

UNIVERSITY OF TARTU  
Institute of Computer Science  
Software Engineering Curriculum

**Yevhenii Sabanin**  
**Aspect/Feature-based Evaluation of**  
**Competing Apps**  
**Master's Thesis (30 ECTS)**

Supervisor(s): Dietmar Pfahl, Faiz Shah

Tartu 2016

## **Aspect/Feature-based Evaluation of Competing Apps**

### **Abstract**

Measurement of software quality is a challenge for many companies. It is very complicated to extract and structure experience that users had. The feedback is usually too general, and it is becoming tough to figure out which problems a piece of software has and in which specific features (e.g. security problem, UI). At the same time, customers are struggling with the problem of comparing applications based on their features. There is no way for customers to know exactly which functionality works well, and which does not, in different applications.

Companies are trying to make customers provide "structured" feedback. However, feedback forms are often filled in superficially and partly or completely ignored. Since selling online is nowadays the typical delivery channel of software applications, most of the customer reviews are stored online and thus publicly available on the web (e.g. Google Play, Apple Store – for mobile software). However, automatically extracting valuable information and separating positive from negative opinions, as well as classifying software apps by feature groups is difficult. Comparing competing applications based on features they have is still a hard problem. One of the problems is the large amount of comments, which makes it difficult to keep track of the reviewers' variety of sentiments. Likewise, it is hard to figure out a summarized opinion about each aspect (also, widely used the word "feature") of the software. That is why approaches to sentiment analysis are becoming more and more popular. Much research has been done in this field, and various methods and tools have been developed and applied. Based on information extracted from app reviews, in this thesis, I tackle three goals:

1. For a given app, identify features that this software application has.
2. Identify those applications that can be considered competing apps with regards to functionality (i.e., the set of features provided).
3. Compare these applications using sentiment analysis.

**Keywords:** Natural language processing, text analysis, topic mining, feature mining, sentiment analysis

CERCS Classification: P170 - Computer science, numerical analysis, systems, control

## **Konkureerivate mobiilirakenduste erijoontel ja tunnustel põhinev võrdlev analüüs**

### **Kokkuvõte**

Tarkvara kvaliteedi mõõtmine on väljakutse paljudele ettevõtetele. Kasutaja kogemuse kogumine ja struktureerimine on väga keeruline. Kasutajate tagasiside on tavaliselt liiga üldine ning tarkvara probleemide leidmine ja seostamine tarkvara võimalustega (nt. kas tegemist on turvaveaga või kasutajaliidese probleemiga) on muutumas keeruliseks. Samal ajal näevad kliendid vaeva rakenduste võimalusepõhise võrdlemisega. Klientidel ei ole ühtegi viisi tuvastamiseks, milline võimekus töötab hästi eri rakendustes ja milline mitte. Ettevõtted üritavad sundida kliente andma „struktureeritud“ tagasisidet, kuid tagasiside vorme täidetakse tihti pealiskaudselt või eiratakse täielikult. Kuna võrgumüük on tänapäeval tüüpiline tarkvararakenduste evituskanal, hoitakse enamikku kasutajate tagasisidest võrgus ja avalikult kättesaadavalt veebis (nt. Google Play, Apple Store mobiilitarkvara puhul). Sellegipoolest on automaatne väärtusliku info eraldamine, positiivsete ning negatiivsete arvamuste eristamine ning rakenduste klassifitseerimine võimaluste rühmade järgi keeruline. Konkureerivate rakenduste võimalustepõhine võrdlemine on jätkuvalt raske ülesanne. Üks probleemidest on kommentaaride suur arv, mis teeb arvustajate hoiakute jälgimise keeruliseks. Analoogiliselt on keeruline leida koondarvamust tarkvara iga aspekti (või ka „võimaluse“) kohta. Sel põhjusel on hoiakuteanalüüs muutumas aina populaarsemaks. Selles valdkonnas on teostatud palju uurimusi ning loodud ja rakendatud on mitmeid meetodeid ja tööriistu. Võttes aluseks rakenduse ülevaadetest eraldatud informatsiooni, püstitatakse antud töös kolm eesmärki:

1. Tuvastada etteantud rakenduste võimalused.
2. Tuvastada need rakendused, mida võib funktsionaalsuse (nt. võimaluste) poolest pidada konkureerivateks.
3. Võrrelda neid rakendusi kasutades hoiakute analüüsi.

**Võtmesõnad:** loomuliku keele töötlemine, tekstianalüüs, teemakaeve, tunnuskaeve, hoiakuteanalüüs.

CERCS Classification: P170 - Computer science, numerical analysis, systems, control

## Table of contents

<b>1</b>	<b>Introduction.....</b>	<b>6</b>
1.1	Motivation .....	6
1.2	Goals.....	6
<b>2</b>	<b>Related work.....</b>	<b>8</b>
2.1	Preprocessing Methods .....	8
2.2	Frequent Features Mining .....	9
2.3	Sentiment Analysis.....	11
2.4	Other Approaches.....	12
2.5	Limitation of the State of the Art .....	13
2.6	Existing Tools .....	13
2.6.1	NLTK.....	13
2.6.2	SentiStrength.....	14
<b>3</b>	<b>Methodology .....</b>	<b>15</b>
3.1	Data Collection.....	15
3.2	Pre-processing of Data .....	18
3.3	Lemmatization and Stemming .....	21
3.4	Extracting Features.....	23
3.5	Sentiment Analysis.....	24
<b>4</b>	<b>Implementation .....</b>	<b>27</b>
4.1	Database Generation .....	27
4.2	Web Application .....	32
4.3	Usage of the Tool .....	36
<b>5</b>	<b>Testing and Evaluation.....</b>	<b>43</b>
5.1	Pre-processing .....	43
5.2	Feature Mining .....	44
5.3	Sentiment Analysis.....	47
<b>6</b>	<b>Summary and Future Work .....</b>	<b>49</b>
6.1	Summary .....	49
6.2	Future Work .....	49
<b>7</b>	<b>References.....</b>	<b>51</b>

**Appendix..... 54**

# 1 Introduction

In this section, the motivation for the research is outlined. In addition, the research goals are specified along with additional issues that have to be addressed.

## 1.1 Motivation

Nowadays selling software online is quite common. However, online shops (app stores) usually provide very limited functionality for comparing software applications (in the following we will refer to this term as “application” or “app”). The first challenge that user face is to identify competing apps in an app store. Often, applications that provide almost the same functionality may differ in particular features. For example, existing software stores (e.g. iTunes, Play Store) do not provide functionality to find GPS trackers that support heart rate, or allow for playing music during exercise, or count calories of the workout. Categories in iTunes and Play Store assemble a huge variety of application within one category. Also, application markets do not provide any information or tool for comparing apps within the same category, except ratings. However, ratings only measure the app as a whole, not its individual features. This makes the comparison of apps complicated for the customer. Potentially our work can help customers to pick application with better functionality and help developers spot features that need to be improved.

## 1.2 Goals

What we need is to figure out how to obtain valuable information from the huge number of reviews in an app store. Based on the users’ feedback we should be able to extract competing apps, extract information about different features of apps and represent this information such that it becomes easy to figure out what satisfies customers and what does not. We defined the following research goals:

1. Which features contains a particular app we have selected?
2. Which applications contain the same set of features and thus can be considered competitors?
3. In which applications was a particular feature implemented better and thus satisfied customers more?

To answer these research questions, additional issues have to be addressed. Firstly, we need to clarify where to find sources of reviews. Secondly, we must "clean" reviews from trash (e.g. smiles, extra punctuation).

Also pre-processing will include performing the part of speech (also known as "lexical category" or "lexical class") tagging for each word. It is quite important because usual features of the software are nouns and opinions about them are adjectives. Besides distinguishing the part of speech we need to perform lemmatization– extracting the root of a word from its form (e.g. walks, walked or walking will be replaced as "walk"). That will simplify sharply further analysis.

One of the main questions and the most challenging one to answer is how to distinguish different features of an app contained in the set of reviews. After extracting features, we need to find features representing the same functionality – synonymous features. The next step would be to find competing apps containing same features.

Nevertheless, we need to clarify how to distinguish people's opinions, then assess the level of satisfaction per feature, and use this information to provide an understandable and valuable overview of the comparison of an app's feature set. As a result, the user will have consistent feedback about features that were perceived positively and those that were not, compared to the competitor. The representation of output should be easy and clear and should allow the developer to understand what exactly have to be changed or improved.

## 2 Related work

In the content of this chapter, current research on this topic will be reviewed and measured. Chapter consists of few parts. First, few subchapters describe in details methods for different steps: pre-processing, feature extraction and sentiment analysis. After that, some alternative approaches will be discussed. Nevertheless, limitations of the state of the art will be reviewed and evaluated. In the end, existing software tools will be described and evaluated.

### 2.1 Preprocessing Methods

Preprocessing is a step which transforms a raw review into valuable data that is ready for analysis. This step usually includes the division of a review into sentences and of each sentence into words. Useless for analysis characters might be removed, and reviews might be filtered. In [1] and [2], the authors used delimiter tokens for parsing the text into sentences. As delimiters for sentences punctuation characters corresponding to the end of a sentence (e.g. ‘!’ and ‘.’) were used. As the authors of [1] were aiming to find features for improvement, they did not include into analysis reviews that had a high rating (i.e., more or equal 4). Moreover, in the work [3] authors used common patterns matching to avoid slang, like “u” instead of “you”, “plz” instead of “please”, typos, and repetitions. In [2], the authors performed some additional steps for further analysis. For further feature mining, they performed extraction of words based on part of speech. Using the Natural Language Toolkit NLTK [4], they tagged nouns, verbs, and adjectives. They assumed that these lexical groups mostly describe features. Also, in this work authors performed stop word removal. The standard stopword list was taken from Lucene [5]. For further analysis, they added to the standard list of stopwords application names and some general words specific to their context and goal, e.g., “app,” “fix”. At this step, the authors also implemented Lemmatization to extract word roots. Lemmatization helps to group words, which can have different forms (e.g. plural, gerund, past tense), into one common form for further analysis.



## 2.2 Frequent Features Mining

Extraction of features is a long process consisting of different steps depending on the purpose of research. For example, in [2] authors used collocation finder algorithm provided by NLTK. However, the task of extracting features includes plenty of difficulties connected to natural language understanding. Some of the features can be mentioned indirectly. For example, instead of referring directly to the size of the camera, the reviewer might say that it is "not fit in a pocket". Minqing Hu and Bing Liu in [6] ignored all of this indirect mentioning of features. Also, they did not have a goal to group features of an entity by categories.

For mining of frequent item sets in a review, association mining [7] was used. Minqing Hu and Bing Liu assumed that nouns and a group of nouns representing a product feature converge. Zhongwu Zhai, Bing Liu, Hua Xu and Peifa Jia proposed in their work [8] a way to cluster features that are represented by an entirely different group of nouns. They offered to unite synonym features as the same aspect. For example, "picture" and "photo" for the camera should be considered as one and the same feature. The grouping of features that are synonymous is also described in [2]. The authors assume that features like <pdf viewer> and <viewer pdf> refer to the same feature because they consist of the same words. In their work, the grouping of features uses the Wordnet [12] dictionary of synonyms. It allowed the grouping of synonymous features. For feature groups that include few aspects, a name of the group was the feature appearing in reviews most often, i.e. which has appeared most frequently in the set of reviews.

Grouping of features can also be done using LDA [9]. However, this approach was criticized in [3]. Topic extraction based on LDA mostly uses the bag-of-words assumption and does not analyze a sentence itself. As a result, the customer is not able to verify the result of topic mining and use it to assess the application.

