university of groningen / faculty of arts

Detecting Income Level of Dutch Twitter Users using Stylometric Features

Léon Melein

Bachelor thesis
Information Science
Léon Melein
S2580861
January 13, 2017
Supervisor: B. Plank

ABSTRACT

Income prediction is a relatively undiscovered aspect of author profiling. Early research on English (Flekova et al., 2016) linking Twitter users to occupations and their respective average incomes, obtained promising results. There is no comparable research for Dutch speakers yet. In this thesis, we explore to what extent author profiling can predict the income level of Dutch users.

We do so by creating a dataset of 2000 Twitter users. These are divided into two income classes as there currently is no complete income data available for individual occupations in The Netherlands. We use *distant supervision* to annotate users with their occupational class and their income. We then extract a number of surface, readability and n-gram features from the users' posts. Using logistic regression, we try to classify the users on their income class with those features.

After testing various feature groupings, the classifier proved to be the most robust with uni-, bi- and trigram features, reaching an F1-score of 0.72. Although this indicates that profiling can predict a user's income class to a very large extent, this can only be seen as a first indication as the scope of this study is limited. With clear directions for future improvement, we hope that this study may be a stepping stone towards the prediction of individual incomes for Dutch authors.

CONTENTS

PREFACE

I would like to take this opportunity to thank a number of people who have been instrumental to finishing my thesis. First of all, my supervisor Barbara. Thanks to her relentless support and guidance, I was able to tackle my research question. Even when my research proved more challenging than expected, she helped me to tackle the problems step by step.

I also want to thank fellow student Reinard van Dalen for his advice and support. As our thesis subjects partly overlap, it really helped to look at certain problems together and then solve them in the most efficient way possible. And a final thank you to Stijn Eikelboom, who proofread my thesis.

Their support could obviously not prevent the toll my thesis took on my keyboard. It broke down while I was writing the first chapters. It turned out to be a worthwhile sacrifice for completing my thesis.

1 INTRODUCTION

As we rely more and more on the internet and its applications in our daily lives, the opportunities for author profiling continue to grow. Social media services, and the text-based communications they facilitate, provide an ever-growing corpus of texts, which are linked to their author. Furthermore, these services provide additional self-disclosed information about the authors like education, occupation and relationships. If we use this data the right way, we can perform research that simply wasn't possible before, at least not on this scale (Sloan et al., 2015).

An aspect of author profiling is income prediction. A potential use for these estimations of income is in customer relations. Just as the rest of our life, our contact with companies and their customer services moves online. People want to be helped quickly, without giving a lot of information in advance. Customers expect companies to "know" them. Likewise, companies want to know their customers so they don't have to ask for a lot of information and can help quickly. Income can be a key factor in this, because it can help in predicting which products or services a customer probably already has and in which they are likely interested.

An example: a cable company customer with a relatively low income probably has a low tier service package. He or she will only be interested in cheap(er) options or options that provide significant extra value in comparison to their additional cost. Knowing this, offering an expensive movie channel package to this customer does not make a lot of sense. Such a package is nice to have, but isn't of much value - unless the customer is a true movie addict.

Despite the existing research into English speaking Twitter users and their respective incomes, there is currently no comparable research for Dutch speaking Twitter users. We will therefore focus our research on the following question: to what extent is it possible to accurately predict the income level of a Dutch Twitter user and which stylometric features are predictive?

The rest of this thesis is organized as follows. Chapter 2 gives an overview of earlier work on income classification with English speaking users, as well as all prerequisites for studying income classification. Chapter 3 details our data gathering efforts and goes into detail about our labeling algorithm. Chapter 4 builds on that, with our method to find the best combination of features for a classifier on income. In chapter 5, we will discuss the results of the different feature setups tested. Finally, in chapter 6 we will look back at the study as a whole and answer our research question.

2 BACKGROUND

Research into author profiling on Twitter is relatively new, especially in the field of income prediction. All prior work has focused on English-speaking Twitter users.

The most recent study was performed by Flekova et al. (2016). Their goal was to find a viable writing style-based predictor for age and income. For each dimension, a different data set was used. Regarding income, an off the shelf data set was used, which will be discussed in more detail later on. It contained almost six thousand Twitter users labeled with their occupation. The researchers labeled each user with the mean UK yearly income associated with their occupation. This was done regardless of the location of the users involved (e.g., an English-speaking user from Amsterdam would also be assigned a UK income). As the exact income for every occupation was known, the machine learning task was framed as regression. Flekova et al. codified stylistic variation into a number of features, which were grouped into four categories: surface, readability, syntax and style. After performing a ten-fold cross validation with both linear and non-linear regression methods, they discovered that readability metrics like the *Flesch Reading Ease* metric and the relative use of pronouns correlated with income over age. They concluded that the differences in style can be used to "tailor the style of a document without altering the topic to suite either age or income individually".

