Utrecht University

Bachelor Thesis

Systematically extracting data from mobility solution reviews using natural language processing

Koen Niemeijer

superviser
Dr. Slinger Jansen

second superviser
Dr. Fabiano Dalpiaz

June 29, 2017

Abstract

When buying a vehicle from a website, there are millions of choices to be made. Customers do therefore often not know what is best suited for them. In this thesis we show how to apply natural language processing tools to systematically retrieve and process data from car reviews and present the most essential features based on their needs. This helps customers find the best mobility solutions tailored to their preferences in a systematic way. This work of research implements a solution of opinion phrases and coreNLP, which shows that natural language processing is a solid and reliable way of extracting and rating data from reviews, but also that there is still work to be done.

Keywords: Natural language processing, mobility solution, car review, sentiment analysis

Contents

Chapter 1

Introduction

When the first T-Ford was built in 1908, Henry Ford wrote in his autobiography: "Any customer can have a car painted any color that he wants so long as it is black" (Ford & Crowther, 1922). More than a hundred years later, the U.S. car market is booming with hundreds of different types of cars in every possible colour. Thus, making a choice between seemingly infinite amounts of options has become ever so problematic. Fortunately there are dozens of magazines, books, and websites that help customers pick the best mobility solution. However, not all comparators give their rating to a car and even so, it is impossible to read every source for car comparison before making a well-informed choice.

This problem has even grown more problematic after the invention of the Internet in 1991 and the rapid advance of the mobile phone. The knowledge of the world of cars in the palm of your hand has been a relief for many users, even if this leads them to information overflow. This problem has been picked up by many magazines and websites that review cars by hand and present an opinion and rating to the customer. However, just like everything done by humans, it is prone to errors, prejudices, and fallacies.

This has led to the need of developing a scientific method and solution for systematic extraction of data in car reviews. What is needed is a system that works through a vast number of reviews and extracts their opinion without the human fallacies or prejudices. To top it off, this data can also be used by to incorporate into a larger, more user friendly system.

To start things off, chapter 2 describes a research approach to the problem. More specifically, we define a set of research questions that provide guidance towards developing a systematic way of extracting data from reviews. It defines keys questions, aspects, and requirements that are essential to fulfil customers needs.

In chapter 3, a number of well-suited, potential natural language processing tools are described and compared that may be used in the solution. Later, various implementation tasks are proposed that these tools must possess, and an extraction model is construed for the final output based on literature. When the foundations of the potential system have been laid out, we outline an implementation based on

this foundation.

Finally, the last three chapters address results of the system, lessons they teach, and future work to be done. We make recommendations for other systems with perhaps a slightly different use or different set of tools. In the final chapter, we discuss what improvements and extensions can be made to the system.

Acknowledgements. The author thanks Slinger Jansen for the opportunity and guidance on this project, and Fabiano Dalpiaz reviewing and providing useful feedback.

Chapter 2

Research approach

In chapter 1 we have described how the increasingly growing problem of information overflow on the mobility solution market is a problem, and why this is a problem. This research focuses on finding data in reviews and extracting them in a systematic way. Reviews have been chosen as the source to focus on, since they do not only contain more information than user ratings or meta data, but may even unravel properties or features that otherwise may not be asked about. Furthermore, the goal is to create a system that can make this data useful for other systems, such as system which visualises it for a user. Therefore, the research question is:

How can we find data in car reviews and extract it in a systematic way to support people in finding a mobility solution?

To answer this question, we must first work through multiple sub questions:

- In what data formats do car reviews come and how do we handle them?
- What techniques do we use to extract data from a text?
- How do we bundle together the found data into meaningful variables?

First, we need to know what kind of data formats to look for. Something to keep in mind is that the system needs to be able to extract data from many sources, such as different types of databases, different review types, or potentially even the quality of the photo that is included. Ideally, the system may even be able to extract data from car advertisements. The system needs to have a way to interpret the text from various different sources. This is a big step, but we are aided by systems that support a wide variety of text inputs in many languages. In one case, experts have written the car reviews, so there is no need to worry about quality of the reviews. Alternatively, if they are written by users, quality becomes less of a problem as the amount grows larger.

Next, we need to know how to extract data from reviews. In other words, what techniques do we use to extract data from a text? First, it is paramount to realise that we do not have to start from scratch. Analysing naturally written text

is a traditional information science problem, on the cusp of the field. Mining data from a text is called natural language processing, and there already is a multitude of good research, such as Bird's et al. (Bird, Klein, & Loper, 2009) natural language processor in Python, or Manning's et al. (Manning et al., June 2014) natural language processor. In addition, there is also good groundwork when it comes to natural language processing (NLP), such as Manning & Schütze's prestigious research (Manning & Schütze, 1999). In conclusion, a wide variety of techniques and frameworks already exist for NLP solutions. The laborious part is to apply it to often-biased car reviews, databases, and advertisements. For example, if an advertisement says the design looks 'great', its meaning is not the same as when a non-biased review says its design looks 'great'. After all, the advertisement is trying to sell something. For this reason, we need to find a composition of techniques that will provide a means to extract data in a systematic non-biased way. What's also mentioned in the research question is that it needs to be extracted in a systematic way, so that it is also usable for other systems and it treats every car the same way. Once again, both quality and quantity play a role here, as more of each means better quality of data. This is important for the system, since the accuracy needs to be as high as possible.

Finally, the system needs a way to bundle together the data it has found into variables. That is, how do we describe a car as a set of variables. We can say the car design is good or not, how it performs on multiple aspects, or maybe some other vaguely described measurement for a car. What keywords in a text represent these variables and how do we decide which keyword belongs to which variables? After all, not every writer has the same writing style. There is no point in trying to find something in a text if you do not know what to look for. In addition, there can be a strong variance in intensity of a review statement. If both reviews say a car is 'good' on a certain aspect, can we give the same rating for both reviews if we consider the context? Once we have decided which variables describe a car best, the system can use these variables (and their keywords) to gather information from a car review. Furthermore, simply stating which variables are good or bad may not be enough. After all, the goal of the system is to support people in finding a mobility solution. In order to do this, the system needs to present its results in the most clear manner as possible. This means that output is not only accurate and valid, but also easy to comprehend.

Chapter 3

Foundations in literature

The previous chapter concludes that natural language processing (NLP) is a traditional solution for extracting data from text, as it is able look at patterns in a text. Of course, we do not have to build a tool for NLP ourselves, as there are already many tools available.

3.1 Available tools

Natural language processing is a field in computer science, and has been developing for decades. As said before, there are a great many good researches available when it comes to information extraction. However, we consider only a few modern, major tools used for extracting data from text. Arguably, the Stanford CoreNLP natural language processing toolkit has been the most used NLP toolkit and offers "a simple, approachable design, straightforward interfaces, the inclusion of robust and good quality analysis components" (Manning et al., June 2014). CoreNLP is a pipeline system that makes use of several other NLP components. The system tunnels through an 'Annotation object', which takes raw text as input, processes it through the pipeline, and has an annotated text as output. Some advantages are that it is lightweight, easy to use, and most important of all, it is useful for many languages from many different sources. If need be, external modules (annotators) can be added so that the system is extended. CoreNLP partly solves for the goals in the first and second step, namely how to extract text from different sources. However, it lacks lexical understanding as to what is decisive to a car and what is not. In other words, CoreNLP is useful for syntactical purposed, but it may not fully understand the semantics of a text.