In [6] Hu and Liu used different methods for frequent mining and pruning. They downloaded reviews for various products and manually found features in there. They compared the result of the algorithm at each step with their manual selection. The results are shown in Table 1.

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

Table 1. Recall and precision at each step of feature generation [6]

Using the CBA association miner, which is based on Apriori algorithm [7] with minimal support of 1%, they found out that not all frequent items were features. That is why after obtaining the results shown in the “Frequent features” column of Table 1, they used two different pruning algorithms to cut frequent item sets, which are not featured. As the Apriori algorithm does not influence the position of words in a sentence, compactness pruning was used. This method allows for reducing the number of frequent itemsets if the words within it do not appear in the same order. For the next pruning, p-support (pure support) was used. P-support pruning allows reducing features that are a subset of other features or do not have the meaning. For example, life makes sense only in a context of battery life. More explanation on p-support pruning can be found in [6].

In [10], the authors point out that product feature extraction as proposed in [6] is fully automated and unsupervised but not anchored from a semantic point of view. Consequently, it is not able to remove non-product features and features that are opinion-irrelevant. In [10], the authors propose semantic-based feature extraction. Like product feature extraction, semantic-based feature extraction performs extraction of common features and two types of pruning. However, in addition, it also performs semantic-based refinement. In [10], the authors collected all adjectives from General Inquirer [31] from both groups: positive and negative. That allowed knowing the sentiment of the feature in advance. Then they process each subjective opinion word independently to find the closest noun that is supposed to be a feature. However, this approach does not work all the time correctly. Example, provided in [10] is the sentence "The picture quality is not rich in color". The closest noun to subjective adjective "rich" would be "color". However, the word

"quality" is the real one that needs consideration. That is why authors took the first word after the subjective adjective, and if it is not a noun, they started to mine noun phrases appearing before the adjective. Also, there is a possibility to extract features based on conjunction. For example, if we have a sentence like "Camera has a good screen, quality, price", authors assumed that knowing that "good screen" is already a feature, can help us to determine "quality" and "price" as features as well.

In [2], the authors also used topic modeling to produce a summary of results. Latent Dirichlet Allocation (LDA) [9] was used for grouping of features. Using this method each review has some probability to belong to a certain topic. Themes in this method are described by a set of words that happen to occur more often. It is worth mentioning that the authors did not use a standard approach and did not extract vocabulary from whole reviews. Instead, they used a vocabulary based on extracted features. Therefore, features consisted of two words (e.g. "pdf\_viewer") was counted as one term for LDA.

### **2.3 Sentiment Analysis**

The next step is to find opinion words. Based on [11], Minqing Hu and Bing Liu use the presence of adjectives in a sentence as a marker that a sentence might contain opinion about the feature. Besides, a sentence could be classified as valuable only in the case when it describes an opinion related to a specific feature. After extracting opinion words, the next important step is to find the semantic orientation of the word, i.e., whether the opinion is positive or negative.

Hu and Liu faced the problem that dictionaries and systems, such as WordNet [12], do not provide semantic information. Moreover, some adjectives might have positive or negative opinion dependently on context. For example, the word "small" does not allow for making a conclusion about the feature itself. In [6], the authors propose to use WordNet's organization of adjectives that provides information about synonyms of adjectives and antonyms. Based on this, they managed to determine the semantic orientation of an adjective. For this, they looked up synonyms and antonyms of adjectives for which the semantic orientation is known. Minqing Hu and Bing Liu started from predefined 30 adjectives and in the process of mining, the system added new adjectives on its own with their semantic orientation.

Also, it is worth to mention the approach proposed by Wei Wei and Jon Atle Gulla [13]. In this work, authors analyzed reviews on cameras. They proposed grouping of features and creating of sentiment ontology tree for features. This method solves the problem of the overlapping of reviewed features. Overlapping happens when features are synonymous or include into each other. For example, “noise” and “resolution” are sub-attributes of “picture quality”. This work is highly related to our goal of a synonymous feature grouping. Authors propose a tree-like structure representing aspects of the product. On top of this representation Wei Wei and Jon Atle Gulla proposed an algorithm that tackles sentiment analysis.

Moreover, in [6], the authors propose a way to find features that are not included to the frequent item set, but which might be interesting for developers and users. Therefore, they move from opinion words to the closest noun. From this, the authors managed to obtain infrequent features, which represent up to 20% of the total amount of features.

The next step is to predict the orientation of opinion sentences based on the orientation of opinion words. Authors used term *effective opinion*. The most *effective opinion* considered is the one that is closely located to a feature in the sentence. Based on the sentiment of *effective opinion* Hu and Liu in [6] determined the sentiment orientation of the whole sentence. Based on their practical results the method of using effective opinion shows high performance. The result Hu and Liu achieved was represented as follows: each feature has two associated sets of sentences, each set containing only sentences with the same semantic orientation (i.e., positive or negative).

## **2.4 Other Approaches**

The number of approaches that have been proposed for extracting features, sentiment analysis is huge. In the most recent work in this field [14], the authors reviewed most of the existing methods in this area and grouped them by the problem they solve.

Processing of user reviews is a very complex task. As stated in [14], some researchers prefer to concentrate on solving sub-problems without trying to resolve the whole problem. While this approach does not address the problem as a whole, it might increase performance in some cases. The first idea, called “comparative opinions”, is to find in the

text related words (with the help of WordNet [12]) and use them to identify the user's opinion about a feature.

The second idea, mentioned in [14] is reviewing of conditional sentences. The sentences were grouped into four categories; then support vector machine was applied to identify semantic of the adjective. Based on this, adjectives were weighted based on their distance to the topic. The most interesting outcome of this approach is that the most successful classifier was the one, which tried to classify only consequent part, not a conditional one. However, the whole-sentence classifier performed much better than only-conditional-part classifier. Next research analyzed in [14] concentrated on negation and other classifiers. Negation is extremely interesting for sentiment analysis because it can reverse to opposite meaning of the sentence. Also, other methods like irony and sarcasm can totally change the significance of the sentence (for example overly positive or negative expression change meaning of the sentence to opposite).

## **2.5 Limitation of the State of the Art**

The state of the art provides useful methods for analyzing reviews and extracting users' opinions. Feature extraction is described well in [14] and [2]. Different approaches and patterns for aspects extraction along with sentiment analysis provide a good basis for extracting user opinion. However, most of the research concentrates only on analyzing features and feedback about one single application. Tools and approaches for generating competitors and performing feature-base analysis for sets of similar apps have not been described. As a result, users only learn about the strengths and weaknesses of one application and are not provided with information about other implementations of the same aspect of an app.

## **2.6 Existing Tools**

In content of this part, the functionality of existing tools will be described.

### **2.6.1 NLTK**

For the purpose of analyzing, one of the most useful tools is Natural Language Toolkit for Python [4]. It implements functionality for processing natural language and aggregates

interfaces for more than 50 dictionaries and language resources like WordNet [12]. This tool also helps to perform stemming, part of speech (POS) tagging, extracting the semantic meaning of words and implementation for working with natural language processing (NLP) libraries. NLTK is open source and based on community support. Worth mentioning that as part of NLTK WordNet Lemmatizer is provided.

### **2.6.2 SentiStrength**

SentiStrength [15] is a tool for counting sentiment outcomes of sentences. It processes sentences and evaluates them based on an internal database of adjectives and “booster words” that can increase or decrease the level of evaluated sentiment. It mostly works at the sentence-based level. It can return evaluated feedback in a different format (binary, trinary, with marks from -4 to +4). Tool has been used for social medias like twitter and quality of output was measured in [16]. In [17], it has been found that 80.4% of the reviews in App Store contain less than 160 characters so that SentiStrength can be used. This tool was evaluated in plenty of academic articles which can be found on the official page of the tool [15].

### 3 Methodology

In this chapter, the methodology will be presented. Firstly, the source of reviews will be discussed and explained. There will be given an introduction to the database used for developing the tool. Secondly, theoretical background will be introduced. Will be explained algorithms used in the tool, introduce the main principles of them and give motivation why each of this algorithm had been chosen.

#### 3.1 Data Collection

Obtaining reviews database became quite challenging. At this stage, three main possible sources of application reviews have been considered: iTunes, Google Play, and Windows Store. Unfortunately, the amount of feedback in Windows Store for each application does not allow performing any deep analysis. From the other side, iTunes does not allow to fetch reviews. During research for the source of data, database by Kon Mouzakis from the Swinburne University of Technology in Melbourne, Australia [18], was found. A number of applications and review was considered sufficient to perform the analysis. The database contains 106539 reviews from iTunes for 25 different applications, the language of reviews is English. Reviews text is raw and requires some cleaning, for instance, cleaning of smiles, extra punctuation. Database provides all necessary information for research about review and contains columns:

- Application Identifier – unique identifier of the application, which corresponds to the id of application in iTunes.
- Review Identifier – unique identifier of the current review.
- Date of Review
- Title Word Count, Title Character Count, Body Word Count, Body Character Count – counters of words and characters in title and review itself responsively.
- Rating – rating that reviewer set for this application.
- Author Handle – name of the author
- Author Identifier – unique identifier of author
- Review Title Text, Review Body Text – raw title and review text responsively

For our research, only a few of these fields was considered significant: Application Identifier, Body Word Count, and Review Body Text. Other columns were marked irrelevant for our purposes. Also, in our work we do not concentrate on giving a general rating of application and interested more into comparing features of it, that is why column Rating irrelevant for this research.

Manual checking of applications, mentioned in the database, showed that some of the apps are not presented in iTunes anymore. We removed these applications from the database because there was no possibility to verify what kind of application it is and prove that it might be competing app. Therefore, after cleaning and manual reviewing, the database contained 87904 reviews for 19 apps. Number of reviews for each application shown at Figure 1.

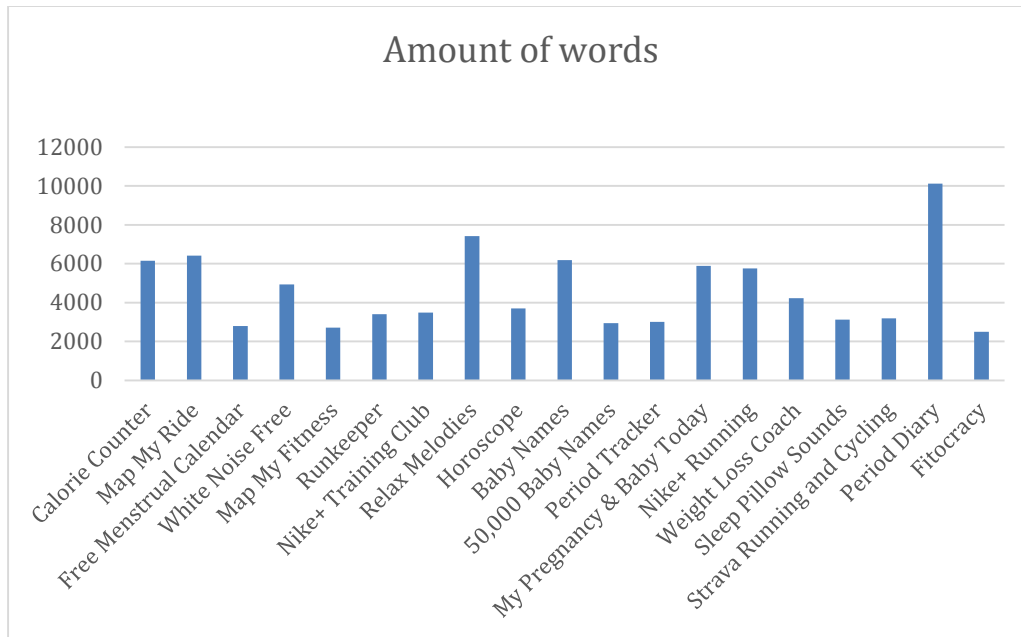


Figure 1. The number of reviews for each app

Reviews in the database have good quality, contains many words and have an opinion not only about the app in general but specific aspects of it as well. Next two charts (Figure 2 and Figure 3) show average length of review in words, and the number of unique words in all reviews for each app.



