A Method for Hebrew Stylometry, Based on Short Messages

Anonymous ACL submission

Abstract

@asi

Given a set of short text messages in Hebrew, we attempt to classify each sentence to it respective author, according to its style of writing. We use Tweeter as our data set, and we've checked 10 users with approximately several thousands of tweets per user to train our model. We then use a different set of several hundreds of tweets per user in order to evaluate our model's accuracy.

1 Introduction

@asi
Stylometry is the practice being used to identify an author based on his style of writing. While stylometry has been used historically to identify authors of long texts such as novels and plays, legacy practices were proven inefficient when applied to short text messages such as SMS[1] and social media posts. Recently, more novel approaches were suggested for stylometry that utilize algorithms in machine learning and deep learning, to improve accuracy of stylometry on short texts. These works rely on several learning models, such as Bayesian statistics *(INSERT CITATION)*, Support Vector Machine *(INSERT CITATION)*, Neural Networks *(INSERT CITATION)*, and others. These works also suggest different approaches to features selection. The study of feature selection may not be suitable to any language, as different languages may hold different features that may be suitable to a more accurate system. In this work we started an initial research and benchmarking on the influence of different learning models and features, in order to learn more about which of these models and features influence on the accuracy of a possible stylometry system for short messages in Hebrew. Code and resource can be found on our github repository: https://github.com/asilev/NLP_FinalProject

[1]Short Message Service

2 Article Structure

@asi

This document is arranged according to the following order: We begin by describing related work this article relies on in section ??. We then proceed by describing the architecture of our utility, that implements a framework for model benchmarking in section ??. After that, we describe our approach of research and benchmarking in section ??. We then describe our results in section ??. Further work is suggested in section ??. Finally, we conclude our work in section ??.

3 Related Work

@ronen
There has been significant amount of work and progress achieved in the past 10-15 years in the field of author classification and identification based texts, initially ranging from a few paragraphs to complete books. As the usage of emails in general, and social media in particular, became increasingly popular, significant amounts of work were invested in shorter texts. On longer texts, usually satisfactory-high results were achieved. Qian, He and

Zhang [2017, (?)] showed that deep learning models on authorship identification and authorship verification can yield an accuracy of 69.1% on Reuters_50_50 (Reuters data-set) and 89.2% on Gutenberg data-set (a data-set established by the authors), while Siamese network achieved a verification accuracy of 99.8% on both Reuters_50_50 and Gutenberg data-sets.
However, despite the vast amount of work invested, it is still challenging to achieve reasonably-high detection results on messages derived from the aforementioned social media (Twitter, Facebook, etc.), emails and such, especially as the number of authors increases; for a small set of authors (sometimes as low as 2) good results can be achieved. Green and Sheppard (?) experimented with both BOW and Style Markers, and showed that SM is superior (average classification accuracy of 92.3% vs. 76.7%). However, this high classification rate was achieved on 2 authors, whereas increasing from 2 to 5 authors dropped the accuracy quickly (around 55%) up to the lowest accuracy of 40% reached on 12 authors. Schwartz, Tsur, Rappoport and Koppel (?) experimented (among other criteria) with varying training set sizes and number of authors. They demonstrated an improvement in accuracy from 49.5% (50 tweets per 50 authors) to 69.7% (larger amount of tweets per 50 authors), but with a large number of authors (1000) showed 30.3% accuracy (using 200 tweets/author), which correlates to the trend shown in other works as well. Albeit, 30.3% is still distinguishably better than the random baseline of 0.1%. This trend is further validated in the work of Brocardo, Traore, Saad and Woungang (?), albeit their work was founded on emails from the Enron data-set (200K emails from 150 users, average number of words per email is 200). Their techniques were based on a combination of supervised learning and n-gram analysis. Even after applying a few pre-processing operations on the text (e.g. e-mails were combined to produce a single long message per individual, and then divided into smaller blocks used for authorship verification), their experiments yielded an EER (an Equal Error Rate metric used by the authors which corresponds to the operating point where False Acceptance and False Rejection rates have the same value) of 14.35% for 87 users for relatively small block sizes.
It is important to notice that thus far the related works described herein are mostly involved with messages in the English language. In an overall perspective, little work was done on Semitic languages such as Hebrew and Arabic. The work done by Rababah, Al-Ayyoub, Jararweh and Aldwairi (?) on tweets in Arabic showed the best accuracy when employing SVM technique on a unified collection of features (more than 900) using tweets from 12 authors, usually around 3000 tweets per author. The best accuracy achieved was 68.67%.
An excellent and comprehensive overview of modern authorship attribution methods can be found in (?).

