Author Verification: Basic Stacked Generalization Applied To Predictions from a Set of Heterogeneous Learners

Notebook for PAN at CLEF 2015

Erwan Moreau[1], Arun Jayapal[1], Gerard Lynch[2], and Carl Vogel[3]

[1] CNGL and Computational Linguistics Group, Centre for Computing and Language Studies
School of Computer Science and Statistics
Trinity College Dublin, Ireland
moreaue@cs.tcd.ie, jayapala@cs.tcd.ie
[2] Centre for Applied Data Analytics Research
University College Dublin, Ireland
gerard.lynch@ucd.ie
[3] Computational Linguistics Group, Centre for Computing and Language Studies
School of Computer Science and Statistics
Trinity College Dublin, Ireland
vogel@cs.tcd.ie

Abstract In this paper we present the system we submitted to the PAN 2015 competition for the author verification task. We consider the task as a supervised classification problem, where each case in a dataset is an instance. Our approach combines the output from multiple learners using basic stacked generalization. The individual learners are obtained using five distinct approaches, each trained using a generic genetic algorithm. Our system performed well on the test set: the macro-average score was 0.61 (2nd best).

1 Introduction

In the PAN 2015 author verification task [11], a training set is provided for each of the 4 datasets: Dutch, English, Greek and Spanish. Each training dataset consists of a set of 100 problems, and each problem consists of a small set of "known" documents written by a single person and a "questioned" document: the task is to determine whether the questioned document was written by the same person. More precisely, the system must provide its prediction as a value in the interval $[0, 1]$, which represents the probability that the answer is positive (same author). The intended interpretation is for 0 to mean "different author" with maximum certainty, and for 1 to mean "same author" with maximum certainty, and any intermediate value describes the likeliness of a positive answer, with 0.5 equivalent to the system saying "I don't know". The predictions are evaluated using the product of the area under the ROC curve (AUC) and the modified accuracy measure c@1 [8], which treats 0.5 answers as a special case, in order to penalize such cases less than a wrong answer.

We consider the task as a supervised classification/regression problem[4], at the dataset level; in other words, each problem is an instance and we expect to find some regularities among the instances which belong to the same dataset. It is worth noticing that this view is compatible with, for instance, unsupervised learning at the problem level: the latter can be seen as an initial answer, which is then used as input in the dataset-level supervised system.

We propose a relatively complex solution[5], which consists in training a set of multiple (mostly) heterogeneous learners, later combined using a simple stacked generalization method. A genetic algorithm is used in two stages:

1. to learn an optimal combination of parameters for each of the five approaches from which the individual learners are obtained;
2. to learn the optimal way to combine the learners together.

The main difficulty in this approach is to avoid overfitting, especially with such a small number of instances. The genetic algorithm always tries to maximize the performance in any available way, because this is the criterion used to select the breeders for the next generation. But by doing so it can easily end selecting parameters which are optimal only for the set of instances seen by the algorithm, even if the performance of every individual is evaluated using cross-validation. This is why we try to minimize the risk of overfitting by using various additional techniques (see §3.2).

This paper is organized as follows: in §2 we detail the reasons why we chose this approach. In §3 we describe the architecture of our system, then in §4 we give a brief overview of each of the five approaches we take as individual learners. In §5 we present some of the techincal issues we faced with this approach. Finally in §6 we analyze the main observations that we made based on the training process.

2 Motivations

We had participated in the previous edition of the author identification task of the PAN competition [7]. The system that we implemented then was using a genetic algorithm in order to find an optimal configuration of parameters for each dataset. The genetic algorithm gave good results, but was naturally prone to overfitting.

This is why we had also prepared two distinct approaches: the *fined-grained* one, which contains a lot of parameters in order to maximize the performance, and the *robust* one, which was on the contrary a very simple method, intended to minimize the risk of overfitting. For each dataset, the final choice between the two approaches was made manually, based on the observations and statistics available to us.

Our system was ranked 4th overall, with 2nd or 3rd rank in 4 out of 6 datasets [12]. Given these promising results, we tried to improve the approach in three ways:

[4] The general task definition is closer to a classification problem, the output being either "yes" or "no". However, since the system has to provide a score in the range $[0, 1]$ instead, it is treated as a regression task in practice.

