university of groningen

faculty of arts

# Review the review
## Distant supervised aspect based sentiment analysis

Jorrit Bakker

**Bachelor thesis**
Informatiekunde
Jorrit Bakker
s1857843
June 12, 2016

# ABSTRACT

Aspect Based sentiment analysis (ABSA) is concerned with mining opinions from text about specific entities and their aspects. Nowadays, people easily spill their opinions on the internet about anything. In this thesis, we focused on customer reviews of smartphones to do ABSA. This could be done by using supervised machine learning, however annotating is very time consuming. A lot of web shops summarise reviews by listing positive and negative points above the reviews. So in fact, the annotation has already been done. In this thesis, we researched whether distant supervised machine learning can achieve comparable results as with supervised machine learning. As data, 26,565 customer reviews of smartphones from the web site pdashop.nl were used. This data consisted of 94,307 distant supervised annotated points. Twelve categories were defined based on the aspects of a smartphone. For each category a word list was created to match sentences of a review to a category. Using a Support Vector Machine we created for each category one model. As features we used POS tags, different types of N-grams, word frequencies and TF-IDF-scores. 887 sentences annotated by two annotators, divided over the twelve categories were used to evaluate the system. Dependent on the category, achieved accuracies were between 0.22 and 0.72. In this research we didn't achieve results comparable to supervised machine learning (0.79). Highly skewed data to positive making it hard to detect negative sentiments and not having the ability to train on neutral sentiments were the main causes for low accuracies.

# CONTENTS

# LIST OF TABLES

# PREFACE

This Bachelor's thesis is written by Jorrit Bakker, premaster student Information Science at the Rijksuniversiteit Groningen. I would like to thank my supervisor Dr. Prof. Malvina Nissim for her advise, support and supervision. Also, I would like to thank the annotators and in special Maike Tromp for helping me annotating the data. Lastly, I would like to thank the Information Science Department making it possible for me to do a premaster Information Science. Writing this thesis I unexpectedly discovered I like doing research, although this project didn't end up as good as I hoped for. With this thesis, I complete my premaster Information Science after one year of interesting lectures and assignments. September 2016 I'll start with the master Informational Science, yet new plans are already being made.

# 1 INTRODUCTION

The last couple of years the world wide web has expanded very fast. With this growth, the available content has increased dramatically. Shopping on the internet has increased in popularity, so much physical shops struggle in their existence. Slowly physical shops are disappearing and are replaced by web shops on the internet. Advice by expert shop personal, although maybe subjective, is replaced by readable content like blogs and product reviews. A lot of the web shops have integrated a reviewing system on their website. Doing this, giving their customers the option to advise each other. Reviews by customers, are probably more objective, although those reviewers are not always experts about the product. The fact that everybody can post a review makes that their is a lot of product information available. However, the fact that anybody can review a product and the fact companies are stimulating this do, might causes a reduction of the quality of those reviews. So there is a lot of content, but often without any quick filtering option. A potential customer got to read a lot of reviews to form his opinion about the product. Some web shops tackled this problem by listing a summary of positive and negative points per product or review. This makes it much easier to see whether the review is worth reading.

Aspect Based sentiment analysis (ABSA) is mining opinions from text about specific entities and their aspects. A way to do this is using supervised machine learning. Supervised machine learning requires annotated data to train on. However, annotating is a time consuming task and possibly it isn't necessary. After all, on some web sites the data has already been annotated by their users. An example of such web sites, having reviews annotated by their users, are those from CoolBlue.nl.

Having the possibility to use distant supervised annotated data raised the question what the quality would be compared to the more common approach using supervised annotated data.

**Can we achieve the comparable results using distant supervised machine learning as by using supervised machine learning (79,34%) for aspect based sentiment analysis?**

This paper is structured as follows: In chapter 2 we discuss the background. Chapter 3 will describe the data collection and annotation. This is followed by an overview of the used methods in chapter 4. The achieved results are presented in chapter 5. In chapter 6 the results are discussed. Chapter 7 contains the conclusion and suggestions for further research. In the appendices the most important code snippets are presented.

# 2 BACKGROUND

Aspect Based sentiment analysis (ABSA) is mining opinions from text about specific entities and their aspects. This has been done in different domains like consumer electronics, restaurants and movies (Pontiki et al., 2015). The first important steps of ABSA are extracting the aspects from the content (Mukherjee and Liu, 2012). Within this first task, there are two important subtasks. The first subtask is to extract the aspects and the second task is to categorise the aspects in aspect categories. However categorising aspects it not always easy. An aspect could fit into multiple categories depended on the context (Zhai et al., 2011). Task 12 during the SemEval 2015 campaign (Pontiki et al., 2015) consisted of 4 subtasks whereof one of the tasks was extracting entities as an addition to only aspects.

A subtasks of (Pontiki et al., 2015) was extracting the sentiment polarity for two domains, laptops and restaurants. This has been done by multiple teams for both domains. In the laptop domain the highest accuracy was achieved, 79.34%. An accuracy of 78.69% was the highest in the restaurant domain. Both highest scores were reached using a Maximum Entropy (MaxEnt) classifier with features based on N-grams, POS tagging, lemmatization, negation words an available sentiment lexica. A SVM model with N-grams, PMI-scores, POS tagging, parse trees, negation words and scores based on 7 sentiment lexica as features ended on the second place with a score of 78.29% and 78.10% for the laptop and restaurant domain.

In 2015, the teams scored slightly better in the domain of laptops compared to the restaurants domain, probably due the fact the training data was skewed to positive were the test data wasn't. However in 2014 the laptop domain proved to be harder (Pontiki et al., 2014). A big difference between the domains is that people tend to write about restaurants with expressing more sentiment compared to the laptop domain. In the laptop domain people often mention features instead of expressing a clear sentiment. In this research we'll use smartphone reviews because people seem to have a stronger opinion about smartphones compared to laptops. When we compared the number of reviews (900) of the 12 best sold laptops on laptopshop.nl with the number of reviews (3.507) of the 12 best sold smartphones of pdashop.nl, we found 2.607 more reviews for smartphones.

