A Straightforward Author Profiling Approach in MapReduce

Suraj Maharjan, Prasha Shrestha, Thamar Solorio, and Ragib Hasan

University of Alabama at Birmingham
Birmingham, Alabama
{suraj, prasha, solorio, ragib}@cis.uab.edu

Abstract. Most natural language processing tasks deal with large amounts of data, which takes a lot of time to process. For better results, a larger dataset and a good set of features are very helpful. But larger volumes of text and high dimensionality of features will mean slower performance. Thus, natural language processing and distributed computing are a good match. In the PAN 2013 competition, the test runtimes for author profiling range from several minutes to several days. Most author profiling systems available now are either inaccurate or slow or both. Our system, written entirely in MapReduce, employs nearly 3 million features and still manages to finish the task in a fraction of time than state-of-the-art systems and with better accuracy. Our system demonstrates that when we deal with a huge amount of data and/or a large number of features, using distributed systems makes perfect sense.

1 Introduction

In natural language processing (NLP) as with any task, producing good results takes higher precedence over getting the results faster. But most of the time the runtime performance is so overlooked that it is almost never mentioned in publications. A good approach is of course of utmost importance to get good results. But if testing out the approach takes a large amount of time, we will be spending our time waiting for results rather than improving our approach or trying out newer ones. Also, a good runtime performance is important to have practical solutions.

By nature, most NLP tasks such as parsing, POS tagging, and named entity recognition are computationally expensive. When we perform these tasks on large datasets, we have to wait hours, even days, to see preliminary results. However, since most of the work in NLP entails performing the same tasks over and over again on different texts, the tasks can be easily parallelized and distributed. Theoretically, each document or independent piece of text could be computed on a single machine, which would allow the whole work to be completed within the time taken for a single document.

The PAN workshop series is one of the few shared tasks where the test runtimes for the systems submitted to the competition are published. There are different tracks in the competition, namely, plagiarism detection, authorship attribution and author profiling. We chose the author profiling task because along with the runtimes being very high for some systems, the accuracy for this task was very low and we wanted to see if we could improve along these two fronts. Moreover, author profiling is in itself an important problem. Analyzing of an author's profile is useful in fields such as security, forensics, literary research, and marketing. In forensics, finding out the profile of the author of an email containing threats, spam or malware can help to narrow down the number of suspects. Another scenario where author profiling can be useful is when there are a lot of old texts for which the author is unknown. If we are to directly perform authorship attribution, the list of authors will be too long. If we profile the author of the text first, we can cut down the list of prospective authors on which authorship attribution can then be performed. In the field of marketing, companies can target their ad campaigns at certain groups of people who match the demographics and profile of current customers. Author profiling can also be applied to the feedback on online shopping websites and on reviews of their products [1].

Although author profiling has many practical applications, it is also time intensive. Having a large amount of training data helps in generating a good model for the profile of an author. In the PAN 2013 competition [1], the size of training data was 2.2GB. Other previous works do not generally mention their data sizes. However, they do mention the number of files in their dataset. Estival et al. [2] have used 9,836 email messages of at least 200 words each. Schwartz et al. [3] have performed what they claim to be the largest study on Facebook data. They have trained their system on 700 million words. These statistics show that the task of author profiling usually deals with a large quantity of data. For this reason, author profiling tasks are generally very slow.

Most authors do not mention how long their system takes to run, so we are unable to compare directly against them. In the PAN 2013 competition, however, even to get results from an already trained model, it took up to 17 days for some systems. The time required for training these models was not reported. There were 312,500 training documents, while there were only 33,600 test documents. The number of training documents is nearly 10 times more than the number of test documents. Also, all of the methods used in the competition are supervised learning methods; so all of the systems must have performed model training. Thus training is bound to have required more time by an order of magnitude. Here, we have taken on the same task as of the PAN 2013 competition i.e. given a document, finding out the age and gender of the author. Although in the competition, the focus was more on accuracy than on speed, we have given both equal emphasis. We have used nearly 3 million n-gram features, which produce a well-rounded representation of the text. Finding and using this large number of features for a large data set consumes a huge amount of time and memory. So, we have used Hadoop MapReduce so that even with large number of features, we can run our experiments with on all of the data in an acceptable amount of time. We also developed a web interface for users, so that they can submit a sample text to our system and get a response in real time.[1]

