PAN 2016 shared task
Author identification

Can a clustering sandwich help a neural network

in establishing authorship links?

Olivier Louwaars (s2814714), Department of Information Science
Rijksuniversiteit Groningen

Abstract

Faced with the problem of having an unknown number of authors writing an unknown number of documents that might but do not necessarily have the same topic, we came up with a pipeline of a K-Means clustering algorithm informing a neural network to establish document similarity and which then informs a Meanshift algorithm that outputs the final clusters (each cluster representing an author and enveloping one or more documents). The data provided by PAN consists of 18 problems that each consist of 50-100 documents from various authors. Every problem has either news articles or reviews, and is written in English, Dutch or Greek. Document length differs from 130 to 1000 words. This is not enough training data for a neural network, so character based features were created for training. Also, the task itself limited feature selection to character-only as well. If words or word n-grams would have been used, the initial assumption was that the system would be tricked easily into clustering on topic instead of author. As all documents within a problem have one genre, it is not unimaginable that documents from different authors have the same topic, and are therefore grouped together. Preventing the system from topic clustering and aiming it at author clustering was one of the biggest challenges in this task. The most promising feature for this task were character based skipgrams [1] pairing every character with either a neighboring one (positive sample) or one further away (negative sample). The embeddings thus created informed the neural network with underlying information regarding character sequences and structures. Although the task is to cluster all documents of a single author, the chosen approach was to train the neural network with every possible document pair, like the PAN 2015 task for author identification. The baseline thus created was very high (94%), resulting in a default decision to most frequent class (negative) for all samples. The K-Means clusterer was added to see if the number of pairs could be cut back, as a set of 50 documents already results in over 1200 possible pairs. K-Means was run with an iterative setting of [1:n-1] clusters, after which the total of document clusters was counted. If two documents were never clustered together, they were stripped from the input for the neural network. This lead to a 50% reduction in document pairs, but also a 25% reduction of correct pairs. Although this improved the baseline slightly, the data was still to biased for the neural network to be able to detect correct pairs in the training data, creating useless output of the network, which gave the Meanshift no additional features to work with.

During development and tweaking of the neural network, the initial deadline for the shared task expired without submitting a working system. When the overview paper was published, the used method (on development set) could still be compared to the test set results of the other team. Based on only character based preprocessed data, as this lead to the best result in the PAN task of 2015 [2], Meanshift output had a precision of 0.12 on average. Using a Scikit-learn Countvectorizer for preprocessing the data with both word Ngram counts and normal word counts, the BCubed precision was 0.256, with a BCubed F-score of 0.376 (8th out of 9 participants). This measure was also applied by the task committee and although the system was not submitted, results could be compared with other participants afterwards [3]. The second part of the task, the ranking of links within clusters resulted in a mean average precision of 0.014 (5th place).

Contents

1. Introduction

With the publishing and sharing of documents becoming easier and easier via the internet, the need to verify what was written by who becomes apparent. Not only to prevent plagiarism, but also to prevent texts of unknown authorship from being attributed to an author they do not belong to. Since 2011, PAN[1] contains a shared task regarding automatic authorship identification. Where in recent years the task focused on determining whether or not a certain document belongs to a set of known documents of an author, the 2016 task is to cluster documents per author, without knowing the number of contributing authors. Additionally, the task also comprises a second step, in which the certainty of links has to be established between the different documents within a cluster/author. This second step is comparable with the earlier shared tasks, as it is a one on one comparison of documents. Depending on the similarity of two documents, the certainty of the author of both can be established (Figure 1). If the results of this shared task are satisfactory, the method can be applied on, for example, a portfolio of documents of students, to see if the author of all documents is the same. To make sure both steps are executed properly and no work is duplicated, the following research question will be the foundation of this research:

" Can a recurrent neural network help traditional clustering algorithms in clustering documents per author?"

To back up the main questions, the following sub questions can be formulated:

- What features are important for the initial clustering per author?
- What features are important for establishing links between documents?
- Can the system be prevented from clustering based on obvious but wrong patterns such as topic?

In this thesis, at first related work and literature will be explored, based on which the approach to tackle the problem and answering the research question will be explained, followed by the results and the conclusion based on those results.

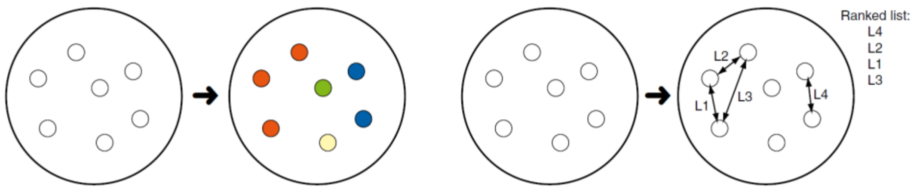

Figure 1: Examples of complete clustering (left) and authorship-link ranking (right). [3]

[1] http://pan.webis.de/

2. Method

Plagiarism detection is a hot topic in scientific research, as it is now easier than ever to copy work someone else did and claim the results for yourself. Detectors are available for a long time, with plenty of research done on best practices and approaches, depending on the structure and the type of the available data.

2.1. Data

For this task, the data is split into 18 problems. Every problem has 50-100 documents that all have the same genre (news or review) and language (English, Dutch or Greek), equally distributed over the problems (Table 1). Every problem also has a R of either 0.5, 0.7 or 0.9. R in indicative for the number of clusters with multiple authors; the lower r, the higher the chance that the problem has multi-author clusters. R correlates with max C, the maximum number of documents in a cluster. Included is a JSON file that shows language and genre per document. Also, a truth directory is available per problem. In this folder are two JSON files, one with gold data for clustering, and one with gold data for link ranking.

