Author Profiling Using Support Vector Machines
Notebook for PAN at CLEF 2016

Rodwan Bakkar Deyab, José Duarte, and Teresa Gonçalves

Departamento de Informática, Escola de Ciências e Tecnologia,
Universidade de Évora, Rua Romão Ramalho, 59, 7000-671 Évora, Portugal
d34642@alunos.uevora.pt, d10401@alunos.uevora.pt, tcg@uevora.pt

Abstract The objective of this work is to identify the gender and age of the author of a set of tweets using Support Vector Machines. This work is done as a task for the PAN 2016 which is a part of the CLEF conference. Techniques like tagging, removing stopwords, stemming, Bag-of-Words representation were used in order to create a 10 classes model. The tuning of the model was based on grid-search using k-fold cross-validation. The model was tested for precision and recall with the corpus from PAN 2015 and PAN 2016 and the results are presented. We have experienced the Peaking Phenomenon with the increment of the number of features. In the future we plan to try the term frequency-inverse document frequency in order to improve our results.

Keywords: PAN, CLEF, Author Profiling, Machine Learning, Twitter, Support Vector Machines, Bag-of-Words

1 Introduction

Author profiling problem is about detecting some characteristics (age, gender, for example) of the author of some piece of text depending on the features (eg. lexical, syntactical) of this text. Men and women, and of different ages, write in different ways. Having a dataset in hand, written by different authors of different characteristics, we can train the machine using this dataset so it can predict these characteristics of an unseen piece of text fed to it. PAN 16[1] author profiling task provides a dataset of tweets for the sake of developing an author profiling system. The task is about predicting the age and the gender of the author. Machine learning technique suits to achieve this goal. Support Vector Machines (SVMs)[3] can be used as a multi-class classifier which could be trained using the dataset provided to produce a model which can be consulted on an unseen set of tweets written by some author to predict his age and gender. Bag-of-Words (BOW)[14] is a simplified representation of the text corpus which contains all the words used in it with their frequencies. BOW representation is used in many areas like *Natural Language Processing*[13], *Information Retrieval*[5], *Document Classification* and among others[14]. In our work we use SVMs and BOW representation. We use the python machine learning library, scikit-learn[7]. After we produced the best possible model trained on PAN 16 author profiling dataset, we ran some tests over the test

[1] http://pan.webis.de/clef16/pan16-web/author-profiling.html

sets provided by Tira[2,9]. The work presented in this paper was reviewed and is part of the PAN 2016 overview[11].

This paper is organized as follows: in section 2, the Implementation is described; in section 3, we present the results with features selected and evaluation criteria; in section 4, a retrospective analysis of the work is preformed and a future vision is suggested.

2 Implementation

In this section we describe all the steps of creating the model. We first analyse the dataset, then we present the architecture of the system and at the end we explain the implementation of it.

2.1 The dataset

We used the dataset[2] provided by the PAN 2016 in our study. The corpus contains 436 files, each file contains a set of tweets and these files are written by different authors. The information about each file written by which author is indexed by a file called *turth file*. The file structure is shown in (1) and is explained in Table 1.

$$AID ::: G ::: AR \tag{1}$$

Table 1: Truth file description

Token	Value	Description
AID	Hexadecimal String	Unique author identifier
G	[MALE; FEMALE]	Gender (Male or Female)
AR	[18-24; 25-34; 35-49; 50-64; 65-xx]	Age Range

The Table 2 shows the distribution of the data after analysing it. For example, the corpus contains 14 files written by female authors which have ages between 18 and 24.

2.2 System Architecture

Our system has three modules: *preprocessing*, *training* and *testing* modules. In figure 1 we show the architecture of the system in the training phase. In figure 2 we present the architecture of the system in the testing phase. Both of them use the preprocessing module.