The contributions of this paper are threefold. First of all, we show that by using MapReduce we can dramatically increase the runtime performance of an author profiling system. Since most NLP systems deal with large amounts of data and most of these systems are parallelizable at least on the document level, we believe that this increase in performance will translate to any such system if parallelization is used. Secondly, we were also able to improve on the best accuracy of the competition, even though we used

[1] http://coral-projects.cis.uab.edu:8080/authorprofile/

simple features as compared to other systems. Finally, our system is useful for someone who wants to use distributed computing for NLP but does not want to get into MapReduce. The Naive Bayes implementation and other jobs that we wrote can be useful in that scenario. The tokenizer, filter, IDF counter, and Naive Bayes implementation can all be used fairly easily by someone who does not have any knowledge of MapReduce. Our code is freely available for use and extension.

2 Related Work

The PAN Overview paper for author profiling [1] outlines all of the methods used by the participants in the competition. All of the approaches use a combination of several linguistic features such as lexical, stylistic, syntactic and readability features. Since the dataset contains a lot of spam, some of the approaches also use a spam filter. Some have used URLs and emoticons as features as well, while others simply removed them. Although the training runtimes are not reported, the PAN competition organizers have listed the testing runtimes for them, which range from 10.26 mins to 11.78 days. Since all of the approaches use supervised learning, we believe it is safe to assume that the training runtimes are a lot higher than the test ones. The winning approach for the English language data was by Meina et al. [4] also used Naive Bayes for classification. Their testing runtime was 4.44 days. The winning approach in Spanish was by Santosh et al. [5] and they used decision trees for classification. Their tests took 4.86 hours. The overall winner was the system by Lopez-Monroy et al. [6] and they used second level meta-features. Their testing runtime was comparatively low at 38.31 mins.

Schwartz et al. [3] explored gender, age and five personality traits on data obtained from Facebook users. They used n-grams and LDA topics as features and obtained an accuracy of 91.9% for gender. They modeled age as a continuous variable and obtained linear regression coefficient *R* of 0.84. Estival et al. [2] collected English email texts and along with gender, age and personality traits, they also tried to predict the first language and the country of the authors. They tried experimenting with a variety of classifiers including SVMs using SMO, Random Forest and rule based learners among others. While no one classifier was best for all traits, SMO performed the best for both age and gender. They obtained accuracy of 56.46% and 69.26% for age and gender respectively.

The accuracy obtained by the above two systems is a lot higher than any of the systems in the PAN 2013 competition. This might be due to the difference in the dataset. The PAN data is mostly blogs, which while mostly informal, retains a little more formality than say, Facebook statuses. In blogs, people have to write on a certain topic and are restricted in the kind of language that they can use. In Facebook statuses and messages, people are completely free to express themselves and use any kind of language. So, they are more likely to use words that are predictive of their profiles. We have further explored this overall low accuracy in later sections.

We also wanted to compare the runtime performance of our system with other author profiling systems implemented in MapReduce. But we could not find any such research. However, there were papers that dealt with using MapReduce for natural language processing tasks. Eidelman et al. [7] have investigated the use of MapReduce for large-margin structured learning. They used MapReduce for statistical machine translation and showed that their approach scaled linearly with the size of the data.

3 Dataset

We have used the dataset provided in the PAN 2013 competition. The data has been collected from different blog posts where the authors also mention their age and gender [1]. There are a total of 236,600 files in the English dataset and 75,900 files in the Spanish dataset of the PAN 2013 training corpus. There are 3 classes across age: 10s, 20s and 30s and 2 classes for gender: male and female. We considered this as a 6 class problem rather than considering the 2 sets of classes independently. The distribution for these 6 classes for both English and Spanish is shown in Table 1. The dataset is balanced across gender and imbalanced across age. The size of training, early bird and test datasets are shown in Table 2. PAN has a provision in which during the course of the task, participants' software can be pre-evaluated against a separate test dataset, called the early bird dataset. The datasets are large and will take a large amount of time to process by using standard off-the shelf machine learning toolkits.

Table 1. Training, Early bird, Test documents distribution.