The small amount of data excludes the most promising solutions, as artificial neural networks combined with word embeddings are state of the art in solving practically any language related problem but need huge amounts of data to do so. Therefore, a less explored approach will be sought in this chapter.

Table 1: Evaluation datasets (left=training, right=test).

ID	Lang	Genre	r	N	k	Links	max C	Words	ID	Lang	Genre	r	N	K	Links	max C	Words
001	English	articles	0.70	50	35	26	5	752.3	001	English	articles	0.71	70	50	33	5	582.4
002	English	articles	0.50	50	25	75	9	756.2	002	English	articles	0.50	70	35	113	8	587.3
003	English	articles	0.86	50	43	8	3	744.7	003	English	articles	0.91	70	64	7	3	579.8
004	English	reviews	0.69	80	55	36	4	977.8	004	English	reviews	0.73	80	58	30	4	1,011.2
005	English	reviews	0.88	80	70	12	3	1,089.7	005	English	reviews	0.90	80	72	10	3	1,030.4
006	English	reviews	0.50	80	40	65	5	1,029.4	006	English	reviews	0.53	80	42	68	5	1,003.7
007	Dutch	articles	0.89	57	51	7	3	1,074.7	007	Dutch	articles	0.74	57	42	24	4	1,172.1
008	Dutch	articles	0.49	57	28	76	7	1,321.9	008	Dutch	articles	0.88	57	50	8	3	1,178.4
009	Dutch	articles	0.70	57	40	30	4	1,014.8	009	Dutch	articles	0.53	57	30	65	7	945.2
010	Dutch	reviews	0.54	100	54	77	4	128.2	010	Dutch	reviews	0.88	100	88	16	4	151.7
011	Dutch	reviews	0.67	100	67	46	4	134.9	011	Dutch	reviews	0.51	100	51	76	4	150.3
012	Dutch	reviews	0.91	100	91	10	3	125.3	012	Dutch	reviews	0.71	100	71	37	4	155.9
013	Greek	articles	0.51	55	28	38	4	748.9	013	Greek	articles	0.71	70	50	24	4	720.5
014	Greek	articles	0.69	55	38	25	5	741.6	014	Greek	articles	0.50	70	35	52	4	750.3
015	Greek	articles	0.87	55	48	8	3	726.8	015	Greek	articles	0.89	70	62	9	3	737.6
016	Greek	reviews	0.91	55	50	6	3	523.4	016	Greek	reviews	0.73	70	51	24	4	434.8
017	Greek	reviews	0.51	55	28	55	8	633.9	017	Greek	reviews	0.91	70	64	7	3	428.0
018	Greek	reviews	0.73	55	40	19	3	562.9	018	Greek	reviews	0.53	70	37	44	4	536.9

2.2. Related work

Most recent work focuses on comparing a new document to a set of known documents from one author, as that is the most logical approach if that data is available. The decision making was binary which does not make it a good source for this research. Instead, the method used by the winning team of PAN 2015 offers an interesting view on the problem. Using multi headed artificial neural networks[4], he was able to link two documents together if they belonged to the same author. Using this approach for clustering would of course take a lot of processing time as all possible pairs must be compared, but if it works it might be very good at clustering all the right documents together. Although primarily applied on other topics, clustering itself is of course well explored. Many algorithms have been developed that can find the center of dense clusters and compute to what range the cluster extends. In most cases however, the algorithm needs to know on beforehand how many clusters it is supposed to use to be able to find the right centers. An exception on this is Meanshift [5], a hill climbing algorithm that keeps looking for better centroids and is able to add more clusters if necessary. Meanshift is proven to be effective in determining the number of clusters and has several software implementations that can be used off the shelf. Before Meanshift was developed, the research of Holmes and Forsyth already tried to achieve a very similar result [6]. They tested their method on the very famous (and notorious in NLP tasks) federalist papers, a set of articles about the American constitution written by three authors. Their dataset is comparable to the one used for this research, which makes their approach relevant. Many of the features used by Holmes and Forsyth are quite common these days, such as word frequency counting, like tf-idf now, and trying to find stylistic patterns for authors in documents. This last feature is especially important for this task; most clustering algorithms will look for word or ngram similiarity when trying to find similar documents. Similar words still can be an indication for similar documents, but there is a risk that although the documents are alike, they are not from the same author because they all have the same genre. A field in which stylistic features are even more important is engineering, where many articles are written on the same topic and genre. This makes the research of Berry and Sazonov about the clustering of technical documents an interesting and reliable source. The nature of the documents in their dataset makes them highly structured and restricted [7], making the effect of stylistic feature selection extra visible. Although less technical, the documents in the current dataset can also be identical in structure and covered subjects. Berry and Sazonov say that sometimes the preference of an author for one word over another can be enough to distinguish who wrote what. This high influence of small features is something to keep in mind in this approach, as it can change the outcome in a very strong way.

2.3. Approach

Based on the earlier work and proven concepts, a combined approach of all will be attempted. The largest restriction is that the entire system will have to be built in Python, as both the thesis and the shared task are on a tight schedule. This leaves no room for learning a new programming language. Python does provide all necessary tools for the task, and has modules for all desired features. Scikit-learn[2] provides an excellent API for several clustering algorithms and preprocessing steps, and Keras[3] allows for building an artificial neural network on either the TensorFlow or the Theano backend.