[5] It is important to notice that this approach is computationally expensive. We address the task in the perspective of accuracy only, mostly ignoring the efficiency criterion.

– Since the robust and fine-grained approaches in our previous system are naturally complementary, it seems relevant to combine their answers (instead of only choosing one or the other).
– Furthermore, as part of their analysis of the participants systems, the organizers of PAN 2014 evaluated a meta-model based on all the individual systems answers. They found that *" this meta-classifier was better than each individual submitted method while its ROC curve clearly outperformed the convex hull of all submitted approaches. This demonstrates the great potential of heterogeneous models in author verification, a practically unexplored area."* [12]. This observation encouraged us to upgrade our system into a meta-learning system which combines the output of heterogeneous learners.
– The General Impostor approach [5,4] was a natural first candidate to use as individual learner, since it was the approach taken by the best system in PAN 2014 [3]. We also implemented two additional strategies (see §4).
– Finally, because of the genetic algorithm, the previous version of our system contained most of the basic components needed to generate multiple independent learners, which made it technically easier to take this quite complex approach.

3 General Architecture

Our approach consists in training a set of $5 \times N$ individual learners, with N learners obtained from each of the five distinct methods (we call them *strategies*) that we have implemented (see §4). Every model obtained at the end of this stage is a full "authorship verification system", in the sense that, given a set of problems as input, it provides a set of scores in $[0, 1]$ as answers. Then the training of the *meta-learner* takes place, which consists in selecting the optimal subset of individual learners and the optimal way to combine their outputs together. Both training stages are carried out with a generic genetic algorithm, which evaluates models according to their performance obtained with cross-validation.

3.1 Genetic Algorithm

We call *configuration* a set of parameter-value pairs: $C = \{p_1 \mapsto v_1, \ldots, p_n \mapsto v_n\}$. For each strategy, we define a set of parameters in such a way that a given configuration defines exactly how a model is learned (or applied, when associated with a corresponding model). In particular the configuration describes which features are used and how they are combined (possibly using a ML model). In training mode, a configuration C and a set of instances (problems) S define a model M in a unique way[6]: $f_{train}(C, S) = M$. In testing mode, a configuration C, a model M and an instance s define a unique prediction: $f_{test}(C, M, s) = p$. The space of all possible configurations can be very large, depending on the strategy. This is why we use a genetic algorithm, in order to learn the optimal configurations.

[6] This is not entirely true because of the randomness in some strategies. It is however theoretically true in the limit, that is with sufficiently random runs.

Our genetic algorithm works with configurations as “individuals”: each configuration describes the meta-parameters of a strategy. A *multi-configuration* associates multiple values to one parameter:

$$MC = \left\{ p_1 \mapsto \{v_1^1, \ldots, v_{m_1}^1\}, \ldots, p_n \mapsto \{v_1^n, \ldots, v_{m_n}^n\} \right\}$$

Thus, a set of multi-configurations can be used to describe the set of meaningful combinations of parameters, in a way similar to a disjunctive normal form.[7] This union of multi-configurations is the input of the genetic algorithm:

- The first generation of configurations is initialized randomly: N configurations are selected among the possibilities described by the union of multi-configurations.
- Then every generation is obtained based on the individual performance of the configurations from the previous generation:
  - The “breeders” are selected in a way such that the probability of a configuration being selected is proportional to its rank by performance.
  - For every new configuration, two parents are selected randomly among the breeders and every parameter is assigned the value of either one of the parents value, with the possibility of mutation.

Additionally, the algorithm allows for a proportion of the new generation to be selected fully randomly, and for a proportion of the best previous configurations to be cloned to the next one (elitism). We had observed in [7] that the meta-parameters of the genetic algorithm do not have a major impact on its success or its speed. Thus we used the following fixed values:

- Proportion of selected breeders: 10%.
- Probability of mutation (by gene/parameter): 0.02.
- Proportion of cloned “elite” configurations: 10%.
- Proportion of new fully random configurations: 5%.