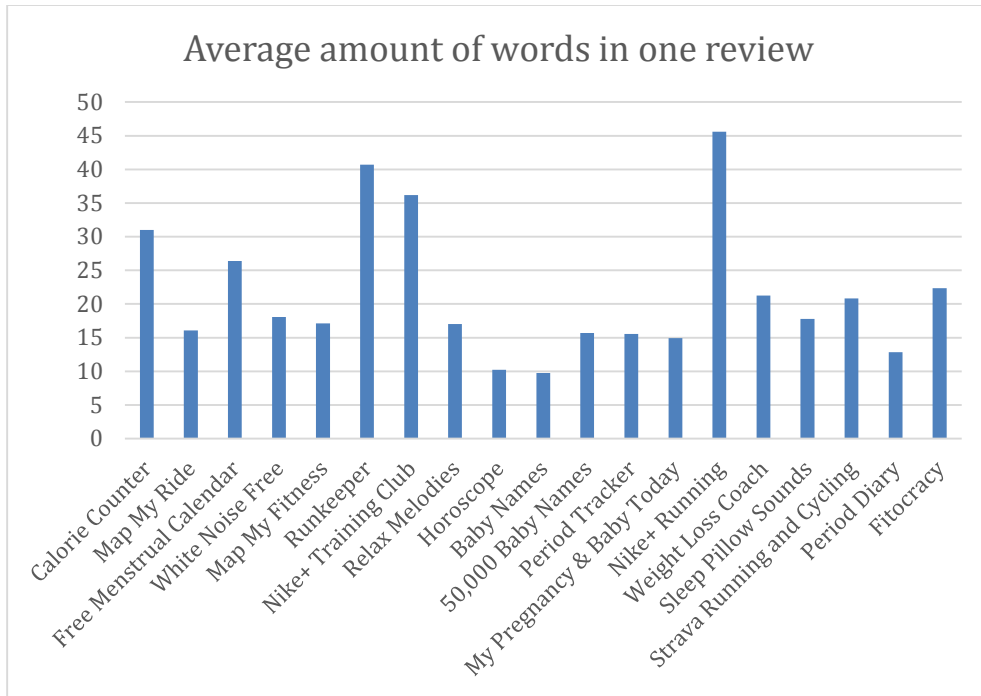


Figure 2. Average amount of words in each review

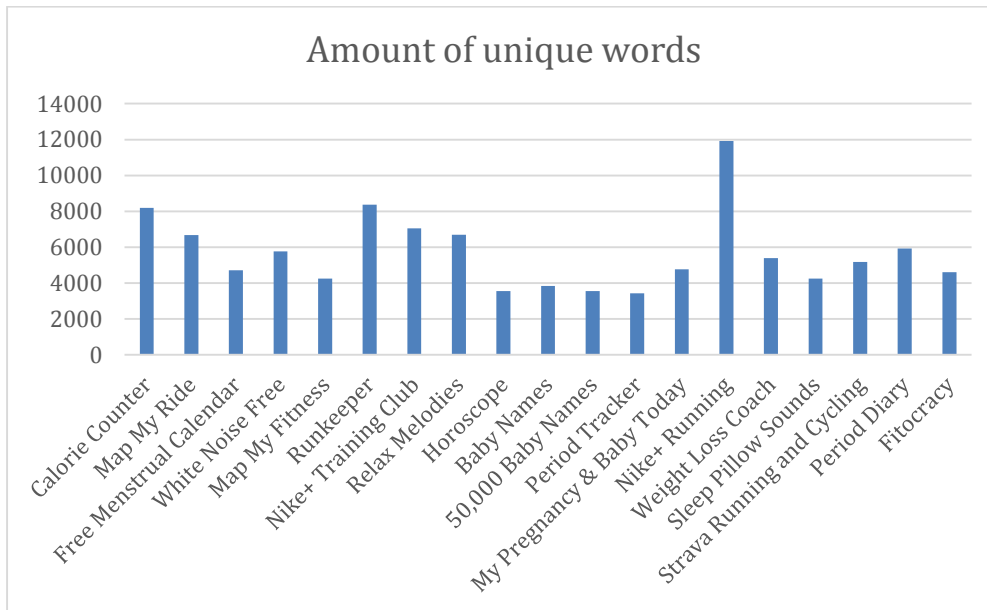


Figure 3. Amount of unique words

Also, manual checking helped to verify that there are few groups of applications in the database which can be considered as competitors, for example, Runkeeper and Nike+ running – both are GPS trackers which help the customer to record run and provide some extra features on top of it.

## 3.2 Pre-processing of Data

The first step of processing data was cleaning of reviews. Reviews which database contains are given in raw format. That means that no pre-processing had been done before. Therefore, it is very hard to perform any further analysis. Pre-processing includes few steps.

First, a division of reviews by sentences happens. Further stages of the algorithm do not concentrate on the whole review but concentrates more on sentence-based analysis. The reason for that is: when a customer talks about the feature, he describes his opinion inside a sentence. In work [6] authors used the method of “effective opinion” which means that closest adjective to feature mostly contains opinion about this feature. That is why further sentiment analysis will measure sentence with the feature, and extract customer opinion from that, rather than analyze the whole review as one. NLTK [4] sentence tokenizer was used to perform separation by sentences. One of the main profit of using NLTK tokenizer is that it does not only help to separate raw text by sentences, but also it keeps original punctuation like question marks and exclamation marks, which is necessary for sentiment analysis. The extra punctuation allows performing more accurate sentiment analysis due to better representing of customers emotions.

Secondly, a division of each sentence by word-tokens happens. Word-tokens are not only words itself, but it also includes punctuation and numbers as separate tokens. Every word is lowercased, so in future analysis words like “Calorie” and “calorie” would not be counted as different words. This step also includes two highly important tasks for further analysis:

1. **Part of speech tagging.** As was specified before, features of the application are mostly nouns or group of nouns. From the other side, opinion is usually contained in adjectives. Therefore, to distinguish words that might be a feature, part of speech tagging needs to be performed. Part of speech tagging tool from NLTK python library was used. NLTK uses dictionaries to find out the lexical group for word. NLTK part of speech tagging is working so that it can determine the right lexical category of a term based in the context of the sentence. For example, in sentence “*I wish it would also work with my nike+ run*” it will determine word “*run*” as a noun

and not as a verb. Moreover, NLTK tagger can guess part of speech even if it cannot find a word in the dictionary, based on the context. For instance, in the example above word “nike” will be tagged as a noun. Part of speech tagger also consider tokens like “.”, “!”, etc. This tokens all tagged like dot “.” which corresponds to the end of sentence punctuation.

2. **Removing of “stopwords”.** Stopwords are those words that happen to appear very often in reviews but do not provide any information. Here is important to notice that word tokenization is used only for feature extraction, but not for sentiment analysis. That is why the list of stopwords can have negotiations, some common adjectives, and verbs. For this purpose standard list of stopwords provided by Lucene [5] was used. Stopwords list contains: articles (e.g. “a”, “the”), common verbs (e. g. all possible forms of verbs “to be”, “have”, “can”) some modal verbs and words with different part of speech which will be useless for further algorithm (e. g. “anyway”, “lot”, “keep”, “same”). Before using of stopwords list, it was manually reviewed to exclude stopwords which might be part of app feature. Nevertheless, some extra stopwords have been added for our specific purposes. These stopwords were inserted after all steps of the algorithm. Words that occur in the final results very often without referring to real features was added to the list. The word “app” has been added because it refers to the very general description of the app and does not refer to a specific aspect. Also the words “love” and “fun” was added, because in this case it mostly describes customers feelings or emotions, however, by part of speech tagging algorithm this words often appeared to count as nouns. This words was added to stopwords list and was excluded in aspect detection to forbid possibility for them to stand in the feature list.

It is highly critical to implement last two steps in the order specified above. The reason for this is that after stop words removal, the context of the sentence might change and lexical categories tagging might work wrong. For example, word “to” is present in stopwords list, however, removing it from sentence might alter the tag of next word from the verb to the noun. After performing these two steps, the output would be separate lowercased sentences with part of speech tags for every token. Nevertheless, NLTK provides information about each possible tag it uses. For example, JJ corresponds to an adjective, RB to an adverb, VB

to the base form of the verb. For feature extraction, we are interested in keeping words with following tags:

- NN - singular noun
- NNS – plural noun
- NNP – singular proper noun
- NNPS – plural proper noun
- VB – Verb base form,
- VBD – Verb, past tense
- VBG – Verb, gerund or present participle
- VBN – Verb past participle
- VBP – Verb, non-3rd person singular present
- VBZ - Verb, third person singular present

As output we have lowercased sentence and part of speech for each of token in sentence:

<p><i>“numerous studies have shown that the only way to lose weight and keep it off is to maintain a food diary.”</i></p>	<pre>[[{'pos': 'JJ', 'word': 'numerous'}, {'pos': 'NNS', 'word': 'studies'}, {'pos': 'VBN', 'word': 'shown'}, {'pos': 'VB', 'word': 'lose'}, {'pos': 'NN', 'word': 'weight'}, {'pos': 'VB', 'word': 'maintain'}, {'pos': 'NN', 'word': 'food'}, {'pos': 'NN', 'word': 'diary'}, {'pos': '.', 'word': '.'}]]</pre>
---	---

Table 2. An example of output after pre-processing step.

In the Table 2 it is easy to notice that words “have”, “that”, “the”, “only”, “way”, “keep”, “and”, “it”, “is”, “to”, “a” has been removed due to presence in stoplist. Other words remain (for example, “food” and “diary” that are both nouns and possibly represent a feature of application).

### 3.3 Lemmatization and Stemming

For the further analysis, it is crucial to have each word in the same form for all reviews. On the previous step of the algorithm, every word has been put to lowercase, so our algorithm would consider words “Feature” and “feature” as the same word – this process called normalization.

At this step, our task is to find an infinitive version of word and use it for further analysis. For example, in third person singular, most of the verbs end with “-s” (e.g., “uses”, “tracks”). Nevertheless, nouns in plural also end with “-s” or “-es”. Therefore, features “calorie counter” and “calories counter” will be counted as different features, which is, of course, not true. Our goal is to decrease the possible amount of various forms and use one common form for each related forms. For instance:

- “Use”, “uses” should become “use”
- “Calorie”, “Calories” should become “calorie”

Two possible solutions were considered to target this issue: stemming and lemmatization.

Stemming is a fast method that simply cuts away the ending of a word, expecting that the result will be acceptable in most of the cases for further analysis. Stemmers are language-specific. However, it does not require a full vocabulary to perform the operation. Many algorithms have implemented stemming. Different algorithms can produce different output for the same list of words and can target different situations better or worse, compare to others. Therefore, stemming cannot be used to identify the morphological root of the word.

Lemmatization is usually using vocabularies to identify the word and perform morphological analysis to determine the root of the word. The result of lemmatization is called lemma.

