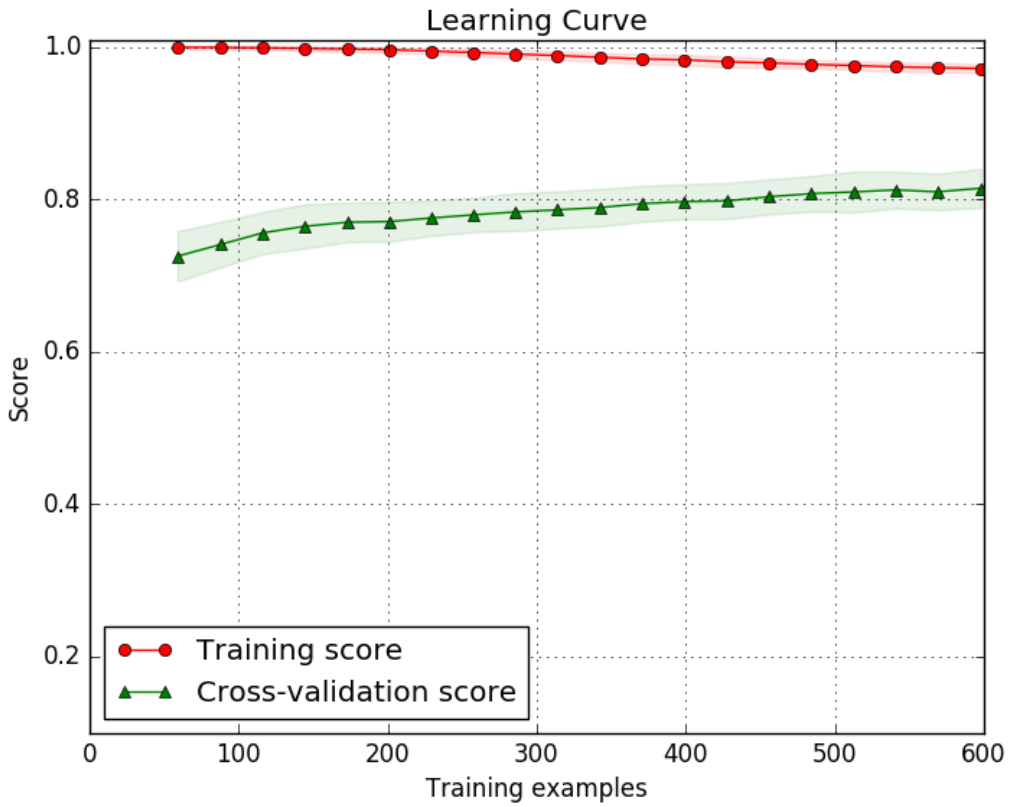

Figure 4.1: Learning curve for the main classifier

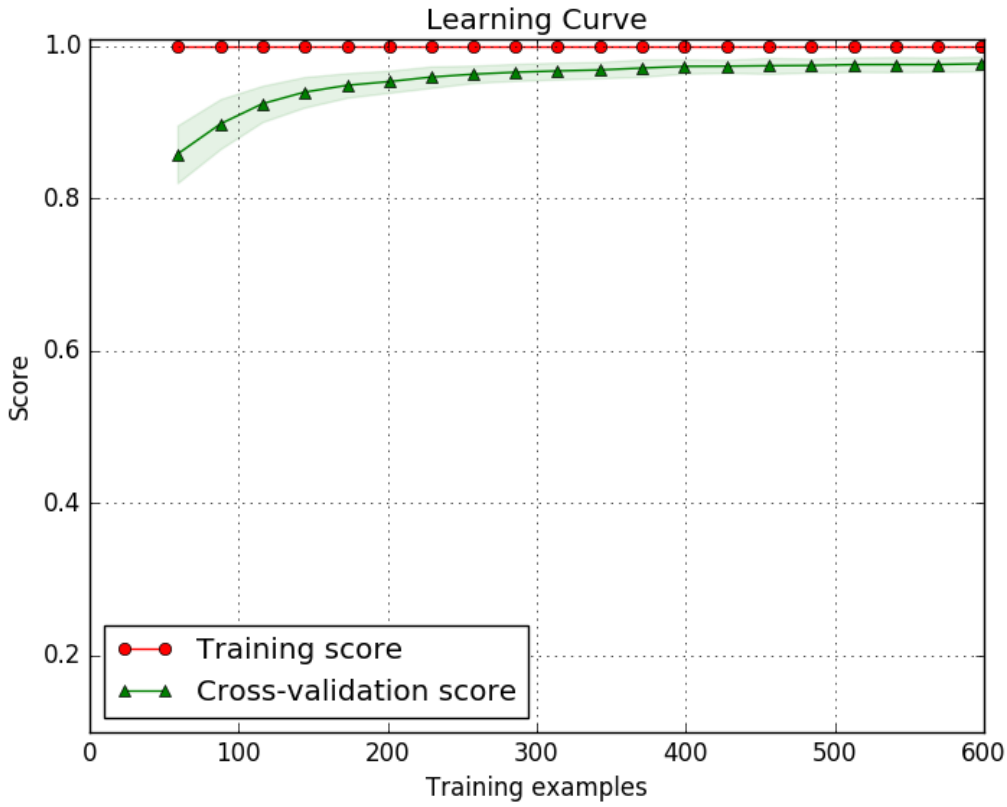

Figure 4.2: Learning curve for the chunking classifier

While learning curve is a way of getting an understanding how the classifier performs given an increase in training samples, a confusion matrix visualizes what classifications are frequent, so that improvements can be implemented to correct miss-classified samples. In Figure 4.3 we can see the confusion matrix for the main classification.

	-	PF	DESC
-	592	28	78
PF	29	133	15
DESC	85	12	238

Figure 4.3: Confusion matrix for the main classification. We can see that the most common mixup is to differentiate between "-" and "DESC"

	-	b.PF	i.PF	b.DESC	i.DESC
-	698	1	0	1	0
b.PF	2	129	6	0	0
i.PF	0	9	30	0	0
b.DESC	0	0	0	151	2
i.DESC	0	0	0	3	178

Figure 4.4: Confusion matrix for the chunking classification. There are very few mixups.

4.3 End-To-End

The end-to-end evaluation process is described in Section 4.3. The end-to-end test set is comprised of 30 reviews from 5 different categories. In the review text, we selected 306 relations that we found relevant. In total, there are 451 PFs and 703 descriptors to be found. We fed the test set through our system to get a sense of how it performs on real data. Our main objective was to get a high recall. The more relations we found, the better. The results (recall only) can be found in Table 4.3.

Fully correct	Almost correct	Missed	Total
105	62	146	313
33.55%	19.81%	46.64%	100%
53.36%		46.64%	100%