[2] Corpus available in http://pan.webis.de/clef16/pan16-web/author-profiling.html

Table 2: Distribution of the data in the corpus

Gender	Age Range	Number of files	Total
Females	18-24	14 (3%)	218
	25-34	70 (16%)	
	35-49	91 (20%)	
	50-64	40 (9%)	
	65-xx	3 (0.6%)	
Males	18-24	14 (3%)	218
	25-34	70 (16%)	
	35-49	91 (20%)	
	50-64	40 (9%)	
	65-xx	3 (0.6%)	
	Total		436

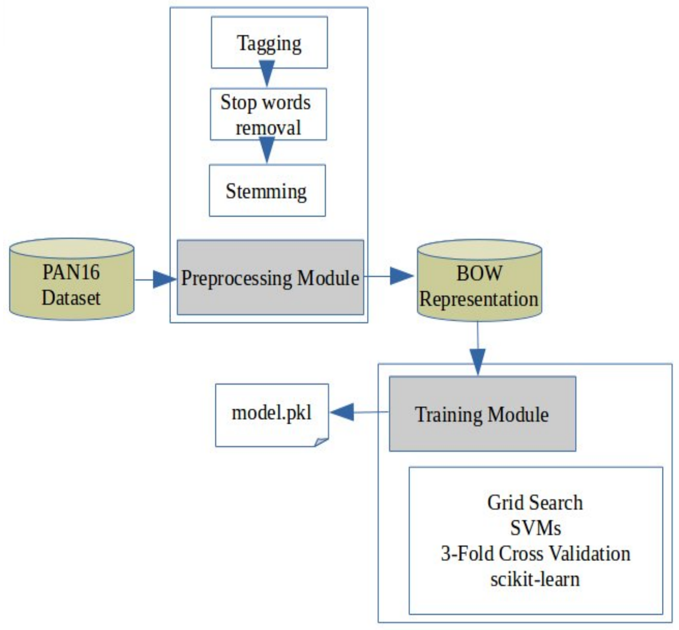

Figure 1: The architecture of the system: training phase

Preprocessing Module

Social Media like twitter is a very noisy environment where informal texts can thrive. As the space is noisy and it does not comply with the syntactic rules of the natural language, NLP (Natural Language Processing)[13] can not be exploited to the best extent.

In our study, we use the BOW[14] representation of the corpus as set of features. Before the BOW generation the data had been transformed. The objective is to opti-

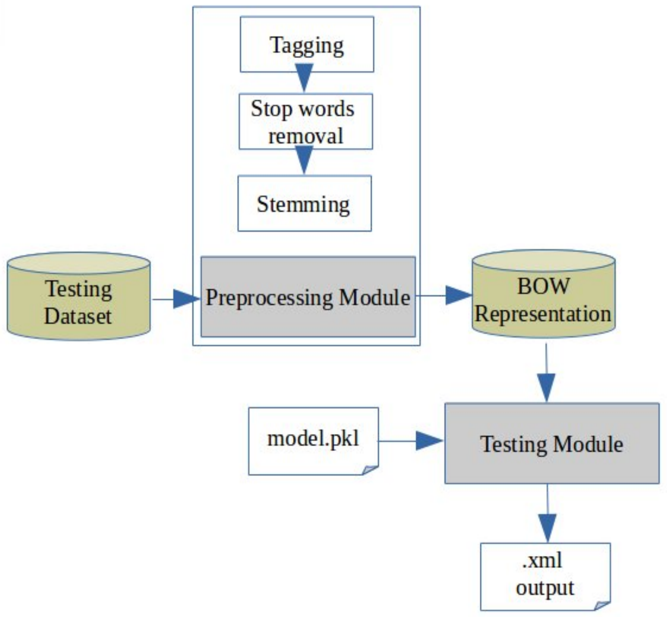

Figure 2: The architecture of the system: testing phase

mize the BOW representation by reducing the words set of the corpus without losing information. This preprocessing is done in *three steps*.

The data in the corpus comes from twitter and has the nature of being noisy containing a lot of abbreviations and special expressions. These special expressions can hold important clues that can differentiate the characteristics of the authors. A *regular expression parser* has been created in order to replace all of these special expressions with predefined tags. This *first step* allows to group expressions and reduce the words set, without losing information. The list of tags with few examples of tokens replaced by them is shown in Table 3.

The *Second step* consists of removing the stopwords form the corpus. Stop words are a set of words like prepositions ("in", "on", "to") and conjunctions ("and", "or"). Usually they carry no information and they are used a lot in the context. The Natural Language Toolkit (NLTK)[1] has a list of English stop words and the scikit-learn[7] too. In the work presented, two lists were merged and used to filter out the corpus.

The *third step* in the stemming. Stemming[6] is the process of finding the root (the lemma) of a given word. Stemming is used in *Information Retrieval*[5] such that, for example, words like "connect, connected, connecting, connection and connections" would be considered as one search word which is the stem of these words "connect". It is useful for the BOW representation such that it reduces the number of the tokens as it may reduce many words to their root and use them as if they were one word. NLTK provides many algorithms for stemming. We used the SnowballStemmer[8] algorithm in our work.