The data set used in Flekova et al. (2016) was created during an earlier study by Preotiuc-Pietro et al. (2015). They used the corpus to classify users according to their occupational class. The occupational titles and classes used were gathered from the UK Standard Occupational Classification (SOC) (Office for National Statistics, 2010). The SOC is a hierarchical classification of occupations. It consists of four levels, starting with nine very general classes and terminating in hundreds of very specific classes. Each level is indicated with a different number of digits. The coarsest level is indicated with one digit and the finest level with four digitals (e.g., class 1: 'managers, directors and senior officials' and class 1116: 'elected officials and representatives', respectively). The classification is based on the International Standard Classification of Occupations (International Labor Office, 2013). For each occupation the *Twitter REST API*[1] was used to find at most 200 users for each occupation. The accumulated users were divided into the three-digit groups that they belong to. Users that were companies, had no description or had a contradicting description, were removed from the collection by hand. Furthermore, three-digit groups with less than 45 users were discarded. The final collection contained 5191 users, divided into 55 three-digit groups.

Preotiuc-Pietro et al. (2015) mention two papers in their related work section that describe different labeling strategies. Both do not annotate users manually. They all employ *distant supervision*, a method that labels data automatically, given an existing form of ground truth. This approach was first used by Read (2005) for sentiment classification of tweets. In the Preotiuc-Pietro study, Twitter users were labeled by looking for a self-disclosed occu-

[1] https://dev.twitter.com/rest/public

pational title. In order to detect such titles they relied on a list of occupational titles from the SOC. These list forms were used by the ground truth that is needed for labeling.

The first is a paper by Li et al. (2014) which tries to label users with the name of their employer, amongst other things. In this study, Twitter profiles were cross-linked with profiles on a different social networking site, Google Plus. To ensure both profiles are interrelated, they looked at their friend connections on both sites and made sure that there was a large enough intersection between them. The employer name was extracted from the Google Plus profile and used to label the Twitter user. Using this method, they were able to collect 7208 users with a known employer. This strategy relies heavily on profiles from networking sites other than Twitter, like Google Plus or LinkedIn. Without unfettered access or a large number of cross-linkable profiles on both platforms, it cannot be used for labeling data.

The second strategy relies solely on the occupational titles from the SOC and profiles on Twitter. Sloan et al. (2015) labeled users with class in the National Statistics - Socio-Economic Classification (NS-SEC), a classification closely related and interoperable with the SOC. They extracted users from a feed provided by the *Twitter REST API*, which constitutes a one percent representative sample of public tweets. For each user, they looked for a self-disclosed occupational title in the biography line. The titles they used for detection were gathered from the SOC. After finding an occupation, the user would be labeled with their NS-SEC class by looking up the SOC class for their occupation and consequently looking up the corresponding NS-SEC class. In case a user mentioned multiple occupations, the authors hypothesized that the first one found would be most important and therefore should be used. Using this method they were able to label 32,032 users with a NS-SEC class. A random survey of 1000 users resulted in an accuracy of 57.8 percent. The authors mention several caveats of this method. Hobbies and former occupations may be falsely detected as current occupations. Commonly occurring phrases like "Doctor Who fan" may also result in false matches.

This second strategy depends a lot less on outside data. It is therefore easier to implement and use in further research, given that the needed data is available. For this thesis, a corpus of Dutch tweets and their authors is already available at the University of Groningen and data on occupational titles and incomes is available from the Dutch government bureau Statistics Netherlands[2]. It therefore makes sense to use this second strategy for our data collection and annotation efforts. The implementation details of this strategy follow in the next chapter.

[2] https://www.cbs.nl/en-gb

3 DATA AND MATERIAL

3.1 COLLECTION

The primary data set for this research is a corpus of Dutch Twitter users with their 500 latest tweets, categorized by income class. As there was no suitable data set available off the shelf, a new corpus was created. As a starting point, the University of Groningen (UG) *twitter2* corpus was used to gather user profiles. The twitter2 corpus contains all Dutch tweets provided by Twitter's *Streaming API*, which constitutes a one percent representative sample of public messages posted on Twitter.

In order to gather user profiles, all tweets from September 1st till September 5th, 2016 were used. For each tweet in the corpus, the user was looked up using the UG's in-house *tweet2tab* tool to extract the user ID, username, real name and biography line for each user from the corpus. The user's biography line was used to find an occupational title. That title was then linked to an occupational class and consequently the average hourly income for that occupational class. The average hourly income was then multiplied by the number of hours worked by the average Dutch worker per year, to compute the average yearly income. All users with a known occupation were labeled with their average yearly income. This resulted in a collection of 36,113 users with suggested occupations and incomes.

After removing no longer existing accounts, private accounts and accounts with less than 1000 tweets, 21,862 users were still available. These users were divided into two income classes, high (above €34,500) and low (below €34,500). Afterwards, 1500 users were randomly selected from each group and their tweets were gathered using the Twitter REST API. Retweets and non-Dutch tweets (as explained in the next section) were left out of the collection. Users with less than 500 Dutch, self-written tweets were discarded.[1] From the remaining users, a thousand were randomly selected per class for further use in this research. More details about processing users for use in the collection will follow in the section on *processing*.