Table 4.3: End-to-End evaluation recall results. 53.36% of the relations were extracted either fully correctly or almost correctly.

However, in order to give the full story of how our system performs, we provide some interesting statistics of the results. Table 4.4 shows some statistics of the results for the end-to-end evaluation. We treat noise as the situation where our system suggests a relation, but there was in fact no relation to extract according to the test set.

Metric	Relations
Fully correct	105
Almost correct	62
Missed	146
Noise	534

Table 4.4: End-to-End evaluation statistics. Our system suggests 534 relations that were not supposed to be found, indicating that noise is a substantial problem.

Table 4.5 shows the accuracy, recall and Fscore. The results were computed by using a relaxed setting, meaning that we treat a relation as correct if it was found to be fully correct, or almost correct (as explained above).

Accuracy	Recall	Fscore
23.82%	53.35%	32.94%

Table 4.5: End-to-End evaluation results including accuracy and Fscore (relaxed)

4.4 Pre-processing

The end-to-end evaluation results with pre-processing can be seen in Figure 4.3 above, to be differed against the results without pre-processing, seen below in Figure 4.6.

Fully correct	Almost correct	Missed	Total
94	75	144	313
30.03%	23.96%	46.00%	100%
53.99%		46.00%	100%

Table 4.6: End-to-End evaluation recall results without spell correction. 53.99% of the relations were extracted either fully correctly or almost correctly.

The major numerical differences when applying pre-processing are thus as follows: Fully correct: +3.52 %. Almost correct: -4.15 %. Total recall: -0.63 %.