Age	**Gender**	**English**			**Spanish**		
		Training	**Early Bird**	**Test**	**Training**	**Early Bird**	**Test**
10s	male	8600	740	888	1250	120	144
10s	female	8600	740	888	1250	120	144
20s	male	42900	3840	4608	21300	1920	2304
20s	female	42900	3840	4608	21300	1920	2304
30s	male	66800	6020	7224	15400	1360	1632
30s	female	66800	6020	7224	15400	1360	1632
Total		236600	21200	25440	75900	6800	8160

Table 2. Sizes of the Dataset.

Language	**Training**	**Early Bird**	**Test**
English	1.8 GB	135 MB	168 MB
Spanish	384 MB	34 MB	40 MB
Total	2.2 GB	169 MB	209 MB

4 MapReduce for Author Profiling

We have built our system by using MapReduce on Hadoop. We chose MapReduce because it provides a high level abstraction for writing distributed applications. Programmers do not have to manage the communication as with MPI or p-threads. Hadoop handles race conditions, fault tolerance, deadlock, and failures automatically so that programmers can focus on the application rather than the on the cluster details. Similarly, Hadoop Distributed File System (HDFS) provides a distributed file system, which takes care of the data distribution and replication. This provides a benefit over some other cluster management software such as SGE, because for SGE, we have to manage the data ourselves. All parts of our system have been written as MapReduce jobs, from preprocessing to training and testing, which are described below.

4.1 Preprocessing

Since the PAN 2013 dataset is in XML format, we removed all the XML tags. Most of the training and test files were very small in size: only a few kilobytes. Hadoop is optimized to work with large files rather than with lots of small files. So, we merged them into larger sequence files with the filename as key and the file content as value. The tokenizer job (Algorithm 1) takes this sequence file, removes all HTML tags from it, changes all the tokens to lowercase and finally generates unigrams, bigrams and trigrams. This is a map only process and not having a reducer eliminates shuffle and sort that could have hindered the speed.

4.2 IDF Counts and Filtering

Algorithm 2 works on the n-gram tokens generated in the tokenization phase. IdfMapper emits the partial count of each unique token in the document and IdfReducer sums these counts grouped by keys. We also used combiner in between mapper and reducer which performs local aggregation and hence reduces data transfer across the network, thus making the process faster. The output of this phase are the tokens with their IDF scores, which are passed on to the filtering job (algorithm 3). The filtering job first creates a dictionary file as a mapping between tokens and their unique integer id. It also has an entry for *UNSEEN_TOKEN*, which accounts for any unseen tokens that we may encounter in the test documents. It then reads the idf files and based on the threshold supplied, filters out infrequent tokens. After obtaining the final tokens, TFIDFVectorizeMapper (algorithm 4) creates term frequency-inverse document frequency (TF-IDF) vectors for each document by making use of the dictionary file and idf scores.

4.3 Training

For training, we used Naive Bayes as our classifier. We explored Mahout's Naive Bayes implementation, but it did not meet our requirements. First of all, to use it we have to have the training and test data together so that it can create the dictionary file and vectors. So, the vector creation process needs to be done by combining the training and test sets. The resulting vector is subdivided into training and test sets by Mahout and the ratio of training and test can be given to it as parameters. But in a lot of cases, we have separate training and test data like in the PAN competition. Also, as of Mahout version 0.7, the Naive Bayes implementation does not consider prior probability and unseen

Algorithm 1 Document Tokenizer

```
1: class TokenizerMapper
2:   method Map(docname $a$, doc $d$)
3:     tokens $T \leftarrow$ new Tuple
4:     for all term $t \in$ Tokenize$(d)$ do
5:       $T$.Add$(t)$
6:     end for
7:     Emit(docname $a$, tokens $T$)
8:   end method
9: end class
```

Algorithm 2 IDF Count

```
1: class IdfMapper
2:   method Map(docname $a$, tokens $T$)
3:     uniqueWords $U \leftarrow$ new List
4:     for all term $t \in$ tokens $T$ do
5:       if term $t \notin U$ then
6:         Emit(term $t$, count 1)
7:         $U$.Add$(t)$
8:       end if
9:     end for
10:  end method
11: end class
1: class IdfReducer
2:   method Reduce(term $t$, counts$[c_1, c_2 ...]$)
3:     idf $i \leftarrow 0$
4:     for all count $c \in$ counts $[c_1, c_2 ...]$ do
5:       $i \leftarrow i + c$
6:     end for
7:     Emit(term $t$, idf $i$)
8:   end method
9: end class
```

words. Although Mahout can be frustrating due to lack of proper documentation, it has implementations for a lot of machine learning algorithms and is worth exploring.