4 Architecture

@asi

Figure ?? illustrates the architecture of the benchmarking platform we've built for this work.

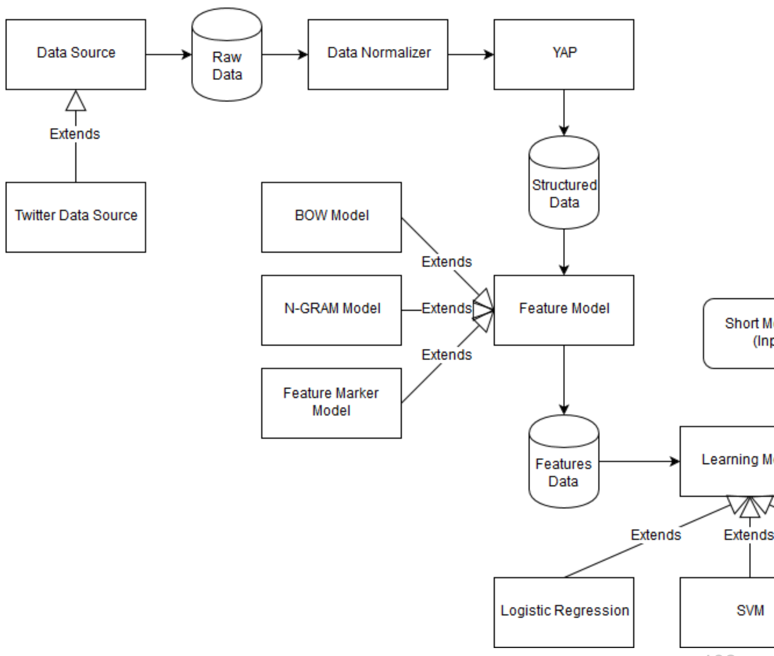

Figure 1: Platform Architecture

We use "Data Source" as the abstract utility to pull text from short messages source, and save the messages with the correct tagged author for each short message in a well formed structure. In here we've implemented the "Data Source" with a "Tweeter Data source", that pulls information according to parameters such as time period and a list of users from tweeter. This data is then being processed by "Data Normalization" utility, that reads this data, filters unwanted cases (as will be described in section **??**), and transforms the data to a structure that can be used by YAP *CITE*. We then use YAP to provide initial parsing for our data, and provide more information about it, such as morphological segmentation, stemming, POS tagging, etc. Then, we choose our features according to some model. This component takes the list of sentences, and builds for each sentence a list of features, where each feature is assigned with a value that describes that feature for the particular sentence. We create two lists of sentences: one for the training (gold list) and one for the evaluation (test list). Then, we use some learning model to train on the gold list, and produce the parameters that will be used for the encoder. Lastly, we use the encoder on the test list, and compare its results with the actual authors from the test list using the evaluation component, to test the system's accuracy (hit rate).

5 Approach

@asi
We now describe the benchmarking approach step by step, and describe how each step is implemented in our work.

5.1 Data Acquisition

@ronen
We used the Twitter4j library (see section **??**) in order to extract the tweets for our 20 authors. Twitter4j is an unofficial yet a very popular library for using the Twitter API. As we needed to get thousands of tweets per user, we used the "getUserTimeline()" API using paging (as Twitter API limits the number of tweets one can get in a single API call). Overall we executed 50 user timeline page API calls, with 200 tweets for each page. For most users this ended up with about 3000-4500 tweets, but obviously some had fewer and one had significantly more (around 10.5K). See table **??** for more details.

5.2 Data Exploration

@ronen
In order to conduct our experiments we could not use the data we extracted via the Twitter API "as-is". The tweets we extracted directly from the Twitter API were in JSON format containing an abundance of information (content and meta-data) such as creation date, links, re-tweets, mentions, etc. This format not only includes more information than needed for the task at hand, but is also impossible to be processed by a morphological analyzer (such as YAP, see section **??**). As such, we needed to apply a series of operations (i.e. "normalize", see next section) on the raw data in order to be able to run YAP on it and get the suitable output we needed to continue with the other steps in the model (see figure **??**).