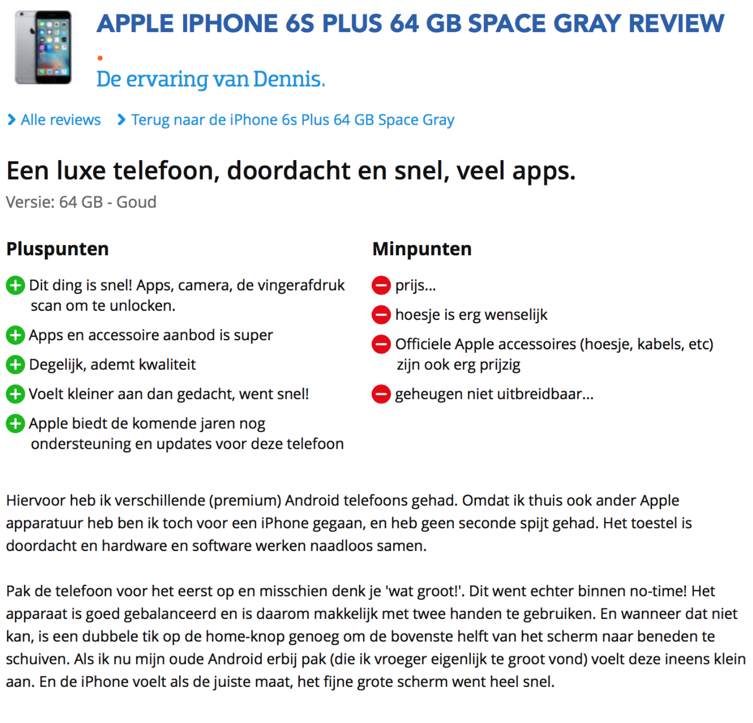
Aspect based sentiment analysis consists of two main task: entity extraction and sentiment extraction. In this research we want to explore the possibilities by extracting the sentiment using distant supervised sentiment analysis (Go et al., 2009). We won't focus on entity extraction, we'll use a list of categories and predefined words to extract entities.

# 3 DATA AND MATERIAL

## 3.1 COLLECTION

The data that is used are Dutch reviews from the web site pdashop.nl, a web shop selling smartphones. On their web site, Pdashop.nl is currently selling 276 different phones from 21 brands. On average, each phone has been reviewed 96 times. In total, pdashop.nl has 26,565 customer reviews. All reviews have a title, content, positive and negative points, whether the reviewer recommends the product, rating, votes of review readers whether the review is useful or useless and a product id. By using a web crawler we collected all 26,565 customer reviews. In total, those reviews contained 94,307 listed points divided in positive and negative aspects about the product. Out of the 94,307 points, 68,446 are positive and 25,861 are negative.

An example of an review is shown in figure 1. Positive points are listed under *Pluspunten*, negative points are listed under *Minpunten*.



APPLE IPHONE 6S PLUS 64 GB SPACE GRAY REVIEW

De ervaring van Dennis.

› Alle reviews › Terug naar de iPhone 6s Plus 64 GB Space Gray

Een luxe telefoon, doordacht en snel, veel apps.

Versie: 64 GB - Goud

**Pluspunten**

- Dit ding is snel! Apps, camera, de vingerafdruk scan om te unlocken.
- Apps en accessoire aanbod is super
- Degelijk, ademt kwaliteit
- Voelt kleiner aan dan gedacht, went snel!
- Apple biedt de komende jaren nog ondersteuning en updates voor deze telefoon

**Minpunten**

- prijs...
- hoesje is erg wenselijk
- Officiele Apple accessoires (hoesje, kabels, etc) zijn ook erg prijzig
- geheugen niet uitbreidbaar...

Hiervoor heb ik verschillende (premium) Android telefoons gehad. Omdat ik thuis ook ander Apple apparatuur heb ben ik toch voor een iPhone gegaan, en heb geen seconde spijt gehad. Het toestel is doordacht en hardware en software werken naadloos samen.

Pak de telefoon voor het eerst op en misschien denk je 'wat groot!'. Dit went echter binnen no-time! Het apparaat is goed gebalanceerd en is daarom makkelijk met twee handen te gebruiken. En wanneer dat niet kan, is een dubbele tik op de home-knop genoeg om de bovenste helft van het scherm naar beneden te schuiven. Als ik nu mijn oude Android erbij pak (die ik vroeger eigenlijk te groot vond) voelt deze ineens klein aan. En de iPhone voelt als de juiste maat, het fijne grote scherm went heel snel.

**Figure 1:** An example of a review on pdashop.nl.

All data is collected by crawling pdashop.nl using the framework Scrapy. This is a framework for crawling web sites and extracting structured data based on Python. The collected data was stored in an SQLite database.

## 3.2 ANNOTATION

### 3.2.1 Categories and word lists

In this thesis, we focussed on sentiment analysis and not on entity extraction. Therefore, we created a predefined set of 12 categories. For each category we defined a list of words matching this category. We created those lists by tokenizing all collected content of the points into unigrams. After tokenizing the data we did have 15,395 unique words. We reduced the number of unique words to 1169 by using a Part-of-Speech tagger and selecting only the nouns. An annotator reviewed all unique words and categorised them in one of the following 12 categories: *Accu, Beveiliging, Camera, Connectiviteit, CPU, Design, Geheugen, Geluid, Geheugen, Opslag, Overig, Prijs, Scherm* and *Software*. The *Overig* category was added for words that are not relevant to any of the other categories. The words in the *Overig* category aren't used in this research.

In Table 1, an overview of the number of words in each category is presented. Except from *Overig*, the category *Design* contains the most words followed by *Software* and *Accu*. The categories *Geheugen* and *CPU* have the least number of words.

**Table 1:** Categories and number of words per category.

| Category | Number of words |
|---|---|
| Accu / Battery | 81 |
| Beveiliging / Security | 48 |
| Camera / Camera | 60 |
| Connectiviteit / Connectivity | 53 |
| CPU / CPU | 15 |
| Design / Design | 160 |
| Geheugen / Memory | 12 |
| Geluid / Audio | 69 |
| Opslag / Storage | 31 |
| Overig / Other | 466 |
| Prijs / Price | 25 |
| Scherm / Display | 52 |
| Software / Software | 97 |

### 3.2.2 Distant supervised annotation

All the reviews include listed positive and negative points that summarise the review. The created word lists are used to label one or multiple categories to the list item. Since those points are given an positive or negative sentiment by the review writers, the sentiment is already determined (Go et al., 2009).