Two types of stemming were used: Porter stemming and Lancaster stemming. For lemmatization, WordNet [12] Lemmatizer was used. Some examples of the algorithms’ outputs are shown in Table 3.

Original word	Porter Stemmer	Lancaster Stemmer	Lemmatizer
calories	calori	cal	calorie
time	time	tim	time
iphones	iphon	iphon	iphones
tracking	track	track	tracking

Table 3 Example of Stemming and Lemmatizing

For making a decision on which algorithm is better for further analysis, we perform a manual comparison of common words . The most common words for each application were chosen, and manual comparison of results has been made. In the most of the cases, Lemmatizer showed much better output result. Even though WordNet Lemmatizer used vocabulary and showed better output result, there are some issues:

1. WordNet Lemmatizer works very poorly with proper nouns. For instance, it was unable to make singular from “iphones” to “iphone”. On the other hand, stemming algorithms was able to do from both words “iphones” and “iphone” the same output “iphon”.
2. WordNet Lemmatizer sometimes counts words of different type of speech as the same lemma. For example, it considers noun “track” and verb “to track” as the same lemma “track”. However, it identifies present participle or gerund as different lemmas, for example, “tracking”. The most of these issues were eliminated in a further step of synonym identification.
3. Lemmatization is a time-consuming procedure, compared to stemming. One of the main advantages of stemming is that it works much faster, mostly because it use simple patterns to cut away ending of the words. However, time was not a criterion for choosing one approach over other.

Finally, lemmatization is performed only for those words that could be possibly part of a feature.

### 3.4 Extracting Features

Feature extraction is done using collocations. Collocations are words that often appear next to each other in one sentence. For our purpose, we only concentrate on collocations of two words. For our purposes, we use NLTK collocations. This tool helps to extract collocations from the sentence, the output of this step give all possible combination of words in a sentence. We only concentrate on collocation of words with part of speches specified at the end of sub-section 3.2.

Our input at this step would be a sentence represented as follows:

```
[{'pos': 'JJ', 'word': 'numerous'}, {'pos': 'NNS', 'word': 'studies'}, {'pos': 'VBN', 'word': 'shown'}, {'pos': 'VB', 'word': 'lose'}, {'pos': 'NN', 'word': 'weight'}, {'pos': 'VB', 'word': 'maintain'}, {'pos': 'NN', 'word': 'food'}, {'pos': 'NN', 'word': 'diary'}, {'pos': '.', 'word': '.'}]
```

From this set of words we extract only some parts of speech and perform collocation searching. As output at this step we would have:

```
[('food', 'diary'), ('lose', 'weight'), ('maintain', 'food'), ('shown', 'lose'), ('study', 'shown'), ('weight', 'maintain')]
```

As one can see, word “numerous” was not considered as part for any collocation because it is adjective. Also, worth noticing that the word “studies” became “study” after lemmatization.

After all possible collocations have been extracted we need to verify that there are no synonymous collocations. The problem is that people often talk in a review about similar features using different words. To target this issue, synonym comparison of features needs to be performed. We would consider two features to be synonymous if both words are similar or the same with two words of other feature.

For example, features “count calorie” and “calorie count” are synonyms, because they consist of the same words. All features “calorie exercise”, “calorie work“, “calorie workout” are synonyms, because the words “exercise”, “work” and “workout” are synonyms. However, “calorie count” and “calorie exercise” are not synonymous, because the word “count” is not a synonym of the word “exercise”. That approach allows grouping

of features with the same meaning to the group and to count them as one. For synonym extraction, WordNet [12] vocabulary is used.

The label for this feature group is the feature with the higher frequency. For instance, if feature “workout music” appears 67 times in reviews and “exercise music” appears 36 times, then the common name for these two synonymous features becomes “workout music”.

After all collocations have been found and grouped by synonyms, we are only left with those that happen to appear in more than three reviews. We choose a threshold to avoid words that have occurred together by accident; the same approach was used in [2]. As a result we have a list of collocations and their frequencies, united by synonymous groups (Table 4). In the left column of the table are all the features with the same meaning (synonymous features), in the right column is the most frequent collocation from the left column, which is the name for this aspects group.

[('great_ride', 30), ('ride_great', 23)]	('great_ride', 30)
[('relax_sleep', 74), ('sleep_relax', 21), ('relaxing_sleep', 19), ('restful_sleep', 12), ('sleep_relaxing', 11)]	('relax_sleep', 74)
[('track_cycle', 496), ('track_bike', 33)]	('track_cycle', 496)

Table 4. Features table

Having collocations for each sentence and features table is enough to implement next step of the algorithm.

### 3.5 Sentiment Analysis

Sentiment analysis helps to figure out the actual opinion about a feature. We run sentiment analysis against sentences with a known feature in it. Sentiment analysis usually works by analyzing adjectives in a sentence and check for each adjective how much positive or negative score it produces. However, sentiment analysis may also use some extra information like smiles, extra punctuation. That is why sentiment analysis should be run for sentences without normalization, lemmatization and part of speech tagging.



For the purpose of sentiment analysis, the tool SentiStrength is used. SentiStrength analyzes each sentence and returns two values for each sentence: negative emotions rating and positive emotion rating. Negative emotion rating has a scale from -1 (neutral) to -5 (strongly negative). Positive emotion rating has a scale from 1 (neutral) to 5 (strongly positive). Each aspect gets a score equal to total sentiment value (both positive and negative) of the sentence. The sentence is considered as positive if the absolute value of positive score is greater than the absolute value of the negative rating. Otherwise, the sentence is counted as negative. In the case of equality, the sentence is counted as negative. An example of SentiStrength output specified in Table 5.

Positive	Negative
i love entering my food and exercise and i find myself wanting to go exercise just so i can enter it in! [+4][-1]	the food and exercise databases need to be expanded and a little more extensive. [+1][-1]
great food and exercise tracker. [+3][-1]	in real life, loss is much slower than predicted by this app based on the food and exercise you record. [+1][-3]
i like this app its very fun and easy to understand and has a lot of exercises to enter along with lots of different foods. [+3][-1]	it does not work all the time which can be annoying. [+1][-3]

Table 5. Example of SentiStrength output

As SentiStrength by itself is able to recognize sentences and not normalized words (without lemming), it was run against uncleaned reviews. Moreover, SentiStrength uses emphasis in punctuation (like exclamation marks, emoticons), so running it against unclean sentences provide us more accurate sentiment analysis.

All data, collected at previous steps and the output of SentiStrength collected all together allows for generating data that will be used in the tool. Generated data includes:

- Application ID – identifier of application in iTunes
- Comment – Sentence from review of this application
- Feature – feature that contains in this application
- Frequency – Frequency of feature all over reviews
- Main feature – name of the synonymous group this feature belongs
- Negative – negative sentiment score of the sentence
- Positive – positive sentiment score of the sentence

If a sentence has more than one feature, this sentence will appear in the data as often as the number of features is it contains. An example of a data table is shown in Table 6.

Application ID	Comment	Feature	Frequency	Main feature	N	P
287529757	it is gets frustrating trying to enter new foods.	enter_food	130	enter_food	-3	1
509253726	not sure how fitocracy calculates points, but it's a great motivator to do a more challenging workout.	great_motivator	131	great_motivator	-1	3
466847531	lots of helpful tips!	helpful_tip	79	helpful_tip	-1	3

Table 6. Final data example (N – negative sentiment score, P – positive sentiment score)

## 4 Implementation

In content of this chapter, the implementation will be described. This block will contain information about technical background, tools that have been used and all the architecture decisions. Generating of data and all steps of the algorithm, which was specified in chapter 3 were done using Python. For user interface part and easy interaction with data Java has been chosen. For persistence layer, PostgreSQL was used. The content of chapter divided into two parts. The first part describes database generation using Python, NLTK [4] library and direct working with persistence layer. The second part describes Java web based application and concentrates on main views, REST API available and JavaScript libraries which were used to represent feature-based comparison. Finally, the usage of the tool will be shown. All views of the tool will be shown and explained.

### 4.1 Database Generation

The Python part consist of few scripts which run in certain order. The output of one script usually is input for the next one. This modularity helped to verify at each step quality of result and provided easy access to change parts of the algorithm without interrupting into others. Source code can be accessed in the references in [20].

The first script is *preprocess.py*. It is responsible for few initial steps of data processing:

1. Using CSV module of python, it obtains review record from raw data. From each record, it extracts only required information: Application Identifier and Review Body Text.
2. It tokenizes raw review into sentences using NLTK sentence tokenizer. Also at this step, each review is lowercased.
3. For each sentence in a review, using NLTK word tokenizer, it cut the sentence by words. Nevertheless, each word is checked if it belongs to the list of stop words.
4. For each word, it applies NLTK part of speech tagger to identify to which lexical category word belongs.
5. Finally, it saves all output of this step in the table “reviews” using the *psycopg2* module for working with PostgreSQL database. Each record of database corresponds to one sentence of review and has a unique identifier, an application identifier, the sentence itself, and dictionary of words with respective lexical class.

The second script is *colocations.py*. It works with “reviews” table, created with the previous script and as a result produce collocations, which potentially can be considered as features. Nevertheless, during extracting of aspects this script needs to perform critical tasks:

1. For each word of the sentence it checks the lexical class of the word and check if feature might contain this part of speech. All words that contains other lexical categories do not consider for generation of collocations.
2. Every word from the previous step is lemmatized using WordNetLemmatizer. At this point, we get base forms of the term and ready to implement collocation extraction
3. All set of base forms for each sentence is passing to BigramCollocationFinder. BigramCollocationFinder is part of NLTK, which provides all necessary functionality for bigram generation. A bigram is collocation of two words, which happens to occur together. Bigrams are all possible collocations; however, not all of them end up being a feature.
4. Nevertheless, each sentence is going through SentiStrength library to produce sentiment scores of the sentence. SentiStrength is used as separate Java Archive File (JAR). Each sentence is processed by SentiStrength and measured by two parameters: positive and negative sentiment score. SentiStringth uses vocabularies; however, it also provides functionality to manage some of them. This setting contains in text files of SentiStrength\_Data folder:
  - a. BoosterWordList.txt contains words which provide extra points for adjectives or reduce the amount of score for the adjective. For example, word “extremely” will give additional 2 points to adjective value (either positive or negative).
  - b. EmoticonLookupTable.txt contains list of Emoticons and its value for sentiment scores. For instance, emoticon “:)” gives +1 to positive sentiment value.
  - c. EmotionLookupTable.txt contains a list of words (mostly adjective) with its positive or negative sentiment score.

- d. EnglishWordList.txt provides a list of words without any additional values for sentiment score for using this words.
  - e. IdiomLookupTable.txt contains idioms and sentiment scores for each of it.
  - f. NegatingWordList.txt contains short forms of few words, like “wouldn’t”, “aren’t”. It does not include any information how each of it influent to sentiment score.
  - g. QuestionWords.txt contains question words like “how”, “who”, “why”. It does not include any information how each of it influent to sentiment score.
  - h. SlangLookupTable.txt contains a table which helps to resolve some slang words and idioms and contributes to resolving it to literature synonyms. For example, “btw” resolves to “by the way” and “otoh” resolves to “on the other hand”.
5. In the end, for a sentence with a not empty set of bigrams, all produced collocations are saved to PostgreSQL database. The separate table “colocations” has been created and each record contains a unique identifier, an application identifier, raw text of the review, all possible collocations in this sentence, negative sentiment score, and positive sentiment score.