In order to train the system, the most promising features must be selected and applied. As the approach will be based on Bagnall' s, his feature selection also applies for this problem. Training a neural network requires vast amounts of data, so the only way to do this is by looking at characters instead of words. On character level you can apply almost all techniques used on words, like the relative and absolute frequency of characters per document and the full corpus (tf-idf). Metadata about the documents will also be added, informing the system about the average sentence and word length, punctuation usage and number of mid-sentence capital letters. These stylistic features can be very indicative about an author, but according to Bagnall they should not be fed raw into the neural network. This could lead to the network assigning a too great weight to a small feature, and negatively influence decision making. Therefore, Bagnall proposes to normalize all uncommon characters [4]. Different commas, ellipses (…), quotation marks (single and double) and dashes (longer and shorter) all should be converted to a single style. Furthermore, additional whitespace must be stripped and all numbers and Latin characters in Greek texts should be normalized to a common placeholder to keep their weight evenly distributed. As final step, Bagnall recommends to convert every character into the NFKD unicode normal form. This form describes the character instead of displaying it, splitting it up if it has an accent on it to describe the accent separately (Figure 2).

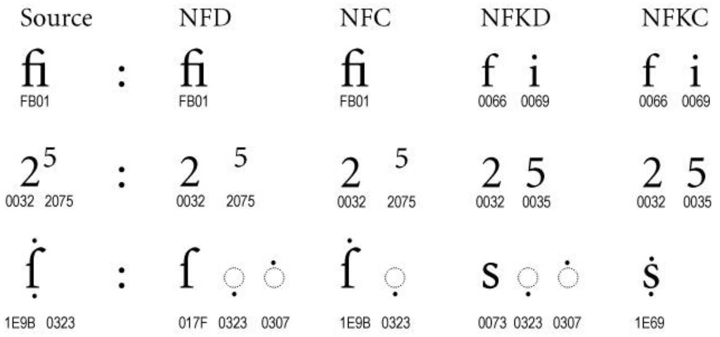

Figure 2: Different unicode forms and their output.

[2] http://scikit-learn.org/stable/
[3] http://keras.io/

The result of all normalization steps is a human unreadable list of strings per document, that can be used by both Keras an Scikit-learn for further preprocessing.
The assumption is that clustering and an artificial neural network can inform each other in order to achieve better results on the data (Figure 3). The idea of Bagnall to use a neural network for ranking document similarity is very promising, but time consuming on this many document pairs. Each of the 18 problems has at least 50 documents, which leads to at least 1225 unique pairs per problem that need to be processed. It would therefore be very useful to remove certain pairs that are highly unlikely from the initial set. This removal should be done discreetly, as it is better to remove too less faulty pairs than too much pairs that should be together. This will make the training data less skewed, and cut back the processing time as well. The pairs can be shifted using a K-Means clusterer with an unspecified number of clusters. K-Means expects the user to set the desired number of clusters (K), so by iterating through a K of [1:N-1], all possibilities will be tried. The cluster output can then be added together, to see which documents are never clustered together. That particular pair can then be removed from the full set. Once the set of document pairs has been trimmed, the remaining texts can be preprocessed to be fed into Keras' s Long Short-Term Memory, or LSTM, neural network [8]. LSTM is a type of recurrent neural network that is especially good in processing and predicting texts, as it looks back at everything it learned until now. This means that an LSTM is able to learn rules from a correct pair and apply them on the current pair, even if there is a high number of incorrect pairs in between. Given the skewed data for this task, it is important that the network remembers the sparse correct pairs as good and as long as possible. LSTM' s do have limitations on length however, so it might be that the long sequences of characters are too much for it to keep learning correctly. But just like any other implementation in Keras, LSTM' s are stackable and should be able to fit the entire sequence of characters in the documents. LSTM expects the data in three dimensions of (nb˙sequences, nb˙samples, input˙dim). Sequences is defined by the total number of sequences, samples by the length of one document and input dimension by the total of different characters in the vocabulary, so 26 + some special ones.

The data itself must be one-hot encoded, with each character being represented by a number in range[input˙dim]. Once encoded, the entire dataset will be converted into a 3D matrix by Keras' s preprocessing tools for the network to use. As an extra feature, character embeddings will be constructed using skipgrams [1]; by encoding all possible character pairs with either 1 if they are neighbors, and 0 if the pairs are far apart, a vector can be built of what characters are likely and unlikely to occur together.
The ratings of one document versus every other will be used as an additional feature for the Meanshift algorithm. Experiments will have to point out whether the feature should be shaped like a list with 1' s and 0' s, or more like a dictionary with one document as key and all its matches as values. The Meanshift implementation in Scikit-learn offers few parameters, and is able to calculate the

bandwidth it should use based on the data. A too small bandwidth results in many clusters while there might be overlapping ones, and a too large bandwidth merges too many clusters, resulting in only a few final clusters. The output of Meanshift will be a list of documents per cluster, that can be transformed in the same JSON format the task committee provides the truth data in.