3.2 ANNOTATION

In order to divide the users into income classes, they need to be annotated with their average yearly income. Distant supervision is used to find the average yearly income of a user. We look for an occupational title of a user in the user's biography line. In case a user has multiple occupations, we use the first one we can find. With the found title, we look up the user's occupational class and the average hourly income for that class. We then label the user with the average yearly income by multiplying the average hourly income with the average total hours worked in The Netherlands.

There are three additional data sources needed in order to make our annotation process work.

[1] The tweets were collected on December 10, 2016 between 00:44 and 9:30.

First, we use a list of occupational titles and their respective classes from Statistics Netherlands (2014a) to look up the occupation of a user. These classes correspond with classes in the International Standard Classification of Occupations (International Labor Office, 2013), just like the earlier mentioned SOC. As this file was never meant for machinal consumption, the file had to be modified. All titles formed one long string, which had to be split in order to get the individual titles per class. Furthermore, the titles contained a lot of shorthand notations, e.g. "assistent-, coach" for the similar occupations "coach" and "assistent-coach" and slashes for synonyms like "typist / tekstverwerker". These were removed by hand as there was no suitable way to do this correctly in an automated manner. Second, we use a list of occupational classes and their respective average incomes from Statistics Netherlands (2014b) to look up the average hourly income for a particular class. We use the two-digit classes, as the incomes corresponding to almost all of them are known[2]. For most three- and four-digit classes, incomes aren't provided by Statistics Netherlands. Furthermore, the incomes amongst the two-digit groups vary enough to create a viable two-class split of our data. Finally, to derive the average yearly income we need to know the average worked hours per year in The Netherlands. According to the European Observatory of Working Life (2015) the average Dutch worker makes 1677 hours a year.

To evaluate the performance of our distant supervision method, a random survey of 100 users per class was taken. Their labels were manually checked in a two class setting, as mentioned in the previous section. The labels were considered correct if they appeared in the biography of a user, the user was a human and the occupational title was used to indicate a paying occupation, not a hobby or study. The accuracy over the whole group of 200 users was 74.5 percent, with 70 percent in the low class and 79 percent in the high class. In 17 cases, the labels were wrong because the account was simply not used by a person but by a company. As there is no surefire way to distinguish between human and non-human users, we disregard these cases. The overall accuracy without these cases is 81.4 percent. The last few cases that were wrongly labeled consist of the following cases. In 16 cases, hobbies or voluntary work were labelled as an occupation. In 14 cases, occupational titles got bodged during the transformation of the needed file and therefore detected the wrong occupation. Finally, in four miscellaneous cases users described an internship or former occupation in a non-trivially detectable way.

These results confirm that our distant supervision method yields enough correctly labeled data for our research, even though it is far from perfect. Possible future improvements of the method will follow in the *Future Work* section of chapter 6.

3.3 PROCESSING

Concerning the users, after extracting users with a known occupation from the twitter2 corpus, a number of processing steps are involved to bring the entire group of 36113 users to a manageable number of suitable users. First, users that no longer exist, have their profile set as private or have less than 1000 tweets are removed, to ensure we can get enough data per user for our

[2] There were no average incomes available for the two-digit (submajor) groups 62, 82, 92 and 95. They are therefore left out of the rest of our research.

research. The users are checked by using the Lookup API of Twitter, which allows us to check users in batches of 100. The remaining users are saved in a Python dictionary and written to disk with the built-in Pickle library. Afterwards, the remaining users are divided in the chosen income classes. From each class 2000 users are randomly selected for further processing by using the random.choice function of the NumPy Python library (van der Walt et al., 2011). For these users, their latest 1000 tweets are collected. Retweets and non-Dutch tweets were discarded. The language classification of each tweet was performed by the langid Python library (Lui and Baldwin, 2012). Users with less than 500 suitable tweets were left out of the data set. For all remaining users, the tweets are written to a text file per user per class. From the remaining set of users, 1000 were randomly selected per group to be used in this research.

After the processing of the users is completed, the tweets are prepared for further use. URLs, hashtags and usernames are removed and the tweets are tokenized, so that relevant features can be derived from the tweets. The processing relies on the *TweetTokenizer* included in the NLTK Python library (Bird et al., 2009), a popular library for natural language processing in Python. As some features need to be generated from text tokenized per sentence, all tweets were run through a pre-trained NLTK sentence tokenizer created by Kiss and Strunk (2006). It was trained on the Dutch part of the Multilingual Corpus 1 (ECI), particularly on articles from the "De Limburger" newspaper. The resulting data was collected in a dictionary and written to disk with the Pickle library.

4 METHOD

Flekova et al. (2016) was a major source of inspiration for our methodology. However, as our research is limited in available time (7 weeks) and resources, we had to make some compromises, given all constraints. We mainly rely on the scikit-learn library (Pedregosa et al., 2011) for the concrete implementation of our method. It provides a lot of tools for machine learning, from full blown classifiers and regression models to tools for evaluation.