Table 3: Special tags used to preprocess the data corpus

Tag	Examples
_LINK_TAG	http://t.co/jtQvfIJIyg
_NOSY_EMOJI_TAG	:-) :-D :-(
_SIMPLE_EMOJI_TAG	:) :D :(
_FIGURE_EMOJI_TAG	(K) <3
_FUNNY_EYES_EMOJI_TAG	=) =D =(
_HORIZ_EMOJI_TAG	*.* o.O ^.^
_RUDE_TALK_TAG	F*** stupid
_LAUGH_TAG	haha Lol eheheeh
_PUNCTUATION_ABUSE_TAG	!! ????
_EXPRESSIONS_TAG	ops whoa whow
_SHARE_PIC_TAG	[pic]
_MENTION_TAG	@username
_HASHTAG_TAG	#Paris
_NEW_LINE_TAG	a new line in the tweet

The result of the preprocessing module is the BOW model as a list of lists such that each list represents a file of the dataset. The list length is equal to the number of features chosen. The numbers in the list represent the frequency of each word of the Bag-of-Words (the features) in each file in a descending order.

Training Module

Our training module is the core of the work done. It uses the data preparing module to convert the training dataset to the BOW representation as explained before. Each word in the BOW is considered as a feature. We do not use the whole BOW as features but we limit the number, this will be discussed in the result section. After getting the BOW representation of the dataset we divide it into two parts; one part for training (it is two thirds of the whole dataset and we call it the *development set*) and another part for testing (it is one third of the whole dataset and we call it the *evaluation set*). We divide the dataset using the scikit-learn function *train_test_split*.

Then, this module seeks to get the best parameters to train an SVMs classifier on the *development set*. The parameters we seek to get for our SVMs classifier are the *kernel*, *gamma* and *C* parameters. To achieve that we do a hyperparameter tuning through a grid search provided by the scikit-learn library using *GridSearchCV* function. We define a set of parameters to be used by the grid search function as we show in Table 4.

Grid search uses stratified cross validation once for each pair of the parameters provided keeping track of the results it gets. We used a k-fold cross validation with $k = 3$. It is more usual to use this technique with $k = 10$ but due to the small number of some classes, it was not possible as can be seen in Table 2 there are some age ranges with only 3 elements (files). In other words, with classes of small number of files, it was not possible to apply a stratified cross validation with $k = 10$ correctly.

With “rbf(radial basis function)” kernel in our work. We explain how grid search works by a pseudo code (Code 1).

Table 4: Grid Search values for *Gamma* and *C*

	1	2	3	4	5	6	7	8	9
Gamma	0.0001	0.001	0.01	0.1	1	10	100	1000	10000
C	0.0001	0.001	0.01	0.1	1	10	100	1000	10000

Code 1: Grid Search pseudo-code

```
for each_c in c_list:
    for each_gamma in gamma_list:
        results[i] =
            3-fold_cross_validation(each_c ,each_gamma)
```

Results include the *cross-validated-mean-score* and the *standard deviation*. The best parameters are those which produce the highest mean and the lowest standard deviation. For example, (2) is the result which refers to the best parameters after doing the *grid search* over the PAN 16 dataset.

$$kernel : rbf, gamma : 0.0001, C : 100 \tag{2}$$

After we get the model trained on the *development set* using the best parameters we do a test on the *evaluation set*. Getting the result of this test, we produce a classification report to show the results in terms of *precision*, *recall*, *f1-score* and *support*. This will be discussed in the result section. We then used the best parameters we obtained from the grid search to train a classifier on the whole PAN 16 dataset and produce our model which we used to do the Tira tests.

The Testing Module

This module will take again the benefit of the preprocessing module to get the dataset in a suitable format (BOW representation) to consult the model was produced by the training module. It will consult the produced model to predict the age and gender of the author of each file of the test dataset and it will produce an XML file for each one of them. The description of the XML file format is shown in Description 1.

Description 1: XML file format description

```
<author id="author-id"
        type="not relevant"
        lang="en|es|nl"
        age_group="18-24|25-34|35-49|50-64|65-xx"
        gender="male|female"
/>
```

The set of XML files will be the input of the Tira evaluation where accuracy will be calculated as a performance measure.