As was specified above, not all collocations are going to be counted as a feature. Nevertheless, some of the collocations are synonyms between each other. That is why frequency for each collocation should be counted, and synonyms features should be grouped. *Synonym\_looking.py* does both of these tasks:

1. Working with “colocations” table, it reads bigrams generated with the previous script. For each collocation, algorithm counts how many times it appears all over reviews and generates the feature frequency dictionary.
2. The pool with collocations created. Only collocations with a frequency above a certain threshold are going to be presented in the pool.
3. For each collocation, it checks if synonyms of this collocation exist. For each word of bigram, all synonyms are generated. Nevertheless, the word itself is also stored in synonyms list. After that, the algorithm is going through all collocations in the pool and check in pairs words to be in the list of synonyms. To access a list of synonyms for each word, WordNet module from NLTK

Corpus used. As output, it provides a list of synonymous names of lemmas for word. There are two checking performed: the first one with a straight sequence of words and the second one with reverse. The reason for this is that collocation with different word order, but same meaning, are considered to be a synonymous feature. The synonym collocation finding process is described in Figure 4. Arrows represent belonging to word to the list of synonyms. Conditions, which represented by arrows of the same color, should happen simultaneously for two features to be counted as synonyms. However, fulfillment of both (red and black) condition sets is not required.

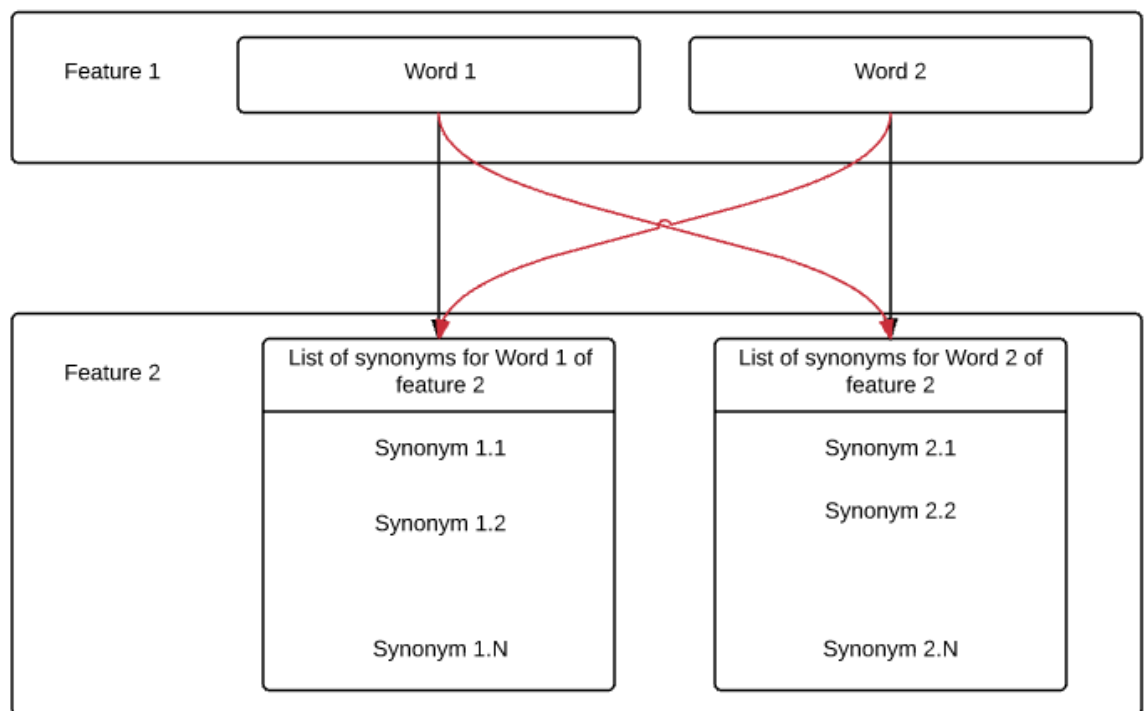


Figure 4. Synonym looking procedure. Black arrows represent condition with a straight sequence of words, red arrows – reverse sequence.

After all synonyms for collocations features had been found, all the features are grouped. The name of the group is represented by feature with maximal frequency all over reviews. In case, if there are no synonyms exists for the feature, a group with one collocation is created. All features that included in the group are deleted from pool and procedure repeats until no bigram left in the pool.

4. All groups, created at the previous step, are recorded into “synonyms” table using `psycopg2` module. Each row of table consists of features list with frequency for each feature and main feature that represents group (feature with maximal frequency)

Finally, all data is from tables collected and passing to the final table. This happens through REST API calls to Java application and performed by `final.py` script:

1. The script is going through all records of table “synonyms” and creates a mapping table between feature and group name. Therefore, there will be no need to check through all synonyms table every time to which group feature belongs.
2. From every sentence of “colocation” table, it gets collocations and checks it for presence in the mapping created at the previous step.
3. Finally, it gets all other required data from “colocation” table (sentiment scores, application id) and creates REST API call to Java service. POST request is created using Requests library [19].

The total lifecycle of data at persistence level is shown in the table below (Figure 5):

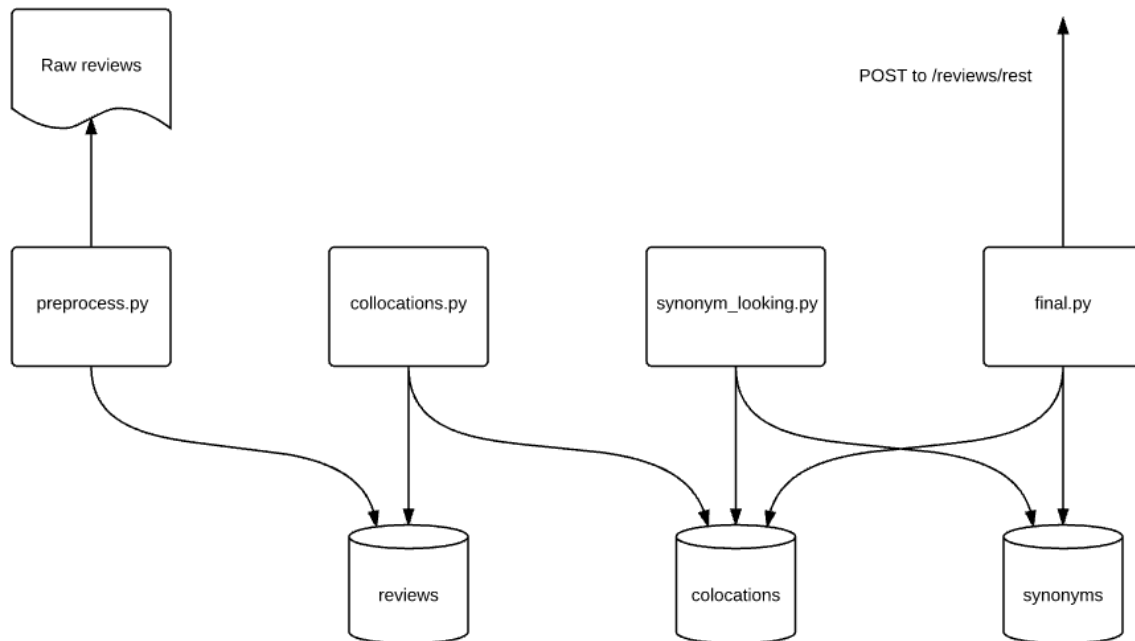


Figure 5. Database generation process.

## 4.2 Web Application

The web application provides the possibility to compare apps based on chosen features and generates views that summarize data obtained during generation. The tool allows for choosing one software application, obtaining a list of features and finding the closest competitors based on the feature. Moreover, it provides functionality to compare the chosen application with its competitors. It also offers filters that allow for optimizing results presented to the user. The web application was developed using Java Spring Framework; persistence level relies on PostgreSQL and for views generation, Thymeleaf [21] template engine was used. The tool implements the Model-View-Controller pattern. The application works with two tables from databases: “data” and “apps”. For accessing the persistence level in Java, JpaRepository was used. The source code can be found in Bitbucket repository [22].

The “data” table contains all information for each sentence. Worth noticing that at the step of data generation in python, script *final.py* does not write into the database, but instead, it makes API call to our web application. The reason for this is because we wanted to have the possibility to run Web Application remotely in the cloud, but have all data generation locally. That would allow sharing the result of the production with the world, but still being able to make some changes and testing concerning data generation.

The “apps” table consist of all necessary information about the app itself: Application identifier in iTunes, Name of the app and link to the image. This data is also can be filled using API calls: this was done manually, using Postman tool. For each application from the list, it was found in iTunes; required information was extracted and passed through API calls to Web Application. Finally, it was saved in the database.

There are two controllers in the application: one is responsible for view calls (*ViewController.java*), and other (*RestController.java*) for REST API calls.

At Figure 6 shows a high-level architectural view of the created Java Web Tool. Initially, data is obtained via POST calls to *RestController* by *final.py*. This request contains data that was generated and that needs to be saved in the “data” table. *ViewController* reads



data from both tables of the database and returns views to render. Views do also issue REST API calls to RESTController to obtain valuable information (mostly regarding graphs data and filtering of content based on frequency and sentiment level).

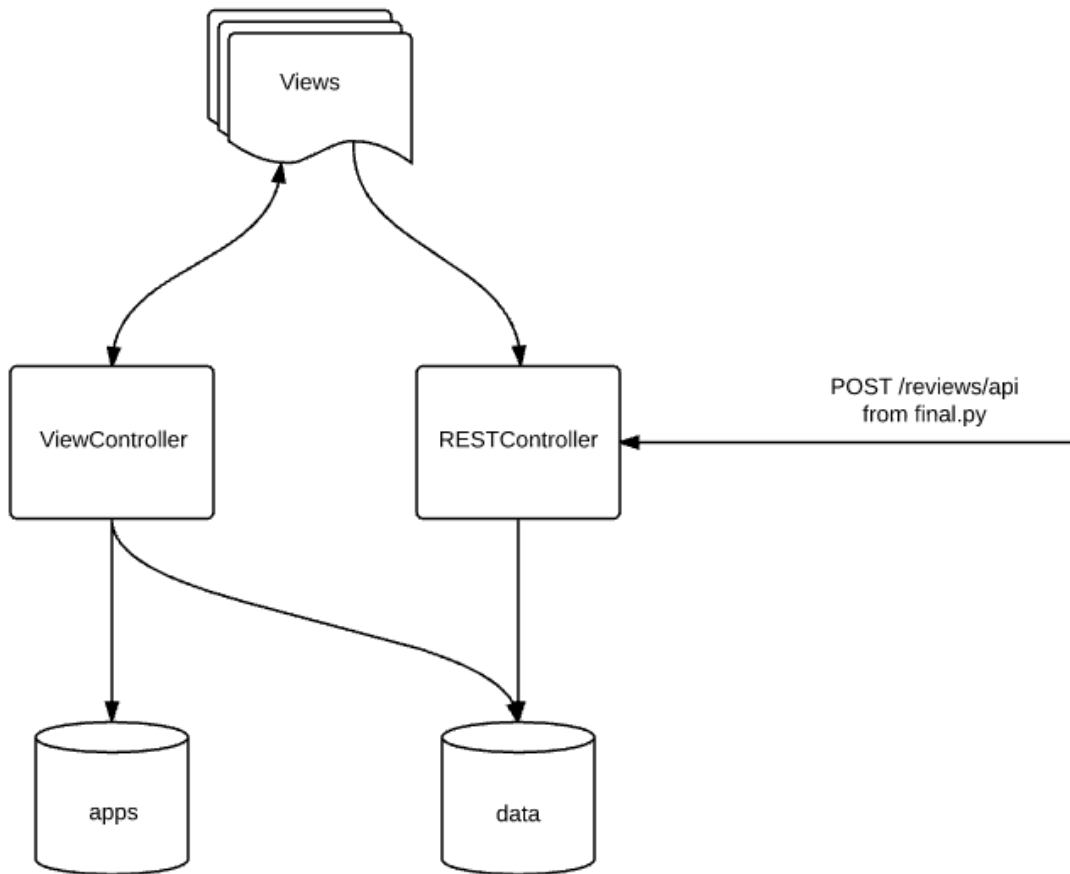


Figure 6. Structure of Web Application

Routing with description of respective views for *ViewController*:

- GET /reviews – it is the start point of the tool. This page contains all applications for which exists reviews in the database. It also works with “apps” database to provide information about applications.
- GET /reviews/features/{appid} – is a view that returns all features for application with identifier appid. View also provides a filter that allows decreasing amount of features based on frequency level.