The convergence of the algorithm is tested by looking at the n latest windows of m generations (length), starting from the $(n \times m)^{th}$ generation: the mean of the performance is computed for every window, and the stop criterion is met if the first window obtains the maximum performance (that is, the performance didn’t improve on the last $n - 1$ windows). This simple method offers some useful flexibility: depending on the characteristics of the process (e.g., size of the hypothesis space, time needed to compute a generation), the process can be set to favor speed or a more exhaustive exploration of the search space.

3.2 ML Setting

The disadvantage of using a genetic algorithm, especially with a vast set of possible configurations, is the risk of overfitting. We use cross-validation *inside* the genetic algorithm: in other words, each configuration generated is evaluated using k-fold cross-validation, and the resulting performance is used as the fitness function by the genetic

[7] In practice, however, we have used only one multi-configuration by strategy, leaving the selection of relevant combinations to the genetic algorithm.

algorithm. It is worth noticing that k-fold cross-validation *outside* the genetic process is not a good option for two reasons: first, the genetic process is very long, hence hardly practicable for any decent value of k; but it is also very unlikely that two distinct genetic processes would find the same optimal configurations, given the size of the search space[8] and the fact that a genetic algorithm can only return a local optimum.

We use various techniques to keep overfitting to a minimum:

- The partitioning for the k-fold cross-validation is randomly (re-)generated at every generation.
- The system allows to chain multiple stages of genetic learning with different parameters, in particular different values of k and different values for the size and number of windows in the stop criterion. At every new stage, the previously selected set of configuration is used as first generation, and re-evaluated under the new parameters. This allows the process to check and progressively refine the optimal configurations and/or adjusting the trade-off between the precision of the process and the required computing power.
- At the end of the last stage, the N best configurations are re-evaluated using a 10x2 cross-validation setting, in order to control the influence of the cross-validation partitioning on the performance variance.

At the end of the first stage, for every dataset we obtain N strategy configurations and their corresponding models for each of the five strategies. Then the second stage of meta-learning is carried out, using the the predicted scores of the strategy configurations as features and the same genetic algorithm. A special *meta-configuration* is built, which contains only:

- For each of the selected strategy configurations, a binary parameter which indicates whether its prediction is used or not (feature selection).
- A parameter which indicates how the strategy predictions are combined. In order to avoid another potential source of overfitting (see discussion below), we restricted this combination to the safest methods: the algorithm can select only the arithmetic mean, geometric mean or the median. Of course, using a regression algorithm might provide better results.
- A parameter which indicates whether to apply the C@1 optimization again on the output score (see §4).

We were hesitant as to what would be the most reliable approach regarding the use of instances:

- The most simple option consists in using a first subset of the instances (e.g. 50%) for the strategy training stage, then the second one for the meta training stage. Optionally, the second set of instances can be split further in order to control for overfitting in the meta training stage with a new subset of fresh instances (e.g. 25% and 25%).
- A proper nested cross-validation setting seems a methodologically more solid approach. There are, however, two major issues with that:

[8] From $124,000$ (*robust* strategy) to 10^{21} (*fine-grained* strategy); see §4.

- Training the genetic algorithm for every strategy and every dataset ($5 \times 4 = 20$ cases) requires a lot of time; multiplying that by k in the case of k-fold cross-validation is likely to be prohibitive.
- In the case of an inner cross-validation process with respect to the genetic algorithm, overfitting arises because, even if a given model at generation t is tested on instances that it did not see, the instances have been seen in the previous generations and influence greatly[9] the configuration from which the model was generated. The case of outer cross-validation is discarded for the reasons explained above.

This is why we opted for an hybrid (maybe somewhat unorthodox) setting which consists in doing the three stages of the first option using 2-fold cross-validation. That is, in both runs:

1. The strategy genetic training uses inner cross-validation on 50 instances. As a result we obtain the 10 best models by strategy (50 models);
2. The meta training uses 25 fresh instances, in order to avoid the overfitting issue caused by inner cross-validation. We use only unsupervised combinations methods in order to minimize overfitting, which is very likely on such a small number of instances.
3. The best meta-models are evaluated on the fresh last 25 instances (as well as the best strategy models, mostly for comparison purposes). This part of the training data is called the *meta test set*.