For a more contextual understanding, one may find DKPro TC (Daxenberger, Ferschke, Gurevych, & Zesch, June 2014) more useful. In contrast to Stanford's CoreNLP, DKPro TC works with features, allowing a deeper understanding of context. Another advantage is that it takes annotated context as input, which is the output of CoreNLP. This annotated text then goes through five stages: reading input data, pre-processing, feature extraction, machine learning, and evaluation

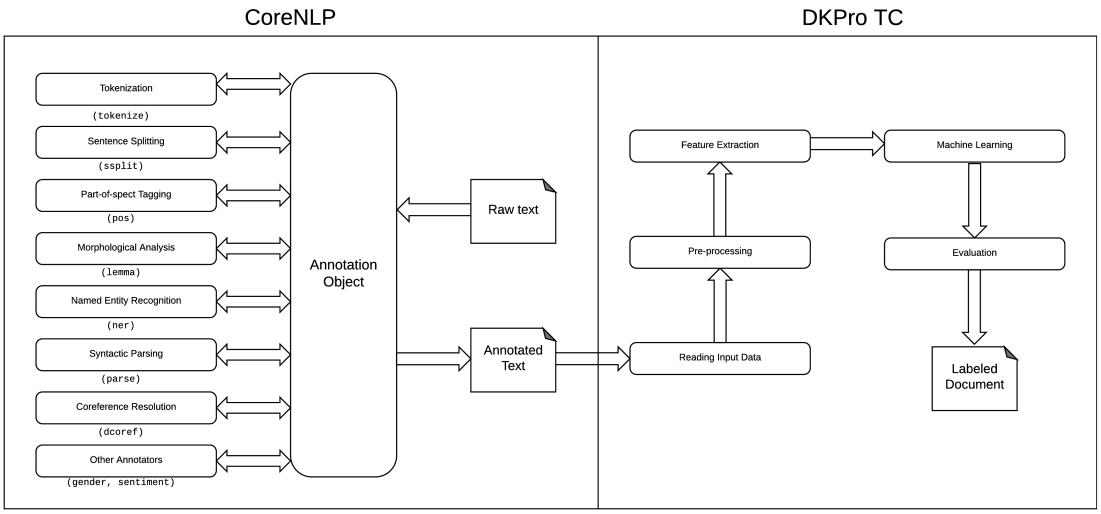

Figure 3.1: A tool model of the proposed NLP tools. CoreNLP model by Manning et al. (Manning et al., June 2014).

(Daxenberger et al., June 2014). The first two stages, reading input data and pre-processing, are to prepare the system to analyse the text. In the feature extraction phase, DKPro TC classifies each feature found with a label, multiple labels, or real numbers, depending on which mode is selected. For this research's purposes, classifying features with labels or numbers already goes a long way to rating a car based on a review. Figure 3.1 visualises a proposed solution by combining Stanford CoreNLP and DKPro TC.

Now that we have described some major key points in the process of developing a system, it is useful look at feasibility. The research approach describes some obstacles that the system must overcome. One of those obstacles is automation of input types and processing of those different types. CoreNLP already solves this issue by being able to take multiple formats as input and annotate this text in such a way that it is usable for other systems, for instance DKPro TC. A second obstacle for the system is that it needs to know which key words in a text belong to which properties of a car. A partial solution can be achieved by implementing DKPro TC, as this system makes an educated guess for each feature to assign it a value or label. However, there are still many problems. One earlier mentioned problem is the need to overcome the article writer's bias. Generally, it is next to impossible to completely overcome this bias. This, however, does not mean there is nothing to be done to mitigate it. A possible solution is to automatically account for exaggerations made about a car in an advertisement.

Yet another NLP toolkit is Bird's et al. Natural Language Toolkit (NLTK). NLTK is an easy to learn suite of Python modules which provides various NLP tasks, such as tagging, chunking, and parsing (Bird, 2006). One of NLTK's advantages is that, because of Python's easy syntax, it is easy to learn and use for people who do not have much experience with NLP tools. A downside of this is that the tasks and

Feature	CoreNLP	DKPro	NLTK	spaCy	UIMA
Tokenization	X	X	X	X	X
Sentence Segmentation	X	X	X	X	X
Lemmatization	X	X	X	X	X
Stemming	X	X	X	X	X
Morphology	X	X	X	X	X
POS	X	X	X	X	X
Lexical Semantics		X	X		
Named Entity Recognition	X	X	X	X	
Sentiment Analysis	X	X	X	X	
Dependency Parsing	X	X	X	X	
Coreference Resolution	X	X			
Feature Extraction		X			

Table 3.1: Comparison of features used in various NLP tools

evaluations embedded in NLTK are relatively basic and therefore may not offer all of the functionality required for the envisioned system.

It is now apparent that before starting to apply one of these tools to car reviews, it is fundamental to first understand more about how text extraction works. A first thing to understand is that text extraction has its roots in literature and more specifically, in statistics. Nearly every NLP tool contains elements that have its roots in statistics, such as tokenization, morphological segmentation, or n-gram models. Most of these elements new to NLP sparked from a revolution in statistical natural language processing in the late 1980s and mid-1990s. Instead of using hand-written rules, researchers started to experiment with using machine learning and algorithms, such as decision trees. However, research then was mostly focused on using statistical and probabilistic models to analyse sentences, instead of analysing the deeper contextual understanding that a sentence conveys (Manning & Schütze, 1999; Berger, Della Pietra, & Della Pietra, 1996; Brown et al., 1990).

Table 3.1 provides an overview of the most used NLP toolkits. There are two things that can be learnt from this table. First, the toolkits have largely the same features, save lexical semantics, coreference resolution, and feature extraction. There are several reasons why this occurs. Most toolkits provide a basic number of features that help to analyse a sentence. When it gets more difficult however, there are only a number of tools that can do this. One of these difficult things is lexical semantics, or in other words, understanding not how something is said (the relations between words) but the semantic meaning a sentence conveys. Another interesting feature not all systems possess, is coreference resolution. In short, coreference resolution occurs when two or more expressions in a text refer to the same person or thing (Wikipedia, 2017). For people this is often easy to figure out, but for computers this ambiguity is hard to overcome. Finally, there is feature extraction, which is

only implemented by DKPro. Unique about this, is that "feature extractors have access to the document text as well as all the additional information that has been added in the form of UIMA stand-off annotations during the preprocessing step" (Daxenberger et al., June 2014).

The second thing that stands out in the table is the seemingly limited set of features of Apache's UIMA. Unstructured Information Management Architecture, or UIMA for short, is a framework for NLP tools that allows others to build plugins or other toolkits on top of it. As a result, UIMA itself is not so powerful as a fully build NLP toolkit and gets its strength from the plugins available, which are not incorporated in this table. If they were, UIMA would have many more, if not all, of the features listed in the table. An example of a toolkit built on top of UIMA is DKPro. As a matter of fact, DKPro is not much more than a pipeline of UIMA plugins and several other toolkits, such as CoreNLP and some features of NLTK.

3.1.1 Tasks and evaluations

Ever since the statistical revolution in the '80s, nearly every NLP toolkit uses an amalgamation of the most commonly researched tasks in NLP. This section discusses some useful evaluations and tasks for the research purpose, so that it may use this as a criteria of evaluation for NLP tools.

Lemmatization and stemming. Both of these techniques are used for different forms of a word, such as its stem or closely related words.

Part-of-speech tagging (POS) and morphology. Determine the usage of a word in a sentence, such as a noun, verb, or adjective. Doing this task gives a better syntactical understanding of a sentence and thereby helps other tasks such as lexical semantics.

Lexical semantics. After having analysed the sentence syntactically, it is much more crucial to understand its meaning. Lexical semantics often uses real values assigned to words to calculate the most probable relationship between them. This gives a better computational meaning of individual words in a context.

Clustering. The input text gets split up into related groups called clusters. These clusters can then be joined with each other to create a new cluster. Every cluster is a set of objects represented as tokens. Objects are normally described and clustered using a set of features and values. As a result, clusters can be represented in a data representation model, which provides more insight into the text (Manning & Schütze, 1999).

Note that the list of the tasks and evaluations listed above is non-exhaustive and that there are many more tasks a toolkit may use to be useful. For now, however, this is seen as a minimal set of evaluations a system must have.

3.1.2 Supervised and unsupervised learning

When NLP systems were in their earliest form, there was only one type of learning: supervised. This means that the systems were build and tested based on corpora of real-world examples. An advantage of this is that systems can be reliably tested and options can be made exhaustively. However, as systems became more sophisticated and as processing power increased, the need for non-given language processing increased.

Whereas supervised learning systems are based on decision trees, unsupervised learning systems work with statistical probabilities and machine learning, which utilises language models to make predictions about a given text (Manning & Schütze, 1999). Even though supervised learning systems can be more easily tested because the output is known, unsupervised learning systems thrive when there is no predetermined text and therefore no possibility to make a decision tree. This is especially useful when trying to find new relationships in a text of unlabelled data, such as reviews.