Chapter 5

Discussion

As we started this thesis, we defined that the system would consist of six separate parts in total. The first of which is considered pre-processing, which turned out to be crucial to make sense of some of the sentences that were fed to the system. The following three making up the foundation of the thesis and the system. With the final two, aggregation and presentation being post-processing. Realizing that there were three natural steps in the foundation, we also made the decision to separate them programmatically. Resulting in two classifiers and a set of manual methods. With this approach we could feature engineer with each classifier and insert manual rules as we found appropriate, creating a modular design. We found that not all features fitting the first classifier fit the second one, and vice versa. The linking stage is manual to remain as flexible, simplified and transparent as possible. This turned out to be a difficult task manually and if more time and thought would have gone into the linking process, it is possible that would have boosted the end-to-end results.

5.1 Data Selection

In the beginning, we wanted to analyze every product on Amazon. Due to time limitations we had to pivot from that idea and instead focus on something a bit smaller. Using the full data set would have included over 34 million reviews, which would have slowed us down too much. Just working on the electronics category, we just had to deal with around 1.7 million reviews. This simplified things, since handling a smaller portion of the data is a lot easier and more efficient to work with. Furthermore, electronics is an area we know well. Having some kind of domain knowledge is important when it comes to continuously analyzing and improving the system.

5.2 The Annotation Process

Even if annotation is semi-automatic, one of the main challenges is to keep annotating in a consistent and stable manner. Every sentence is manually checked by a human, meaning that it may have been annotated in a faulty way. It is especially hard to annotate when the review text itself is poorly written or has bad grammatical structure. Should you include such a sentence? Probably not, but if the actual domain consists of review text like this, should you just ignore it? These are some of the issues we've been trying to make sense of. We ended up building the annotation set from real review data, but with minor justifications so that sentences would be grammatically correct, at least to a certain point. Another problem with annotation is when to stop adding words to a certain chunk. For example, given the sentence: *The sound in the back is not very good for parties*, what should be labelled as PF, and what should be descriptors? Certainly it's important to include that it is the *sound in the back* that is not very good. But should you include the part about the parties? Well, maybe. This is definitely interesting information. The problem that arises with labelling too freely is that the classifier suddenly starts labelling a majority of the words as either PFs or descriptors. This is fine until a certain point, because this implicates a lot of relations that can be treated as noise. However, when the noise because the main part of your extracted relations, you know you have a problem. Therefore, we decided to not label descriptors following words such as *for, of, in* etc. In the example, we would label the descriptor as *not very good*. The example highlights some of the decisions you need to make when performing manual annotation, with the point being that there may be some examples across the whole training set that were not annotated in the same manner, especially when the annotation is made by two different persons. Either way, using manual annotation to create a training set proved to be very powerful because it allowed us to define certain examples and then explicitly state what piece of information is valuable to extract, and then let the classifiers decide how it wants to perform the separation. We will talk more about this in the next chapter, which relates to the classification process.

5.3 Pre-processing

Not particularly suprising, the pre-processing part only changed the end-to-end results by a few percentage. Probably because not a excessive amount of time was spent on this matter and a lot of grammatical and problems remains along with many misspelled words. What is interesting is *how* the result differed. We can see that the overall recall was almost the same with pre-processing but that the amount of 'fully correct' relations were increased, meaning that the amount of 'almost correct' also decreased. The reason for this is open for interpretation but a logical conclusion is that our pre-processing actually helps the system with ambiguous sentences, to a draw fully correct conclusion. But also that it can not help the system make correct interpretations on 'lost-causes'.

5.4 The Machine Learning Process

5.4.1 Features

The feature selection process is crucial in order to provide the models with the information they need to make good predictions. Extracting the features algorithmically from the data points was neither time consuming or difficult in our scenario. What's hard, however, is to know which features should be included. You can't afford to miss any important patterns in the data, so doing feature selection thoroughly is essential. We did this very experimentally, and in the end, we could probably have implemented a few more features if we had a larger data set and more time. For example, we do not include the word as a feature. This stems mainly from the fact that our data site is too small (1226 samples) in order to perform consistently with these features. The model would probably be overfitted since the number of features would exceed the number of training samples.

5.4.2 Best Classifier

For chunking, decision trees proved to be the best classifier. For the first classification step, SVM produced the best results. So why did not the other classifiers come up to the same accuracy and Fscore? Is it obvious that SVM should yield the best results, given our data set? These are hard questions, but we will try to answer them in this chapter. First and foremost, learning algorithms make very different assumptions about the data. They converge with different speeds, meaning that they try to minimize different cost functions, and this cost function converges with different rate for different algorithms. The algorithm that converges good enough to its error rate and also make consistent assumptions about the data generally performs well.