The last stage of evaluation uses bagging (bootstrap aggregation): the models are evaluated 20 times against a different random 50% subset of the instances, so that we can control for performance variance. But there are still two problems left:

– The last evaluation stage is done on a small set of instances, and therefore not very reliable.
– The resulting models from the two cross-validation runs are not comparable together (see above). Moreover, it is likely that one of the runs will perform better in absolute value because of the different subsets they use for training and testing.

This is why we carry out an additional stage of evaluation of all the resulting models on the whole dataset, using bagging again. The rationales for this are: (1) this is a pragmatic workaround for the two issues above, and (2) the meta-models have not seen *directly* the first 50 instances used for the strategy training stage; testing proved that the performance on these instances was, in general, not overevaluated compared to the actual fresh instances of the meta test set. Of course, being aware of the potential biases, we do not consider the results of this final evaluation stage as totally reliable. We rather consider these as a useful indication of the models behaviour, that we study together with the results on the meta test set. This is why the very last stage of model

[9] We have tested this option and the results of the meta training stage were very bad: the resulting meta learner would expect too much reliability in the scores ("too good" scores) it uses as features, hence it performed badly on regular ("mediocre") scores, as obtained on unseen data.

selection is not currently automatic: we make the decision based on several statistics, the comparison to the other models and what we know about the possible biases.

There are probably other relevant options that we did not consider, due to lack of time or by ignorance. This naive and pragmatic approach is for us a rather reasonable first attempt at using multiple learners for this task.

4 Individual Strategies

Five distinct strategies are used to generate the individual learners. Two of them, namely the *robust* and the *fine-grained* strategies, come from the system that we submitted at the previous PAN task (2014). A brief summary of these approaches is given below; for more details, see [7].

- The *fine-grained* strategy contains many possible parameters. It is intended to try as many configurations as possible, in order to maximize the performance.
- The *robust* strategy is a simpler method which uses only a small set of parameters. It is intended to be safer (in particular less prone to overfitting), but probably not to perform as well as the fined-grained strategy.

We also implement three new strategies: *GI*, *Universum Inference* and *Topic Detection*, which are described below.

Technically, every strategy takes as input a set of problems with their answers (training mode) or a set of problems and a model (testing mode), and optionally some additional resources (e.g. impostors documents). As output, the strategy returns a set of features for every problem[10]; these features are fed to a ML algorithm[11], which is used to train a regression model (or to apply a previously trained model), using each problem as an instance ("Yes" answers are converted to 1, "No" to 0).

An additional optional step can be carried out, which consists in optimizing the answers in order to maximize the C@1 score, i.e. assign 0.5 scores to ambiguous cases. Two options are provided: the most simple option consists in finding the optimal accuracy threshold, then testing if assigning 0.5 to the instances with close scores improves the C@1 score. The second option consists in training an additional ML classification model, which tries to determine which cases are ambiguous based on the features and the predicted score.

Each application of a strategy is governed by a configuration (see §3.1) which defines not only the specific parameters for this strategy, but also some parameters which are common to all strategies. These include:

- some general-purpose parameters, like the kind of observations (words or characters n-grams, POS n-grams, ...), the minimum frequency, etc. (see [7] for more details).

[10] Currently we use only numerical features. The number of features depends on the strategy and its parameters, but is generally around 5 and 15, since more features would be likely to overfit the model given the low number of instances.

[11] It is also possible for a strategy to output a single value by problem, to be used directly as the score; we do not use this possibility in the current version. As regression algorithms we use only SVM and decision trees regression.

– the ML algorithm to apply to the features and its hyper-parameters[12], and whether the optional C@1 optimization stage should be done, and if yes how.

4.1 General Impostor (GI)

This method is described fully in [5], used in previous PAN workshops by [10] and in a modified form by [6] and [3]. Similarly to these authors, we use the result of Google queries formed by words from the training set documents as impostors. We define the Python implementation of the method used in the system as follows.