The second step was using those points for distant supervised annotating the content of the review. The content of the review was tokenized into sentences using the NLTK's sentence tokenizer. While iterating over the sentences we searched for words in the review content which matched the words in the categories obtained from the listed points. When there was a match found, the sentence was labeled either positive or negative for the specific category. For example, a review with a positive point *"Helder* *scherm"* having in the review content the sentence: *"De display is erg mooi en heeft helder beeld"* would be labeled as positive for the subject category *Display*. Since there are not neutral listed points, all sentences with a category match are labeled either positive or negative. We weren't able to train on a neutral class. Sentences, matching more then one different category, were labeled multiple times, possibly with a different sentiment. For example: *"Het scherm is een amoled scherm en de camera is uitstekend door de snelle focus en de heldere foto's."*, would be labeled *positive* for the categories *Scherm* and *Camera*.

In Table 2, an overview of the number of positive and negative sentences in the trainings data set is presented. Each row represents a category with in the columns the category, number of positive sentences, number of negative sentences and total number of sentences.

**Table 2:** Number of sentences per category in training data.

| Category | Positive | Negative | Total |
|---|---|---|---|
| Accu / Battery | 4977 | 1879 | 6856 |
| Beveiliging / Security | 543 | 209 | 752 |
| Camera / Camera | 11223 | 2512 | 13735 |
| Connectiviteit / Connectivity | 324 | 442 | 766 |
| CPU / CPU | 213 | 8 | 221 |
| Design / Design | 4948 | 3322 | 8270 |
| Geheugen / Memory | 310 | 334 | 644 |
| Geluid / Audio | 600 | 563 | 1163 |
| Opslag / Storage | 318 | 368 | 686 |
| Prijs / Price | 2378 | 1046 | 3424 |
| Scherm / Display | 6953 | 1585 | 8538 |
| Software / Software | 3420 | 2231 | 5651 |
| Total | 36207 | 14499 | 50706 |

### 3.2.3 Gold standard

To create the gold standard, test and development data, we used sentences from the review content. We picked a review randomly and sentence tokenized the content of this review. We checked every sentence whether it contains one of the words defined in the word categories. If the sentence fitted a category we labeled it as development or test data. We repeated this process until we did select at least 3000 sentences. As a result we ended up with 3004 sentences.

Five annotators, named as *JB, MT, AB, TG, IP*, did annotate the data. *JB* refers to the author of this thesis, *MT* has a bachelor linguists and the other annotators completed a high education in an other field. The annotators did see a sentence and the category the sentence was selected for. Using a web interface the annotators labelled the sentence for the given category as positive, negative or neutral. JB and MT annotated most of the data, 2534 sentences. The other 470 sentences were annotated by the other three annotator. Because of low kappa scores, < 0.5, we only used the data annotated by the *JB* and *MT*. The calculated kappa was 0.77 based on 34 sentences. Although it is calculated with a small amount of sentences, we consider this kappa as acceptable. After removing the data of the other annotators, we further split the data into development (1647, 65%) and test (887, 35%) data.

In Table 3, an overview of the number of positive and negative sentences in the development set is presented. Each row represents a category. The columns show the category, the number of positive sentences, the number of negative sentences, the number of neutral sentences and the total number of sentences.

**Table 3:** Number of sentence per category in development data.

| Category | Positive | Negative | Neutral | Total |
|---|---|---|---|---|
| Accu / Battery | 161 | 58 | 39 | 258 |
| Beveiliging / Security | 21 | 4 | 4 | 29 |
| Camera / Camera | 164 | 32 | 50 | 246 |
| Connectiviteit / Connectivity | 22 | 12 | 26 | 60 |
| CPU / CPU | 10 | 4 | 3 | 17 |
| Design / Design | 149 | 49 | 65 | 263 |
| Geheugen / Memory | 22 | 9 | 9 | 40 |
| Geluid / Audio | 28 | 10 | 17 | 55 |
| Opslag / Storage | 19 | 20 | 15 | 54 |
| Prijs / Price | 77 | 29 | 25 | 131 |
| Scherm / Scherm | 136 | 34 | 57 | 227 |
| Software / Software | 140 | 31 | 96 | 267 |
| Total | 949 | 292 | 406 | 1647 |

In Table 4, an overview of the number of positive and negative sentences in the test set is presented. Each row represents a category. The columns show the category, the number of positive sentences, the number of negative sentences, the number of neutral sentences and the total number of sentences.

**Table 4:** Number of sentence per category in test data.

| Category | Positive | Negative | Neutral | Total |
|---|---|---|---|---|
| Accu / Battery | 72 | 26 | 19 | 117 |
| Beveiliging / Security | 16 | 2 | 4 | 22 |
| Camera / Camera | 106 | 11 | 13 | 130 |
| Connectiviteit / Connectivity | 16 | 15 | 9 | 40 |
| CPU / CPU | 4 | 2 | 1 | 7 |
| Design / Design | 91 | 18 | 41 | 150 |
| Geheugen / Memory | 7 | 3 | 10 | 20 |
| Geluid / Audio | 15 | 13 | 4 | 32 |
| Opslag / Storage | 8 | 5 | 9 | 22 |
| Prijs / Price | 37 | 11 | 12 | 60 |
| Scherm / Display | 76 | 19 | 28 | 123 |
| Software / Software | 84 | 30 | 50 | 164 |
| Total | 532 | 155 | 200 | 887 |

# 4 METHOD

## 4.1 SCIKIT-LEARN

Scikit-learn is a open source Python platform build on NumPy, SciPy, and matplotlib. Scikit-learn can be used for data mining and analysing tasks. It supports most machine learning algorithms, both supervised and unsupervised. Evaluation calculations and plotting are easily preformed.

### 4.1.1 Supported Vector Machine

The Support Vector Machine (SVM) with a lineair kernel of Scikit-learn uses the LIBSVM library (Chang and Lin, 2011). The LIBSVM is normally used in two steps. First trainings data is used to create a predicting model. The model uses vectorized data and places it in a vector space. A hyperplane is calculated between the data points. The hyperplane with the largest distance between the different classes is used. Secondly the the created model, based on the trainings data, is used to predict information. It is also possible to get the probability estimates for output information.

### 4.1.2 Vectorizers

Scikit-learn has different classes to vectorize text data into a matrix. The Countvectorizer converts text data to a matrix based on token's frequency in the text data. The TfidfVectorizer converts the text to a matrix of TF-IDF features. The TF-IDF gives a reflection of the importance of token in to document in a corpus. Both vectorizers can apply N-grams while vectorizing the data.