- POST /reviews/getapps (parameters: {features}, {appid}) – will return the view with applications that contains one or more aspects from list of {features}. {appid} is a parameter which contains the id of “base app” – application which was chosen first. This view also shows closest competitors at the beginning. In this case, we consider the closest competitor to be an application with contains the most of the feature from parameters list.
- POST /reviews/getstatistic (parameters: {features}, {apps}, {appid}) – return view with generated statistic. That is the first view of generated result. The statistic is represented using charts by CanvasJS [23]. View consist of 3 charts, each of that showing average sentiment score point – sum of sentiment scores divided by the amount of review for this application. The difference between graphs is that one is representing the only sum over positive sentiment, the second one over negative sentiments and the last one is showing the sum of both sentiments. Deeper explanation of user interface and possibilities provided in chapter 4. Parameters representing {features}, which are being compared for selected application in {apps}. {appid} stands for “Base app”.
- POST /reviews/gettable (parameters: {features}, {apps}, {appid}) – return view with table view of statistic. Table view is generated using CanvasJS. Parameters stays the same as in previous view.
- POST /reviews/getgraph (parameters: {features}, {apps}, {appid}) – return view which represents graph that shows which applications share common features. Parameters are the same as in two previous views. For drawing graph in this view Dracula Javascript Graph Library was used [24]
- GET /reviews/get?{appid}&{feature}&{level} – return view which shows all reviews for application with identifier {appid} about feature {feature} with sentiment level above {level}. Reviews divided by 2 columns – negative and positive reviews.

*REST Controller* is used as for outer calls, for instance, feeding of databases. Also, *REST Controller* is used for managing some inner calls for obtaining data in statistic views and filtered data.

- GET /reviews/rest – returns JSON [25] with all reviews in the database.
- POST /reviews/rest (Request Parameters: appid, comment, feature, main feature, frequency, negative, positive) – this is the call which writes to table “data” all necessary information about the review. This call is triggered by python script final.py.
- POST /reviews/rest/app (Request Parameters: appid, name, img) – this call writes to table “apps” information about the application. Parameters correspond to the identifier of application, name of application and image link respectively.
- POST /reviews/rest/piechart (Request Parameters: feature, app, levelsentiment, levelsupport) – This call returns data for the pie chart. It counts positive and negative comments for a feature for application with id equals app. It only counts sentiments with absolute values of sentiment scores above levelsentiment and feature frequency above level support.
- POST /reviews/rest/sentiment (Request Parameters: features, apps, baseapp, levelsentiment, levelsupport, type) – This method returns data for the bar chart. It goes through all applications from apps and all features from features. For each entry, it obtains all reviews and counts sum of sentiment level of it. Dependently on the type, it counts only positive, only negative or both sentiment scores. After this, it divides sentiment level calculated at previous step by an amount of review. It is worth mentioning that it only counts features with a frequency above levelsupport. Simultaneously, it does not count sentiment level for a sentence with the absolute value of sentiment scores below levelsentiment.
- POST /reviews/rest/graph (Request Parameters: features, apps, baseapp, levelsentiment, levelsupport) – This method provides data for the application-feature graph. Identical to the methods mentioned above, it only counts reviews filtered by sentiment level and does not consider features with frequency level below the threshold.

All views use Bootstrap stylesheets [26] for a nicer representation and the JQuery library [27] for handling some actions and filtering.

### 4.3 Usage of the Tool

In content of this chapter, we will concentrate on the user experience of using tools. Basic scenarios of usage will be clarified. All user interface decisions will be covered, and detailed manual of using tool will be provided. Each view will be described, and typical usage scenario will be provided. Tool can be accessed at the link in the references [28].

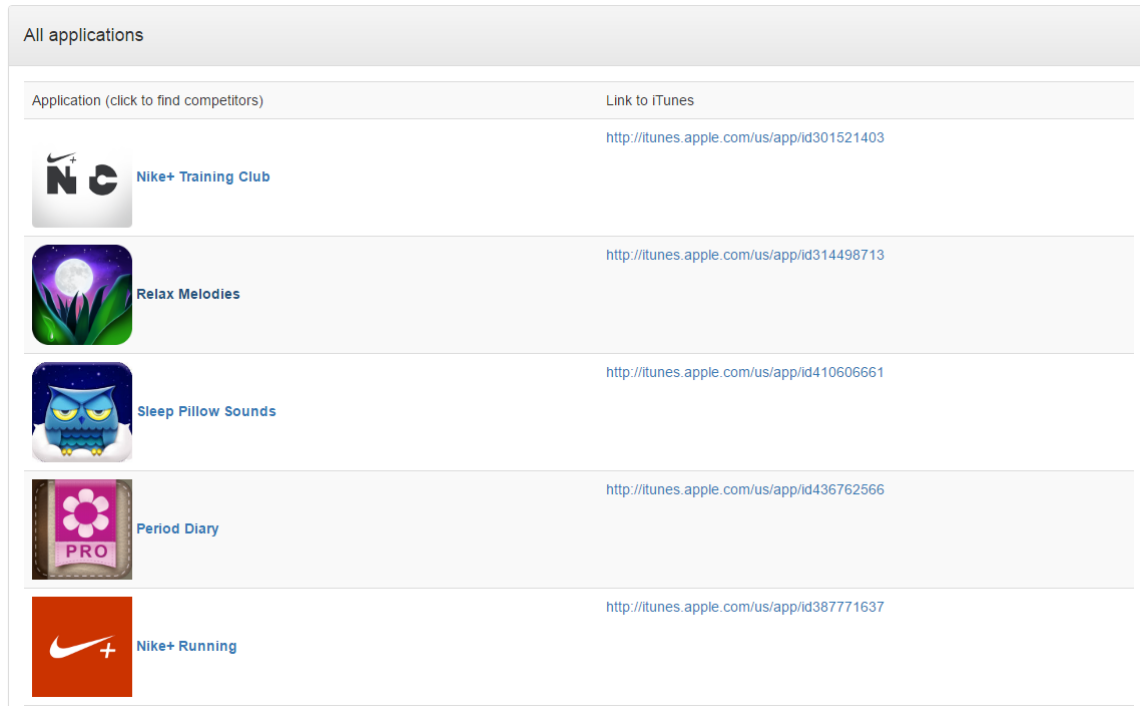


Figure 7. First screen of the application.

Experience with the tool starts from the screen of choosing **Base Application**. For this application, we will extract features and will be looking for competitors. At the First screen of the application, the user can see application name and link to iTunes. As was specified above only applications which still exist on iTunes was included in the analysis. Screenshot of the first screen showed at Figure 7.

These features was extracted for you from reviews of [Nike+ Running](#)






<input type="checkbox"/>	Feature	Frequency	Synonymical features
<input type="checkbox"/>	track_cycle	414	track_bike, bike_trail,
<input type="checkbox"/>	track_workout	366	track_exercise,
<input type="checkbox"/>	time_distance	356	distance_time,
<input type="checkbox"/>	track_calorie	312	
<input type="checkbox"/>	day_period	292	run_day, day_run,
<input type="checkbox"/>	work_time	287	time_shape, time_exercise, time_work, shape_time, exercise_time, time_play, clock_work, time_make, play_time,
<input type="checkbox"/>	track_progress	277	
<input type="checkbox"/>	count_calorie	239	calorie_count,
<input type="checkbox"/>	heart_rate	219	pace_heart,
<input type="checkbox"/>	distance_pace	186	pace_distance,
<input type="checkbox"/>	time_pace	181	place_time, time_order, pace_time, time_rate,
<input type="checkbox"/>	play_music	176	run_music, work_music, music_track, music_run, music_work, music_play, music_running,
<input type="checkbox"/>	phone_run	170	turn_voice, run_phone, turn_phone, phone_turn, work_phone, phone_work, voice_turn,
<input type="checkbox"/>	selection_name	167	option_call, option_list, make_option,

Feature support count: 1 - 414

Figure 8. Feature-choosing screen.

After choosing an application, we will be sent to screen with a list of features. This screen contains all features, which Base Application contains. Features are sorted using total frequency all over reviews. At Figure 8 is an example of feature extraction for “*Nike+ Running*” application. Worth mentioning that tool provides the possibility to filter results by feature support count. Nevertheless, this view contains information about all feature that are synonyms for each other. All features in synonymous feature column – is a synonymous group. As was specified above, the synonymous group is united and counted as one feature, and the name of the group is the feature with the highest frequency. Also, features that consist of the same words counted as synonymous, for example in Figure 8 it is possible to see that “*time\_distance*” and “*distance\_time*” united in one field.

These are closest competitors based on features you choose

	Application	Common features with Base application
<input checked="" type="checkbox"/>	 Nike+ Running (Base app)	7 out of 7 features you choose
<input type="checkbox"/>	 Map My Fitness	7 out of 7 features you choose
<input type="checkbox"/>	 Map My Ride	7 out of 7 features you choose
<input type="checkbox"/>	 Strava Running and Cycling	6 out of 7 features you choose
<input type="checkbox"/>	 Runkeeper	5 out of 7 features you choose

Back Next

Figure 9. Choosing of competitors.

After choosing some features from the list, we can go to the next screen, which is shown in Figure 9.

At this screen represented applications, which contains features that we choose at the previous step. All applications are sorted by the common feature frequency. At the Figure 9 can be seen that “Map My Fitness” and “Map My Ride” are the closest applications based on features we choose at the previous step. All of them share together with Base Application (which is “Nike+ Running”) seven out of seven features we choose. These applications are considered to be **closest competitors** of Base Application. After choosing applications, we finally can proceed to page with results of the comparison. Also, it is possible at this step to exclude Base Application from the final comparison.

Results of the comparison are consist of three views, each of it provides different information about comparison of the app.