4.1 FEATURES

Originally, we planned to implement all of the feature groups used in Flekova et al. (2016). Due to the constraints of our research, we could only adopt two of the four groups: surface and readability. To compensate for this, we added a third groups of features: n-grams. All groups will be discussed in more detail below.

4.1.1 Surface

From this group, we chose and implemented the following features:

- Length of a user's tweets in words;
- Length of a user's tweets in characters;
- Average word length in a user's tweets;
- Ratio of words longer than 5 characters in a user's tweets;
- Type-token ratio.

As these features mostly rely on counting words and performing some basic calculations, they were relatively easy to implement. They are all based on tokenized tweets without punctuation. The last two features partly overlap with the group of readability features, as they are considered predictors of readability (Flekova et al., 2016).

4.1.2 Readability

Flekova et al. (2016) used a host of readability measures. They all have some commonality in the way they are calculated, but differ in measuring scale and intended application. Instead of implementing each measure with all its peculiarities by hand, we relied on the *readability* library (van Cranenburgh, 2016) for their calculation. This library provides a function that takes sentence tokenized text as its input and outputs the scores for several readability metrics. As sentence tokenization has already been performed by NLTK, we only need to provide the right input data for each user to calculate its scores. These texts do include punctuation symbols, as they are needed for the calculation of some measures.

The readability metrics included in our research are:

- Automated Readability Index;
- Coleman-Liau Index;
- Flesch-Kincaid Grade Level;
- Flesch Reading Ease;
- Gunning-Fog Index;
- LIX Index;
- SMOG Index.

4.1.3 N-grams

For the final feature set, we looked at the syntax and style sets in the study by Flekova et al. (2016). The syntax set proved unimplementable in this short timeframe. Although the Alpino parser (Bouma et al., 2001) could provide us with the needed part-of-speech tags, the parser could not do so reliably. The parser would sometimes crash on input it could not handle.

We therefore moved on to the style set. Whilst some features like the ratio of words with numbers were easily implementable, others would prove more problematic. A lot of features depend on part-of-speech tags. Without a proper functioning parser we cannot implement those features in this research. We had no other choice than looking elsewhere for a possible feature set to add.

Inspired by the work of fellow student Reinard van Dalen on classifying users based on their political preference, we decided to opt for word n-grams as our third and final feature set. We use a modified version of a function provided by our supervisor in order to generate the n-grams. It calls on NLTK's *ngrams* function to compute all possible n-grams for a given text.

We chose to include unigrams, bigrams and trigrams in our research. They are counted in a binary fashion: the fact that a certain n-gram occurs in a user's text is recorded, but the amount of occurrences is not taken into account.

4.2 CLASSIFICATION

As stated in the previous chapter, incomes are not available for every occupation, but only for a select number of occupational classes. The machine learning task ahead of us should therefore be framed as classification, instead of regression. Handling in the spirit of Flekova et al. (2016), we tried to find a classification method that is closest to the methods they have used, despite the difference in prediction task. They used linear regression and vector support regression with an RBF kernel as its non-linear counterpart for comparison.

Two classification methods came to mind: a support vector machine (SVM) and logistic regression. We chose to only use the latter. Scikit-learn's documentation on SVM's states that "if the number of features is much greater than the number of samples, the method is likely to give poor performance" (scikit-learn, 2010). As one of our feature sets is n-grams, one can expect thousands of features, while the data set only consists of two

thousand samples. We therefore shifted our focus to logistic regression and implemented it in our classifier using scikit-learn's *LogisticRegression* module.

4.3 EVALUATION

After implementing all features, the performance of the classifier and its composing features is assessed by using k-fold cross validation. In our case we apply a k of 10, like Flekova et al. (2016).

For each fold, the results were printed to screen, including a list of most informative features. After each validation run the precision, recall and F1-scores are calculated in order to compare the different classifier setups. We test a host of different setups in order to find the most effective combination (i.e. giving us the highest F1-score with the least amount of features). The tested feature compositions and their results are shown in Table 1.

The classifiers will all be compared to a baseline. As the data set consists of two equally large groups, a majority baseline cannot be used. We therefore use a random baseline. Given that there are two classes and they are equal in size, the chance that a single instance belongs to a certain class is 50 percent ($P_{correct} = 0.5$). We rely on scikit-learn's *StratifiedKFold* module to split our data for each of the ten folds. The stratification makes sure that the equality in class size in our data set will be carried through to the individual test and training sets, and thus guarantees that our baseline will be the same on each run.

5 RESULTS AND DISCUSSION

All evaluation results for the different classifier setups and our baseline method can be found in Table 1. All setups outperformed our baseline, but the extent differs quite a lot.

Overall, it is clear that the n-gram feature groups are by far performing the best. Surface and readability feature groups, either alone or combined, reach an F1-score in the higher fifties, whereas the n-gram feature groups score at least in the higher sixties. It seems that the limited scope of features, provided by the surface and readability sets, is extensively outweighed by the n-gram sets.

