A Simple Approach to Author Profiling in MapReduce
Notebook for PAN at CLEF 2014

Suraj Maharjan, Prasha Shrestha, and Thamar Solorio

University of Alabama at Birmingham
Department of Computer and Information Sciences,
Campbell Hall, 1300 University Boulevard,
Birmingham, Alabama 35294-1170
{suraj, prasha, solorio} @cis.uab.edu

Abstract Author profiling, being an important problem in forensics, security, marketing, and literary research, needs to be accurate. With massive amounts of online text readily available on which we might need to perform author profiling, building a fast system is as important as building an accurate system, but this can be challenging. However, the use of distributive computing techniques like MapReduce can significantly lower processing time by distributing tasks across multiple machines. Our system uses MapReduce programming paradigm for most parts of the training process, which makes our system fast. Our system uses word n-grams including stopwords, punctuations and emoticons as features and TF-IDF (term frequency inverse document frequency) as the weighing scheme. These are fed to the logistic regression classifier that predicts the age and gender of the authors. We were able to obtain a competitive accuracy in most categories and even obtained winning accuracies for two of the categories each in both test corpus 1 and test corpus 2.

1 Introduction

The process of identifying age-group, gender, native language, personality and other aspects that constitute the profile of an author, by analyzing his/her writings is called author profiling. Since most of the text is now online and written behind a curtain of anonymity, author profiling has become a very important problem. In fields like forensic, security, literary research and marketing, finding out an author's demographic and personality information has proven to be very helpful. In literary research, author profiling helps to resolve disputed authorship for unknown documents. In forensics, author profiling can be used to shortlist potential suspects, given a piece of writing from them. In marketing, ad campaigns can be directed towards the demographic of users that review certain products the most.

The PAN'14 author profiling task requires us to predict the author's age-group and gender. The PAN'14 corpus contains data collected from authors' writings in English or Spanish. The data has been divided into different categories according to the source of the data. For Spanish there are three categories: blogs, social media and twitter. Whereas for English, there is one more category along with those three, namely reviews. The task is to create a system that can predict an author's age and gender for all these categories when we are given their writings.

Prior work has tackled the task of author profiling by employing a range of lexical, stylistic, syntactic and readability measures. Schwartz et al. [5] used n-grams and LDA topic features to profile gender and five personality traits on Facebook user's data and obtained 91.9% accuracy. Burger et al. [1] used word and character n-grams along with Twitter user profile (full name, screen name and description) as features to determine the gender of the users. They experimented with Naive Bayes, LIBSVM and balanced Winnow2 classification algorithms and found that Winnow was better in both accuracy and speed. Likewise, Estival et al. [2], in addition to age and gender prediction, tried to predict the first language and country of an author. They experimented with different classification algorithms like SVM using SMO, Random forest and rule based learners and concluded that SMO performed the best for both age and gender.

In this paper, we have experimented with word and character n-grams as features. We also analysed the use of five different classification algorithms viz Naive Bayes, Cosine Similarity, Weighted Cosine Similarity, LIBLINEAR [3] Logistic Regression with L2 regularization, and LIBLINEAR SVM with linear kernel.Since feature computation as well as classification can be very slow, we have implemented most our system in MapReduce. All of the feature computation has been implemented in MapReduce, which make our system very fast. We also implemented Naive Bayes, and weighted as well as non-weighted Cosine Similarity in MapReduce. Our code is available for use and extension for people who want to make use of the computing prowess of a distributed system without having to go into much detail about MapReduce.

2 Methodology

Most of our training takes place in Hadoop and uses MapReduce. For our training process, we started by randomly dividing the available data into training and cross validation dataset in a ratio of 70:30. The training data was then preprocessed to filter out all the HTML and XML tags. The plain text files thus obtained were then combined into sequence files because MapReduce jobs run faster when we use a small number of large files rather than when using a large number of small files. We then used MapReduce jobs to compute feature vectors.

2.1 Features Vector Creation

Since most of the training data was from some sort of social media, we used Ark-Tweet-NLP tokenizer [4] to tokenize the document because it is well adapted for online text. We retained all the stopwords, punctuations and emoticons as these are important features for the author profiling task. Since any MapReduce job requires key and value, the tokenizer job uses the filename with all the class information as the key and the file content as its value. After tokenizing with Ark-Tweet-NLP, this job generates the necessary n-grams.

After obtaining the n-grams, we compute the inverse document frequency (IDF) count for each token. Based on these counts, we filter out the tokens that have not even been used by at least two authors. We have an idf MapReduce job to compute the idf. The mapper computes the partial count of each individual ngram and passes them to the reducer, which sums these partial counts to produce the final idf counts. The filter job takes the idf counts and a threshold and filters out all the tokens whose idf score is less than the threshold. Also, this job creates a dictionary file that contains all the unique tokens mapped to integer ids. After the tokens are filtered out, our TF-IDF vector creation job takes in the idf counts and dictionary file to compute the tf-idf scores for the remaining tokens. This map only job outputs a tf-idf vector for each document.