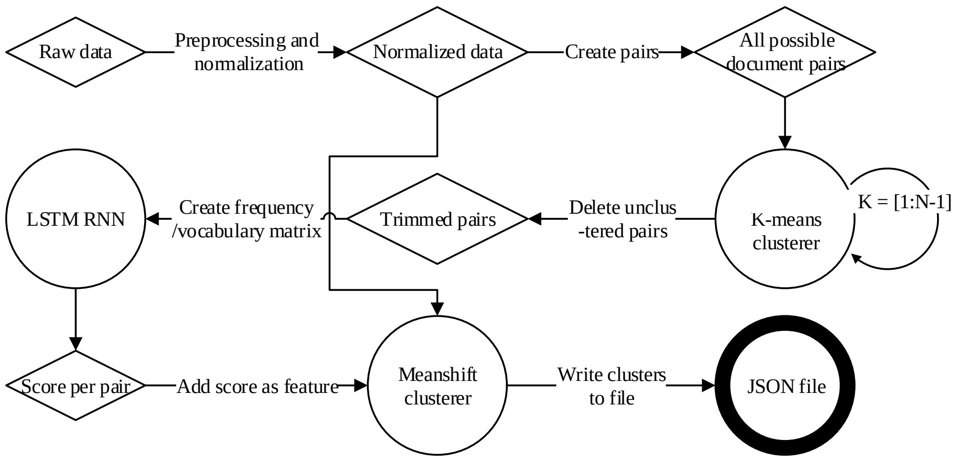

Figure 3: Dataflow troughout the system.

Using the online review environment Tira [9], automatic evaluation of the answers given versus the gold standard data will be done. Evaluation results will be according to the Bcubed score [10], a measure that combines the scores within a cluster with the ones across clusters for computing precision and recall (Figure 4). This results in different scores per language and per genre, with a total score over all problems that all will be described in the next chapter.

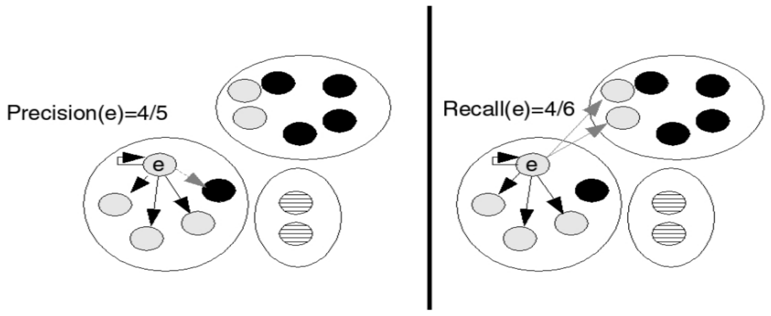

Figure 4: Calculation of Bcubed precision and recall

3. Results

When coding the software based on the dataflow from the previous chapter, it soon turned out that the sequences of characters were in fact too long for a LSTM recurrent neural network to process. The correct pairs were just too sparse for the network to remember the previous one when encountering a new one, resulting in a preference for the most frequent class with no apparent learning. This was only worsened by using the pairwise comparison in the neural network. With the aforementioned example of 1200 pairs, created from just 50 documents, only about 70 of 1200 would be correct pairs. This gives a tough to beat baseline of 94%, with far too little correct data for the network to learn from. Using the K-Means clusterer to strip unlikely pairs was successful, removing about half of the total, but at a cost of 25% of correct pairs. In absolute numbers it was a very successful method, but relatively without effect to lower the most frequent class baseline.

Meanwhile, during the development of the system and continuous tweaks to get the neural network going, the deadline for the shared task expired. No working software was submitted because the pipeline could not be completed in time, which made official evaluation at that point impossible too. Unofficial measures show a precision of 0.1 per problem on average over the 20 most common pairs in a cluster. This means that, of all clusters returned from the K-Means iterations, only the top 20 is taken into account. Of this 20, only 10% (so 2 pairs) would usually be correct according to the gold standard. So K-Means was an outstanding preprocessor if executed before the neural network, but performed bad if its output was used as final data. The iteration over all possible number of clusters made the output unfit for comparison with the gold standard, so a single K should be chosen or computed first. The returned data after iteration gave no indication when it performed best, so only a rule based K could be implemented. Exploring the development data showed that in most problems less than one third of all articles share clusters, so two third of all clusters are only populated by a single document or author. Implementing K-Means with K=2/3(N) worked out well in F-score (Table 2), it is a highly unstable method on unseen testdata, as it assumes that the shape will always be the same as that of the development data. The good results in Table 2 are only because of rules based on the observations, and working with a rule based system is never recommended if sufficient data for machine learning is available. Despite performing significantly less than K-Means, the idea was that Meanshift is better prepared for unseen data, making it a better candidate for final implementation.

With the neural network out of the system, a different set of features was selected to optimize the clustering algorithms. The data provided was not very extensive, but big enough for word based features for clustering. Changing from characters to words greatly influenced clustering performance, with acceptable results. However, adding features outside of stripping accents and ngrams did not contribute to a higher score. Clustering performed the same, regardless of adding word or punctuation counts. The system seemed to ignore topic similar documents by itself, clustering based on stylistic features instead in many cases. Once again, this was primarily the case with K-Means that performed much better than expected.

Table 2: Experiment results per system

System	Complete clustering			Link ranking	Runtime (s)
	B3 F	B3 rec.	B3 prec.	MAP	
MS Word	0.376	0.877	0.256	0.014	208
MS Char	0.065	1	0.034	0.020	174
KM Word	0.696	0.732	0.676	0.008	191
KM Char	0.065	1	0.034	0.020	135
Baseline	0.811	0.697	1	0	120

Because no ranking within clusters could be made without a neural network, it was decided to give every link of two documents within a cluster a certainty of 1, trusting on the accuracy of Meanshift. Due to the presentation of the overview paper [3] of the 2016 shared task, the evaluation script and other results also were made available for comparison (Table 3). Note that the results in the table are presumably on the test set, while our results are on the development set. As seen before the two datasets are very alike (Table 1), so the results also might be comparable, but this should be said with great caution.