Now, it has to be determined which classifier is the most robust. There are five setups that showed equal performance, all reaching an F1-score of 0.72. These setups are:

- #7: N-grams (n=1-2).
- #10: N-grams (n=1-2-3);
- #18: Readability + N-grams (n=1-2);
- #25: Readability + N-grams (n=1-2-3);
- #32: Surface + Readability + N-grams (n=1-2-3);

In order to draw conclusions on robustness, the standard deviations of the intermediate F1-scores, provided by the individual folds of the 10-fold cross validation, are calculated. The calculation was performed by IBM SPSS, a popular statistical software package. The outcomes can be found in Table 2.

Although all setups yield roughly equal minimum and maximum F1-scores, there is some difference in standard deviation. An interesting finding is the negative influence of adding readability or both surface and readability features to the n-gram features. In all cases considered, the n-gram setups have a lower standard deviation than comparable setups with additional features.

The setup with uni-, bi- and trigrams clearly stands out as the most robust method, having a standard deviation of 0.01581. The results make clear that simply adding more and more features does not necessarily have a positive effect on a method's robustness. We can see that the standard deviation drops when we add trigram features to a setup with uni- and trigram features. This improvement is diminished when we add even more features, like readability or both surface and readability features.

Table 1: An overview of precision, recall and F1-scores for the different classifier setups.

#	Setup	Precision (P)	Recall (R)	F1-score
1	Baseline	0.50	0.50	0.50
2	Surface	0.56	0.56	0.56
3	Readability	0.57	0.57	0.57
4	N-grams (n=1)	0.70	0.70	0.70
5	N-grams (n=2)	0.69	0.69	0.69
6	N-grams (n=3)	0.68	0.68	0.68
7	N-grams (n=1-2)	0.72	0.72	0.72
8	N-grams (n=1-3)	0.71	0.71	0.71
9	N-grams (n=2-3)	0.70	0.70	0.70
10	N-grams (n=1-2-3)	0.72	0.72	0.72
11	Surface + Readability	0.59	0.59	0.59
12	Surface + N-grams (n=1)	0.70	0.70	0.70
13	Surface + N-grams (n=2)	0.70	0.70	0.70
14	Surface + N-grams (n=3)	0.67	0.67	0.67
15	Readability + N-grams (n=1)	0.70	0.70	0.70
16	Readability + N-grams (n=2)	0.69	0.69	0.69
17	Readability + N-grams (n=3)	0.68	0.68	0.68
18	Surface + N-grams (n=1-2)	0.71	0.71	0.71
19	Surface + N-grams (n=1-3)	0.70	0.70	0.70
20	Surface + N-grams (n=2-3)	0.70	0.70	0.70
21	Surface + N-grams (n=1-2-3)	0.71	0.71	0.71
22	Readability + N-grams (n=1-2)	0.72	0.72	0.72
23	Readability + N-grams (n=1-3)	0.71	0.71	0.71
24	Readability + N-grams (n=2-3)	0.70	0.70	0.70
25	Readability + N-grams (n=1-2-3)	0.72	0.72	0.72
26	Surface + Readability + N-grams (n=1)	0.69	0.69	0.69
27	Surface + Readability + N-grams (n=2)	0.70	0.70	0.70
28	Surface + Readability + N-grams (n=3)	0.66	0.66	0.66
29	Surface + Readability + N-grams (n=1-2)	0.71	0.71	0.71
30	Surface + Readability + N-grams (n=1-3)	0.70	0.70	0.70
31	Surface + Readability + N-grams (n=2-3)	0.69	0.69	0.69
32	Surface + Readability + N-grams (n=1-2-3)	0.72	0.72	0.72

Table 2: An overview of the minimum and maximum F1-scores as well as the standard deviation of each selected setup.

#	Minimum F1	Maximum F1	Average F1	Standard deviation
7	0.68	0.75	0.72	0.02221
10	0.69	0.74	0.72	0.01581
18	0.68	0.75	0.72	0.02321
25	0.69	0.74	0.72	0.01767
32	0.68	0.75	0.72	0.02119

6 CONCLUSION

6.1 SUMMARY

We started our research with the question: *to what extent is it possible to accurately predict the income level of a Dutch Twitter user and which stylometric features are predictive?* Earlier work on English speaking users looked promising. Flekova et al. (2016) concluded that differences in style could be used to "tailor the style of a document without altering its topic to either income or age individually". This matched our hopes for a potential application of this technique: tailoring customer service communication to customers without having to ask them for background information.

As a comparable study has not been done yet, we had to create a dataset from scratch. By employing the labeling strategy established by Sloan et al. (2015), we were able to create a corpus of 21862 users with known incomes. They were divided into two income classes, one above and one below €34.500. From each class thousand users were selected for further research. Their 500 latest tweets were preprocessed for use in our classifier.