Given a feature set of size *N*, a source text *X*, a target text *Y* and a set of impostors $I = \{Z_1....Z_i\}$, we select a random percentage of features *a* of N and repeat this selection *n* times.

We then obtain for each system run a matrix P with dimensions $(i + 1) \times n$ where each cell is the *value*[13] for a particular distance metric for a comparison (i + 1) (columns) between the source *X* and either the target *Y* or one of the set of impostors for each of *n*(rows), each row a randomly selected subset of *N*.

This distance metric employed in each run is either the cosine distance, Kullback-Leibler divergence or the Average Jaccard Index between two texts using a metric subset *a* of the full metric set *N*. The distance metric is randomly selected at the beginning of each run. The metric representation *N* used in a run of the system is also defined in a previous step. Finally the matrix values are combined into a small set of features based on various options, including the variants used in the literature.

4.2 Topic Modelling

The topic modelling is used to exhibit topics from the underlying text in a corpus. The intuition is that the Latent Dirichlet Allocation (LDA) [1] topic model will be able to exhibit the styles from the underlying text. Every author's text was split into different character n-grams, which were considered to represent the style of the author and LDA model was constructed using [9] to find 5 topics from each author's text; say $\boldsymbol{A} = \{T_{A1}, T_{A2}, T_{A3}, T_{A4}, T_{A5}\}$, with each topic containing top 20 n-grams. Similarly, 5 topics were constructed from the document which is to be verified; say $\boldsymbol{V} = \{T_{V1}, T_{V2}, T_{V3}, T_{V4}, T_{V5}\}$ with each topic containing the top 20 n-grams. Then, we used a overlap metric between each of the topics of $\boldsymbol{A}$ and $\boldsymbol{V}$ to find how close the topics of the author and the document to be verified are, in terms of the style. This got a matrix of overlap metric values for $\boldsymbol{A} \times \boldsymbol{V}$. From this we obtained the overall mean of the overlap metrics as a feature for the genetic algorithm.

The use of topic modelling might seem counter-intuitive in this task. However, our idea is to let the genetic algorithm select the features used by the algorithm, hoping that it will select features which are relevant for style distinctiveness rather than topic.

[12] We actually restrict these to a very small set of possibilities.
[13] To five decimal places.

4.3 Universum Inference

This strategy follows the idea described in [13]: in this paper, a large corpus containing several "categories" (known authors in our task) was split into small chunks. A chunk of category A was compared against many chunks from other categories and from category A as well, picked randomly. It was shown that a reliable measure of the category homogeneity can be derived from examining how different the level of similarity is between comparing A to A and comparing A to some distinct category X.

We adapted the approach to the case of smaller documents in the following way. Let A and B be two documents, the following process is repeated N times with different random subsets:

1. The two documents are split into three parts randomly: A_1, A_2, A_3 and B_1, B_2, B_3; one of the thirds is split again into two parts: $X_3 = X_3', X_3''$.
2. The pieces of text are re-organized under three categories, each containing two parts and all the parts being of similar size: $C_A = \{A_1, A_2\}$; $C_B = \{B_1, B_2\}$; $C_{mixed} = \{A_3' \cup B_3', A_3'' \cup B_3''\}$.
3. The three categories are compared in the same way as [13], that is, both against itself (using the two parts belonging to this category) and against each other category (picking one of the two parts randomly).
4. The general idea consists in measuring how much confusion there is between the categories: assuming that the goal is to correctly classify each category to its own category, more classification errors means more similar documents. This can be achieved numerically with different methods (a particular method is selected by the configuration parameters).

5 Practical Issues

During the development, training and testing of our system, we faced a number of practical issues. These issues probably had an impact on the quality of the results, but it is impossible to properly measure to what extent.

Of course, the most important constraint was the time schedule of the competition. We started the development quite late. For the most part, the system was coded during the time that the competition was taking place. This left relatively little time to train the 40 runs of the genetic algorithm to their best (4 datasets $\times$ 5 strategies $\times$ 2-fold cross-validation). Thus, it is possible that for some complex strategies we did not reach the optimal models[14]; more importantly, we did not have enough time to make the models run through enough different stages of cross-validation under restrictive convergence conditions (see §3.1), which could have provided more stability in the models.