The results of Meanshift are surprisingly good for the simple implementation via Scikit-learn, although only Bcubed recall for clustering and mean average precision for ranking are above the random baseline. Especially Singleton baseline performed very well on clustering, as the large majority of clusters only consisted of a single document. Only Bagnall, also winner of last year and following a similar approach as this research, and Kocher were able to beat it in Bcubed F-score, and only just. Cosine baseline on the other hand seemed very low and easy to beat at first sight, because it would purely focus on word similarities between documents. This feature was exactly what all participants wanted to avoid as it would rank by topic instead of by author, but even by actively only selecting stylometric features only two participants were able to beat it.

Table 3: FInal results of the PAN 2016 shared task for author clustering. [3]

Participant	Complete clustering			Link ranking			Runtime
	B3 F	B3 rec.	B3 prec.	MAP	RP	P@10	
Bagnall	**0.822**	0.726	0.977	**0.169**	**0.168**	**0.283**	63:03:59
Gobeill	0.706	0.767	0.737	0.115	0.131	0.233	00:00:39
Kocher	**0.822**	0.722	**0.982**	0.054	0.050	0.117	00:01:51
Kuttichira	0.588	0.720	0.512	0.001	0.010	0.006	00:00:42
Louwaars MS	0.376	0.877	0.256	0.014	-	-	00:03:28
Mansoorizadeh *et al.*	0.401	0.822	0.280	0.009	0.012	0.011	00:00:17
Sari & Stevenson	0.795	0.733	0.893	0.040	0.065	0.217	00:07:48
Vartapetiance & Gillam	0.234	**0.935**	0.195	0.012	0.023	0.044	03:03:13
Zmiycharov *et al.*	0.768	0.716	0.852	0.003	0.016	0.033	01:22:56
BASELINE-Random	0.667	0.714	0.641	0.002	0.009	0.013	–
BASELINE-Singleton	0.821	0.711	**1.000**	–	–	–	–
BASELINE-Cosine	–	–	–	0.060	0.074	0.139	–

Detailed results per genre and language for clustering can be found in Table 4. Interesting in these results is the high difference in languages where most teams have an almost equal score for all three. Although the system was only working with character features, somehow it was much better in clustering Dutch texts than Greek or English. Also, the ratio between scores of articles and reviews differs per participant, with no clear pattern in which genre is easier or tougher to cluster. Contrary to most participants, a lower R only leads to higher F-scores. The difference between the participants and baselines is even clearer in this table, and shows that despite the naivety of the baselines they perform far better than most teams.

Table 4: Evaluation results (mean BCubed F-score) for the complete author clustering task. [3]

Participant	Overall	Articles	Reviews	English	Dutch	Greek	$r≈0.9$	$r≈0.7$	$r≈0.5$
Bagnall	**0.822**	**0.817**	**0.828**	**0.820**	**0.815**	0.832	0.931	0.840	**0.695**
Kocher	**0.822**	**0.817**	0.827	0.818	**0.815**	**0.833**	**0.933**	**0.843**	0.690
BASELINE-Singleton	0.821	**0.819**	0.823	**0.822**	**0.819**	0.822	**0.945**	0.838	0.680
Sari & Stevenson	0.795	0.789	0.801	0.784	0.789	0.813	0.887	0.812	0.687
Zmiycharov *et al.*	0.768	0.761	0.776	0.781	0.759	0.765	0.877	0.777	0.651
Gobeill	0.706	0.800	0.611	0.805	0.606	0.707	0.756	0.722	0.639
BASELINE-Random	0.667	0.666	0.667	0.668	0.665	0.667	0.745	0.678	0.577
Kuttichira	0.588	0.626	0.550	0.579	0.584	0.601	0.647	0.599	0.519
Mansoorizadeh *et al.*	0.401	0.367	0.435	0.486	0.256	0.460	0.426	0.373	0.403
Louwaars MS	0.376	0.386	0.367	0.27	0.465	0.394	0.356	0.376	0.397
Vartapetiance & Gillam	0.234	0.284	0.183	0.057	0.595	0.049	0.230	0.241	0.230

Table 5 makes the results for authorship ranking insightful per genre and language. Again, no genre is consequently ranked better than the other, but there seems to be a tendency of Dutch ranks scoring the lowest throughout all teams. Although the achieved mean average precision of 0.014 seems incredibly low, in this table it turns out to be halfway between the best and worst results. The fully random baseline was narrowly beaten, but cosine similarity appeared to be far more indicative of link ranks than assumed, especially for Greek. Table 5 also shows the same rising scores as Table 4 with a descending r, but for the ranking all other teams show the same behavior.