## 4.2 NLTK

NLTK is Python platform to process human natural language. It contains different libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning and wrappers for industrial-strength NLP libraries. NLTK has over 50 corpora and lexical resources such as WordNet.

### 4.2.1 NLTK POStag

NLTK uses by default the Standard treebank POS tagger. This tagger is trained on the Penn Treebank. This means the POS tagger is trained on English. Still, when using Dutch input, it returns POS tags but with a lower accuracy.

#### 4.2.2 Dutch Stemmer

NLTK supports Dutch stemming with the snowball DutchStemmer. When stemming, the mophological affix is removed from the word. The stem of the word is the result of stemming.

## 4.3 MACHINE LEARNING

Although the MaxEnt classifier achieved the highest accuracy during the (Pontiki et al., 2015), in this research we used a Support Vector Machine (SVM) with a lineair kernel. We did this, because SVM is easy to implement using Python Scikit-learn and the achieved results didn't differ a lot compared to MaxEnt in the reviewed papers. We created for each of the twelve different categories (Table 1) a SVM model based on the distant supervised annotated data (table 2). We did this, by training the model with the sentences that matches one of the word (Section 3.2.1) in the category. The possible labels were *positive* or *negative*. We selected the sentiment by using listed positive and negative points of the review. During development we used the gold annotated development data to evaluate the system (Table 3). After the development we evaluated the system with the gold annotated test data (Table 4).

## 4.4 FEATURES

To add features we used the pipeline in Scikit-learn. By using the NLTK POS-tagger we appended the POS-tags as features. The snowball DutchStemmer in NLTK was used to stem the words of the sentence. Stop words were removed from the sentence. Word frequencies were added as feature by using the countvectorizer. Lastly TF-IDF scores were used as feature, calculated by using the tfidfvectorizer. Both vectorizers used 1-, 2-, 3-, 4-grams.

## 4.5 DETECTING NEUTRAL SENTIMENTS

Because the crawled data from pdashop.nl only has positive or negative points, we only could train on positive and negative sentiments. This resulted in a model that is not able to detect neutral sentiments. Even though neutral was used in manual annotation and is quite a frequent category (Table 4). Therefore, we added neutral sentiments by overriding manually the predicted class based on probablitiy score. To achieve this, we set a threshold fixed at 0.65. While iterating over the probability estimates we checked whether the probability was in between 0.35 and 0.65. In those cases, we manually changed the predicted sentiment to neutral.

## 4.6 EVALUATION

Having manually labelled data makes it possible not only to evaluate our system, but also to evaluate the quality of data that we acquired automatically.

### 4.6.1 Data

Accuracy, recall and F1-score are calculated for the positive, negative and neutral labels to evaluate the gold annotated data with the trainings data. Those scores are calculated by comparing the labels for a sentence annotated by the annotators with the annotated labels based on the listed positive and negative points.

### 4.6.2 SVM Models

To evaluate the created SVM models we calculated for each category accuracies, recall scores and F1-scores. This is done by comparing the labels of the gold annotated data with the predicated labels of the SVM model. The precision, recall and F1-score is calculated for the positive, negative and neutral labels. Averages of those scores are also reported. The system is compared to a baseline of 59.98% for the positive labels (Table 4).

# 5 RESULTS

In Table 5-16, an overview of the achieved results per category is presented. The results are achieved by twelve models, for each category one, trained on the silver data (Table 2) and evaluated on the gold test data (Table 4). In each table, the rows present the scores for positive, negative, neutral sentiments. The last row is presenting the average scores. The first column presents the precision, the second column the recall and the third column the F1-score. In the last column the number of sentences in the test data set is presented.

## 5.1 ACCURACY

In all categories the highest accuracies are achieved for the positive label. Those accuracies were between 1.00 and 0.20. The category *Connectiviteit* did have the highest accuracy with 1.00 followed by *Beveiliging* (0.87, Table 6) and *Camera* (0.81, Table 7). The categories *Geheugen* and *Opslag* did have the lowest accuracies with scores of 0.27 and 0.20 (Table 11, 13).

The negative label, presented in the second row of each table, did have quite lower accuracies compared to the positive label. Accuracies between 0.90 and 0 were achieved. The category *Scherm* did have the highest accuracy (0.90, Table 15) for the negative label followed by the category *Accu* (0.88, Table 5). The categories with the lowest accuracy, all with a accuracy of 0, were *Beveiliging*, *Camera*, *Geheugen* and *CPU* (Table 6, 7, 11, 9).

The neutral label did have accuracies between 0 and 0.43. In none of the categories the accuracy of the neutral label was higher then the positive label. Only the categories *Beveiliging*, *Design* and *Geheugen* did have a higher score for the neutral then negative sentiment. The categories *Beveiliging* and *Prijs* did achieve the highest accuracies with a accuracy of 0.43 (Table 6) and 0.40 (Table 14). The lowest accuracies for the neutral label were found for *Camera*, *CPU* and *Scherm*. All of them with an accuracy of 0.

The average accuracies are between 0.22 and 0.72. The best accuracies were found for the categories *Geluid* (0.72, Table 12), *Beveiliging* (0.72, Table 6). The categories *Geheugen* and *Opslag* did have the lowest accuracies, 0.22 (Table 11) and 0.29 (Table 13).

## 5.2 F1-SCORES

F1-scores for the positive label did vary between 0.13 and 0.86. The highest F1-score for the positive label was found in the category *Camera* (0.86, Table 7). The categories *Beveiliging* (0.84), *Accu* (0.80) and *Geluid* (0.80) did also have high F1-scores.

The negative label did have a F1-scores in the range of 0 to 0.64. The highest F1-scores are achieved in the categories *Geluid* (0.64, Table 12) and *Scherm* (0.62) (Table 15). Lowest scores were found in the categories having 0 as accuracy for the negative label, *Beveiliging*, *Camera*, *Geheugen* and *CPU*.

F1-scores between 0 and 0.55 were achieved. In the most categories, the neutral sentiment did have an F1-score lower then the positive F1-score. The categories *Opslag* and *Prijs* are the exception. The highest scores are achieved in the category *Beveiliging* (0.55, Table 6) and *Software* (0.40, Table 16). Lowest F1-scores were found for the categories *Camera*, *CPU* and *Scherm*, having an accuracy and F1-score of 0 for the neutral label.