3.2 Extraction model

The goal of this research is to produce a system that supports people in finding a mobility solution. We have already proposed several tools that the system can use to achieve this goal. So far, however, we have neglected the input and the output, which in turn helps people find a mobility solution.

As for the input, it is easy to imagine this is nothing more than raw text. Whether this is car review from a magazine, website, database, or advertisement is something the system has to assess for itself. A better question is how the system assesses what is essential to a customer and how it should rate this variable.

A more difficult issue is how to assess the output. The most logical solution would be to analyse many car reviews and determine what people find most significant, for example by counting how many times a property (or feeling about the car for that matter) has been mentioned. Though this solution is good because there is such a vast variety of car reviews, this is also its biggest weakness. Figure 3.2 is a word cloud made out of almost 9000 car reviews from multiple car types. There are some good properties to use such as 'comfort', 'look', or 'new', but it is extremely general and therefore practically useless.

Figure 3.2: A word cloud of car reviews in the shape of a car

Not all cars are the same, so neither can essential properties be. For example, if

you want to buy a mini van, it is good to know whether it is spacey or not. This, however, is probably irrelevant for a sports car. One solution is to make candidate keywords which are selected based on car reviews from all cars. In other words, a non-exhaustive list of general car properties that may or may not be applicable to a specific type of car. The actual keywords are selected based on the general property list combined with the word count (or combination of words) of car reviews of one specific type. This, in turn, leads to a stable list of variables that is only be applied if reviews show it is relevant to that kind of car.

Honda Pilot 2016		Honda CR-V 2016		Maxda CX-5 2016		Honda Accord 2016		Subaru Outback 2016		Honda Civic 2016		All cars	
Word	Frequency	Word	Frequency	Word	Frequency	Word	Frequency	Word	Frequency	Word	Frequency	Word	Frequency
car	439	car	206	car	348	car	394	car	310	car	376	car	15886
pilot	420	honda	161	5	223	honda	201	outback	182	civic	176	like	4952
honda	406	crv	121	mazda	205	accord	151	subaru	160	honda	153	great	4846
vehicle	211	vehicle	100	cx	175	great	112	great	109	great	95	get	4102
like	189	great	94	great	170	like	88	like	101	new	70	vehicle	3977
system	169	2016	85	like	153	get	76	vehicle	99	get	64	drive	3865
it's	158	new	79	drive	126	good	75	would	95	like	63	one	3643
get	152	good	76	love	121	it's	65	system	88	it's	60	driving	3469
new	143	like	76	get	115	one	63	get	81	would	59	it's	3313
transmissio	143	cr	75	one	108	2016	61	2	73	one	57	would	3301
great	142	get	72	really	106	love	60	better	72	love	54	good	3101
would	132	v	71	driving	101	v6	58	driving	72	drive	54	new	3060
one	131	love	65	good	100	new	57	road	70	good	54	back	3029
2016	131	one	57	back	98	drive	56	seat	68	ex	53	love	3002
back	122	driving	52	system	97	really	55	well	68	2016	51	seats	2702
screen	118	back	52	would	93	system	55	2016	67	back	50	well	2651
good	118	much	51	it's	93	would	53	3	66	first	49	miles	2600
driving	110	seat	50	vehicle	89	driving	49	good	66	really	48	much	2587
time	107	noise	50	touring	89	gas	48	back	66	well	46	really	2568
drive	107	would	49	seats	88	ex	46	one	65	system	45	better	2551

Figure 3.3: A word count of the most used words in 6 different types of cars and all cars

Figure 3.3 represents a word count of 6 different car types. The figure points out that even a word count of a specific type of car might not be enough. After all, when you leave common words out, even the most often mentioned words are still pretty insignificant and general. The figure also has some highlighted cells, which represent car properties that may be most relevant to this particular type of mobility solution. This, however, does not mean a word count is useless. If you would have some predetermined list of car properties and would select, say, the five most often mentioned words, you have a fairly good list of features for that specific car type.

Chapter 4

Implementation

In the previous chapter, we have laid out some problems with the envisioned system and some solutions and opportunities found in literature. We also have presented an extraction model as a way of deciding which features are decisive for a car. In this chapter, we take apart the problem of giving a rating for each feature and use some suggested methods from literature to do so. First, we discuss a bag-of-words tool to get a better understanding of the text. After that, it is necessary to interpret output of the model.

4.1 Opinion phrases

In chapter 3.2 we have argued for and against several extraction models. One should note that the extraction model of counting words says what is meaningful about a car, but not how this is meaningful (as in, good or bad). For example, if a customer says 'the seats are terrible', all the extraction model mentions is that there is something about the seats, but not if it is good or bad. After all, the fact that 'seats' has been mentioned substantially in a review does not imply a sentiment of this property.

Nowadays, every product sold on website contains numerous reviews with opinionated text. In fact, there is so much opinionated text, it is almost impossible for a customer to read and compare all of them. After all, to make an informed decision, one has to get to know as much as possible about every option in order to make a logical choice. Even if one has the time and motivation to read through all of it, it is still hard to make an objective, statistical comparison from the different properties and opinions presented to the customer.

To overcome this problem, we introduce a method called opinion phrases. This method has been proposed in a PhD thesis by Moghaddam (Abbasi Moghaddam, 2013) and executed by Vlad Sandulescu (Sandulescu, 2014). Opinion phrases are created by using a bag-of-words model, and more specifically by using latent dirichlet allocation (LDA), which is a statistical method to create a topic model. Under the hood, this tool relies on Stanford CoreNLP (Manning et al., June 2014) to tokenize,

split, assign part-of-speech, and parse the text.

To test this tool, we have gathered almost 9000 reviews from 489 different types of cars. The reviews are collected in a JSON file, which is parsed in Java because the tool is also written in Java. To make the tool work with car reviews, we have extended the program to parse a JSON file and output results to different files together with a total count of all opinion phrases.

Without too many modifications and extensions the results are adequate, but not great. There are some results which are highly usable, but also nonsense such as '!! 500e' or 'row 3rd'. Furthermore, there are still some general comments, such as 'car great', or 'car love', which of course do not mean much to the qualities of a car, except for the overall rating. Another issue that can be pointed out is that after a total count of opinion phrases (a collection and count of the same phrases), the highest count is at 8, of which only a handful has been counted more than once. This implies that the phrases are too different from each other to be used, even in an abundance of data. A possible solution is to standardise the opinion phrases, and only select those which have relevant words.

However, this tool also shows many possibilities. For examples, one car (Fiat 500x 2017) got the following opinion phrases:

['vehicle uncomfortable', 'space sacrificed', 'vehicle love', 'acceleration adequate', 'ride adequate', 'ride comfortable']

Even though 'space sacrificed' or 'vehicle love' may not be so useful, the rest gives a good impression of what people think about the car.

4.2 Interpreting output

In the previous section, we found a system that can extract so-called opinion phrases from reviews. A crucial question we have to ask ourselves is: How do we bundle together the found data into meaningful variables? Before we can answer this question, let us first go over two different yet related problems.

4.2.1 Ratings

First, we need to know what the output is going to look like. In the previous chapter we have already described a set of features that somehow resembles the state of a car. What is not yet described, is what form this set of features exactly has. In popular literature, there are several ways one can rate a set of features. We consider the following three:

- Numbered scale
- Worded scale
- Star rating

Let us start by discussing differences between a numbered scale and a worded scale. A numbered scale works with numbers, for example 1-5. One advantage is that the system can directly output a number, without having to convert it to words first. Another advantage is that numbers are unbiased. A worded scale, in contrast, is much more open to interpretation as it may be unclear how big the difference is between 'good' and 'very good'. On the other hand, numbers may give an uncanny feeling to the reader, whereas a word feels much more natural. Finally, there is the star rating, which is not much more than a number represented in stars. This gives the output a better look, but may also confuse things as they are harder to read and less precise than just numbers or text.

Of course, there is no "correct" output for the system. Therefore, we choose for numbers as they are easy to modify for the system. A good practice that is also applied by Stanford's CoreNLP is to convert the numbered scale to a worded scale, which can easily be done at the end of this project.