Table 5: Evaluation results (MAP) for the authorship-link ranking task. [3]

Participant	Overall	Articles	Reviews	English	Dutch	Greek	$r≈0.9$	$r≈0.7$	$r≈0.5$
Bagnall	**0.169**	**0.174**	**0.163**	**0.126**	**0.109**	**0.272**	**0.064**	**0.186**	**0.257**
Gobeill	0.115	0.119	0.110	0.097	0.079	0.168	0.040	0.105	0.198
BASELINE-Cosine	0.060	0.063	0.057	0.053	0.053	0.074	0.019	0.054	0.107
Kocher	0.054	0.047	0.061	0.032	0.044	0.085	0.042	0.058	0.063
Sari & Stevenson	0.040	0.033	0.047	0.009	0.042	0.069	0.017	0.041	0.062
Louwaars MS	0.014	0.015	0.013	0.016	0.007	0.018	0.004	0.01	0.027
Vartapetiance & Gillam	0.012	0.010	0.014	0.014	0.006	0.016	0.010	0.008	0.017
Mansoorizadeh *et al.*	0.009	0.013	0.004	0.006	0.010	0.010	0.002	0.009	0.014
Zmiycharov *et al.*	0.003	0.002	0.004	0.001	0.000	0.009	0.002	0.003	0.004
BASELINE-Random	0.002	0.002	0.001	0.001	0.002	0.002	0.001	0.001	0.002
Kuttichira	0.001	0.002	0.001	0.001	0.002	0.001	0.001	0.002	0.001

4. Discussion

In this final chapter all unexpected results will be presented, with a possible explanation if there is one at hand. Also, a general reflection on the designing and development of the system will be given with an answer on the research questions.

4.1. Evaluation

The results presented in chapter 3 are far worse than anticipated, but seeing them in context with the other teams helps understanding that they are not that bad. Given the failure of the most important processing part of the system, the recurrent neural network, the result is quite satisfactory and in line with the others. One thing that stands out however, are the irregularities in scores in comparison with other teams. Where most teams have an equal score for all languages, our system has a clear preference for one over the other, and is especially good in Dutch and bad in English. An explanation for this can be given by the unicode normalization step in the process. unicode encodes accents on characters as separate characters, and as Dutch has more accented letters than English, these encodings might be highly informative.

R values are also the reverse of most teams, with better results with a lower r. This can be explained by the conservatism of Meanshift, as it tends to use as few clusters as possible, and therefore performs better in problems with a small k. Finding a way to increase the number clusters found by Meanshift would therefore increase the overall performance of the software and seems promising for future research. Another recommendation is to apply cosine similarity in link ranking, because it worked surprisingly well for cosine baseline. At first the thought was that the baseline would be misguided by topic of documents, as even the task committee underwrote, but this turned out to be false as the baseline had the third score of all teams. An indication of the similarity preferences of the system was already given in the clustering process, as clustering did not improve with adding stylistic features. Using cosine similarity thus would be a good first step in ranking, combined with a more sophisticated way of ranking documents based on similarity.

4.2. Conclusion

Looking back at working on this shared task, it really is a pity that no system was ready for submitting at the time of the deadline. With no submission the opportunity of competing with other teams was gone, making the development and outcome less exciting. Still, the achieved results are not too bad given shape of the final system. The proposed approach with two clustering methods as input and output of a recurrent neural network seemed very promising based on earlier results. This is underlined by the fact that Bagnall’ s system of this year is based on his submission and suggestions of last year. Unfortunately, the shared task’ s individual papers are not released yet, so it is not possible to see how Bagnall got his system to work so well, and what steps were missed in our software. His results do show however that it is possible and useful to use recurrent neural networks for author clustering and link ranking, but this cannot be used to answer the research question here.

The answer to “ Can a recurrent neural network help traditional clustering algorithms in clustering documents per author?” according to this research is negative, as the approach originally proposed could not be implemented. The resulting neural network did not help at all to inform the clustering or to rank the links in the output. An interesting finding though, is that a K-Means iterative approach for stripping away non-likely pairs is working well. By using this preprocessing step, half of all pairs fed to the neural network could be removed, greatly decreasing processing time.

To support the research question, several sub questions regarding features and their effect were formed. Before development started, it was feared that the unexplored field of clustering on author would be easily tricked into clustering on topic, as most clustering algorithms heavily rely on document similarity. Suggesting punctuation counts as important feature was part of the tactic to help the system distinguish different authors, but it turned out that K-means and Meanshift were perfectly capable of that themselves. The addition of stylistic features did not contribute to a higher score, which suggests that either the documents never had the same topic, or that in clustering a higher weight was given to words that were not topic related.

Bibliography

1. Mikolov, T., et al., *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781, 2013.
2. Stamatatos, E., et al., *Overview of the Author Identification Task at PAN 2015.*
3. Stamatatos, E., et al., *Clustering by Authorship Within and Across Documents.* CEUR Workshop Proceedings, 2016. **Working Notes Papers of the CLEF 2016 Evaluation Labs**.
4. Bagnall, D., *Author identification using multi-headed recurrent neural networks.* arXiv preprint arXiv:1506.04891, 2015.
5. Comaniciu, D. and P. Meer, *Mean shift: A robust approach toward feature space analysis.* IEEE Transactions on pattern analysis and machine intelligence, 2002. **24**(5): p. 603-619.
6. Holmes, D.I. and R.S. Forsyth, *The Federalist revisited: New directions in authorship attribution.* Literary and Linguistic Computing, 1995. **10**(2): p. 111-127.
7. Berry, D. and E. Sazonov. *Clustering technical documents by stylistic features for authorship analysis*. in *SoutheastCon 2015*. 2015. IEEE.
8. Hochreiter, S. and J. Schmidhuber, *Long short-term memory.* Neural computation, 1997. **9**(8): p. 1735-1780.
9. Potthast, M., et al. *Improving the Reproducibility of PAN's Shared Tasks*. in *International Conference of the Cross-Language Evaluation Forum for European Languages*. 2014. Springer.
10. Rosales-Méndez, H. and Y. Ramírez-Cruz. *CICE-BCubed: A new evaluation measure for overlapping clustering algorithms*. in *Iberoamerican Congress on Pattern Recognition*. 2013. Springer.