For example, is the data spread into multiple small clusters? Well then KNeighbors would probably perform well since it makes the assumption about the data that it is clustered in small neighborhoods, whereas SVM separates the data with high-dimensional hyperplanes, meaning that some clusters might generalize away from it's original class. However, SVM is very popular in text classification because it is extremely good at working in high-dimensional spaces. Since we don't have a lot of training instances, the algorithms that perform best are the ones that are able to adapt to the small training set and the feature space. This is especially true for the chunking classifier, where one feature, namely if the word has been classified as a PF or descriptor before, is more prominent and important than all the others. Decision Trees adapt well to this, since the major decision can be done early in tree. SVM adapts well as well, since the high-dimension hyperplane can be chosen to make distinct boundaries where this feature is present. They perform very close in terms of Fscore (98.14% vs 98.24%). The worst classifier over both classification processes is AdaBoost in the chunking classification step with a mediocre 89.92% Fscore.

The variance in AdaBoost is high at 8.99%. A high variance indicates that it causes a lot of overfitting; it models the random noise in the data. In contrast, if a model underfits the traning data, it fails to capture important patterns or regularities in data, causing a lower accuracy and recall.

5.4.3 Confusion Matrix

A confusion matrix reveals what classification errors were made. This is a great tool to use for debugging. As we can see in Figure 4.3 and Figure 4.4 the highest numbers appear on the diagonal. This means that they were correctly classified. For example, the cell in the middle in 4.3 shows how many of the instances that should have classified as PFs (row), were actually classified as PFs (column). Similarly, the cell in the bottom left corner shows how many instances that were labelled to be classified as "DESC", but were actually classified as "-". This appears to be our main problem, since this is our highest value that doesn't lie on the diagonal. Conversely, the cell in the top right corner indicates the opposite, i.e the instance should have been classified as "-", but were in reality classified as "DESC". If further improvements should be implemented, it should focus on sorting this out.

The chunking classifier has an easy job, because it already knows if the instance was classified as a PF or as a DESC in the first classification step. This can be visualized in Figure 4.4. Very few values lie outside the diagonal, meaning that it classified almost all instances correctly.

5.4.4 Learning Curve

Looking at the learning curves gives us insight about the bias and variance in our model. When we increase the size of the training set, we can see that the cross validation score increases while the training score doesn't drop significantly, for both classifiers. As seen in other machine learning experiments, the convergence of the two curves is often a good sign. Assuming that they converge at a low error score. Training error is determined when applying the model to the same data from which we trained. Explaining why training error usually increases when increasing the size of the training set and thus the complexity of the model. Maintaining a low training error, as in our case, therefore means that our model fits the training data reasonably well, that the bias is low. In other words it indicates that we have no major issues concerning underfitting. The validation score curve in our diagrams implies low variance, that the model is generalizing well and hence combats overfitting with increasingly better predictions on new data, the more training samples involved. Thus, these curves indicate very satisfying results and shows that the model overall is reasonable. With the validation score curve leveling out we can also assume that more annotated training data will only benefit the model slightly in terms of cross-validation. More training data would benefit the end-to-end evaulation, since more training examples are seen.

5.5 End-to-End Evaluation

The end-to-end evaluation could be seen as the true evaluation of the complete system. With the purpose of determining what information a raw input text would give the end user. Though, this is not entirely true looking at the purpose of the project itself. The project aims to analyze the possibility of creating a system that can perform review summarization. Classifying is, although the most prominent one, only a part of that goal. The

summarization goal is harder to evaluate and would probably be more relevantly evaluated through a user perspective. To proceed with such an evaluation, one would first need to summarize a number of reviews. It is a likely feature that our system provides more relevant results to the user with an increased number of analyzed reviews. It is reasonable to assume that the more reviews the system is fed with, the less noise will surface and the relevant result will prevail. Thus the process of manually summarizing the reviews to achieve a justified result for the system would probably take a significant time. This would be the target values. The user would then need to evaluate the system for the said reviews, the user's perception can then be measured against the pre annotated summarizations. This would give a more appropriate, although subjective, result. Naturally, this would take a lot of time to accomplish and was therefore not included in the thesis. We regret to inform that our system captures a lot of noise. This is the core reason why the accuracy is so low. However, what's truly relevant is the recall, since a majority of that noise gets filtered out in the aggregation step.