2.2 Training

For training, we tried five different classification algorithms. For Naive Bayes, cosine similarity, and weighted cosine similarity, we created another MapReduce job that computes all the statistics needed for classification. The weighted cosine similarity algorithm multiplies the cosine similarity score by prior probability score for each class. The mapper emits class label as key and tf-idf vector as value. Instead of building separate models for age groups and gender, we considered the problem as a 10-class problem by combining the age and gender classes. The class labels were extracted from the document names and were mapped to unique integer ids. The mapper also emits a special key -1 and a vector that contains partial counts of number of documents with that class label. We used VectorSumReducer provided by Mahout as reducer, which groups vectors by their class id and sums them. For both logistic regression and SVM, we first transformed the tf-idf score vectors into a format as expected by LIBLINEAR. Then we trained on these feature vectors by using the LIBLINEAR command utility.

2.3 Testing

The PAN'14 shared task provided us with a Virtual Machine(VM) with 4GB of memory. So, for testing, we created a normal java application (not MapReduce), that would read the trained models and predict class labels for the test documents. For testing, we need to create test vectors similar to trained vectors. Hence, we applied the same steps as we did for the training data. After the test document was preprocessed to removed HTML tags and tokenized using Ark-Tweet-NLP tokenizer, we generated word or char n-grams by using Lucene [1].

3 Experiments and Results

We setup a local Hadoop [2] cluster with 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. Since the data is large, mapreduce is ideal for feature extraction from this data. We were able to finish training in a short amount of time even though the data is large. We were also able to train five different models because we did not need to spend

[1] http://lucene.apache.org/
[2] http://hadoop.apache.org/

a lot of time for feature extraction. In order to find the best model, we tried different classification algorithms and also compared the use of word vs character n-grams. We also performed experiments to find out if building separate models for different categories: blogs, social media, twitter and reviews produce better results than building a single, combined model for all categories. The test was performed in cross validation dataset, which was obtained by randomly separating 30% of the training data.

3.1 Separate Word N-gram Models for Each Category

Table 1 shows the result for word ngram experiments (unigrams, bigrams and trigrams combined) for cross validation set. Here, we built separate models for the different categories and different languages. So, we ended up with 14 models total. The results show that different classifiers perform better for different categories. Also, if we are to choose a classifier, either Naive Bayes or logistic regression seems to be more promising across different categories.

Table 1. Accuracy for word n-grams for cross validation dataset.

	English (%)				**Spanish (%)**		
Classification Algorithm	**Blog**	**Reviews**	**Social Media**	**Twitter**	**Blog**	**Social Media**	**Twitter**
Naive Bayes	27.50	21.55	20.62	28.89	55.00	20.48	34.78
Cosine Similarity	20.00	23.64	19.72	27.78	35.00	26.33	36.96
Weighted Cosine Similarity	30.00	23.16	19.97	26.67	40.00	22.07	32.61
Logistic Regression	27.50	23.08	20.62	33.33	35.00	25.80	32.61
SVM	25.00	22.28	19.80	32.22	30.00	26.33	34.78

3.2 Separate Character N-gram Models for each Category

Table 2 shows the results for character ngram experiments (bigrams and trigrams combined) for the cross validation set. We again built separate models for different categories and different languages. The results show that word ngrams are better features than character ngrams. In all experiments, we found that the word ngram method beats the character ngram method. One possible reason for this might be that we just considered character bigrams and trigrams. If we had considered higher character n-grams, we might have achieved similar results for both experiments. In addition, different people may have used different spelling for same words when using social media. This might have caused character n-gram not be able to capture the style of authors.

3.3 Single Word N-gram Model for all Categories

Since we already figured out that word n-grams are better at predicting the profile of an author, we decided to use them as features rather than character n-grams. The next

Table 2. Accuracy for character n-grams for cross validation dataset.

	English (%)				Spanish (%)		
Classification Algorithm	**Blog**	**Reviews**	**Social Media**	**Twitter**	**Blog**	**Social Media**	**Twitter**
Naive Bayes	25.00	18.99	18.33	24.44	40.00	19.68	23.91
Cosine Similarity	20.00	21.63	17.90	30.00	50.00	21.81	26.09
Weighted Cosine Similarity	20.00	21.15	16.78	23.33	40.00	19.68	28.26
Logistic Regression	22.50	21.71	16.78	25.56	35.00	23.67	17.39
SVM	20.00	20.83	15.92	24.44	35.00	23.14	17.39

decision we needed to make was either to build a separate model for each category or to build a single, combined model for all the categories. Since we had already run experiments for separate models, we built a single combined model next. But we still built two separate models for English and Spanish. Table 3 tabulates the accuracy obtained by running combined model and separate model for the cross validation set. It is clear from the table that having a single model is better than using separate ones. This might be because since all the data has been obtained from some sort of social media, authors might use similar writing styles across all of them. So, having a model trained on all categories or genres performs better than when separate models are trained on each category. We also found logistic regression to be the best classification algorithms for this task. So, our final model uses word n-grams as features and logistic regression as the classification algorithm.