The first screen of comparison is a Feature-based comparison of competing for the application. (Figure 10)

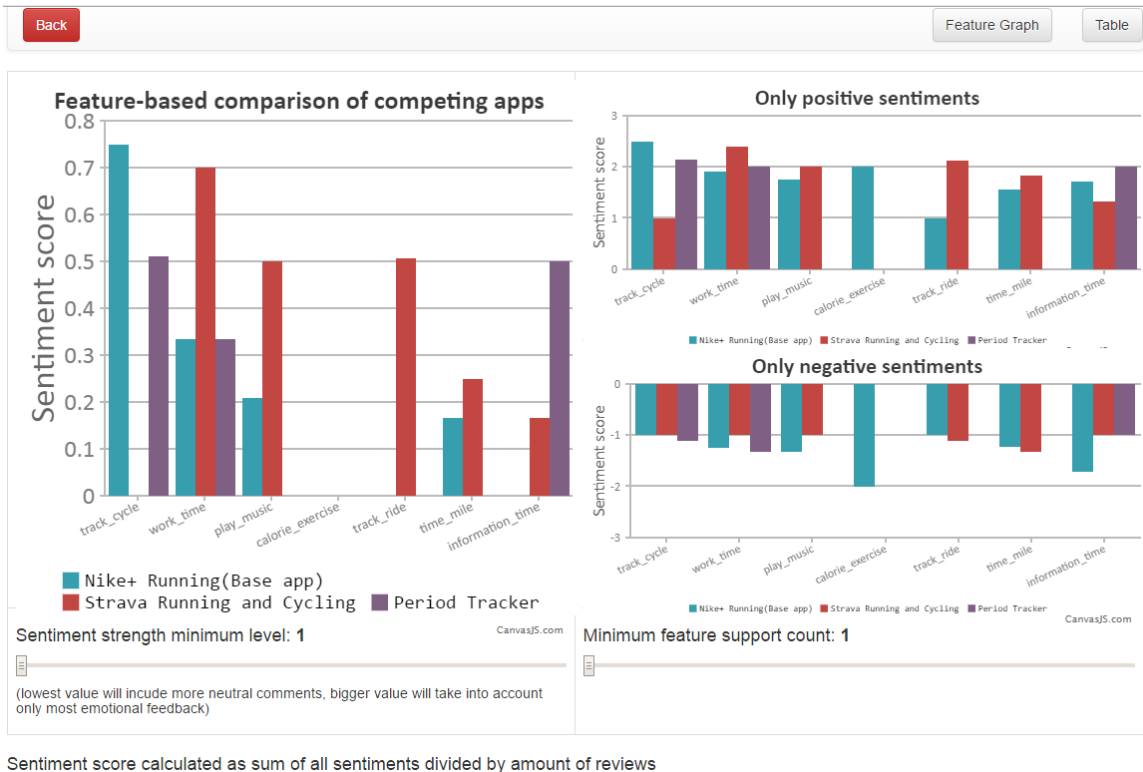


Figure 10. Bar Charts

At the left side of the screen are total sentiments divided by a number of reviews for each of feature. Total sentiment counts as the sum of positive and negative sentiment score. Average sentiment score lays at the y-axis. At the x-axis lay all features that were chosen at previous steps. Different colors of bars represent various applications over those we choose. On the right side, there are two charts. This bar diagrams represent only positive and only negative sentiment scores. The axis structure remains identical. All sentiment scores at this charts are greater than 1 or smaller than -1. The reason for this is that SentiStrength sentiment analysis tool has minimal score equal 1 for a positive score and -1 for the negative rating.

Nevertheless, view allows the user to filter which reviews are counted to the final result. For example, it provides functionality to eliminate reviews with low sentiment scores. Usually, this is neutral reviews, whom authors do not have a strong opinion about aspect. Also, it allows hiding aspects for an application if the amount of comment is not enough from the customer point of view.



Figure 11. Table view

Next view (Figure 11) allows for looking deeper into application comparison. Rows of this table correspond to features and columns represent applications. At the intersection, a pie chart is shown. Each diagram shows the distribution between positive and negative reviews. For example, for the “*Map my fitness*” application and the feature “*track\_calorie*” people mostly tend to talk negatively. However, reviewers mostly talk positively about the “*play\_music*” feature.

It is worth noticing that filters (“Sentiment strength minimum level” and “minimum feature support count”) at the bottom of the screen are not set to 1. Therefore, features which do not have reviews with “strong” opinion (where the absolute value of a positive or negative sentiment is above sentiment strength level) was not included in the table. Also, features about which only limited amount of reviews was done was not showed. If we put this filters to a higher value, even more, charts can disappear and will not be counted. An example of this showed at Figure 12.





Figure 12. Filtering of output

More diagrams have disappeared. The only diagram that is left is satisfying conditions of the criteria. It is worth mentioning that filter values are now higher. The feature “*play\_music*” of application “*Nike+ Training Club*” contains at least 15 reviews with a score of sentiment (positive or negative) equal or above 3. That kind of sorting helps to avoid neutral reviews and opinions where a number of reviews is low.

In case we are interested in the review text, we can go into details and see all reviews and sentiment scores for each review (Figure 13). Two columns with positive and negative reviews represented in this view

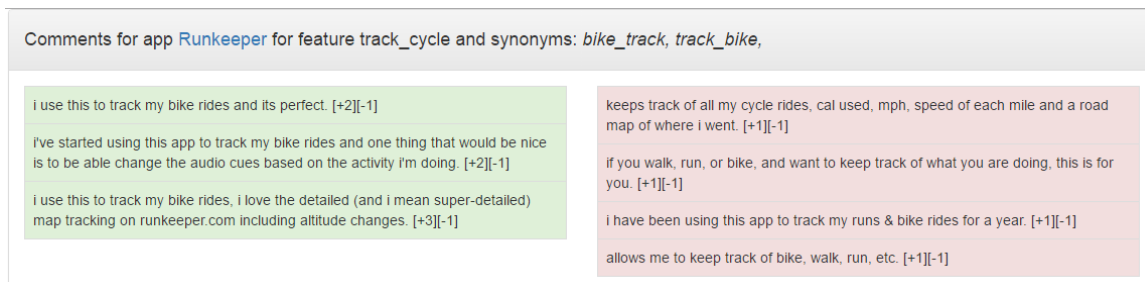


Figure 13. Reviews for application Runkeeper for feature `track_cycle`

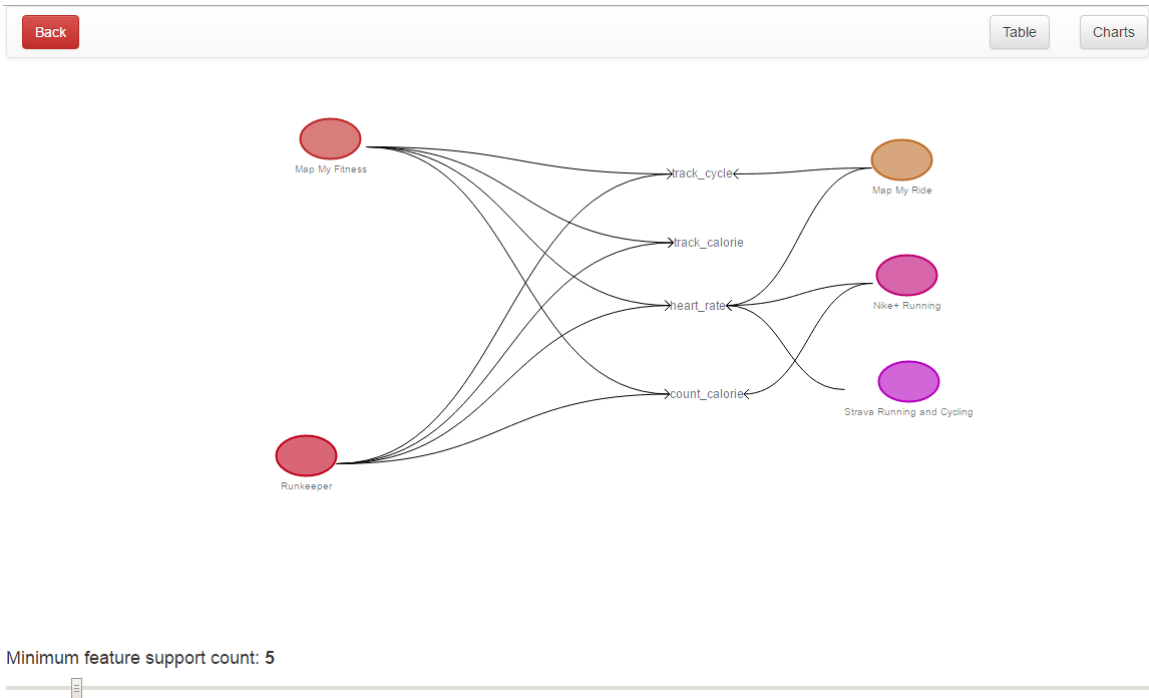


Figure 14. Feature graph

The third view (Figure 14) with results represents application – feature graph. This view shows applications that share the same features. Circles correspond to applications and text to features. This view also provides the possibility to exclude feature appearing only in a small number of reviews for the application. This filter works similarly to the filters in the two previous views.

Based on three views specified above, the customer can make a decision which application has positive feedback for a particular feature.

## 5 Testing and Evaluation

For each step of the algorithm, it was run against the small random dataset. The small dataset was generated randomly from existing reviews in the database. After that, the output of each step of the algorithm was manually checked to verify that it produced a valid result.

Manual verification allowed checking every step of the algorithm including working with a database, extracting correct data from the dataset, Division of raw review by sentences and words. Also was checked if the correct identification of lexical category for each word happens. For each collocation was verified that only right part of speeches involve into collocations generation. For features extracted from collocations, synonym groups generation was tested. Finally, was checked that right data generated, sent to Java Web Application and correctly written to the database. After tool was tested using provided UI and was compared to the database to verify that produced results are correct.

Measurement of the output of the algorithm is quite hard and tricky. Evaluation of used methods should depend on comparing actual output with expected result. The main problem is that expected result is often unpredicted or difficult to measure. In content of this chapter approaches to evaluation will be discussed. Each step of algorithm produced output which was saved separately to different tables, that is why measuring of result was done for each step of algorithm:

1. Pre-processing
2. Feature mining
3. Sentiment analysis

### 5.1 Pre-processing

The main task for pre-processing was to clean reviews, perform separation by sentences and by words. Nevertheless, it performed Part of speech tagging. At this point, we highly depend on two parameters. First one is how clean initial raw text of the review is. Often people do not pay attention to review platform about grammar and punctuation. Therefore, NLTK tokenizer for sentences depends on punctuation and is not able to recognize some sentences to be separate. For example, constructions like “Best. App. Ever.” will be recognized as three different sentences. In this case, we most probably will lose some

opinion about aspect or feature itself, because opinion and feature will appear in different sentences. Alternatively, there are constructions, where the reviewer uses smiles and emoticons instead of the end of the sentence. For instance, “This app is amazing => I recommend it to all my friends”. In this case, the feature will be extracted correctly. However, opinion will include sentiment scores for the whole sentence and most probably will lose in accuracy. Handling of this issue is still an open question and requires deeper analysis and using of tokenizers that are more complex. This topic might be an improvement for future research.

Part of speech tagging is also performed during the pre-processing step. Recent studies show that state of art taggers produce a slightly worse result for software artifact, compared to ordinary English texts. Achievable accuracy of existing methods for bug reports is from 83.6% to 90.5% against 97% for regular corpus [29].

## **5.2 Feature Mining**

To evaluate the feature mining process, we should understand what has to be considered a feature. In our work, we assume that all features contain exactly two words and consist of a combination of a noun and a verb. However, we cannot consider all of the combinations to be features. Few approaches have been examined to measure the quality of feature mining.

First, a possible approach is to find existing reviews in thematic magazines about the application and manually extract features discussed from there. This approach would also allow seeing which feature is acknowledged as a good one. That might have helped with further steps of measurement of sentiment analysis. Unfortunately, this approach contains many drawbacks:

1. We cannot affirm that outcome of professionals reviews is correlating with customers’ opinion. Therefore, this approach cannot produce valuable output for sentiment analysis comparison.
2. Professional reviews are mostly very general and not specific to concrete features of the application. Articles usually provide an overview of the app and describe user experience without evaluating the application.