5.6 Stanford CoreNLP Issues

Since Stanford CoreNLP uses statistical methods to decide how to build the dependency tree, it sometimes gives incorrect results. Stanford CoreNLP is extremely good, with 92.2% accuracy [7] on the English Penn Treebank. However, reviews are seldom written with the same quality as the English Penn Treebank, leading to reduced accuracy for our text. Obviously, this is not Stanford CoreNLP's fault per se, but it affects our results, since our whole analysis is based on the output from CoreNLP. If the parsed text gets tokenized wrongfully, we might miss an entire relation just because someone wrote a sentence with weird grammatical structure, and CoreNLP couldn't make sense of it.

5.7 Sentiment Analysis

As a final addition to our system we applied an analysis of sentiment on the descriptors. Time spent on this task was limited, due to it not being a fundamental part of the classification. Therefore, no extensive analysis was made on the result of the sentiment. Instead it is to be seen as an indicator and to give a better overview in a possible extended UI, built on our system. Sentiment analysis is a big field of interest in the world of NLP research at the moment. With many different options it is a big possibility that our take on sentiment analysis is not the most optimal one. Our approach is thusly a more ease of implementation and availability oriented rather than accuracy oriented. With more time and a bigger project scope, different sentiment analysis methods could have been evaluated to find the one more suitable for our system. It would have been possible to extend the sentiment analysis, adding more classes like "Very good" / "Very bad", resulting in a more relevant distinction between sentiments.

5.8 Alternate Approaches

Our first and initial thought was to use a word cloud (constructed using TF-IDF) in order to find product features in a review. In hindsight, this would probably have worked just as well. We built a word cloud for some products, and found that it included a lot of words that were not very interesting or unrelated to the product's features. Removing these manually would be tedious, especially if needs to be done for each product, or even category. Since our aim was to do the extraction automatically and dynamically, we ended up not using the word cloud.

There are also minor difference in the classification pipeline that could have been made. For example, another library could have been used for the NLP processing, which might give different results. Scikit-learn could also have been replaced with another machine learning library like NLTK[25].

5.9 Future Work

The scope of our thesis ranged only to experimenting and creating a foundation of a product feature classification system. It is thus possible to add many things in future work. In terms of machine learning accuracy it is possible to come up with approaches to combine with ours, both in terms of extra features and manual additions like a list of features among other things.

To further increase the value for the potential user, our approach has been quite restrictive when including long descriptors, a suggestion for a more complete system would therefore be to in some way allow for longer descriptors to included, capturing more context. With the example sentence "The sound level is very good for parties", it would then not only capture "Sound level" -> "very good" but instead "Sound level" -> "very good for parties". To increase the end-to-end result even further, it would be suitable to in the future experiment using a classifier to perform the linking step in our system, instead of the current manual implementation.

In terms of non-machine learning, the system would benefit from a well composed extended UI to filter the noise still prominent in the simple UI. Moreover also evaluate the system from a user's perspective.

As stated in the limitations section, using neural networks as classifiers was not included in our scope. To optimize the classification steps in the future, it is suggested to also evaluate the result using it. Since it generally is considered a prominent classifier in machine learning.

Naturally, increasing the size of the training set, or at least reiterating and correcting annotations even more, is a final suggestion to achieve at least marginally better results.

Chapter 6
Conclusions

We managed to finish an end-to-end system, although we would have liked to implement the improvements mentioned in Section 5.9. Overall, we managed to implement a fully dynamic relation extraction system with classifiers that perform in a consistent manner with an FScore at 80%. The end-to-end evaluation proves that about half (53%) of all relevant relations are found, which can be aggregated to summarize a review. However, our model produces a lot of noise, and even if this could be filtered out by either a list of stop words or aggregation with relative ease, it is a problem that we hoped we would have had time to deal with. Furthermore, had the annotation set been larger, we would have covered a lot more variations that reviewers use to express opinions about product features. Pre-processing the data turned out to be crucial to make sense of some of the sentences that were fed to the system. Given more time, we would have improved this part of the system to break sentences into smaller sub-sentences using the link structure from the dependency parse tree, as described by Angeli et al, 2015 [2].

It turned out to be very hard to stay on target and be consistent in annotation both training and test sets. We often changed the way we wanted to annotate, and this step was particularly prone to human error. After annotating together for a while and coming to a shared understanding of what should be labelled as what, we probably should have let one person do the annotation by himself. This would have reduced some of the inconsistencies in the annotation set.