We wrote our own Naive Bayes classifier in MapReduce. We did use Mahout's *Vector* class in order to take advantage of *mahout-math*, Mahout's well-designed math library. Also, Mahout's Naive Bayes is well-structured, so we based our class structure upon it. The TrainNBMapper reads in the vectors per document created by TFIDFVectorizeMapper, groups the vectors by their class labels, and then computes frequencies of tokens per class label, total tokens in a class and vocabulary size. During the mapping phase, it extracts class labels from document names and maps them to unique integer values. It then emits the class label id as key and the TF-IDF vector as value.

At the clean up stage, we emit a special key -1 to indicate that this key holds the partial counts for documents with the same class label. Finally the reducer needs to sum up the vectors grouped by the label id for which we used Mahout's VectorSumReducer. These final vectors hold all the information we need for Naive Bayes. After completion of this phase, we read in all of the final vectors and create a Naive Bayes Model and then save it to a model file. This model file also contains other details like vocabulary count, total number of classes and Laplace smoothing coefficient.

4.4 Testing

We tested our system on both the early bird dataset and the test dataset released by PAN 2013. Similar to what we did with the training files, we first removed the XML tags

Algorithm 3 Filter out Least Frequent Tokens

```
1: class FilterLeastFrequenMapper
2:   method Setup
3:     $H \leftarrow$ new HashSet
4:     Read dictionary file
5:     $H.\text{Add}(tokens \in \text{dictionary file})$
6:   end method
7:   method Map(docname $a$, tokens $T$)
8:     finalTokens $F \leftarrow$ new Tuple
9:     for all term $t \in$ tokens $T$ do
10:      if term $t \in H$ then
11:        $F.\text{Add}(t)$
12:      end if
13:    end for
14:    Emit(docname $a$, finalTokens $F$)
15:  end method
16: end class
```

Algorithm 4 TF-IDF Vectorizer

```
1: class TFIDFVectorizeMapper
2:   method Setup
3:     $H \leftarrow$ new HashMap
4:     $IDF \leftarrow$ new HashMap
5:     Read dictionary file
6:     $H.\text{Add}(tokens, id \in \text{dictionary file})$
7:     Read idf file
8:     $IDF.\text{Add}(tokens, idf \in \text{idf file})$
9:   end method
10:  method Map(docname $a$, tokens $T$)
11:    vector $V \leftarrow$ new RandomSparseVector
12:    Compute token count
13:    for all unique term $t \in$ tokens $T$ do
14:      $V.\text{Set}(H.get(t), tfIdfScore(t))$
15:    end for
16:    Emit(docname $a$, vector $V$)
17:  end method
18: end class
```

from the test files and then converted them to sequence files. Algorithm 6 takes the sequence file as input. The TestNBMapper loads the model created during the training phase, the dictionary file and the idf scores files. Then TestNBMapper tokenizes the test document and creates TF-IDF vectors using dictionary and idf files. It then computes the scores for each class label using the model and emits the actual label and the score vector. After the completion of this process, we read in all the score vectors and used Mahout's ResultAnalyzer to compute accuracy and create the confusion matrix.

5 Experimental Settings

After we had our MapReduce jobs in place, we performed our experiments on our local cluster. It consists of a master node and 7 slave nodes, each node having 16 cores and 12GB memory. We are running Hadoop version 1.0.4 and Mahout version 0.7.