The average F1-scores did vary between 0.71 and 0.23. *Beveiliging* achieved the highest F1-score with 0.71, followed by *Camera* (0.70). The lowest, average F1-score of 0.23, was achieved by *Geheugen* (Table 11)

**Table 5:** Results Accu.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.71 | 0.90 | 0.80 | 72 |
| Negative | 0.88 | 0.27 | 0.41 | 26 |
| Neutral | 0.22 | 0.21 | 0.22 | 19 |
| avg / total | 0.67 | 0.65 | 0.62 | 117 |

**Table 6:** Results Beveiliging.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.87 | 0.81 | 0.84 | 16 |
| Negative | 0.00 | 0.00 | 0.00 | 2 |
| Neutral | 0.43 | 0.75 | 0.55 | 4 |
| avg / total | 0.71 | 0.73 | 0.71 | 22 |

**Table 7:** Results Camera.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.81 | 0.92 | 0.86 | 106 |
| Negative | 0.00 | 0.00 | 0.00 | 11 |
| Neutral | 0.00 | 0.00 | 0.00 | 13 |
| avg / total | 0.66 | 0.75 | 0.70 | 130 |

**Table 8:** Results Connectiviteit.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 1.00 | 0.31 | 0.48 | 16 |
| Negative | 0.50 | 0.40 | 0.44 | 15 |
| Neutral | 0.22 | 0.56 | 0.31 | 9 |
| avg / total | 0.64 | 0.40 | 0.43 | 40 |

**Table 9:** Results CPU.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.57 | 1.00 | 0.73 | 4 |
| Negative | 0.00 | 0.00 | 0.00 | 2 |
| Neutral | 0.00 | 0.00 | 0.00 | 1 |
| avg / total | 0.33 | 0.57 | 0.42 | 7 |

**Table 10:** Results Design.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.66 | 0.59 | 0.62 | 91 |
| Negative | 0.15 | 0.22 | 0.18 | 18 |
| Neutral | 0.34 | 0.34 | 0.34 | 41 |
| avg / total | 0.51 | 0.48 | 0.49 | 150 |

**Table 11:** Results Geheugen.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.27 | 0.43 | 0.33 | 7 |
| Negative | 0.00 | 0.00 | 0.00 | 3 |
| Neutral | 0.25 | 0.20 | 0.22 | 10 |
| avg / total | 0.22 | 0.25 | 0.23 | 20 |

**Table 12:** Results Geluid.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.80 | 0.80 | 0.80 | 15 |
| Negative | 0.78 | 0.54 | 0.64 | 13 |
| Neutral | 0.25 | 0.50 | 0.33 | 4 |
| avg / total | 0.72 | 0.66 | 0.68 | 32 |

**Table 13:** Results Opslag.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.20 | 0.12 | 0.15 | 8 |
| Negative | 0.38 | 0.60 | 0.46 | 5 |
| Neutral | 0.33 | 0.33 | 0.33 | 9 |
| avg / total | 0.29 | 0.32 | 0.30 | 22 |

**Table 14:** Results Prijs.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.65 | 0.89 | 0.75 | 37 |
| Negative | 0.25 | 0.09 | 0.13 | 11 |
| Neutral | 0.40 | 0.17 | 0.24 | 12 |
| avg / total | 0.52 | 0.60 | 0.53 | 60 |

**Table 15:** Results Scherm.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.66 | 0.97 | 0.79 | 76 |
| Negative | 0.90 | 0.47 | 0.62 | 19 |
| Neutral | 0.00 | 0.00 | 0.00 | 28 |
| avg / total | 0.55 | 0.67 | 0.58 | 123 |

**Table 16:** Results Software.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.72 | 0.61 | 0.66 | 84 |
| Neutral | 0.36 | 0.44 | 0.40 | 50 |
| Negative | 0.31 | 0.33 | 0.32 | 30 |
| avg / total | 0.54 | 0.51 | 0.52 | 164 |

## 5.3 EVALUATION SILVER VS GOLD DATA

In Table 17, an overview of the comparison between the gold annotated data, development and test, and the distant supervised annotated is presented. The positive sentiment has a precision of 0.66 with a F1-score of 0.72, the negative sentiment has a precision of 0.35 with an F1-score of 0.46. The neutral class scores 0 on all scores.

**Table 17:** Evaluation supervised- and distant supervised annotated data.

| - | Precision | recall | F1-score | support |
|---|---|---|---|---|
| Positive | 0.66 | 0.79 | 0.72 | 1191 |
| Negative | 0.35 | 0.67 | 0.46 | 383 |
| Neutral | 0.00 | 0.00 | 0.00 | 572 |
| avg / total | 0.43 | 0.56 | 0.48 | 2146 |

# 6 DISCUSSION

In this research, we tried to achieve comparable accuracies using distant supervised training instead of supervised training. The achieved accuracy scores are between 0.22 and 0.72 with F1-scores between 0.23 and 0.71.

The in most cases relative high accuracies for the positive labels can be explained due the fact the data is highly skewed to positive. The silver training data consisted of 71.40% positive sentences (Table 2). With a baseline of 59.98% for positive in the gold annotated test data (Table 4), higher accuracies can be expected.

Although we expected that people would have a strong opinion about there newly bought smartphone, quite a large part of the test data consisted of neutral sentences. It seemed that a lot of people write about aspects of the phone without expressing a clear positive or negative opinion about it. Instead, they just mention the the aspects. We can see this in the gold annotated data, having a lot of neutral labels (22%).

Having a lot of neutral sentences about the aspects is probably one of the main causes of the low accuracies. Using distant supervised learning we couldn't train on neutral sentiments because they are not listed above the review. We tried to solve this problem by adding a threshold of 0.65. Although this made it possible to detect neutral sentiments, it wasn't enough to increase the total accuracy drastically.

A comparable problem happened with the negative sentiments. We can see that the data is highly skewed to positive. This made it very hard for the model to predict negative sentences. We think that people that buy a phone, they really like and know what they are buying based on reviews. This makes it less likely that customers will write about negative aspects. This results in mainly positive reviews about a product.

In Table 17 we can confirm the low quality of the trainings data. With an average accuracy of 0.43 it's not possible to create an accurate model. The reduction of accuracy is caused by lack of neutral sentences to train on. However, it is also caused by a difference between the sentiment of review content and the listed points. A neutral expression in the review content could have been annotated as positive or negative when it's listed in the points. Hereby, the model has neutral sentences labeled as positive and negative making it hard to predict the right class.