In the end, SVM was the overall winner in the classification comparison, winning the relation extraction classification (80.78% FScore), and also closely followed Decision Trees in the chunking classification step (98.14% vs 98.24%).

We were content with the way we performed product feature and descriptor disambiguation by using lemmatization to allow words with different word endings to reference the same product feature or descriptor. Additionally, this was improved upon by using sequence permutations to link entities together (as described in Section 3.4.2), while preserving the original information.

Our iterative approach has been crucial for the success of the project. Building an infor-

mal baseline and then improving upon that with clever preprocessing, annotation, feature engineering and classifier comparison in small steps proved to be an efficient approach. This is particularly true when dealing with complex systems such is this, where there are so many small parts of the system that affect the end results.

In conclusion we are satisfied with our results, even if we left some loose ends for future work due to time restrictions. We successfully fulfilled the purpose of the thesis, answered our research questions and, most importantly, completed our goal of fully dynamic relation extraction.

Glossary

ASIN Amazon Standard Identification Number is a product identification number. 28

BIO Begin Inside Outside. A way of annotating words indicating that they belong to the same group. E.g the in the following sentence: "The sound(B) quality(I) is amazing(B)", "sound" and "quality" is in the first group and "amazing" is (alone) in the second group.. 27

confusion matrix A confusion matrix is used to display the spread of the classifier predictions compared to the expected result. Thus one matrix axis corresponds to the expected results and the other to the actual results. This matrix is commonly used to reveal what classification errors were made, to be able to combat them.. 26

POS Part-of-Speech tag. Defining the word-category. 24

Scikit-learn Open Source Machine Learning library in Python. 9, 44

tokenization Tokenization is the process of extracting phrases, words and symbols from a sentence. This includes breaking out commas, whitespace and other characters that builds up a sentence. Each element extracted results in a token. It's essential to perform tokenization, since tokens are what's passed to a parser or tagger in order to e.g assess part of speech tags to each token. [29]. 14

Acronyms

DESC descriptor. 7

ML Machine Learning. 7, 8

NLP Natural Language Processing. 7

PF product feature. 7

Bibliography

[1] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[2] Gabor Angeli, Melvin Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. *Linguistics*, (1/24), 2015.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204, 2010.

[4] Nguyen Bach and Sameer Badaskar. A review of relation extraction. *Literature review for Language and Statistics II*, 2007.

[5] Brat rapid annotation tool. `http://brat.nlplab.org/index.html`. [Online; accessed 14-May-2016].

[6] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[7] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[8] Stanford CoreNLP, on github. `https://github.com/stanfordnlp/CoreNLP`. [Online; accessed 14-May-2016].

[9] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[10] Kushal Dave, Steve Lawrence, and David M Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM, 2003. [Online; accessed 29-june-2016].

[11] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *Proceedings of LREC*, volume 6, pages 417–422. Citeseer, 2006.

[12] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

[13] Marti A. Hearst, Susan T Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[14] David W Hosmer Jr and Stanley Lemeshow. *Applied logistic regression*. John Wiley & Sons, 2004.

[15] Minqing Hu and Bing Liu. Mining opinion features in customer reviews. In *AAAI*, volume 4, pages 755–760, 2004.

[16] Julian McAuley, Rahul Pandey, and Jure Leskovec. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2015.

[17] Amazon Mechanical Turk. `https://www.mturk.com/`. [Online; accessed 17-May-2016].

[18] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[19] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[20] Stanford Neural Network Dependency Parser. `http://nlp.stanford.edu/software/nndep.shtml/`. [Online; last accessed 18-August-2016].

[21] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002. [Online; accessed 29-june-2016].

[22] Ana-Maria Popescu and Orena Etzioni. Extracting product features and opinions from reviews. In *Natural language processing and text mining*, pages 9–28. Springer, 2007.

[23] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[24] scikit-learn, Open Source Machine Learning in Python. `http://scikit-learn.org/`. [Online; accessed 17-May-2016].

[25] NLTK, Natural Language Toolkit. http://www.nltk.org/. [Online; accessed 14-May-2016].

[26] SentiWordNet, A lexical resource for opinion mining. http://sentiwordnet.isti.cnr.it/. [Online; accessed 10-May-2016].

[27] SpaCy, Industrial-strength Natural Language Processing. https://spacy.io/. [Online; accessed 14-May-2016].