In order to classify our users we employed a logistic regression method provided by scikit-learn (Pedregosa et al., 2011) with three sets of features: surface, readability and n-gram features. After testing a host of different feature group combinations on our classifier, we managed to get a maximum F1-score of 0.72 using 10-fold cross validation, outperforming our baseline F1 of .5. The feature set with only uni-, bi- and trigrams proved to be the most robust, providing the highest average F1-score with the lowest standard deviation.

6.2 LIMITATIONS

As our study was limited in the time and resources available, it is far from conclusive. We will describe a few of its limitations below.

6.2.1 Availability of income data

The first important limitation of our study is the level of detail in the income data used to label our users. As Statistics Netherlands does not provide incomes for all occupations, we were forced to use and predict income classes instead of individual incomes. By using the average incomes for each two-digit group, chances are that there is a large gap between the predicted income and the actual income of a user. This might lead to erroneous classifications. As we have no full income data, we cannot assess the size of the problem at this time.

6.2.2 Labeling strategy

Our labeling strategy worked reasonably well, as demonstrated by the random sample in chapter 3. However, it is far from perfect. One problem is that our method cannot distinguish between human and non-human users. Although this is such a complex topic that it would warrant a separate study, adding this capability would help to reduce the noise in our dataset. Another problem is that our labeling method relies on just one heuristic: it will always take the first occupation it can find, even if there are multiple occupations mentioned by a user. It currently has no way to disambiguate between multiple occupations. We will present possibilities for future research on our labeling strategy in the next section.

6.2.3 Class and classifier setup

We originally planned to use two different class setups: a two class setting with high and low classes and a six class setting used by Statistics Netherlands (€0 - €10.000, €10.000 - €20.000, €20.000 - €30.000, €30.000 - €40.000, €40.000 - €50.000 and €50.000 or more). As our data gathering and annotation efforts proved to be quite challenging, given that there was no prior gathering and annotation method for Dutch users, we decided to limit ourselves to the two class setting.

For the same reason, we had to limit our exploration of different classifier setups. We planned to do a thorough review like Flekova et al. (2016), where each individual feature was measured on its predictiveness using two different regression methods. We limited ourselves to one classification method (logistic regression) as the documentation of scikit-learn indicated that *support vector machines* would likely give poor performance on our feature sets (scikit-learn, 2010). This would have mainly affected setups with n-gram features. Exploration of different classification methods may prove worthwhile in improving results.

Regarding the included features, we only assessed the influence of different feature set combinations, including different levels of n-grams, as they greatly influence the amount of features available to our classifier. This limited assessment already yielded 31 distinct feature group combinations to be evaluated. As this proved to be time consuming, more detailed evaluations had to be left out.

6.3 FUTURE WORK

A major boost to future work on profile Dutch authors on income would be the availability of incomes on a full four-digit level so individual incomes can be predicted and not just income classes. However, as we depend on Statistics Netherlands for our income data, it is out of our hands. For now, there are three clear avenues for future research using the available data.

6.3.1 Improvement of labeling method

Although our current labeling method is a good basis, it could become more robust. As already mentioned, disambiguation between multiple occupations by selecting the one with the highest ISCO class would be one possi-

bility for improving our labeling accuracy, as it would also limit the amount of hobbies, studies and volunteer's work that was falsely detected as occupations in cases where multiple candidate occupations are present.

6.3.2 Different class setups

As we only tested a two class setup, it would make sense to test the performance of our classifier on a data set with more than two classes. This would also bring the income classes closer to the actual income of the user, thus resulting in more valuable predictions. It would be interesting to see how the predictive power of certain feature sets hold up in a different situation.

6.3.3 In-depth analysis of features

As we only investigated the influence of entire feature groups on our classifier's performance, it might be worthwhile to research the influence of individual features in the surface and readability groups. It might become clear that an entire different mix of individual features yields better results than the best ones our current setup could provide.

6.4 FINAL STATEMENT

This study has been a first exploration of the possibilities of profiling Dutch authors on their income. We hope that this may be the basis of further research and, in combination with more detailed income data, be a stepping stone towards predicting the income of individual Dutch authors.

BIBLIOGRAPHY

Bird, S., E. Klein, and E. Loper (2009). *Natural Language Processing with Python*. O'Reilly Media.

Bouma, G., G. Van Noord, and R. Malouf (2001). Alpino: Wide-coverage computational analysis of dutch. *Language and Computers 37*(1), 45–59.

European Observatory of Working Life (2015). *Developments in collectively agreed working time 2014*.

Flekova, L., D. Preoţiuc-Pietro, and L. Ungar (2016, August). Exploring stylistic variation with age and income on twitter. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Berlin, Germany, pp. 313–319. Association for Computational Linguistics.

International Labor Office (2013). *International Standard Classification of Occupations 2008 (ISCO-08)*. International Labor Office.

Kiss, T. and J. Strunk (2006, 2016/12/18). Unsupervised multilingual sentence boundary detection. *Computational Linguistics 32*(4), 485–525.

Li, J., A. Ritter, and E. Hovy (2014, June). Weakly supervised user profile extraction from twitter. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Baltimore, Maryland, pp. 165–174. Association for Computational Linguistics.