Table 3. Accuracy for separate and combined models.

	English (%)		Spanish (%)	
Classification Algorithm	**Separate**	**Combined**	**Separate**	**Combined**
Naive Bayes	21.21	20.13	23.53	21.04
Cosine Similarity	19.89	17.34	27.83	27.6
Weighted Cosine Similarity	21.32	18.18	23.98	24.89
Logistic Regression	21.83	21.92	26.92	28.96
SVM	20.99	20.48	27.37	28.05

3.4 Test Results

Table 4 shows accuracy along with runtimes when our final system was run on test corpus 1 and 2 for both languages. Despite using simple word n-gram features, our system achieved competitive results both in terms of accuracy and speed. We ranked first in spanish-socialmedia and spanish-twitter for test corpus 1 and in english-socialmedia and spanish-twitter for test corpus 2. We were ranked in the top three for most of the other categories as well. This might even indicate that some problems are better suited by simple solutions. For english-socialmedia, which has largest number of test documents, we ranked second in runtime for both test corpora. Our system took 26 minutes to complete the classification. Whereas, some of the other participants took from around 30 to 69 hours to complete the testing. Although even for other test corpora, our runtime performances are quite fast, we could not obtain equally high ranking because we have a huge model that takes a lot of time to load.

Table 4. Accuracy on test corpus.

		Test 1				**Test 2**			
Language	**Category**	**Both**	**Age**	**Gender**	**Runtime**	**Both**	**Age**	**Gender**	**Runtime**
English	Blog	0.1667	0.2500	0.5417	00:01:50	0.2308	0.3846	0.5769	00:01:56
	Reviews	0.2012	0.2805	0.6280	00:01:46	0.2223	0.3331	0.6687	00:02:13
	Social Media	0.2009	0.3627	0.5332	00:07:18	0.2062	0.3652	0.5382	00:26:31
	Twitter	0.4000	0.4333	0.7333	00:02:01	0.3052	0.4416	0.6688	00:02:31
Spanish	Blog	0.2857	0.4286	0.5714	00:00:35	0.2500	0.4643	0.4286	00:00:39
	Social Media	0.3033	0.4016	0.6803	00:01:13	0.2845	0.4276	0.6449	00:03:26
	Twitter	0.6154	0.6923	0.8846	00:00:43	0.4333	0.6111	0.6556	00:01:10

4 Discussion and Conclusion

In the end, we were able to produce a system that performs the author profiling task with good accuracy. We also observed that analyzing word usage seems to be promising for this task. But character n-grams were not as good of features. This might be because people tend to use their own version of spelling for words especially when the writing is informal as in this dataset. Also, stopwords, punctuation and emoticons proved to be predictive features as well. The fact that we produced good results even with such simple features might indicate that some problems are better suited for simple solutions. Due to our model's load time, we were not able to obtain very high rankings for runtime. But this will not be of concern in a practical system because we need to load the model only once. Also, all of the systems in the competition are likely to be supervised classification systems and all of them must have trained a model, which takes a lot more time than testing. But, there is no mention of the training runtimes so we cannot make a comparison. But, since we used MapReduce for feature extraction, our training time was significantly shortened. So, our system is very likely to have less training runtime. Because of MapReduce, we were able to play with different threshold parameter settings in a reasonable amount of time and thus we can say that MapReduce is ideal for the task of feature extraction.

5 Acknowledgments

We want to thank PAN'14 organizers and committee members for organizing the author profiling task. This research was partially funded by The Office of Naval

Research under grant N00014-12-1-0217 and National Science Foundation under grant 1350360.

References

1. Burger, J.D., Henderson, J., Kim, G., Zarrella, G.: Discriminating gender on twitter. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. pp. 1301–1309. EMNLP ’11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011), http://dl.acm.org/citation.cfm?id=2145432.2145568
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. pp. 263–272 (2007)
3. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
4. Gimpel, K., Schneider, N., O’Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., Smith, N.A.: Part-of-speech tagging for twitter: Annotation, features, and experiments. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2. pp. 42–47. HLT ’11, Association for Computational Linguistics, Stroudsburg, PA, USA (2011), http://dl.acm.org/citation.cfm?id=2002736.2002747
5. 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(9), e73791 (09 2013),