4.2.2 Sentiment analysis

In the previous chapter we came to the conclusion that counting words to rate variables is not enough. To tackle this problem, we introduced a method called opinion phrasing. Even though this has already yielded much better results, we still have not determined how to transition these phrases into a meaningful rating for a car. An aforementioned method is to use opinion phrases and see how significant this is to customers. A major flaw is that a count of phrases does not imply a weight, or a positive or negative value. It only says something about the importance in contrast to other variables, but not if this is good or bad, and how good or bad this is.

A major task in NLP is called sentiment analysis. Socher et al. describe recursive deep models for semantic compositionality over a sentiment treebank (Socher et al., 2013). Sentiment analysis is the task of describing the sentiment or feeling of one or multiple words. As the authors point out, the accuracy gets higher when more words are analysed. Their models use a recursive neural network, instead of a more traditional bag-of-words model.

A bag-of-words model, which is also used for extracting opinion phrases, is one of the most popular representation models in NLP (Zhang, Jin, & Zhou, 2010). This kind of model uses clustering to generate a histogram of visual words. This model, however, ignores word order, as pointed out Wallach (Wallach, 2006). Therefore it is hard to determine the sentiment of a bi-gram because it relays on features such as 'great' or 'awesome'. This may work well in longer documents, but is not suited for purposes described in this work of research.

Stanford's sentiment treebank relies on a recursive neural network. For each sentence analysed, it creates a tree of words in the sentence, each with a sentiment of their own. Word sentiment is determined by training the model on a large data

set of corpus annotated film review data. Its output is sentence sentiment, which gets calculated by taking word sentiment from the top of the tree and working down. One thing about the recursive neural network is that when only a few words are analysed, extreme values are rarely used. In other words, if the model calculates sentiment for each feature, it is likely to be between 2-4, and almost never 1 or 5.

One approach is to calculate overall sentiment from all reviews for a single vehicle, as well as determine sentiment for every feature. This gives a customer insight into both the sentiment for each feature, as well as the overall sentiment from features not listed. As one can imagine, it is virtually impossible to list every feature correctly, as there is a cost of allowing more features. The more features are allowed to get listed, the more useless features are also listed.

In fact, there can be so many features to list, you quickly lose sight of what is essential to the car. If one car has more than, say, 30 features, a customer is unlikely to go through all of them. In addition, some features may say something about a larger feature in a slightly different way. For example, consider the following features:

['soft seats', 'neat dashboard', 'spacey seats', 'smooth steering wheel', 'quality leather']

All of these features say something about the car's interior, though this may be far too specific to list. Consider when this is the case for every aspect of the car, and then every aspect has many more than just 5 features, the list gets colossal in no time.

What this problem illustrates, is that there is a need to categorise without losing any information. By simply taking the average of all features in a category (interior for example), we can reduce the total number of results to a fraction of what it would be. Depending on how much data there is, it is also possible to use this as a summary and keep all independent features when specifics are needed.

4.2.3 A GUI based approach

In the previous sections we have talked about ways to analyse reviews and output them in some meaningful way. What we have left out so far is how to bring it to a customer in an easy to use way if they were to use the program directly.

The program we have created so far is a command-line Java based program. However, many users may find it inconvenient or even impossible to use, as the average car buyer will have not have much knowledge of these kind of programs. Therefore, it is appropriate to develop some kind of interface to use the program with. In his book, Galitz describes the importance of good design and some patterns on how to achieve this (Galitz, 2007). Furthermore, he also emphasises the importance of several elements, such as using the proper kind of window, choosing proper screen-based controls, and providing feedback.

In a related work, Schneiderman (Shneiderman, 2010) argues that it is good practice to keep user interface simple and keep the user away from any distractions. Therefore, we have sought to create a minimalistic user interface with only elements that directly affect the user.

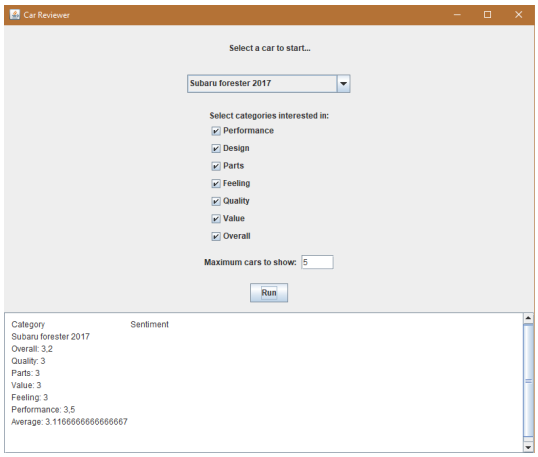

Figure 4.1: A minimalistic interface for the mobility solution review program.

In figure 4.1 you can see this minimalistic interface. The design is simple, yet efficient. When the program starts, it scans the JSON file for all available vehicles and lists it in the combo box. All the user has to do is select a car from the list, categories they are interested in, and press 'Run'. The program then does the rest and outputs the category alongside its sentiment in the text area below.

4.3 Categories and adaptive selection

As explained earlier, there is a strong need to categorise features. The aforementioned reason was that when more reviews are added, the list of features grows so fast that it becomes impractical for a customer to go through. However, another reason not yet mentioned is in the scope of a larger system. Let us imagine a website where you can search and buy mobility solutions, for instance cars.com or autotrader.com. These websites are fairly straightforward in the sense that users need to enter exactly what they are looking for in order to find something. In the era of multibillion dollar advertising, there is only so much a website does not know about its user. Therefore, a search on any of these websites needs not only be based on the search query, but the results should also be tailored to the user's profile. The results do not just come from a simple SQL query, but from a system that takes preferences from a user's profile and search query as input, and generates a list of mobility solutions best suited to the customer.

One should keep in mind that such a system does not yet exist. Most websites that exist now use previous searches and purchases to provide similar products in the form of recommendations. This is vastly different from the system described in this thesis. For some websites, basing recommendations on earlier bought or visited products works very well. For example, recommendations made by Youtube generate 60% of all clicks on the website (Davidson et al., 2010). For mobility solutions, however, this is less useful as customers generally do not want to buy a similar vehicle they recently purchased. Moreover, customers do not purchase vehicles frequently and therefore it is hard to gather enough data to provide a

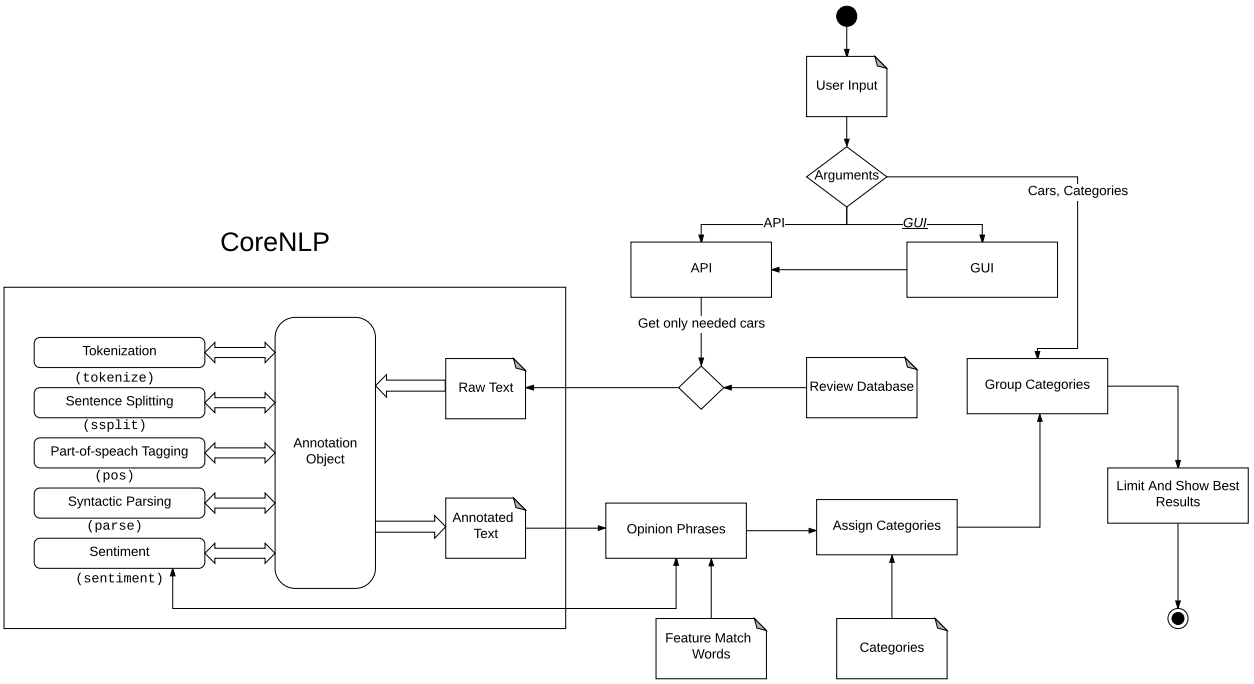

Figure 4.2: A process model of the described system

sensible recommendation.