Lui, M. and T. Baldwin (2012). Langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, Stroudsburg, PA, USA, pp. 25–30. Association for Computational Linguistics.

Office for National Statistics (2010). *The Standard Occupational Classification (SOC) 2010 Vol 1: Structure and Descriptions of Unit Groups*. Palgrave Macmillan.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Preotiuc-Pietro, D., V. Lampos, and N. Aletras (2015, July). An analysis of the user occupational class through twitter content. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China, pp. 1754–1764. Association for Computational Linguistics.

Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL Student Research Workshop*, ACLstudent '05, Stroudsburg, PA, USA, pp. 43–48. Association for Computational Linguistics.

scikit-learn (2010). 1.4. support vector machines. Retrieved from http://scikit-learn.org/stable/modules/svm.html.

Sloan, L., J. Morgan, P. Burnap, and M. Williams (2015, 03). Who tweets? deriving the demographic characteristics of age, occupation and social class from twitter user meta-data. *PLoS ONE 10*(3), 1–20.

Statistics Netherlands (2014a). *Codelijsten ISCO-08*. Retrieved from https://www.cbs.nl/nl-nl/onze-diensten/methoden/classificaties/onderwijs

Statistics Netherlands (2014b). *Uurlonen van werknemers naar beroepsgroep, 2012*. Retrieved from https://www.cbs.nl/nl-nl/maatwerk/2014/15/uurlonen-van-werknemers-naar-beroepsgroep-2012.

van Cranenburgh, A. (2016). Readability. Retrieved from https://github.com/andreasvc/readability.

van der Walt, S., S. C. Colbert, and G. Varoquaux (2011, March). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering 13*(2), 22–30.

A APPENDIX

A.1 LIST OF OCCUPATIONAL CLASSES AND RESPECTIVE OCCUPATIONAL TITLES

As this file is very long, it could not be included in the text. The file is provided in the zip-file included with this submission and is also provided online via:
https://www.thesociallions.nl/thesis/.

A.2 LIST OF AVERAGE HOURLY INCOMES FOR EACH OCCUPATION CLASS

Income class	Average hourly income (€)
11	43.02
12	39.95
13	35.77
14	22.86
21	29.10
22	30.91
23	28.40
24	31.50
25	28.45
26	28.30
31	26.33
32	21.72
33	25.42
34	20.42
35	24.80
41	19.42
42	16.96
43	22.24
44	18.90
51	15.74
52	16.24
53	19.12
54	20.65
61	16.12
71	20.60
72	19.17
73	19.07
74	19.45
75	17.73
81	17.71
83	17.52
91	13.89
93	15.28
94	13.23
96	17.03

A.3 LIST OF COMMANDS FOR USING OUR TOOLSET

The toolset is divided into three modules: *datagathering*, *preprocessing* and machinelearning. All included scripts can be imported as a module in a Python script and be used as a user wishes. Full docstrings are provided in the files themselves. The libraries used can be installed by using the included requirements.txt file, using the command *pip install -r requirements.txt*. The commands below list all scripts that can run standalone in order to recreate our setup.

By default, all created files are written to the *output_files* folder in the *supportdata* directory of the project. Reference output files are provided in the folder *reference_output* in the same directory.

A.3.1 Data gathering

All scripts for data gathering can be found in the *datagathering* folder of the project. The description below will guide you through all needed steps to gather the labeled user data.

A.3.1.1 *Loading income data from a CSV file*

Script: import_incomes.py
Default command python3 import_incomes.py <path_to_input_file>
Input: a comma separated file with submajor groups in the first column and their respective average hourly incomes in the second column. The file used in my research is *hourly_income_per_submayor_group.csv*, which is situated in *supportdata/input_data/*.
Output: a dictionary containing the submajor groups as keys and incomes as values, written to disk with Pickle.
Comments: the method uses ; as delimiter by default, as it is the one used by Microsoft Excel.

A.3.1.2 *Connecting occupational titles with the average income for their submajor group*

Script: connect_titles_with_incomes.py
Default command: python3 connect_titles_with_incomes.py <path_to_input_file>
Input: a comma separated file with submajor groups in the first column and a comma separated string of example occupational titles in the third column. The file used in my research is *edited_occupational_titles.csv*, which is situated in *supportdata/input_data/*.
Output: a dictionary containing the example occupational titles as keys and a Tuple containing occupational class and income as values, written to disk with Pickle.
Comments: the method uses ; as delimiter by default, as it is the one used by Microsoft Excel.

A.3.1.3 *Collecting users with known incomes from the twitter2 corpus*

Script: collect_labeled_users.py
Default command: zcat /net/corpora/twitter2/Tweets/2016/09/20160901..5* | tweet2tab user.id user user.name user.description | python3 collect_labelled_users.py
Input: a tab separated user data from the twitter2 corpus, generated with tweet2tab as demonstrated in the above command. We extract the user id,

username, full name and description for each user. This data is loaded by default via standard input, but can also be read from a similarly formatted file by calling the script with a parameter containing the input file as first argument.
Output: a dictionary containing the user id's as keys and a Tuple containing username, real name, description, found occupation, occupational class and incomes, written to disk with Pickle.
Comments: this script requires output from the *twitter2* corpus formatted with *tweet2tab* or a similar corpus that provided similarly formatted output for use as input data. The *twitter2* corpus can be found on the *Karora* server of the University of Groningen.