5.3 Data Preparation

@ronen
We performed the following operations on the raw tweet data in order to prepare it for YAP morphological analysis:
A) Strip any meta-data information
B) Ignore/omit any re-tweets: As we're interested in the style of only the particular author, text from other authors can only impede the analysis
C) Remove English words and characters
D) Leave punctuation marks
E) Formatting: For each tweet (i.e. in YAP's "eyes" this is a sentence) format it such that each word (or punctuation mark) is in its own line, and tweets are separated from one to another by an empty line.
The above steps generated a "normalized" input file, per author, ready to be processed by YAP. We then executed two YAP commands - morphological analysis and dis-ambiguity. The output from these operations was a file (per author) containing all the tweets after analysis, resulting in the following information for each tweet:

- **OriginalWord**
- **StemmedWord**
- **OriginalWord POS**
- **StemmedWord POS**

- **Gender**

- **Single or Plural**

- **Person Attribute (first, second, etc.)**

Figure **??** summarizes the statistics of the data we used in our experiments after which being run through the steps depicted above, covering the authors (name and user ID), their area of work, the number of tweets we got using the Twitter4j API, and the number of tweets we ended up with after pre-processing (preparing the tweets in the format that YAP requires) and applying the YAP tool:
Focus on Hebrew such as morphemes separation Comment: credit as required by forum

Table 1: Data Statistics

User Name	**User ID**	**Area of Work**
Ben Caspit	bencaspit	Political journalist
Nadav Eyal	NadavEyalDesk	Political journalist
Amit Segal	amit_segal	Political journalist
Ayala Hasson	AyallaHasson	Political journalist
Guy Rolnik	grolnik	Financial Journalist
Tal Schneider	talschneider	Political Commentator
Alon Ben-David	alonbd	Military Journalist
Rino Zror	RinoZror	Political journalist
Or Heller	OrHeller	Political journalist
Baruch Kra	baruchikra	
Udi Segal	usegal	Political journalist
Keren Neubach	kereneubach	Reporter
Shaul Amsterdamski	amsterdamski2	Financial Journalist
Roy Sharon	roysharon	Military Journalist
Akiva Novick	akivanovick	Parliamentary journalist
Sefi Ovadia	sefiova	Political journalist
Yoaz Hendel	YoazHendel1	Political Advisor and Commentator
Dan Margalit	Danmargalit	Journalist and Commentator
Sivan Rahav-Meir	SivanRahav	Political journalist
Zion Nanous	zionnenko	News journalist
Total	**20**	**N/A**

5.4 Feature Selection and Benchmarking

@?

There are several approaches to features selection. In (**?**) two approaches, namely bag of words vs. style markers. In bag of words, each feature is a word, and the associated value is the number of time this word appears on the sentence. In Style Marker, each feature is a selected feature of the sentence. It's value can be any numeric classifier for the feature, such as exist (1) or non exist (0), number of occurrences, or probability (percentage of occurrences). While testing each of these approaches ourselves, we suggest a more generalized approach that also tests combinations of these approaches, by using some features as words from bag of words model, and some *additional* features as style markers. We also tested different approaches, such as N-Grams and POS analysis. The list features we used is:

- ***Simple Word (Bag of Words)***: Each feature is a word, and the value is the number of times the word appears in the sentence.
- ***Stemmed Word***: Each feature is the stem analysis of a word, and the value is the number of times the stemmed word appears in the sentence.
- ***Bigram***: Each feature is a sequence of two words, and the value is the number of times this pair of words appears in the sentence. We can further generalize this feature to any N-gram, but for a small corpus, this is proven as ineffective even for a bigram, since specific pair are very unlikely to reappear.
- ***POS Analysis***: Each feature is a word, combined with its proper POS tag. This helps separate different words that may be written the same way, but used for different meanings.
- ***Number of Morphemes***: A single feature, describing the number of morphemes the sentence holds.
- ***Average Word Size***: A single feature, describing the average size of a word in the sentence.
- ***Number of Characters***: A single feature, describing the number of characters in the sentence.
- ***Number of Punctuation Marks***: a single value, describing the number of punctuation marks used in the sentence.
- ***Average Number of Punctuation Marks***: A single value, describing the number of morphemes that represent punctuation marks, relative to the total number of morphemes in the sentence.
- ***POS Usage***: Each feature is a POS, and the value is the number of times the POS appears in the sentence.
- ***Average Sentence Size***: If the given sentence is divided to more than one actual sentence using a dot, this feature holds the average number of morphemes in each actual sentence in the complete given sentence.
- ***Number of Long Words***: Each feature represents the number of words in the sentences, that are longer than a given threshold.