Appendix I: Scripts

Processor.py

Script for preprocessing data and sending it to network or clusterers.

```
import os, re, sys, ujson, unicodedata, json
from keras.preprocessing import text, sequence
from NNcompare import sharedNN, embedNN
from itertools import combinations,permutations
from collections import defaultdict, Counter
from progressbar import ProgressBar
from clusterer import KMclusterer, MSclusterer
from sklearn.feature_extraction.text import CountVectorizer
import numpy as np
import time


def preprop(token,greek):
        # Replace all quotes to a single one
        token = token.replace("`", '"').replace("′", '"').replace('‶',
'"').replace('″', '"').replace("'", '"').replace("'",




        '"').replace(
                        '"', '"').replace('¨', '"').replace("´", "'").replace("`", "'")
        # Replace apostrophes with standard ones
        token = token.replace("′", "'").replace("'", "'")
        # Replace commas
        token = token.replace('、', ',').replace('،', ',')
        # Replace dashes
        token = token.replace('—', '-').replace('–', '-').replace('—', '-
').replace('—', '-')
        # Replace ellipsis
        token = token.replace('. . .', '…').replace('...', '…').replace('⋯', '…')
        # Normalize according to paper
        #token = unicode(token, "utf-8")
        #token = unicodedata.normalize('NFKD', token)
        # Remove additional whitespace
        token = re.sub('\s+', ' ', token).strip()
        # Replace numbers with 7
        token = re.sub('\d+', '7', token)
        # If language is greek, change latin token for placeholder (s for convenience)
        if greek:
                        token = re.sub('[a-zA-Z]','s',token)
        return token

def main():
        scoreList = [0.0,0.0]
        with open('data/info.json') as j:
                info = ujson.load(j)
        for problem in os.listdir('data'):
                greek=False
                if problem.startswith('problem'):
                        truthPath = 'data/truth/'+problem+'/clustering.json'
                        with open(truthPath) as t:
                                truth = ujson.load(t)
                        print(problem)
                        probTokList = []
                        docList = []
                        docDict = {}
                        X=[]
                        Y=[]

                        path = 'data/' + problem
                        for entry in info:
                                if entry["folder"] == problem:
                                        lang=entry["language"]
                                        if entry["language"] == "gr":
                                                greek=True

                        CV = CountVectorizer(input='filename', strip_accents='unicode',
analyzer='word', ngram_range=(1,4))
                        docs = [path+'/'+x for x in os.listdir(path)]
                        cMatrix = CV.fit_transform(docs)
                        for doc in os.listdir(path):
                                docTokList = []
                                with open(path + '/' + doc) as d:
                                                article = d.readlines()
                                                for sent in article:
                                                        sentTokList = []
                                                        for word in sent.split():
                                                                for token in word:
                                                                        procToken =
preprop(token,greek)

        sentTokList.append(procToken) #Every item of the list is a normalized character
```

```
                                                        docTokList.append('
'.join(sentTokList))#Every item of the list is a sentence
                                probTokList.append(' '.join(docTokList))#Every item of
the list is a document
                                docList.append(doc)
                        tokenizer =
text.Tokenizer(nb_words=None,filters=text.base_filter(),lower=True,split=" ")
                        tokenizer.fit_on_texts(probTokList)
                        seqList = tokenizer.texts_to_sequences(probTokList)

                        uniqueTokens = max([max(x) for x in seqList])

                        print(uniqueTokens,lang)
                        sampling_table = sequence.make_sampling_table(uniqueTokens+1)
                        for i,seq in enumerate(seqList):
                                x, y = sequence.skipgrams(seq, uniqueTokens,
window_size=4, negative_samples=1.0, categorical=False, sampling_table=sampling_table)
                                x = zip(x, y)
                                X.append(x)
                                #Y.extend(y)
                                docDict[docList[i]] = seq
                        strX=[str(x) for x in X]
                        xTokenizer =
text.Tokenizer(nb_words=None,filters=text.base_filter(),lower=True,split=" ")
                        xTokenizer.fit_on_texts(strX)
                        #docMatrix = tokenizer.sequences_to_matrix(seqList,mode="tfidf")
                        docMatrix = xTokenizer.sequences_to_matrix(strX,mode="tfidf")
                        #scores = embedNN(X,Y)
                        pairs = combinations(docDict.keys(),2)
                        cList = []
                        nnDict = {}
                        for cluster in truth:
                                cPairs = []
                                if len(cluster) > 1:
                                        for item in cluster:
                                                cPairs.append(str(item["document"]))
                                        cList.extend(list(permutations(cPairs,2)))
                        for pair in pairs:
                                match = False
                                if pair in cList:
                                        match = True
                                nnDict[pair] = match
                        for i, doc in enumerate(docMatrix):
                                docDict[docList[i]] = doc
                        truthCounter =  Counter(nnDict.values())
                        baseline = 1-float(truthCounter[True])/float(len(nnDict))
                        print("Baseline for {} is {}".format(problem, baseline))
                        clusterCount = Counter()
                        kmclusters = False # Change to False for meanshift
                        if kmclusters:
                                pbar = ProgressBar()
                                for nclusters in pbar(reversed(range(len(docMatrix)-
1))):
                                        #print("{} Clusters".format(nclusters+1))
                                        clusters = KMclusterer(nclusters+1,cMatrix)
                                        for c in range(nclusters+1):
                                                #print(c,"has:",[i for i,x in
enumerate(clusters) if x == c])
                                                for clusterpair in list(combinations([i
for i,x in enumerate(clusters) if x == c],2)):
                                                        combo =
(docList[clusterpair[0]],docList[clusterpair[1]])
                                                        clusterCount[combo] +=1
                        else:
                                clusters =
KMclusterer(int(len(docMatrix)*0.67),docMatrix)
                                #clusters = MSclusterer(cMatrix)#cMatrixdocMatrix
                                for clusterpair in list(combinations([i for i,x in
enumerate(clusters)],2)):
                                        combo =
(docList[clusterpair[0]],docList[clusterpair[1]])
                                        clusterCount[combo] +=1

                        x = 0.0
                        scoreList[0] += truthCounter[True]
                        deleteList = []
                        #print("Most common cluster is in
{}%".format((float(clusterCount.most_common(20)[19][1])/len(docMatrix))*100))
                        for combo in nnDict.keys():
                                if combo not in clusterCount.keys():
                                        deleteList.append(combo)
                        y = 0.0
                        for item in deleteList:
                                if item in cList:
                                        y+=1
                                del nnDict[item]
                        scores = sharedNN(docDict, nnDict)
                        print("Deleted pairs are {}% of total correct pairs, {}% of
deleted pairs was wrongly deleted".format(round(y/len(cList)*100.0,2),
round(y/len(deleteList)*100.0,2)))

                        for combo in clusterCount.most_common(20):
                                if combo[0] in cList:
                                        x += 1
                                        scoreList[1] += 1
                        print("prec: {}".format(x/20))
                        #print("Document score is {} clusters correct out of {}
(accuracy {})".format(x, truthCounter[True], x/truthCounter[True]))
                        #print("prec: {} \nrec: {}".format(x/20,
x/len(nnDict.values())))

        #print("Total precision  is {}, {} clusters
correct".format(scoreList[1]/scoreList[0], scoreList[1]))


                        if not os.path.exists('answers/'+problem):
                                os.mkdir('answers/'+problem)
                        clusDict = defaultdict(list)
                        rankDict = defaultdict(list)
                        for i, cluster in enumerate(list(clusters)):
                                clusDict[cluster] .append({"document": docList[i]})
                                rankDict[cluster] .append(docList[i])
                        with open('answers/'+problem+'/clustering.json', "w") as
jsonFile:
                                ujson.dump(list(clusDict.values()), jsonFile, indent=4)
                        rankList = []
                        for value in rankDict.values():
                                if len(value) > 1 :
                                        pairs = combinations(value,2)
                                for pair in pairs:
                                        rankList.append({"document1": pair[0],
"document2": pair[1], "score":  scores[pair][0]})
                        with open('answers/'+problem+'/ranking.json', "w") as jsonFile:
                                ujson.dump(rankList, jsonFile, indent=4)
```