A more sensible recommendation can be made when the system knows what is substantial to a customer, and select the best cars available. Appendix A contains a list of words the system uses for feature extraction. More words can be added to the list to find more features. Keep in mind, however, that more words may also lead to more 'nonsense' results. Looking at this list, it is impossible for a website to determine which specific features are paramount to a customer and what is not. This is yet another reason why there is a strong need to categorise.

One useful application of this program is to incorporate it into a pipeline of other systems. For this reason it is imperative to have an API a system can call upon. The output of this API depends on a number of parameters. Firstly, the caller needs to specify if they want to use the API or GUI. The parameters for this are identical, namely "GUI" and "API". Note that parameters are not case-sensitive.

Figure 4.2 visualises the system in the form of a process model. The model starts by taking user input as value. This can either go directly to the API (by using parameters), or is selected by a user via the GUI which then calls upon the API. What's next, the cars selected by the user get combined with the review database and processed through the CoreNLP pipeline. The CoreNLP pipeline uses the features tokenization, sentence splitting, part-of-speech tagging, parsing, and finally sentiment analysis. A list of opinion phrases (patterns) gets extracted from this annotated text. These patterns need to match the list of predefined words such as in Appendix A as so to get a limited yet meaningful list of results. To make the results more comprehensive for a user, categories get assigned based on a predefined

list. Only the categories selected by the user will be used (or all if no categories are selected). When the selected cars have been analysed, the top 5 best cars gets displayed on the screen together with their categories.

In more detail, figure 4.3 represents a class model of the described system. What is striking are the classes `Extract` and `Pattern`, as they do not contain any methods or variables. Furthermore, the pipeline of CoreNLP has not been included. The rationale behind this is that `Extract` and `Pattern` are predefined classes as in the system of opinion phrases (Sandulescu, 2014), but only slightly modified. For CoreNLP, it is only called upon once in the `API` class (for every car) after which annotated text is returned. It is not necessary to look deeper into this system as it is outside the scope of this project. The call to CoreNLP looks like this:

```
Properties props = new Properties();
props.setProperty("annotators", "tokenize, ssplit, pos, parse, sentiment");
StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
Annotation annotation = pipeline.process(text.toLowerCase());
```

A more interesting class is `Categories`, which is responsible for assigning, sorting, and grouping categories. The function `makeCategories()` reads the categories and subcategories from the XML file and parses them into lists. This function is called upon first when an API or GUI is created, so that input can be matched with existing categories. Subsequently, `sortCategories()` assigns the appropriate category to each feature (or Pattern as it is called in the program). Finally, `groupCategories(List<Pattern> patterns)` groups categories under the same top-level category (for example design, performance).

4.4 Testing

The data being used for testing is a JSON file from Edmunds.com. It contains information from 489 cars with each 1-25 reviews. In total, there are approximately 60,000 words to be analysed, or 120 pages. For this reason, it takes well over half an hour to process all of the data if no cars are specified. The reason this takes so long is because of Stanford's CoreNLP, which takes 99% of the total time to process. The other 1% of the time is to assign categories, calculate sentiment, and group everything together. To illustrate the dataset and output, Appendix B has a few scenarios accompanied by input and output.

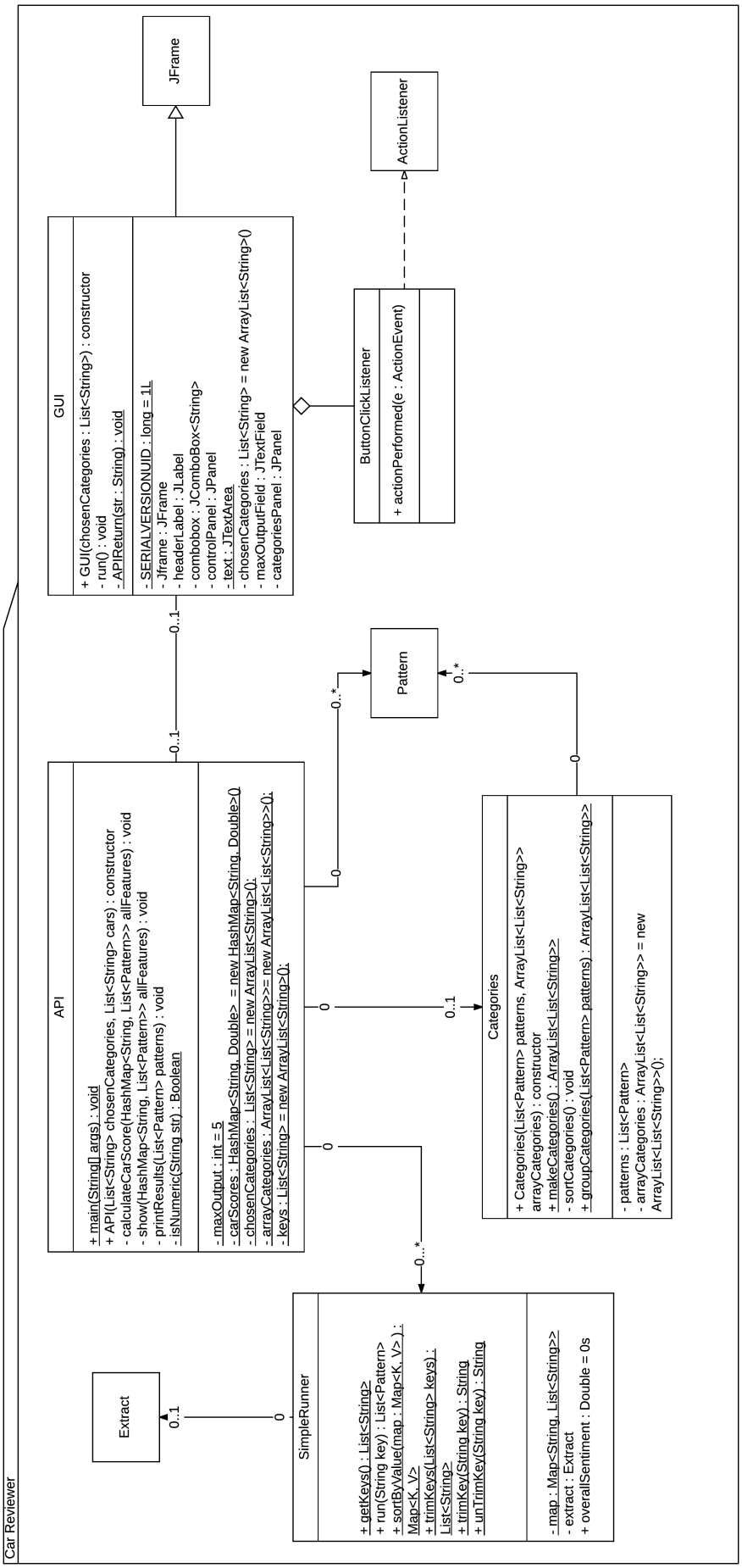

Figure 4.3: A class model of the described system

Chapter 5

Results

Now that a program has been built and tested on various scenarios, accuracy and validity are at issue. Table 5.1 represents the output from scenario 1 in Appendix B. What's striking is that not all categories have been found. This is once again the result of not having enough data to extract features for that category, and thus it shows up empty. One does, however, have to keep in mind this also has to do with the properties of cars itself. For the Ford transit-connect 2016, for example, someone who writes a review is more likely to mention properties about the value or quality than the performance or parts. In contrast, reviews for the Cadillac xts 2016 are more likely to mention performance or specific parts (such as the engine or transmission), as it is more relevant to this kind of car. Leaving categories blank is bad practice, since it is inaccurate to the preferences of the user. After all, a user is interested in both quality and value, not just one of them. This is easily fixed by adding more reviews and thereby adding more features. If we were to make sure now categories are not empty, we would not have many cars left to test.