A second problem in this research was the amount of test data. The initial idea was to create one model that would return a category and an sentiment. Since creating such a model was not possible, we created for every category a model. With the idea having one model, we annotated 3004 sentences whereof less then 1000 sentences as test data. However, we created multiple models, so the test data was split into 12 small parts. Ideally, we should have at least 1000 test sentences for each category to do a good evaluation. Especially the categories *Beveiliging*, *CPU*, *Geheugen* and *Opslag* did have too less data.

# 7 CONCLUSION

In this research we asked the question whether it was possible to achieve the same accuracies using distant supervised machine learning compared to supervised machine learning. After this research, we can't confirm this is possible. In our references, the best achieved accuracy was 79.34%. The achieved accuracies in this research are between 0.22 and 0.72 dependent on the topic.

The low achieved accuracy- and F1-scores we dedicate to low quality of distant supervised annotation. A second important factor is the lack of the ability to train on neutral labels. In a further research we would advise to choose a complete different domain for distant supervised learning with less neutral data. We would suggest something like book, movie or restaurant reviews. We think that people will write more explicit in these domains because they are for one time use. This will probably result in less neutral data and possibly in better results.

# BIBLIOGRAPHY

Chang, C.-C. and C.-J. Lin (2011). Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST) 2*(3), 27.

Go, A., R. Bhayani, and L. Huang (2009). Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford 1*, 12.

Mukherjee, A. and B. Liu (2012). Aspect extraction through semi-supervised modeling. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pp. 339–348. Association for Computational Linguistics.

Pontiki, M., D. Galanis, H. Papageorgiou, S. Manandhar, and I. Androutsopoulos (2015). Semeval-2015 task 12: Aspect based sentiment analysis. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, Denver, Colorado*, pp. 486–495.

Pontiki, M., D. Galanis, J. Pavlopoulos, H. Papageorgiou, I. Androutsopoulos, and S. Manandhar (2014). Semeval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pp. 27–35.

Zhai, Z., B. Liu, H. Xu, and P. Jia (2011). Constrained lda for grouping product features in opinion mining. In *Advances in knowledge discovery and data mining*, pp. 448–459. Springer.

# Appendices

# A PDASHOP.NL CRAWLER

```python
from scrapy.spiders import Spider
from scrapy.selector import Selector
from scrapy import Request
from scriptie_crawler.items import ProductItem, ReviewItem, ReviewPointItem

import re

class PDAShopSpider(Spider):
   name = "pdashop"
   allowed_domains = ["pdashop.nl"]
   start_urls = [
      "http://www.pdashop.nl/category/4214/smartphones.html?items=48",
   ]

   def parse(self, response):

      sel = Selector(response)

      for href in sel.xpath('//a[@class="product-list-item--title-link"]/@href'):
         url = response.urljoin(href.extract())

         yield Request(url, callback=self.parse_as_product)

      # next page
      next_product_list = response
               .urljoin(sel.xpath('//a[@class="pagination next secondary"]/@href')
         .extract_first())
      yield Request(next_product_list)

   def parse_as_product(self, response):

      sel = Selector(response)

      p = ProductItem()
      p["product_id"] = sel.xpath('//*[@id="js-product-scope"]/@data-product-id')
               .extract_first()
      p["url"] = response.url.strip()
      p["name"] = sel.xpath('//*[@id="js-product-scope"]/div[2]/div/h1/span/text()')
               .extract_first().strip()
      product = p.save()

      reviews_url = response
               .urljoin(sel.xpath('//a[@class="rating-summary--reviews-link"]/@href')
         .extract_first())

      #go to review page
      request = Request(reviews_url, callback=self.parse_reviews)
```

```
        request.meta["product"] = product

        yield request

    def parse_reviews(self, response):

        product = response.meta["product"]

        sel = Selector(response)
        for review_url in
            sel.xpath('//div[@class="reviewContent"]/h3/a/@href').extract():
            request = Request(response.urljoin(review_url),
                callback=self.parse_review)
            request.meta["product"] = product
            yield request

        more_reviews_url =
            response.urljoin(sel.xpath('//a[@class="pagination next
            secondary"]/@href')
                .extract_first())
        request = Request(more_reviews_url, callback=self.parse_reviews)
        request.meta["product"] = product
        yield request

    def parse_review(self, response):

        product = response.meta["product"]
        sel = Selector(response)

        r = ReviewItem()
        r["url"] = response.url.strip()
        r["review_id"] = re.findall(r'\d+', response.url)[1]
        r["title"] =
            sel.xpath('//*[@id="layout_content"]/div/div/article/div[1]/h3/text()')
                .extract_first().strip().encode('utf-8')
        r["content"] = ""
        for content in
            sel.xpath('//div[@class="reviewText"]/text()').extract():
            r["content"] += content.strip().encode('utf-8') + "\n"
        recommendation =
            sel.xpath('//*[@id="layout_content"]/div/div/article/div[1]/div[3]/span/text()')
                .extract_first()
        if recommendation != None:
            if recommendation.encode('utf-8') == "Ik raad dit product
                aan":
                r["recommended"] = True
        r["usefull"] =
            sel.xpath('//*[@id="layout_content"]/div/div/article/div[2]/form/strong[1]/text()')
                .extract_first().encode('utf-8')
        r["unusefull"] =
            sel.xpath('//*[@id="layout_content"]/div/div/article/div[2]/form/strong[2]/text()')
                .extract_first().encode('utf-8')
        r["rating"] = sel.xpath('//meter/@value').extract_first()
        r["product"] = product
        review = r.save()

        for pro in
            sel.xpath('//div[@class="pros"]/ul/li/text()').extract():
            rp = ReviewPointItem()
            rp["content"] = pro.strip().encode('utf-8')
            rp["positive"] = True
        rp["review"] = review
        rp.save()

    for con in
         sel.xpath('//div[@class="cons"]/ul/li/text()').extract():
        rp = ReviewPointItem()
        rp["content"] = con.strip().encode('utf-8')
        rp["positive"] = False
        rp["review"] = review
        rp.save()

    return
```