It is difficult to evaluate precisely the amount of computation that we gave to the whole training process, because we used two distinct machines and there were interruptions during the process. We evaluate it roughly between 150 and 300 CPU hours for each of the 40 genetic processes.

[14] Additionally, we discovered after the competition that a bug caused a disruption in the genetic process in the case of the Dutch and English datasets.

6 Results on the Training Set and Observations

6.1 Strategy Observations

Comparing the performance of the five strategies (see 4) shows little surprise; it confirms relatively well the differences between those strategies and their complementarity:

- The *GI* strategy performs very well in general, and its low variance makes it especially robust.
- The *Universum Inference* strategy performs well and is even the best individual strategy on two datasets (Dutch and English); it is however quite unstable.
- The *robust* strategy keeps its promises: albeit low performance in general, it is practically insensitive to overfitting, showing very close performance on training instances and fresh instances.
- The *fine-grained* strategy is vey unstable, but occasionally performs very well.
- The *Topic Modeling* strategy does not seem to work very well in general.[15]

6.2 Global Results

Table 1 shows some statistics about the performance of the selected meta-models on both the *meta test set* and the whole training set set[16]; the final column shows the performance on the actual test set.

The selected models were not necessarily the best performing on the meta-test set (or on all the instances). Our choice of the final model was mostly based on finding the most stable model among the most performant ones. We took standard deviation into account, as well as the difference in performance between the two sets (interpreting a large difference as an indication that the model is not stable).

Dataset	Meta test set			Full training set			Test set	
	mean	median	std. dev.	mean	median	std. dev.	perf.	rank
Dutch	0.710	0.716	0.107	0.722	0.727	0.087	0.635	1st
English	0.405	0.400	0.117	0.421	0.420	0.064	0.453	6th
Greek	0.656	0.671	0.110	0.761	0.765	0.042	0.693	2nd
Spanish	0.950	0.917	0.042	0.952	0.946	0.024	0.661	4th
						Macro-average	0.610	2nd

Table 1. Performance observed on the traning set and test set

It is worth noticing that the relative performance demonstrates fairly well the most visible difficult characterstics of the dataset. For example, the English dataset contains

[15] It has however been selected as individual learner in several cases in the meta-learning stage; thus, such models might actually bring a different kind of information which turns out to be useful to the meta-model (but we did not have time to investigate this question in more detail).

[16] See §3.2 for explanations about the setting in which these performance values are obtained.

only one known document for every problem and has the smallest documents (in average), and it is also the most difficult to predict for the system. On the contrary, there are always four known documents by problem and the documents are the largest for the Spanish dataset, and the system performs very well with it.

The system had the worst difficulties in the case of the English dataset. It might be because, as a strongly supervised system, it does not cope well with scarcity of information. It is also possible that a more complete training would have helped in finding the relevant clues in this more difficult case.

Finally, in most cases the results on the test set are close (within the standard deviation) to the ones obtained on the traning set. This tends to indicate that the approach is good at preventing overfitting. Even in the case of the visibly overfit Spanish model, the results do not fall drastically compared to the other participating systems; this moderation of the effect of overfitting is probably explained by the use of multiple learners, the overfit ones being partly compensated by the others.

7 Future work

There is a lot of room for improvement in our current system: improving the current strategies, adding new strategies, solving efficiency issues, etc. In our opinion, the most interesting question to study is certainly how best to combine the predictions of the individual learners, in the context of a relatively small training set which is expectable in authorship verification. In the current version, the ML setting is not entirely satisfying (see §3.2). Various options could be explored in the future:

- Generating artificial instances from the existing ones, for instance by splitting original documents into multiple sub-documents. This option might not be completely safe, since artificial cases might be less useful in assessing the reliability of a model, and maybe even misleading.
- Finding a way to combine the best models provided by multiple runs of the genetic algorithm. In theory, these models share some similarities, and these similarities correspond to the features that matter, those which allow the model to perform well in a reliable way (the other features can be seen as the result of statistical noise). It is however not necessarily possible to safely "merge" two distinct configurations.[17]

Acknowledgments

This research is supported by Science Foundation Ireland through the CNGL Programme (Grant 12/CE/I2267) in the ADAPT Centre (www.adaptcentre.ie) at Trinity College, University of Dublin.