Clusterer.py

Script for clustering with either K-means or Meanshift.

```python
# !/usr/bin/env python3.5
# coding=utf-8
import numpy as np
from sklearn.cluster import KMeans, MeanShift, estimate_bandwidth

def KMclusterer(nclusters,X):
        cls = KMeans(n_clusters=nclusters, init='k-means++', n_init=10, max_iter=300,
tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True,
n_jobs=-1)
        cls.fit_predict(X)
        return cls.labels_

def MSclusterer(X):
        X = X.toarray()
        bandwidth = estimate_bandwidth(X, quantile=0.04, n_samples=500)
        ms = MeanShift(bandwidth=bandwidth, bin_seeding=False, cluster_all=False)
        ms.fit(X)
        labels = ms.labels_
        labels_unique = np.unique(labels)
        n_clusters_ = len(labels_unique)
        print(n_clusters_)
        return ms.labels_
```

Nncompare.py

Script for the neural network, with either a merged network or for calculating embeddings.

```
# !/usr/bin/env python3.5
# coding=utf-8
from keras.models import Sequential, Model
from keras.layers import Input, LSTM, Dense, merge, Reshape
import numpy as np
from keras.utils import np_utils
from keras.preprocessing.text import Tokenizer
from collections import Counter


def sharedNN(docDict, nnDict):
        pairDict = {}
        keyList = []
        Y = [np.bool_(y) for y in list(nnDict.values())]
        X = []

        for key in nnDict.keys():
                keyList.append(key)
                docTuple = (docDict[key[0]], docDict[key[1]])
                doc1 = docTuple[0].reshape(1, len(docTuple[0]))
                doc2 = docTuple[1].reshape(1, len(docTuple[1]))

                X.append(np.vstack((doc1, doc2)))

        #import pdb; pdb.set_trace()

        X = np.asarray([i[0] for i in X], dtype=np.float32)
        Y = np.asarray([[0, 1] if i == True else [1, 0] for i in Y], dtype=np.int32)

        print(X.shape)
        print(Y.shape)

        model = Sequential()
        model.add(Dense(64, input_shape=(X.shape[1], ), activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
        model.fit(X, Y, nb_epoch=20, verbose=2)
        #import pdb; pdb.set_trace()
        pred = model.predict(X, batch_size=len(X), verbose=0)
        for key in enumerate(keyList):
                pairDict[key] = list(pred)[i]
        return pairDict

def embedNN(X,Y):
        X = np.asarray([i for i in X], dtype=np.float32)
        Y = np.asarray([i for i in Y], dtype=np.int32)
        # print(X.shape)
        # print(Y.shape)

        model = Sequential()
        model.add(Dense(64, input_shape=(X.shape[1], ), activation='relu'))
        model.add(Dense(2, activation='softmax'))
        model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
        model.fit(X, Y, nb_epoch=1,verbose=2)
```