# B MACHINE LEARNING

```python
import numpy as np
from sklearn.base import BaseEstimator

from django.core.management.base import BaseCommand, CommandError

from reviews.models import Product, Review, ReviewPoint, NGram,
    ReviewSentence, Word, Sentiment, WordCategory
from collections import Counter, defaultdict
from django.db.models import Q, Count

from .misc import *
from .nltk_functions import *

from sklearn import svm
from sklearn.naive_bayes import *

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer,
    TfidfTransformer

from sklearn.preprocessing import MultiLabelBinarizer

from sklearn import preprocessing
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.metrics import classification_report

from sklearn.metrics import precision_score, recall_score, f1_score,
    accuracy_score
from skll.metrics import kappa

from nltk.stem import WordNetLemmatizer
from nltk.stem import SnowballStemmer
from nltk.stem.snowball import DutchStemmer

import time

class Command(BaseCommand):

   pointlist = []

   def add_arguments(self, parser):

      # Named (optional) arguments
      parser.add_argument('--dev',
         action='store_true',
         dest='dev',
         default=False,
         help='SVM - evaluating on dev')
      parser.add_argument('--gold',
         action='store_true',
         dest='gold',
         default=False,
         help='SVM - evaluating on dev')
      parser.add_argument('--supervised',
         action='store_true',
         dest='supervised',
         default=False,
         help='SVM - evaluating on dev')
      parser.add_argument('--evaluate-silver',
         action='store_true',
         dest='evaluate-silver',
         default=False,
         help='SVM - evaluate silver')
      parser.add_argument('--kappa',
         action='store_true',
         dest='kappa',
         default=False,
         help='Calculate kappa')
      parser.add_argument('--play',
         action='store_true',
         dest='play',
         default=False,
         help='SVM - Playing with code')


   def handle(self, *args, **options):
      """ Do your work here """

      if options['dev']:
         self.test("dev")
      if options['gold']:
         self.test("gold")
      if options['supervised']:
         self.test_supervised()
      if options['evaluate-silver']:
         self.evaluate_silver()
      if options['kappa']:
         self.kappa()
      if options['play']:
         self.play()

   def tokenize_preprocess(self, x):
      tokenized = nltk.word_tokenize(x)
      newlist = []
      for word in tokenized:
         if is_stopword(word) == False:
            newlist.append(word)
      return newlist

   def preprocessor(self, x):
      tokenized = nltk.word_tokenize(x)

      pos_tags = " ".join([tag for (word, tag) in
         nltk.pos_tag(nltk.word_tokenize(x))])
      sentence = " ".join([DutchStemmer().stem(word) for word in
         tokenized])

      return sentence+" "+pos_tags


   def read_points(self):
      """
      Read all positive and negative points
      """
      print("Read points")

      pointlist = []
      pos = ReviewPoint.objects.filter(review__status="train",
          positive=True).all().order_by('review')
      pointlist.append(pos)

      neg = ReviewPoint.objects.filter(review__status="train",
          positive=False).all().order_by('review')
      pointlist.append(neg)

      self.pointlist = pointlist


   def build_classifier(self, category=None, num_pos=None,
        num_neg=None):
      print("Build classifier")

      count_vectorizer = CountVectorizer(preprocessor =
          self.preprocessor, tokenizer = self.tokenize_preprocess,
          ngram_range=(1, 4))
      tfid_vectorizer = TfidfVectorizer(preprocessor =
          self.preprocessor, tokenizer = self.tokenize_preprocess,
          ngram_range=(1, 4))

      X = []
      y = []

      if category == None:
         words =
             Word.objects.filter(selected=True).exclude(category__name="Overig").all()
      else:
         words = Word.objects.filter(selected=True,
             category=category).all()


      if self.pointlist == []:
         self.read_points()


      total = len(self.pointlist[0])+len(self.pointlist[1])
      counter = 0
      posCounter = 0
      negCounter = 0

      xPos = []
      xNeg = []

      """
      Select all relevant sentences
      TODO: Prevent sentences been marked twice for the same category
      """
      for points in self.pointlist:
         for rp in points:
            counter += 1
            progress(counter, total)

            sentences = get_sentences(rp.review.content)
            for word in words:
               if word.name in rp.content:
                for sentence in sentences:
                    for category_word in
                        Word.objects.filter(category=word.category).all():
                        if category_word.name in sentence:
                            sentiment = rp.positive
                            if sentiment:
                                xPos.append(sentence)
                            else:
                                xNeg.append(sentence)
                            break


        """
        Select number of positive and negative items
        """
        if num_pos != None:
            if len(xPos) > num_pos:
                xPos = xPos[:num_pos]
        X += xPos
        y += ["positive"]*len(xPos)

        if num_neg != None:
            if len(xNeg) > num_neg:
                xNeg = xNeg[:num_neg]
        X += xNeg
        y += ["negative"]*len(xNeg)

        print("\n{} positive\n{} negative".format(len(xPos), len(xNeg)))

        clf = svm.SVC(kernel='linear', probability=True)
        # clf = MultinomialNB()

        pipeline = Pipeline([
            ('features', FeatureUnion([
                ('ngram_tf_idf', Pipeline([
                    ('counts', count_vectorizer),
                    ('tf_idf_ngram', TfidfTransformer()),
                ])),
                ('pos_tf_idf', Pipeline([
                    ('tf_idf', tfid_vectorizer),
                ])),
            ])),
          ('clf', clf)
        ])

        pipeline.fit(X, y)

        return pipeline


    def build_gold_classifier(self, category=None, num_pos=None,
         num_neg=None):
        count_vectorizer = CountVectorizer(preprocessor =
             self.preprocessor, tokenizer = self.tokenize_preprocess,
             ngram_range=(1, 4))
        tfid_vectorizer = TfidfVectorizer(preprocessor =
             self.preprocessor, tokenizer = self.tokenize_preprocess,
             ngram_range=(1, 4))


        test_reviews = Review.objects.filter(status="dev").all()
    total = len(test_reviews)
    counter = 0

    X = []
    y = []

    for test_review in test_reviews:
       counter += 1
       progress(counter, total)

       if category != None:
          ss = Sentiment.objects.filter(
                Q(annotation__sentence__review=test_review) &
       Q(annotation__category=category) & (
          Q(user="jorrit") | Q(user="maike") |
          Q(user="abdul") | Q(user="grote liefde"))).all()
       else:
          ss = Sentiment.objects.filter(
                Q(annotation__sentence__review=test_review) & (
          Q(user="jorrit") |
          Q(user="maike") |
          Q(user="abdul") |
          Q(user="grote liefde"))).all()
       for s in ss:
          if s.sentiment != None:
             if s.sentiment == "positive" or s.sentiment ==
                 "negative":
                if s.sentiment == "positive":
                   y.append("positive")
                if s.sentiment == "negative":
                   y.append("negative")
             else:
                y.append("neutral")
             X.append(s.annotation.sentence.content.strip())

    clf = svm.SVC(kernel='linear', probability=True)

    pipeline = Pipeline([
        ('features', FeatureUnion([
          ('ngram_tf_idf', Pipeline([
             ('counts', count_vectorizer),
             ('tf_idf_ngram', TfidfTransformer()),
          ])),
          ('pos_tf_idf', Pipeline([
             ('tf_idf', tfid_vectorizer),
          ])),
       ])),
      ('clf', clf)
    ])

    pipeline.fit(X, y)

    return pipeline

 def get_test_set(self, category=None, type="dev"):
    print("\n Get test set: {}".format(type))


    test_reviews = Review.objects.filter(status=type).all()
    total = len(test_reviews)
    counter = 0
```

```

      xTest = []
      yTest = []

      for test_review in test_reviews:
         counter += 1
         progress(counter, total)

         if category != None:
            ss = Sentiment.objects.filter(
                  Q(annotation__sentence__review=test_review) &
         Q(annotation__category=category) & (
            Q(user="jorrit") |
            Q(user="maike") |
            Q(user="abdul") |
            Q(user="grote liefde"))).all()
         else:
            ss = Sentiment.objects.filter(
                  Q(annotation__sentence__review=test_review) & (
            Q(user="jorrit") | Q(user="maike") |
            Q(user="abdul") | Q(user="grote liefde"))).all()
         for s in ss:
            if s.sentiment != None:
               if s.sentiment == "positive" or s.sentiment ==
                    "negative":
                  if s.sentiment == "positive":
                     yTest.append("positive")
                  if s.sentiment == "negative":
                     yTest.append("negative")
               else:
                  yTest.append("neutral")
               xTest.append(s.annotation.sentence.content.strip())


      return (xTest, yTest)


   def calculate_with_threshold(self, predicted, probabilities,
        threshold):
      for i in range(len(predicted)):
         if probabilities[i][0] > 1-threshold and probabilities[i][0]
             < threshold:
            predicted[i] = "neutral"

      return predicted


   def test(self, type):

      """Run for categories"""
      threshold = 0.65

      for category in
          WordCategory.objects.exclude(name="Overig").all():
         print("Category: {}".format(category.name))

         classifier = self.build_classifier(category=category)
         testSet = self.get_test_set(category=category, type=type)

         yPredict = classifier.predict(testSet[0])
         yProbability = classifier.predict_proba(testSet[0])
        yPredict_with_theshold =
            self.calculate_with_threshold(yPredict, yProbability,
            threshold)


        print("\n\n")

        report = classification_report(testSet[1],
            yPredict_with_theshold)
        print(report)

        print("\n\n")

    def test_supervised(self):
        threshold = 0.65

        for category in
            WordCategory.objects.exclude(name="Overig").all():
            print("Category: {}".format(category.name))

            classifier = self.build_gold_classifier(category=category)
            testSet = self.get_test_set(category=category, type="gold")

            yPredict = classifier.predict(testSet[0])
            yProbability = classifier.predict_proba(testSet[0])
            yPredict_with_theshold =
                self.calculate_with_threshold(yPredict, yProbability,
                threshold)


            print("\n\n")

            report = classification_report(testSet[1],
                yPredict_with_theshold)
            print(report)

            print("\n\n")

    def evaluate_silver(self):
        """
        methods that compares silver and gold annotated data
        prints precision, recall, f-score
        """
        annotated_sentiment_list = []
        silver_sentiment_list = []

        annotated_sentences = Sentiment
            .objects.filter(
            Q(user="jorrit") | Q(user="maike") |
            Q(user="abdul") | Q(user="grote liefde")).all()

        for sa in annotated_sentences:
            annotated_sentiment = sa.sentiment
            annotation_category = sa.annotation.category
            words =
                Word.objects.filter(category=annotation_category).all()
            review_points =
                ReviewPoint.objects.filter(review=sa.annotation.sentence.review)
            for word in words:
                for rp in review_points:
                    if word.name in rp.content:
```

```
               if rp.positive == True:
                  silver_sentiment = "positive"
               if rp.positive == False:
                  silver_sentiment = "negative"

               if annotated_sentiment != "positive" and
                   annotated_sentiment != "negative":
                  annotated_sentiment = "neutral"

               annotated_sentiment_list.append(annotated_sentiment)
               silver_sentiment_list.append(silver_sentiment)


   report = classification_report(annotated_sentiment_list,
       silver_sentiment_list)
   print(report)

def kappa(self):
   """
   calculates kappa score between different annotaters
   """

   kappa_dict = defaultdict(list)

   objects =
       Sentiment.objects.values(’annotation’).annotate(dcount=Count(’annotation’))
   for row in objects:
      annotations_of_sentence =
          Sentiment.objects.filter(annotation__pk=row["annotation"])
      if len(annotations_of_sentence) > 1:
         users = []
         for annotation in annotations_of_sentence:
            if annotation.user not in users:
               users.append(annotation.user)

         if len(users) > 1:

            u1 =
                Sentiment.objects.filter(annotation__pk=row["annotation"],
                user = users[0]).last()
            u2 =
                Sentiment.objects.filter(annotation__pk=row["annotation"],
                user = users[1]).last()

            if u1.sentiment not in ["positive", "negative"]:
               u1.sentiment = 0
            elif u1.sentiment == "positive":
               u1.sentiment = 1
            else:
               u1.sentiment = -1

            if u2.sentiment not in ["positive", "negative"]:
               u2.sentiment = 0
            elif u2.sentiment == "positive":
               u2.sentiment = 1
            else:
               u2.sentiment = -1

            if users[0]+"_"+users[1] in kappa_dict.keys():
               kappa_dict[users[0]+"_"+users[1]].append((u1.sentiment,
                   u2.sentiment))
            elif users[1]+"_"+users[0] in kappa_dict.keys():
               kappa_dict[users[1]+"_"+users[0]].append((u1.sentiment,
                   u2.sentiment))
            else:
               kappa_dict[users[0]+"_"+users[1]].append((u1.sentiment,
                   u2.sentiment))

   for users, sentiments in kappa_dict.items():
      y1 = [s[0] for s in sentiments]
      y2 = [s[1] for s in sentiments]
      k = kappa(y1, y2)
      print("users: {}, number: {}, kappa: {}".format(users,
          len(sentiments), k))
```