[17] This option could also be considered in the perspective of simply measuring how similar two configurations are: based on such a similarity measure, instead of "merging" heterogeneous configurations, the system could select configurations based on their similarity across different runs. The same idea could be used to maximize the diversity of the selected strategy models, which is likely to be a decisive feature in the success of the meta-learner.

Some of the calculations were performed on the Lonsdale cluster maintained by the Trinity Centre for High Performance Computing. This cluster was funded through grants from Science Foundation Ireland.

Fitting our complex and poorly efficient system to the tira.io framework proved challenging, and we are very grateful to the organizers for their patience and availability.

References

1. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. In: Journal of Machine Learning Research. vol. 3, pp. 993–1022 (March 2003)
2. Cappellato, L., Ferro, N., Halvey, M., Kraaij, W. (eds.): Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014, CEUR Workshop Proceedings, vol. 1180. CEUR-WS.org (2014), http://ceur-ws.org/Vol-1180
3. Khonji, M., Iraqi, Y.: A slightly-modified gi-based author-verifier with lots of features (ASGALF). In: Cappellato et al. [2], pp. 977–983, http://ceur-ws.org/Vol-1180
4. Koppel, M., Seidman, S.: Automatically identifying pseudepigraphic texts. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1449–1454. ACL (2013)
5. Koppel, M., Winter, Y.: Determining if two documents are written by the same author. Journal of the Association for Information Science and Technology 65(1), 178–187 (2014)
6. Mayor, C., Gutierrez, J., Toledo, A., Martinez, R., Ledesma, P., Fuentes, G., Meza, I.: A single author style representation for the author verification task. In: CLEF 2014 Evaluation Labs and Workshop-Online Working Notes (2014)
7. Moreau, E., Jayapal, A., Vogel, C.: Author Verification: Exploring a Large set of Parameters using a Genetic Algorithm - Notebook for PAN at CLEF 2014. In: Linda Cappellato, Nicola Ferro, M.H., Kraaij, W. (eds.) Working Notes for CLEF 2014 Conference. vol. 1180, p. 12. CEUR Workshop Proceedings, Sheffiled, United Kingdom (Sep 2014)
8. Peñas, A., Rodrigo, A.: A simple measure to assess non-response. In: Proceedings of the 49th Annual Meeting of the ACL: Human Language Technologies. pp. 1415–1424. Association for Computational Linguistics, Portland, Oregon, USA (June 2011), http://www.aclweb.org/anthology/P11-1142
9. Phan, X.H., Nguyen, C.T.: . gibbslda++: A c/c++ implementation of latent dirichlet allocation (lda). http://gibbslda.sourceforge.net/ (2007)
10. Seidman, S.: Authorship verification using the impostors method. In: CLEF 2013 Evaluation Labs and Workshop-Online Working Notes (2013)
11. Stamatatos, E., Daelemans, W., Verhoeven, B., Juola, P., Lopez Lopez, A., Potthast, M., Stein, B.: Overview of the Author Identification Task at PAN 2015. In: Working Notes Papers of the CLEF 2015 Evaluation Labs. CEUR Workshop Proceedings, CLEF and CEUR-WS.org (Sep 2015), http://www.clef-initiative.eu/publication/working-notes
12. Stamatatos, E., Daelemans, W., Verhoeven, B., Stein, B., Potthast, M., Juola, P., Sánchez-Pérez, M.A., Barrón-Cedeño, A.: Overview of the author identification task at PAN 2014. In: Cappellato et al. [2], pp. 877–897, http://ceur-ws.org/Vol-1180
13. Vogel, C., Lynch, G., Janssen, J.: Universum inference and corpus homogeneity. In: Bramer, M., Petridis, M., Coenen, F. (eds.) Research and Development in Intelligent Systems XXV, pp. 367–372. Springer London (2009)