The second possible approach is creating of a questionnaire for customers who use the application on daily basis. This survey would ask about what users consider being a feature and what they do not. Poll would also help to measure satisfaction about the feature and might help with the evaluation of sentiment analysis. Unfortunately, during our work, not enough customers was found to provide a reasonable amount of opinions.

The third approach, which was chosen for evaluation of this step, is the manual extraction of features from a sample of reviews and comparing with aspects obtained from the algorithm. The idea behind this approach is described in [2]. The process is based on an independent evaluation of samples by few coders (in work specified above seven coders worked on 2800 reviews). These coders independently from each other marked sentences that contain feature or opinion about the feature extracted aspect mentioned and measured sentiment for each feature. Each coder was also measuring time which he spent on measuring and extracting features. The amount of hours spent on 900 reviews was between 8 and 12.5 hours for every coder. Figures proof that manual extraction of reviews is a process that requires many efforts. Unfortunately, for our work we did not have enough resources to perform this kind of analysis. However, feature extraction process was based on the same approaches with using the same tools (NLTK taggers, WordNet lemmatizer, SentiStrength). Therefore, we can reuse results for feature extraction and sentiment analysis obtained in [2]

For feature measurement, three methods were used: precision, recall, and F-measure.

Formula for precision is  $\frac{TP}{TP+FP}$ , where TP – is amount of true positive results (when feature presented in both manual and automatic extraction), FP – is amount of false positive results (when feature presented in automatic feature extraction but not in manual)

Formula for recall is  $\frac{TP}{TP+FN}$ , where TP – is amount of true positive results, FN - amount of false negative results (when feature is presented in manual review but not in automatic extraction). For F-measure general form was used:  $F = 2 \cdot \frac{precision \cdot recall}{precision + recall}$

An example of the output is shown in the tables below. Authors used different applications from different categories. Table 7 shows amount of features extracted from application:

<b>App</b>	$F_S$	$F_{NS}$
AngryBirds	284	219
Dropbox	612	600
Evernote	3700	3127
TripAdvisor	846	754
PicsArt	290	181
Pinterest	625	465
Whatsapp	383	234

Table 7. Amount of feature extracted [2]

In this table  $F_S$  refers to topic extraction with adjectives and  $F_{NS}$  to extraction without adjectives. As we specified in Chapter 3 in our work we assume that adjectives are mostly represents opinion words and contain sentiments. Therefore, approach used in our work correlate to second column of the table.

<b>App</b>	<b>Precision</b>	<b>Recall</b>	<b>F-measure</b>
	$F_{NS}$		
AngryBirds	0.368	0.321	0.343
Dropbox	0.603	0.473	0.531
Evernote	0.451	0.389	0.418
TripAdvisor	0.403	0.370	0.386
PicsArt	0.815	0.661	0.730
Pinterest	0.658	0.592	0.623
Whatsapp	0.910	0.734	0.813
$F_{NS}$ Average	<b>0.601</b>	<b>0.506</b>	<b>0.549</b>

Table 8. Summarizing of results for feature extraction [2]

Table 8 shows calculated precision, recall, and F-measure for results. Similarly, for the previous table, we only interested in  $F_{NS}$  part of it, therefore other part of table is not provided.

Worth mentioning that average precision achieved is around 60% and recall around 50%. AngryBirds application achieves the lowest recall and precision. This application, according to authors, produced most of the disagreements during manual feature extraction.

Approach for feature mining used in our work, however, has some limitations. Firstly, we only concentrate on features which consist of two words. The real feature name might be any length or even be described by one word. Also, non-frequent features will not be mined using specified method. Features which described in a low amount of reviews will be simply ignored.

### 5.3 Sentiment Analysis

As was specified in chapter 3, SentiStrength tool with default parameters has been chosen for performing sentiment analysis. However, SentiStrength has some limitation in use. Firstly, it measures text by giving to the sentence the score equal to the highest rate of the word in it. It means that for relatively long sentences with many adjectives, total score would contain both negative and positive sentiment score with high absolute value. Secondly, for some features during a manual inspection of results SentiStrength was affected by feature itself. For example, in a sentence which refers to feature “calorie loss”, the word loss has strong negative sentiment score in SentiStrength’s vocabulary. Therefore, the sentence might be possibly counted as negative. As a possible solution, manual configuration of SentiStrength vocabularies may be used. Also, for each feature might be useful to exclude words of feature itself from the dictionary. However, this approaches have not been tested during our work and can be a possible improvement for the future. Finally, SentiStrength is not able to determine irony and sarcasm. Therefore, the sentence might get incorrect sentiment score for such sentences.

To evaluate SentiStrength, we refer to [30]. In this work, authors use SentiStrength with default parameters to identify the accuracy of the tool. As input data, authors used three corpuses based on Twitter: SentiStrength Twitter corpus, Earth Hour 2014 and Earth Hour 2015. Evaluation results of accuracy are shown in Table 9:

Tool	SS	EH2014	EH2015
SentiStrength	59.17	66.20	65.00
ClimaPinion	57.21	86.80	66.33
ARCOMEM	46.04	70.34	47.83
DIVINE	57.33	79.80	60.00

Table 9. Evaluation results.

Except SentiStrength authors also used other tools for sentiment analysis. Columns SS, EH2014, and EH2015 corresponds to SentiStrength corpus, Earth Hour 2014 corpus and Earth Hour 2015 corpus respectively. Worth noticing that accuracy for SentiStrength for all three corpuses is above 59%.



## **6 Summary and Future Work**

This chapter contains a summary of our work. Here will be underlined what has been done and what might be subject to further development of the topic.

### **6.1 Summary**

Our work represents a tool that allows comparing competitive mobile applications based on features. Generation of underlying data and provided tool itself help customers and developers finding competitors and perform the feature-based comparison. In our work was used the latest achievements of the state of the art to perform the best possible comparison. The developed solution consists of two parts that can be used independently. The developed tool allows for performing feature extraction along with sentiment analysis based on raw, uncleaned reviews. The modular system of implementation allows integrating the system easily. The developed application helps to identify features that an application contains and finds other applications with the same features. Moreover, it performs a feature-based comparison of applications and measures the perceived quality of features. For each application and for each chosen feature, it provides information about user satisfaction. Using this tool contributes to finding the overall user opinions about the same aspect in different applications. To conclude with, the instrument brings an important contribution to the field. The proposed solution allow its users to perform the aspect-based comparison of applications.

### **6.2 Future Work**

Due to the fact that the solution algorithm includes many steps, future enhancements are possible in many directions. Firstly, pre-processing and feature extraction can be improved. Sentences separation for raw reviews based on punctuation does not always yield the best result. In addition, division not by sentences but by logical parts containing only aspect and opinion (even in different sentences) might make sense. Moreover, in our work, we concentrated only on features that contain two words. Different word patterns can be researched in the future, which might help to make the feature extraction more precise. Also, better sentiment analysis tools might be developed. Using non-default configurations of SentiStrength might possibly increase the quality of output dramatically. As applications are updating continuously, it also might be valuable to compare applications on the time

scale. Some of the updates of applications might have new features or have some changes in the recent implementation of existing aspects. Therefore, comparing user feedback over time might provide information about the reaction of users to changes in the application.

## 7 References

- [1] Keertipati, S., Savarimuthu, B. T. R., & Licorish, S. A. (2016, June). Approaches for prioritizing feature improvements extracted from app reviews. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering* (p. 33). ACM.
- [2] Guzman, E., & Maalej, W. (2014, August). How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)* (pp. 153-162). IEEE.
- [3] Gu, X., & Kim, S. (2015, November). " What Parts of Your Apps are Loved by Users?"(T). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on* (pp. 760-770). IEEE.
- [4] Natural Language Toolkit <http://www.nltk.org/> (last accessed: 03.08.2016)
- [5] Lucene <http://lucene.apache.org/> (last accessed: 03.08.2016)
- [6] Hu, M., & Liu, B. (2004, July). Mining opinion features in customer reviews. In *AAAI* (Vol. 4, No. 4, pp. 755-760).
- [7] Agrawal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB* (Vol. 1215, pp. 487-499).
- [8] Zhai, Z., Liu, B., Xu, H., & Jia, P. (2011, February). Clustering product features for opinion mining. In *Proceedings of the fourth ACM international conference on Web search and data mining* (pp. 347-354). ACM.
- [9] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan), 993-1022.
- [10] Wei, C. P., Chen, Y. M., Yang, C. S., & Yang, C. C. (2010). Understanding what concerns consumers: a semantic approach to product feature extraction from consumer reviews. *Information Systems and E-Business Management*, 8(2), 149-167.
- [11] Bruce, R. F., & Wiebe, J. M. (1999). Recognizing subjectivity: a case study in manual tagging. *Natural Language Engineering*, 5(02), 187-205.
- [12] WordNet <https://wordnet.princeton.edu/> (last accessed: 03.08.2016)

- [13] Wei, W., & Gulla, J. A. (2010, July). Sentiment learning on product reviews via sentiment ontology tree. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 404-413). Association for Computational Linguistics.
- [14] Schouten, K., & Frasincar, F. (2016). Survey on aspect-level sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering*, 28(3), 813-830.
- [15] SentiStrength, <http://sentistrength.wlv.ac.uk/> (last accessed: 03.08.2016)
- [16] Thelwall, M., Buckley, K., & Paltoglou, G. (2012). Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1), 163-173.
- [17] Pagano, D., & Maalej, W. (2013, July). User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)* (pp. 125-134). IEEE.
- [18] Mouzakis Kon, Hoon Leonard, Vasa Rajesh *Socrates mobile app review dataset*, 2013. <http://hdl.handle.net/1959.3/364882> (last accessed: 03.08.2016)
- [19] Requestst: HTTP for humans <http://docs.python-requests.org/en/master/> (last accessed: 03.08.2016)
- [20] <https://bitbucket.org/sabanin/diplom> (last accessed: 03.08.2016)
- [21] Thymeleaf <http://www.thymeleaf.org/> (last accessed: 03.08.2016)
- [22] <https://bitbucket.org/sabanin/reviews> (last accessed: 03.08.2016)
- [23] CanvasJS: Beautiful HTML5 JavaScript Charts <http://canvasjs.com> (last accessed: 03.08.2016)
- [24] Dracula Graph Library <https://www.graphdracula.net/> (last accessed: 03.08.2016)
- [25] JSON - a lightweight data-interchange format <http://www.json.org/> (last accessed: 03.08.2016)
- [26] Bootstrap <http://getbootstrap.com> (last accessed: 03.08.2016)
- [27] JQuery - user interface oriented feature-rich JavaScript library <https://jquery.com/> (last accessed: 03.08.2016)

- [28] <http://review-sentiment.herokuapp.com/reviews> (last accessed: 03.08.2016)
- [29] Tian, Y., & Lo, D. (2015, March). A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 570-574). IEEE.
- [30] Maynard, D. G., & Bontcheva, K. (2016). Challenges of Evaluating Sentiment Analysis Tools on Social Media. *Proceedings of LREC 2016*.
- [31] <http://www.wjh.harvard.edu/~inquirer/> (last accessed: 03.08.2016)

## **Appendix**

### **I. License**

#### **Non-exclusive license to reproduce thesis and make thesis public**

**I, Yevhenii Sabanin,**

*(author's name)*

1. herewith grant the University of Tartu a free permit (non-exclusive license) to:

1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

#### **Aspect/Feature-based Evaluation of Competing Apps,**

*(title of thesis)*

supervised by Dietmar Pfahl, Faiz Shah,

*(supervisors' name)*

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive license does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **04.08.2016**