5.1 Features

Our method is based upon an n-gram model and we worked with unigrams, bigrams and trigrams. We filtered out the least frequent n-grams and used the rest as features. In the English training dataset, there were 71,710,553 unique n-gram tokens for 236,600 authors. In the Spanish dataset, there were 11,920,013 unique n-gram tokens for 75,900 authors. We removed all of the tokens that were not used by at least 10 of the authors for English and 2 of the authors for Spanish. We then created bigrams and trigrams

Algorithm 5 Naive Bayes Training

```
1: class TrainNBMapper
2:   method Setup
3:     priorPbVector $P \leftarrow$ new Vector
4:   end method
5:   method Map(docname $a$, vector $v$)
6:     $l \leftarrow$ ExtractLabel($a$)
7:     $P\{l\} \leftarrow P\{l\} + 1$
8:     Emit(label index $l$, vector $v$)
9:   end method
10:  method Cleanup
11:    Emit($-1$, priorpb vector $P$)
12:  end method
13: end class
```

Algorithm 6 Naive Bayes Testing

```
1: class TestNBMapper
2:   method Setup
3:     dictionary $D \leftarrow$ new HashMap
4:     $IDF \leftarrow$ new HashMap
5:     Read dictionary file
6:     $D$.Add($tokens, id \in$ dictionary file)
7:     Read idf file
8:     $IDF$.Add($tokens, idf \in$ idf file)
9:     model $M \leftarrow$ load Model
10:    vectorClassifier $C$ $\leftarrow$
       new NBClassifier(M)
11:  end method
12:  method Map(docname $a$, doc $d$)
13:    Compute token count
14:    Create Test Vectors $T$
15:    $l \leftarrow$ ExtractLabel($a$)
16:    vector $V \leftarrow C$.Classify($T$)
17:    Emit(actual label $l$, vector $V$)
18:  end method
19: end class
```

from those tokens. After this, we were left with 2,908,894 n-gram tokens for English and 2,806,922 n-gram tokens for Spanish, each of which acts as a feature. This is a very large number of features and it can be computationally intensive to calculate values for all of them. It is almost impossible to imagine using this many features with a single machine and to expect results in a reasonable amount of time.

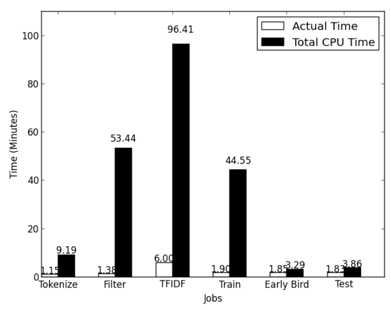
5.2 Classification Algorithm

To choose our classification method, we first divided the training dataset from the PAN 2013 Competition in 60:40 ratio. We created a unigram model of the text and trained a Logistic Regression classifier and a Naive Bayes classifier provided by Mahout [8]. We obtained 29.01% and 38.61% accuracy by using Logistic Regression and Naive Bayes respectively. From these preliminary results, since the difference of accuracy between these methods was so high, we chose Naive Bayes as our classification method.

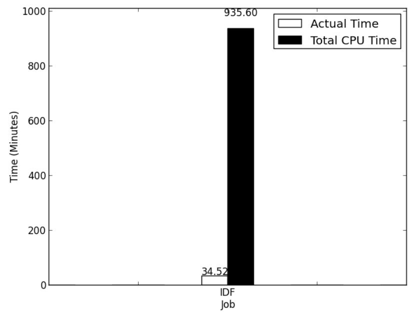
6 Results

Figures 1(a) to 1(d) show the runtimes for the whole process, including testing and training. The CPU time denotes the total time taken by the process to run on all the CPUs available in the cluster. This value should be comparable to the time taken to run the MapReduce tasks on a single CPU. The task of creating IDF was the most time consuming and as expected it was also the task in which the speedup is truly remarkable.

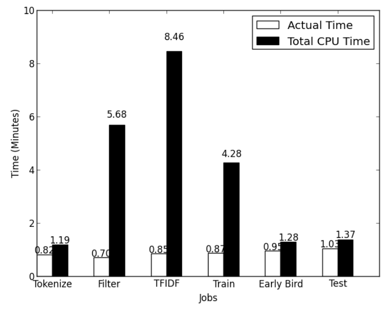
For a task that would have taken at least 1085 minutes as evidenced by CPU time, our system finished it within 55.79 minutes. The total amount of time taken for the whole process was just 72.12 minutes, which is a lot less than the time most of the participant's approach took just for testing. Although both our system and the system used by Meina et al. [4] use Naive Bayes, their testing required 4.44 days, while ours required just 2.86 minutes. However, even our individual machines had high computational power and all the participants in the contest might not have access to such powerful machines.