We hint here that our system was developed just for English language.

3 Results

We did many tests over many datasets (the *evaluation sets* of them) using different sets of features. Our features, as we mentioned before, are the words formed by the BOW such that each word is considered as a feature (taking the frequency of it in each document).

Our results are produced using the classification_report, provided by scikit-learn, over the testing results on the *evaluation sets*. After we obtain the model using grid search over the *development set*, we use it to predict over the *evaluation set* and we run the classification report over the result of prediction. Classification report takes the real target and the predicted target to calculate the precision, recall, f1-score and support for each class was predicted and it calculates the average of these metrics.

First we present some results of the tests on the PAN 16 dataset which has ten classes.

In table 5, we show the results after using a number of features equal to 10000.

Table 5: Results for PAN 16 corpus with 10000 features

	precision	recall	f1-score	support	kernel	gamma	c
class1	0.00	0.00	0.00	2	rbf	0.0001	100
class2	0.22	0.15	0.18	26			
class3	0.31	0.56	0.39	27			
class4	0.25	0.08	0.12	12			
class5	0.00	0.00	0.00	1			
class6	0.00	0.00	0.00	2			
class7	0.43	0.46	0.44	26			
class8	0.29	0.29	0.29	34			
class9	0.33	0.21	0.26	14			
avg / total	0.30	0.31	0.29	144			

In table 6, we show the results after using a number of features equal to 100.

We hint here that the class 10 does not appear in the classification report and that is because of the PAN 16 dataset which contains 436 files, has only 3 files of this class and the way we divided the dataset into a *development set* and an *evaluation set* did not give any file of the class 10 to the evaluation set.

Now we show some results of tests we did on the PAN 15 dataset which has only eight classes. Using a number of features equal to 10000, we present the results in Table 7.

And in Table 8 we present the results after using a number of features equal to 100.

We further discuss the results in the conclusion.

Table 6: Results for PAN 16 corpus with 100 features

	precision	recall	f1-score	support	kernel	gamma	c
class1	0.00	0.00	0.00	2	rbf	0.01	10
class2	0.32	0.23	0.27	26			
class3	0.34	0.67	0.45	27			
class4	0.22	0.17	0.19	12			
class5	0.00	0.00	0.00	1			
class6	0.00	0.00	0.00	2			
class7	0.43	0.35	0.38	26			
class8	0.29	0.29	0.29	34			
class9	0.14	0.07	0.10	14			
avg / total	0.30	0.32	0.30	144			

Table 7: Results for PAN 15 corpus with 10000 features

	precision	recall	f1-score	support	kernel	gamma	c
class1	0.71	0.45	0.56	11	rbf	0.0001	100
class2	0.88	0.58	0.70	12			
class3	1.00	0.29	0.44	7			
class4	0.00	0.00	0.00	3			
class5	0.55	0.75	0.63	8			
class6	0.14	1.00	0.25	3			
class7	1.00	0.67	0.80	3			
class8	0.00	0.00	0.00	4			
avg / total	0.65	0.49	0.51	51			

Table 8: Results for PAN 15 corpus with 100 features

	Precision	Recall	F1-score	Support	Kernel	Gamma	C
Class 1	0.71	0.45	0.56	11	rbf	0.01	10
Class 2	0.65	0.92	0.76	12			
Class 3	0.67	0.29	0.40	7			
Class 4	0.00	0.00	0.00	3			
Class 5	0.60	0.75	0.67	8			
Class 6	0.18	0.67	0.29	3			
Class 7	1.00	0.33	0.50	3			
Class 8	1.00	0.25	0.40	4			
Avg / Total	0.64	0.55	0.54	51			

4 Conclusion and future work

We decided to use the BOW representation as features for our classifier after observing the nature of texts in the social media like twitter. The process of making a parser to replace the special pieces of texts which may mean important in this kind of text and making the BOW (after stemming and stopwords removal) of the resulting tagged text may suit well for this task. But, selecting the right features for SVMs is not an easy task. There are many issues that should be taken into consideration. The scale range of each feature can be a problem[4].

We notice that the results were better for PAN 15 than for PAN 16. That could be because of the tagging process, when we tag the dataset to match special mentions like links and smiles, these special mentions could be found more often in the PAN 15 dataset than in the PAN 16 dataset. In other words, the tagger behaviour is not guaranteed and that depends on the essence of the dataset.