One thing the scenario 2 (table 5.2) makes clear is that the system has a high degree of flexibility. Input can be as simple as no input at all, or as specific as giving the exact preferences of a user. This way, a system or a user may search for (almost) all vehicles or be as specific as needed, until a vehicle is found that suits their preferences best.

Once again, not all categories in scenario 2 have been found. In addition, just as in scenario 1, all the values are whole numbers, whereas with many ratings you would expect at least some decimals. The reason for this is that only one or two

Car / Category	Quality	Value	Average
Cadillac xts 2016	5.0	-	5.0
Mercedes-benz g-class 2016	5.0	-	5.0
Mercedes-benz e-class 2016	5.0	-	5.0
Kia sorento 2017	5.0	-	5.0
Ford transit-connect 2016	-	5.0	5.0

Table 5.1: Results of scenario 1

Car / Category	Value	Parts	Average
Ford fusion-hybrid 2016	5.0	-	5.0
Volkswagen passat 2016	4.0	-	4.0
Hyundai tucson 2016	4.0	-	4.0
Volvo v60 2016	4.0	-	4.0
Mercedes-benz c-class 2017	-	4.0	4.0

Table 5.2: Results of scenario 2

features have been found in that category, leaving it with a very high score. If many categories have been found, it is unlikely to reach a perfect score of 5.0. Just as in scenario 1, we could prevent this by requiring each category to have at least an X number of features, but that would leave us with not enough test data. In non-testing environment, however, this is certainly a requirement to make.

Finally, table 5.3 shows the results of scenario 3. Unsurprisingly, not all categories have been found. What's different from scenario 1 and 2, however, is that values are much more specific. This has to do with more features, and thereby categories, being found for each car. For the Bmw x5-edrive 2016, for example, as much as three categories have been found averaging a score of 3.42. This goes to show that more features lead to much better and accurate results. Because of this, the output evidently shows which car is favourable for customers.

Furthermore, in general the scenarios show that the output is kept well-organised by only showing top-level categories. This can be both a positive and a negative aspect. The rationale behind this representation is that it provides a good overview of the general aspects of a car, as most users will not be interested in the specific categories itself. Also, if a user has selected a multitude of categories, for example 20 or more, not all categories are representative or have enough data to be shown. This, of course, is a problem that needs to be solved by adding more data. What's more, is that the user itself is probably not selecting the categories, but the a website is. This website has a profile and possibly a search query of a user, and infers the relevant categories for its user. Afterwards, the user does not need to know which specific categories have been selected. On the other hand, a user may have a vast knowledge of cars and therefore knows exactly what kind of car they want. It may then be advantageous to show the specific categories.

Looking at these examples, it is important to keep in mind that NLP tools do

Car / Category	Performance	Parts	Feeling	Overall	Average
Bmw 5-series 2016	-	-	-	3.5	3.5
Buick encore 2016	3.0	-	-	4.0	3.5
Bmw x5-edrive 2016	-	3.5	4.0	3.25	3.42
Buick verano 2016	-	-	3.25	3.25	3.25

Table 5.3: Results of scenario 3

not look as specifics, but to dependencies and patterns in a text. Therefore, if a property has been mentioned that is in the list of required words, it does not mean it will automatically appear as a feature. This, however, is a good thing, since it means that a feature really is a feature, and not just an opinion of a single individual and has been mentioned only once. For example, imagine that 'terrible seats' have been mentioned in a single review of a car just once, it will not immediately show as a feature for that car as well. Only if it has been mentioned multiple times NLP will mark it as significant.

One lesson these tests teach is the dependency between quality and quantity of data and results. That is, one cannot exist without the other. If there is not enough data, the quality is low. Vice versa, if the quality is low, more data needs to be added. This once again plays into the system's weakness of needing too much time to process it.

Looking back to the research approach from chapter 2, questions can now be answered based on results found by the system. First of all, data formats have not proved to be as great a difficulty as expected. It is still true that mobility solution reviews can come from many sources, such as an API, database, magazine, or advertisement. For the system, however, it does not matter much where it comes from as long as it is digital and can therefore be analysed. In addition, it has already been stated that there is a difference between expert and user provided data. From the results however, it becomes clear that as long as there are enough reviews to analyse, quality quickly becomes less of a problem.

Next, the results have shown that NLP has been effective at extracting, analysing, and rating mobility solutions based on reviews. As pointed out in chapter 3 and 4, NLP tools are plentiful with each different options that can be implemented to process reviews. There are, however, also many problems with the way this is processed, as is also pointed out in previous and upcoming chapters. While there is still work to be done, the results do show that NLP lays good groundwork for further improvements.

Finally, we have talked about extraction models and bundling data together into meaningful variables in the previous chapters. While it seemed troublesome at first, the problem solved itself step by step. After having extracted features from reviews, it became apparent that there were too many features for each car to look through. Moreover, the features were too different and ample for a human to comprehend. Having too much data to show is hazardous, as it might lead customers to information overflow. This lead to the conclusion that features needed to be grouped together in order to simply data. After all, the ultimate goal of the system is to help customers find a mobility solution that suits their preferences best. Moreover, data also needs to be reusable by another system, as stated in the research approach. In order to do this, we grouped features together in categories along with their sentiment. This has made output understandable, easy to reuse, and most of all helpful to customers.

To test the effectiveness of the tool, we look towards distribution of numerous categories and ratings in a list of all cars sorted by descending average. Figure 5.1 shows a histogram of the average rating for cars. As mentioned before, sentiment analysis on short phrases is more likely to find ratings between 2-4, than 'extreme' ratings such as 1 or 5. This, however, may also be the consequence of the fact that these are average ratings, and therefore are more naturally drawn to the middle. Except for a few outliers, data seems to be equally distributed with a slight positive tendency (>3.0).