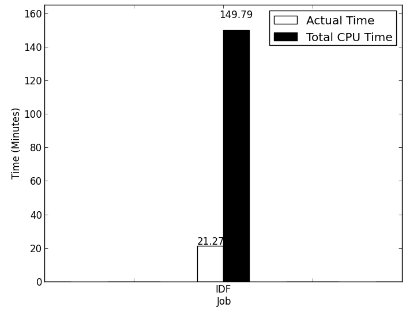
(a) Runtime: English.



(b) IDF Runtime: English.



(c) Runtime: Spanish.



(d) IDF Runtime: Spanish.

Fig. 1. Runtime.

The accuracy of our system, the majority class baseline and the accuracy of best systems for English [4], Spanish [5] and overall [4] in the PAN 2013 Competition are shown in Table 3. In both the early bird and actual test datasets, our accuracy is a lot higher than the baseline for both languages. For age and gender individually, we have the highest accuracy for English in the age category only. But, our overall accuracy and the accuracy for English is the highest, while it is in the top 3 for Spanish. However, the participants in the PAN'13 competition did not have the early bird or test datasets available to them and could not have made any changes based upon them.

Table 3. Accuracy on Test data.

Language	System	Total (%)	Age (%)	Gender (%)
English	Baseline	28.40	56.80	50.00
	PAN 2013 English Best [4]	38.94	59.21	64.91
	PAN 2013 Overall Best [6]	38.13	56.90	**65.72**
	Ours (Test)	**42.57**	**65.62**	61.66
	Ours (Early Bird)	43.88	65.80	62.57
Spanish	Baseline	28.23	56.47	50.00
	PAN 2013 Spanish Best [5]	**42.08**	**64.73**	64.30
	PAN 2013 Overall Best [6]	41.58	62.99	**65.58**
	Ours (Test)	40.32	61.73	64.63
	Ours (Early Bird)	40.62	62.10	64.79

7 Analysis

Although our system is a very simple approach, we were able to obtain good results. We believe this is mostly because we used a high number of features. To analyze this, we performed experiments by varying the minimum IDF count that a token must have for it to become a feature. When we decreased the number of features, the accuracy also declined as shown in Table 4. Here, we can see that having a large number of features is surely helpful, although the increase in accuracy is not necessarily linear.

Table 4. Accuracy on test dataset compared with # of features for English

Filter at IDF	# of features	Accuracy (%)
10	2,908,894	42.57
15	1,885,410	42.37
20	1,403,139	42.26
100	276,902	41.93
200	136,999	37.34
300	90,836	17.23
400	67,938	7.80
500	54,330	5.93
1000	27,151	4.66

We also performed experiments with different types of features and weighing schemes. Our original system runs with the following parameters and settings: n-grams of words with n from 1 to 3, tf-idf weighing scheme, filter at 10 and no preprocessing applied. As shown in Table 3, the accuracy with this setting is 42.57%. We performed different experiments with one of these parameters changed. The changed parameter, its setting and the accuracy obtained are shown in Table 5. When we used character n-grams instead, the accuracy dropped by more than 11%. Although character n-grams represent the writing style of an author and are helpful, people of certain age or gender group are more likely to use certain words than to collectively use a certain style of writing. When unigrams, bigrams and trigrams were used separately, the accuracy was again lower than when they were used together. We obtained the highest accuracy when using bigrams, although we had similar accuracy with all three. When we stemmed the words and also filtered out stopwords, the accuracy again decreased. Stemming and stopword filtering causes a loss of information about the syntactic structure of a text, which is important. We also experimented by using only stopwords as features and again obtained very low accuracy. Although just the syntactic information is not powerful enough in determining the age and gender of an author, it is certainly helpful. When we used term frequency as a weighing scheme rather than using TF-IDF, the accuracy was much lower than that of our original system. Using TF-IDF assigns weight based not only upon how frequently a word is used by an author but also upon how common the word is among other authors. So, it makes sense for TF-IDF to be the better weighing scheme.