Since using many features slows down any learning model, and some of the features approaches are inherently large (such as BOW, N-Grams, etc.), we started by testing each feature approach separately to see its strength individually. We then tried combination of strong feature approaches and tested the contribution of the new features to the list of already decided features. *Here we need to describe each step, and show some results*

5.5 Model Selection and Benchmarking

@asi

In this work we've compared two machine learning models: Logistic Regression and Support Vector Machine. In our platform, Logistic Regression model was implemented from scratch, as it is a very basic model, and for SVM we've used *CITE*. We've compared Logistic Regression and SVM for a feature model that contains all style marker features combined. In both models, features were unscaled. We tested each model for 2-10 authors. For logistic regression, we've used the following hyper parameters selection method: α was chosen by searching for the highest α that does not increase the value of the model's cost function. The value we've found ranges from 0.003 to 0.005, depends on the number of authors. The number of iterations was dynamically chosen, as the number of calculation that stabilizes the cost function on a change of <0.0001 per iteration.

For SVM, we've used the following grid search for hyper parameters C and γ, and we chose a polynomial kernel with 1 degree, as these hyper parameters were proven ideal for small cases we ran. We've used the following values for γ: (0.00003, 0.0001, 0.0005, 0.002, 0.008, 0.03125, 0.125, 0.5, 2, 8, 32, 128), and the following values for C: (0.03125, 0.125, 0.5, 2, 32, 8, 128, 512, 2048, 8192, 32768). For this test, we've seen that SVM provides slightly better results, as we can see in the following graph: INSERT GRAPH

Since we've seen that the Logistic Regression model (at least our implementation of it) was extremely slow to train, and results were less optimal comparing to SVM, we chose to use SVM as our model for further investigations. Our work did not test Neural Networks model, and the more novel Convolutional Neural Networks / Recurrent Neural Networks, which might provide better results, as shown in *CITE*. This, together with more in depth research on other models, may be a good further work on the subject. We now check if scaling the features can improve our results for SVM, for Style Marker and for Bag of Words. For Style marker, scaling features provided an accuracy of 21.81We now test SVM with simple Bag of Words feature set. Non scaled simple Bag of Words provided accuracy of 43.89Next, we turn to test other feature sets that resemble bag of words. These models were all tested with SVM, unscaled, and the same hyper parameters that were found to provide best accuracy in the regular Bag of Words model. We've checked the following feature sets:

- ***Bag of Stemmed Words***: Instead of using the original morphemes, we've used the stemmed version.
- ***Bag of Bigrams***: Each feature is a pair of subsequent words. Start of sentence and End of sentence symbols added to each sentence.
- ***Bag of POS-Words***: Each feature is a morpheme, along with its Part of Speech.

6 Results

7 Further Work

possible under discussion, if more subsections required

8 Conclusion

9 Acknowledgements

@ronen
We would like to acknowledge and thank the following publicly available libraries that we used as part of facilitating our model and experiments:
1. Twitter4j: Twitter4J is released under Apache License 2.0, Copyright 2007 Yusuke Yamamoto. For additional information: Twitter4j site
2. YAP: Yet Another Parser is a parser written in Go, developed by *Amir More and Reut Tsarfaty* and is publicly available for research purposes. We used the major two functions of YAP - Pipeline Morphological Analysis and Disambiguation. For more information: YAP on GitHub site
Full citation (TODO: need to further process the 'InProceedings' field):
@InProceedingsmoretsarfatycoling2016, author = Amir More and Reut Tsarfaty, title = Data-Driven Morphological Analysis and Disambiguation for Morphologically Rich Languages and Universal Dependencies, booktitle = Proceedings of COLING 2016, year = 2016, month = december, location = Osaka
3. SVM
4. Others?