We also notice, from these tests on PAN 16 and PAN 15 datasets that increasing the number of features does not mean necessarily better results. For example, when we used a number of features equal to 100 in the test done for PAN 16 dataset, we got a precision equal to 0.3 and we got the same value of precision for a number of features equal to 10000 for the same test. This is known as the Peaking Phenomenon[12] (PP) and it can occur when using a high number of features. The performance of a model is not proportional to the number of features used, there is a point where the performance deteriorates when more features are added to the model. Procedures already presented in Section 2 like preprocessing the text using tagging, stopwords removal and stemming before creating the BOW representation can help to minimize this problem.

There are many things that could be done or improved in order to continue this study. A true Random Search could be implemented in order to improve the features selection and parameters tuning. It could also be improved by adding features extracted with respect to the natural language (syntactic and semantic features, for example). Natural Language Processing[13] can be exploited to achieve that. But as we mentioned before it may not be possible to exploit it to the best extent as the nature of this environment is noisy.

The use of the *term frequency-inverse document frequency* (tf-idf) technique[10] and tuning the maximum size of the BOW can help too. In fact the *scikit-learn* provides the necessary functions to use tf-idf technique and it could be a good experiment to do as a future work.

Acknowledgement

We want to address our thanks to the Departamento de Informática da Escola de Ciências e Tecnologia da Universidade de Évora, for all the support to our work.

References

1. Bird, S.: Nltk: the natural language toolkit. In: Proceedings of the COLING/ACL on Interactive presentation sessions. pp. 69–72. Association for Computational Linguistics (2006)
2. Gollub, T., Stein, B., Burrows, S., Hoppe, D.: TIRA: Configuring, Executing, and Disseminating Information Retrieval Experiments. In: Tjoa, A., Liddle, S., Schewe, K.D., Zhou, X. (eds.) 9th International Workshop on Text-based Information Retrieval (TIR 12) at DEXA. pp. 151–155. IEEE, Los Alamitos, California (Sep 2012)
3. Hearst, M.A., Dumais, S.T., Osman, E., Platt, J., Scholkopf, B.: Support vector machines. Intelligent Systems and their Applications, IEEE 13(4), 18–28 (1998)

4. Hsu, C.W., Chang, C.C., Lin, C.J., et al.: A practical guide to support vector classification (2003)
5. Lewis, D.D.: Naive (bayes) at forty: The independence assumption in information retrieval. In: Machine learning: ECML-98, pp. 4–15. Springer (1998)
6. Lovins, J.B.: Development of a stemming algorithm. MIT Information Processing Group, Electronic Systems Laboratory Cambridge (1968)
7. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)
8. Porter, M., Boulton, R.: Snowball. On line http://snowball.tartarus.org/.[Visited 25/02/2016] (2001)
9. Potthast, M., Gollub, T., Rangel, F., Rosso, P., Stamatatos, E., Stein, B.: Improving the Reproducibility of PAN's Shared Tasks: Plagiarism Detection, Author Identification, and Author Profiling. In: Kanoulas, E., Lupu, M., Clough, P., Sanderson, M., Hall, M., Hanbury, A., Toms, E. (eds.) Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14). pp. 268–299. Springer, Berlin Heidelberg New York (Sep 2014)
10. Ramos, J.: Using tf-idf to determine word relevance in document queries. In: Proceedings of the first instructional conference on machine learning (2003)
11. Rangel, F., Rosso, P., Verhoeven, B., Daelemans, W., Potthast, M., Stein, B.: Overview of the 4th Author Profiling Task at PAN 2016: Cross-genre Evaluations. In: Working Notes Papers of the CLEF 2016 Evaluation Labs. CEUR Workshop Proceedings, CLEF and CEUR-WS.org (Sep 2016)
12. Sima, C., Dougherty, E.R.: The peaking phenomenon in the presence of feature-selection. Pattern Recognition Letters 29(11), 1667–1674 (2008)
13. Snow, R., O'Connor, B., Jurafsky, D., Ng, A.Y.: Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In: Proceedings of the conference on empirical methods in natural language processing. pp. 254–263. Association for Computational Linguistics (2008)
14. Zhang, Y., Jin, R., Zhou, Z.H.: Understanding bag-of-words model: a statistical framework. International Journal of Machine Learning and Cybernetics 1(1-4), 43–52 (2010)