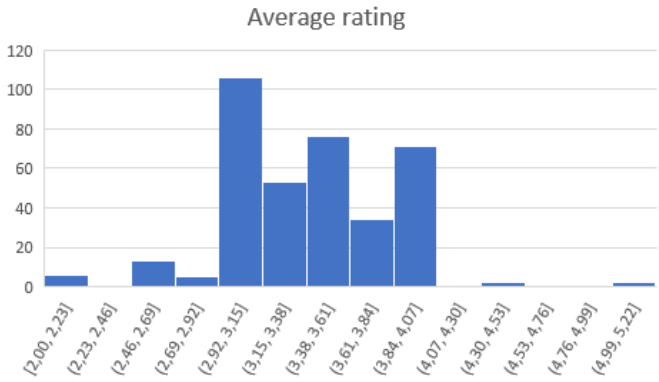

Figure 5.1: Histogram of average rating for cars

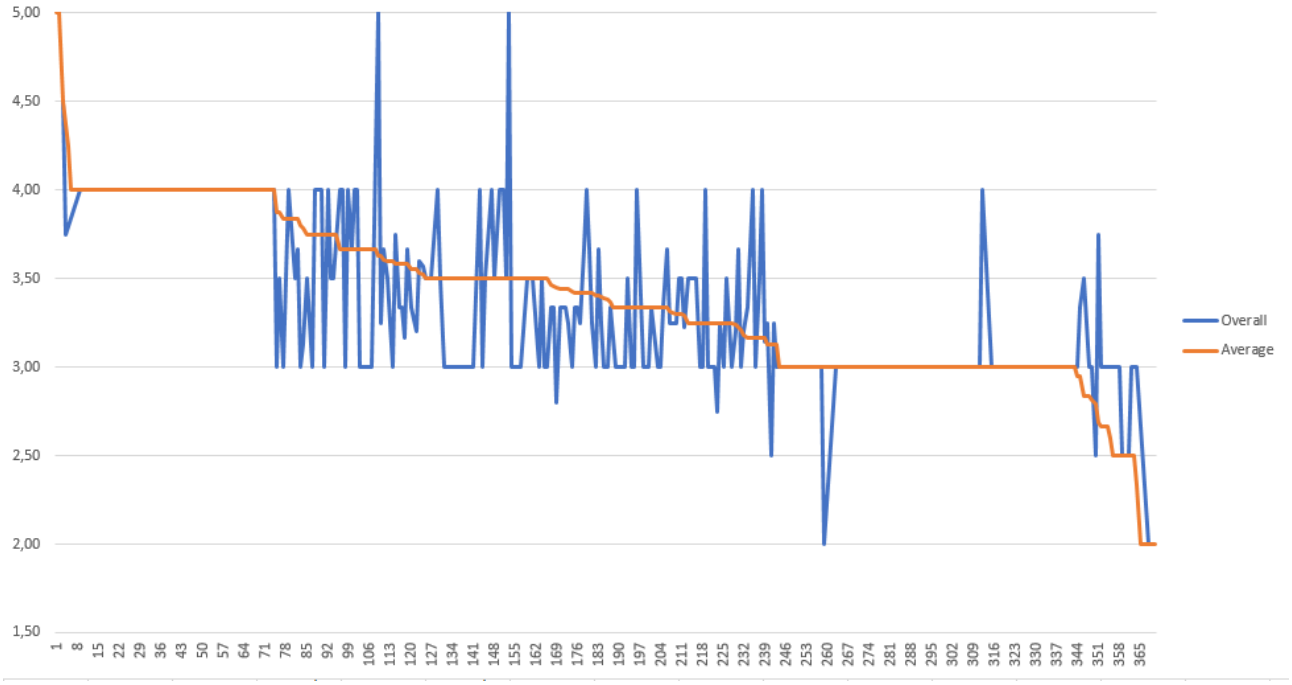

Figure 5.2: Trend between average and overall scores

Another question to be asked about effectiveness is if categories follow the trend of the average score. Figure 5.2 is a graph of both the average and overall rating. In general we can say that the overall score does follow the average score. However, there are many spikes in the overall score, even when the average is not as high. This is largely due to a lack of enough data. What is also noteworthy is that the middle of both lines have much more data more data than in the first or last part. The reason for this is likely that mobility solutions that have a lot of features are located in the middle of the pack, because it is easier for mobility solutions with

not as many reviews to get a very high or very low rating. Finally, both lines are of course downward sloping as the list is ordered on average going high to low.

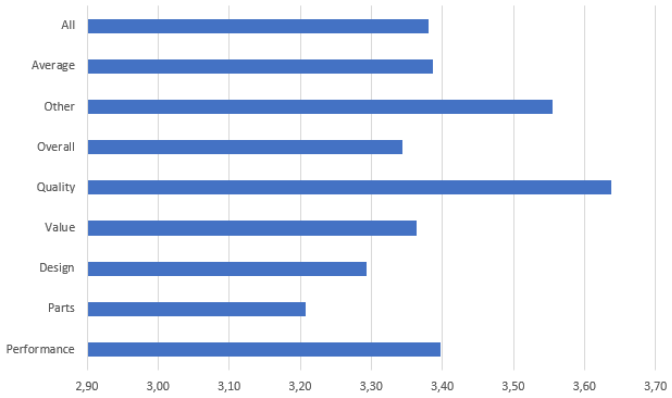

Figure 5.3: Averages for all categories

Figure 5.3 shows what the other figures already make clear. Averages of categories are well above 3.0, which means that customers leave more positive than negative reviews. What is also interesting is that the average of Parts is significantly lower, while the average of Quality is much higher than the rest. One explanation is that people tend to only mention parts when something is not working or is broken, whereas quality is more general and therefore if everything is working well, will be mentioned. Another explanation, however, is that there is not enough and differences in averages are a mere coincidence. After all, the difference in score between Parts and Quality is only 0.43.

Chapter 6

Discussion

One of the problems that has been unmentioned in-depth is that of response time. When using the API, you may notice that it takes a long time for any results to show. This is because all of the work has to be redone for every search. A website cannot afford a customer to wait half an hour because the system need to sift through 10,000 or more reviews. For development purposes however, it is certainly useful to redo all of the work each and every time, as a developer brings on changes in either the data set or the program every time is run. For a website this can also be useful if their data set changes frequently.

This leads to choosing between pre-annotated data and on-the-fly compiling. An obvious advantage of pre-annotated data is that everything has already been through the pipeline of CoreNLP, which takes by far the most time to run. As a result of this time advantage, we feel that pre-annotated data is the way to go for actual use. A company may then replace this data once every month or every week, based on their needs. This also makes for lower requirements in the server hardware and thereby saving costs. For these reasons, we recommend to use pre-annotated data, since compiling on-the-fly is hardly viable.

The problem of having a high time to process is likely to become even larger when different, especially more extensive, NLP tools are being used. If, for example, more annotators get added to the pipeline, time to process quickly becomes larger. Moreover, different NLP tools may yield different annotations or even different output (Al Omran & Treude, 2017). Al Omran and Treude state that most researchers do not justify their choice of NLP toolkit, and that this may severely affect output. In this research we have chosen for CoreNLP over a similar tool such as NLTK, because CoreNLP is easy to use, and also widely used. If we would have implemented NLTK (or spaCy for that matter) instead, results are likely to have been different.

To follow up on the results in the previous chapter, it has been left unsaid how to handle advertisements. In the research approach in chapter 2 we had already noted that advertisements are different from normal reviews, since they are biased. While it is still easy for the system to analyse these advertisements, some values

may be much higher than from other reviews as the goal of the advertisement is to make the mobility solution look as good as possible. One solution is to offset ratings found in advertisements, so that they match results from normal reviews. This is, however, a precarious thing to do, as it is unclear by how much values have to offset to match, especially considering that not all advertisements have the same amount of overstatement.

Chapter 7

Conclusions

In the era of complex websites with a seemingly endless array of choices, only a computer can see the forest for the trees. A system that can analyse user provided feedback in the form of reviews will go a long way towards providing meaningful output from extracted data.

The previous chapters address a number of ways in creating a systematic way to support people in finding mobility solutions by using NLP tools. Chapter 3 speaks of various NLP tools, their advantages and disadvantages, and extraction models. In conclusion, we find that both tool and model need not be too complex as long as the output is meaningful and representative of the data. We recommend to keep both tools and the extraction model simple as the added time to process is unlikely to outweigh better quality. A good practice is to do heavy work such as annotating data first, so that it does not need to be done for each and every iteration.

Chapter 4 discusses one way of implementing a system for analysing reviews using NLP tools. It becomes apparent that, to rate mobility solutions, there are at least two components necessary: features, and a rating of those features. After all, just rating reviews is not enough; A customer needs to be able to make their preferences known and search for them accordingly. A bag-of-words model such as opinion phrases can be used for the features. Their value can be inferred by using sentiment analysis. In addition, it is good practice to also provide an overall rating by using sentiment analysis for all reviews of a car.

Finally, chapter 5 shows the outcome, strengths, and weaknesses of the envisioned system. Tests show that NLP is a solid way of systematically extracting data from car reviews, but also that there is still work to be done to make it more efficient and accurate.

Chapter 8

Future work

The output the program provides look promising, but there is still work to be done. One obvious improvement is to add more data to the database. Potentially, it may be useful to get car reviews from an external source, such as an API call to a website that provides reviews. As a result, more cars can be analysed and more features are provided for each car, leading to better results.

In the scope of a complete mobility solution comparison site, a customer needs much more than the rating of a car. For instance, a customer may want to know in what year it is built, what kind of transmission it has, how many doors the car has, or in what colours it comes. For this reason, having a database of meta data next to a database of reviews helps a customer to know more about a car and thereby supports their choice in finding the best mobility solution. Combining meta data and review data also leads to more search options, since search is now not only based on the content of reviews, but also on other properties.