A.3.1.4 *Removing non-existing users, private users and user with less than thousand tweets*

Script: check_user_profiles.py
Default command: python3 check_user_profiles.py
Input: a Pickle file containing a dictionary with user id's as keys and Tuples with user information as values. This file is generated by the previous script. The file used in my research is *labeled_users.pickle*, which is situated in *supportdata/input_data/*.
Output: a dictionary containing the user id's as keys and a Tuple containing the user data, written to disk with Pickle.
Comments: to use the Twitter API, we need a set of access tokens. These can be requested at `https://dev.twitter.com`. We used six sets of access tokens to speed up our data gathering. The tokens need to be entered in a JSON file named credentials.json, located in the directory *datagathering/twitterapi/private*. An example file is provided in the same folder.

A.3.1.5 *Creating two class split of users*

Script: create_income_classes.py
Default command: python3 create_income_classes.py
Input: a Pickle file containing a dictionary with user id's as keys and Tuples with user information as values, generated by the previous script. The file used in my research is *suitable_users.pickle*, which is situated in *supportdata/output_files*.
Output: a dictionary containing the class labels as keys and Lists containing a List per user with its user id as Int and user information in a Tuple, written to disk with Pickle.
Comments: by default, the users are split in a low and high class, respectively with an average yearly income below €34.500 and above.

A.3.1.6 *Manual evaluation of labels gathered with distant supervision*

Script: evaluate_annotations.py
Default command: python3 evaluate_annotations.py
Input: a Pickle file, containing a dictionary with the class labels as keys and Lists containing a List per user with its user id as Int and user information in a Tuple, as generated by the previous script. The file used in my research is *users_in_classes.pickle*, which is situated in the *output_data* folder of *supportdata*.
Output: the outcomes of the manual evaluation are only printed to screen.
Comments: by default, the script picks 100 users per class.

A.3.1.7 *Preselection of user for retrieval of tweets*

Script: pre_select_users.py
Default command: python3 pre_select_users.py
Input: a Pickle file, containing a dictionary with the class labels as keys and Lists containing a List per user with its user id as Int and user information in a Tuple, as generated by the previous script. The file used in my research is *users_in_classes.pickle*, which is situated in the *output_data* folder of *supportdata*.
Output: a dictionary containing the classes as keys and a List of selected user id's as values, written to disk with Pickle.
Comments: by default, the script picks 2000 users per class.

A.3.1.8 *Gathering of user posts*

Script: gather_user_posts.py
Default command: python3 gather_user_posts.py
Input: a Pickle file, containing a dictionary with the class labels as keys and Lists with the selected user id's as values, as generated by the previous script. The file used in my research is *preselected_users.pickle*, which is situated in the *output_data* folder of *supportdata*.
Output: a corpus of user's tweets, written to text files. Every user has a separate text file with its id in the folder of its class.
Comments: by default, the script loads the latest 1600 tweets of the user. Furthermore, it writes all suitable tweets to disk.

A.3.1.9 *Selection of users for further research*

Script: post_select_users.py
Default command: python3 post_select_users.py
Input: a String containing the location of the gathered tweets. By default, this is the *corpus* folder.
Output: a dictionary containing the classes as keys and a List of selected user id's as values, written to disk with Pickle.
Comments: by default, the script picks 1000 users per class.

A.3.2 Preprocessing

A.3.2.1 *Preprocessing of users and their tweets*

Script: tokenizer.py
Default command: python3 tokenizer.py
Input: a Pickle file, containing a dictionary with the class labels as keys and Lists with the selected user id's as values, as generated by the previous script.
Output: a dictionary containing the classes as keys and a List with a Tuple per user, holding all tokens per tweet in a List and sentence tokenized text as a String, written to disk with Pickle.
Comments: punctuation is removed from the tokens, but not from the sentence tokenized text. The punctuation is needed to calculate the readability measures.

A.3.3 Machine learning

A.3.3.1 *Generating prefeaturized dictionaries for evaluation*

Script: generate_test_dicts.py
Default command: python3 generate_test_dicts.py
Input: none.
Output: dictionaries containing the different feature setups, written to disk with Pickle.
Comments: we extracted the features before running the evaluation set, as this can be quite time consuming on runtime.

A.3.3.2 *Evaluation of different feature setupd*

Script: evaluation.py
Default command: python3 evaluation.py
Input: none.
Output: the outcomes of the cross validation runs are printed to screen.
Comments: the paths to the dictionaries containing the features for every setup by the former script is hardcoded. As the file are very large in size and Nestor does not support the upload of such big files, they had to be left out of this submission. They can always be generated with the prior script.