Table 5. Accuracy comparison for different parameter settings for English dataset

Parameter	**Setting**	**Accuracy (%)**
N-gram	Character Bigrams and Trigrams	31.20
N-gram	Word Unigrams	39.99
N-gram	Word Bigrams	41.17
N-gram	Word Trigrams	39.70
N-gram	Stopword Unigrams	29.14
Weighing Scheme	TF	36.12
Preprocessing	Stopwords filtered and Porter Stemming	35.82

Although we performed well and obtained overall best accuracy in the PAN 2013 competition, the accuracy on this data is generally very low when compared to author profiling done on Facebook [3] and email data [2]. When we analysed the PAN data for possible reasons, we found out that the data contains a lot of spam and some mixed language documents as well, containing both English, Spanish and even some other languages. Along with this, some of the files are very short and do not even contain 5 words. Also, since the data has been collected from blogs, there is no guarantee that authors of the text will provide their correct age and gender. We even found cases where an author mentions his age in the text but it is different from what the file has been labeled as. All these shortcomings of the dataset can introduce errors in the model.

8 Discussion and Future Work

We were able to run everything, which would have taken at least a day at best, within 1.5 hours for the whole process from preprocessing to testing. We were able to outperform all of the systems in the PAN 2013 competition in terms of both speed and combined overall accuracy in the whole corpus. Along with that, even by using a simple approach, we were able to obtain good results. We believe this is in part, because we used a high number of features. Using distributed computing allowed us to use lots of features, which we could not have done with single machine computation.

One aspect we can improve on is cleaning the training data. Although we filtered out a lot of words, we need to filter out some of the documents as well. Removing spam-like documents from our dataset is likely to improve our results. Another aspect is that although we have used a large number of features, they are simple features. In the future, we will look at adding more sophisticated features and making them work in MapReduce. That way, we will be able to test to see if our system obtains even higher accuracy with sophisticated features, while retaining the speed.

As with this task, most NLP tasks deal with a lot of data. People resort to random sampling in order to decrease the amount of data, as done by participants of the PAN 2013 competition. Although filtering the dataset might be useful, random sampling is rarely so. Generally, having a good amount of properly labeled data is helpful in creating good models. So, having to resort to using only a small sample of the data due to computational constraints will keep a system from performing at its best. Our results are a good indication that for most NLP tasks, distributed computing is the way to go.

Acknowledgments

This research was partially funded by The Office of Naval Research under grant N00014-12-1-0217 and by The National Science Foundation under grant 1350360.

References

1. Rangel, F., Rosso, P., Koppel, M., Stamatatos, E., Inches, G.: Overview of the author profiling task at PAN 2013. In: Notebook Papers of CLEF 2013 LABs and Workshops, CLEF-2013, Valencia, Spain, September. (2013) 23–26
2. Estival, D., Gaustad, T., Pham, S.B., Radford, W., Hutchinson, B.: Author profiling for english emails. In: Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics. (2007) 263–272
3. Schwartz, H.A., Eichstaedt, J.C., Kern, M.L., Dziurzynski, L., Ramones, S.M., Agrawal, M., Shah, A., Kosinski, M., Stillwell, D., Seligman, M.E.P., Ungar, L.H.: Personality, gender, and age in the language of social media: The open-vocabulary approach. PLoS ONE **8** (2013) e73791
4. Meina, M., Brodzinska, K., Celmer, B., Czoków, M., Patera, M., Pezacki, J., Wilk, M.: Ensemble-based classification for author profiling using various features. In: Notebook Papers of CLEF 2013 LABs and Workshops, CLEF-2013, Valencia, Spain, September. (2013)
5. Santosh, K., Bansal, R., Shekhar, M., Varma, V.: Author profiling: Predicting age and gender from blogs. In: Notebook Papers of CLEF 2013 LABs and Workshops, CLEF-2013, Valencia, Spain, September. (2013)
6. López-Monroy, A.P., Montes-y Gómez, M., Escalante, H.J., Villaseñor-Pineda, L., Villatoro-Tello, E.: INAOE's participation at PAN'13 : Author profiling task. In: Notebook Papers of CLEF 2013 LABs and Workshops, CLEF-2013, Valencia, Spain, September. (2013)
7. Eidelman, V., Wu, K., Ture, F., Resnik, P., Lin, J.: Mr. MIRA: Open-source large-margin structured learning on MapReduce. ACL System Demonstrations (2013)
8. Owen, S., Anil, R., Dunning, T., Friedman, E.: Mahout in action. Manning (2011)