In general, having more data not only leads to more cars to choose from, but also to more features per car. We expect that more data per car will dramatically improve the data and thus the features output by the system. There are, however, also consequences and concerns. First, a larger data set also leads to longer response times. This has already been addressed in the previous chapter by means of pre-annotated data, but it is uncertain how much longer it will take if the dataset grows exponentially. Secondly, it is not only about the quantity of data, but also about the quality. For example, if a review says "great car", then it does not matter if you have one, or fifty of such reviews. Finally, a larger dataset also leads to a need for more categories or at least for more subcategories. This can be either positive or negative, in the sense that more category versatility is needed, but may also lead to information overflow.

This project has been created as a part of a greater project (Jansen et al., 2017). In this larger project, it becomes more distinct what the role of the system described in this thesis is. For a customer, most of the hard work described provided here goes unnoticed. After all, it is the website itself that calls upon the API and passes parameters that suit the customer's preferences. When results have been found by

this system, it gets added to the web page. For users it does not matter how data is extracted and sorted, as long as it happens in timely and accurate manner.

References

Abbasi Moghaddam, S. (2013). *Aspect-based opinion mining in online reviews* (Unpublished doctoral dissertation). Applied Sciences: School of Computing Science.

Al Omran, F. N. A., & Treude, C. (2017). Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In *Proceedings of the 14th international conference on mining software repositories* (pp. 187–197).

Berger, A. L., Della Pietra, S. A., & Della Pietra, V. J. (1996). A maximum entropy approach. *Computational linguistics*, 39-72.

Bird, S. (2006). Nltk: the natural language toolkit. In *Proceedings of the coling/acl on interactive presentation sessions* (pp. 69–72).

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: analyzing text with the natural language toolkit.* O'Reilly Media, Inc.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J. D., ... Roossin, P. S. (1990). A statistical approach to machine translation. *Computational linguistics*, 79-85.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., ... others (2010). The youtube video recommendation system. In *Proceedings of the fourth acm conference on recommender systems* (pp. 293–296).

Daxenberger, J., Ferschke, O., Gurevych, I., & Zesch, T. (June 2014). Dkpro tc: A java-based framework for supervised learning experiments on textual data. In (p. 61-66). ACL (System Demonstrations).

Ford, H., & Crowther, S. (1922). *My life and work.* Garden City, New York: USA: Garden City Publishing Company, Inc.

Galitz, W. O. (2007). *The essential guide to user interface design: an introduction to gui design principles and techniques.* John Wiley & Sons.

Jansen, S., Voorn, P., Niemeijer, K., Parente, M., Dimovska, L., Andriessen, F. A., & Wesseling, M. (2017, May). *Mobility solutions project.* Retrieved from `mobilityozp.wordpress.com`

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing.* Cambridge: MIT press.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (June 2014). The stanford corenlp natural language processing toolkit. In

(p. 55-60).

Sandulescu, V. (2014). *Opinion phrases.* Retrieved from `http://www.vladsandulescu.com/opinion-phrases` (Online; accessed 17-May-2017)

Shneiderman, B. (2010). *Designing the user interface: strategies for effective human-computer interaction.* Pearson Education India.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., ... others (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (emnlp)* (Vol. 1631, p. 1642).

Wallach, H. M. (2006). Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on machine learning* (pp. 977–984).

Wikipedia. (2017). *Coreference.* Retrieved from `https://en.wikipedia.org/wiki/Coreference` (Online; accessed 3-May-2017)

Zhang, Y., Jin, R., & Zhou, Z.-H. (2010). Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, *1*(1-4), 43–52.

Appendix A

List of words features need to match:

```
abysmal
acceleration
adequate
amazing
appearance
attractive
awesome
awful
bad
bargain
brilliant
buy
camera
car
choice
clever
colossal
comfort
control
deal
decision
design
drive
driving
economy
economy
engine
excellent
experience
experience
feature
feel
feeling
front
fuel
gas
gigantic
good
great
grip
handle
handling
hood
innovative
interior
intuitive
issue
issues
issues
jeep
leather
look
love
magnificent
marvellous
mileage
mileage
miles
miles
miles
mirror
miserable
model
modest
navigation
nice
noise
notorious
oil
overall
performance
performance
poor
power
powerful
powerful
price
pricy
quality
quiet
radiator
reliability
ride
rim
room
safety
satisfaction
screen
seat
seats
shape
simple
size
sluggish
smooth
smoothness
sound
sound
sound system
space
speed
steering
stereo
storage
styling
suv
system
technology
terrible
terrific
torque
towing
trade
trade-off
transmission
truck
trunk
value
van
vehicle
versatile
very
visibility
warranty
wheel
wheels
wonderful
worthy
```

Appendix B

Scenario 1 Franks wants to buy a high quality car for a good price/value. He does not care much about specifics as long as it is a good overall car. Franks is also an application developer and uses the API. From the command line he runs:

```
java -jar "Car Reviewer V2.jar" api Quality price Value
```

The output looks like this:

```
Categories interested in: [Quality, price, Value]
Cars interested in : []
Cadillac xts 2016
Quality: 5.0
Average: 5.0

Mercedes-benz g-class 2016
Quality: 5.0
Average: 5.0

Mercedes-benz e-class 2016
Quality: 5.0
Average: 5.0

Kia sorento 2017
Quality: 5.0
Average: 5.0

Ford transit-connect 2016
Value: 5.0
Average: 5.0
```

As you can see, all cars look excellent to Frank and because he did not have too many demands, this is only logical.

Scenario 2 Let's make it a bit more complicated. Mark, Frank's brother, is looking for a very specific car. What he cares about most is acceleration, control,

handle, transmission, look, engine, seats, safety, mileage, price, and value. The input and output for him will looks as follows:

```
java -jar "Car Reviewer V2.jar" api acceleration control handle transmission look engine seats safety mileage mileage price value
```

```
Categories interested in:  [acceleration, control, handle, transmission, look, engine, seats, safety, mileage, mileage, price, value]

Cars interested in : []
Ford fusion-hybrid 2016
Value: 5.0
Average: 5.0

Volkswagen passat 2016
Value: 4.0
Average: 4.0

Hyundai tucson 2016
Value: 4.0
Average: 4.0

Volvo v60 2016
Value: 4.0
Average: 4.0

Mercedes-benz c-class 2017
Parts: 4.0
Average: 4.0
```

Scenario 3 Now let's get into the details. Mark has done some more research and is now only interested in specific cars and a small number of categories. Moreover, he only needs a top 4 of the cars that suit his preferences best. His input is this:

```
java -jar "Car Reviewer v2.jar" api Performance look engine seats Feeling safety mileage price Overall "Bmw x5-edrive 2016" "Audi sq5 2016" "Audi q5 2016" "Audi tts 2016" "Bmw 5-series 2016" "Buick verano 2016" "Buick encore 2016" 4
```

Worth mentioning is the last 3 types of parameters, namely categories, cars, and the total number of cars in the output. In the categories there are a few parameters with a capital letter, which means that the entire category of Performance, Feeling, and Overall needs to be added instead of just one subcategory. The parameters for cars need to be in quotes if they contain spaces, but not if they are written as in the untrimmed names (bmw_x5_edrive_2016, audi_sq5_2016). The last parameter is

merely a number which represents the total number of best cars to output. Note that parameters do not need to be in any specific order.

What Mark will see is the following list:

```
Categories interested in: [Performance, look, engine, seats, Feeling, safety, mileage, mileage, price, Overall]

Cars interested in : [Bmw x5-edrive 2016, Audi sq5 2016, Audi q5 2016, Audi tts 2016, Bmw 5-series 2016, Buick verano 2016, Buick encore 2016]

Bmw 5-series 2016
Overall: 3.5
Average: 3.5

Buick encore 2016
Overall: 4.0
Performance: 3.0
Average: 3.5

Bmw x5-edrive 2016
Parts: 3.0
Overall: 3.25
Feeling: 4.0
Average: 3.4166666666666665

Buick verano 2016
Overall: 3.25
Feeling: 3.25
Average: 3.25
```