[28] WordNet, A lexical database for English. https://wordnet.princeton.edu/. [Online; accessed 10-May-2016].

[29] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *Proceedings of the 14th conference on Computational linguistics-Volume 4*, pages 1106–1110. Association for Computational Linguistics, 1992.

Appendices

Appendix A
POS-tags

1. CC Coordinating conjunction
2. CD Cardinal number
3. DT Determiner
4. EX Existential there
5. FW Foreign word
6. IN Preposition or subordinating conjunction
7. JJ Adjective
8. JJR Adjective, comparative
9. JJS Adjective, superlative
10. LS List item marker
11. MD Modal
12. NN Noun, singular or mass
13. NNS Noun, plural
14. NNP Proper noun, singular
15. NNPS Proper noun, plural
16. PDT Predeterminer
17. POS Possessive ending

18. PRP Personal pronoun
19. PRP$ Possessive pronoun
20. RB Adverb
21. RBR Adverb, comparative
22. RBS Adverb, superlative
23. RP Particle
24. SYM Symbol
25. TO to
26. UH Interjection
27. VB Verb, base form
28. VBD Verb, past tense
29. VBG Verb, gerund or present participle
30. VBN Verb, past participle
31. VBP Verb, non3rd person singular present
32. VBZ Verb, 3rd person singular present
33. WDT Whdeterminer
34. WP Whpronoun
35. WP$ Possessive wh pronoun
36. WRB Whadverb

EXAMENSARBETE Summarizing Product Reviews Using Dynamic Relation Extraction
STUDENT Mikael Gråborg, Oskar Handmark
HANDLEDARE Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Produktöversikt med hjälp av artificiell intelligens

POPULÄRVETENSKAPLIG SAMMANFATTNING **Mikael Gråborg, Oskar Handmark**

Amazon är världens största säljsajt med upp emot tusentals recensioner för en enskild produkt. Men hur får man som kund en överblick i recensionsdjungeln?

Genom att analysera över 1.7 miljoner recensioner fann vi ett sätt att komprimera brödtexten tillhörande individuella produkter till ett fåtal ord, innefattande endast essensen av textmassan. Med andra ord sammanfattas produktens egenskaper med recensenternas egna synpunkter. Ta ett par hörlurar som exempel: Vad tycker egentligen användarna om batteritiden, designen eller ljudet? Genom att jämföra huruvida dessa egenskaper benämns positivt eller negativt har vi skapat ett verktyg som ger en översikt för produkters fördelar respektive nackdelar.

De senaste trenderna visar att konsumenter inte längre lägger lika stor vikt på vad produkter har för prestanda på pappret, utan påverkas mer av vad andra tycker och tänker. Användare har blivit bättre på att jämföra produkter på webben i takt med att informationen som finns online blivit mer öppen och tillgänglig. I vårt examensarbete ville vi bli av med bruset som finns i skriven information och på så sätt ge användaren kontroll över informationen.

Examensarbetet som berör området artificiell intelligens och språkteknologi skrevs tillsammans med IT-företaget Tactel, som på senare tid börjat utforska möjligheterna att använda dataanalys för att skapa affärsvärde.

Amazon gör själva en del för att underlätta för konsumenter. Bland annat visas de omdömen som markerats av andra användare som "hjälpsamma" överst i flödet. Detta underlättar, men ger inte alltid en rättvis bild av produkten, till exempel lyfts sällan de defekter eller problem en produkt har. Våra efterforskningar tyder på att användare letar efter nya sätt att effektivt jämföra produkter. Vår teknik hjälper till med just detta, och förhoppningen är att kunna underlätta för konsumenter i deras produktletande.

Textanalys är ett komplext problem i sin natur, men blir extra svårt för användarskrivna recensioner eftersom meningsbyggnaden ofta är dålig. Med hjälp av avancerade algoritmer analyseras den grammatiska strukturen av meningar för att hitta mönster som sedan används för att bygga en statistisk modell. Denna model kan sedan förutse om ett ord är en produktegenskap eller något som beskriver en egenskap. Med andra ord lär sig datorn att fatta egna beslut för tusentals omdömen. Det samlade resultatet visualiseras sedan i ett lättsmält format.

Den nya kunskapen från examensarbetet kommer att användas till att utveckla nya spännande tjänster inom området textanalys och artificiell intelligens.