Authorship attribution and link ranking using a clustering/neural network sandwich

A system for the PAN 2016 shared task of author identification

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 (Mikolov, Chen, Corrado, & Dean, 2013) 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 teams. Based on only character based preprocessed data, as this lead to the best result in the PAN task of 2015 (Stamatatos et al.), 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. This measure was also applied by the task committee and although the system was not submitted, results of the experiment on the development set could be compared with other participants's results on the test set afterwards (Stamatatos et al., 2016). Of course, this comparison must be interpreted with the difference between the two sets in mind. The second part of the task, the ranking of links within clusters resulted in a mean average precision of 0.014.

Acknowledgements

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 urgent. Not only to prevent plagiarism, but also to prevent texts of unknown authorship from being attributed to an author they do not belong to.

In order to promote development of systems that can do this automatically, the PAN[1] forum, since 2011, organises a shared task regarding authorship identification. In recent years the task focused on author identification; determining whether or not a certain document belongs to a set of known documents of an author. This is a binary decision, as a document either belongs to an author or not, which has reached its boundaries in how well a computer can do it by the cumulative attempts of teams. The best method for accomplishing this turned out to be an artificial neural network that does not look at the texts or words, but the very characters and stylistics used by authors.

The 2016 task is to cluster documents per author, without knowing the number of contributing authors. This poses new unexplored problems, that can not be solved by the solutions of previous years as there are multiple decisions to be made. The number of authors must be established, followed by a grouping per author off all documents he wrote (Figure 1, left).

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, right).

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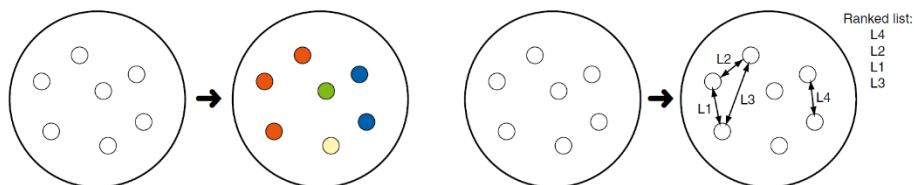
Figure 1: Examples of complete clustering (left) and authorship-link ranking (right). [3]

---

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

1

In this thesis, I wanted to investigate the clustering problem exploiting emerging techniques in NLP[2], namely neural networks. Therefore I formulated and addressed the following research question:

" 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?

---

[2] Natural Language Processing

2 Related work

In this section, several angles to approach the task will be investigated. Although the task of clustering documents based on authors and ranking these links itself is new, various researches in the past have focussed on problems that come together in this shared task. Authorship attribution, clustering documents and one on one comparison of documents are three subsections of this years PAN task, and all will be explained and investigated in this section.

2.1 Authorship attribution

Developing automatic systems to determine whether or not a specific text was written by a certain author or not has been subject to many researches in the past few years. The PAN forum alone already accounts for around ten studies per year for the past five years. During these years the approach on the problem slightly changed, increasing the difficulty of the task.

    The original task was authorship verification; comparing a new document to a set of known documents from one or more authors, as that is the most logical approach if that data is available. The approach and data made it a multiclassification problem, where the system could base its decision on plenty of data per author. This does not make it a good source for the method of this research, because the data per author is not known.

    Despite the method not being relevant for this research, the features for comparing documents are always the same. It is important to only look at the structure of documents and author specific words instead of paying attention to the topic or topic related words. Otherwise the system could be tricked into ranking based on topic while that might not be indicative for authorship. From the overview paper of PAN 2014 (Stamatatos et al., 2014), it is clear that features are to be sought in the linguistic and structural analysis of texts, such as POS-tags and word counts.

    In 2015, participants had to tell the probability of two documents being written by the same author. This task was more difficult then previous years due to the amount of data; only the data within the two documents could be used as input, causing accidental stylistic features to have a too large influence. Interestingly, the 2015's winner did not use a Support Vector Machine (SVM) that used to win these tasks up to then. Instead, an artificial neural network was used to compare the documents (Bagnall, 2015). The features that were selected by this winning team were also new: fully character based instead of words, exploiting sequences of characters without confusing the system with uninformative words.

    This year the task was altered again, but as the 2016 task has two subtasks, the method used by the winning team of PAN 2015 offers an interesting view on the part of ranking the links and is very similar to that. Using multi headed artificial neural networks(Bagnall, 2015), 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. Every single pair gets a score using the method from Bagnall, and pairs that score above a certain threshold then can be grouped together as having likely the same author.

2.2 Clustering

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 (Comaniciu & Meer, 2002), 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 (Holmes & Forsyth, 1995). 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 (Berry & Sazonov, 2015), 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 says 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 Neural networks

Although neural networks seem very new and keep achieving state-of-the art results on well-explored NLP problems, the initial idea dates back to 1944 (Fitch, 1944). Fitch et al. thought of a way to recreate a brain for making mathematical decisions by using artificial neurons that would be activated if a certain calculated threshold was achieved. The reason neural networks only became popular recently is that computer power is becoming larger by the year, making calculations of challenging mathematical problems affordable for most computers and. Also, large amounts of (training)data are freely available. This leads to huge amounts of parallel conducted studies, using well developed API's accessible to every researcher, like Keras (Chollet, 2015).

While neural networks are great for calculating mathematical probabilities, every time it is fed a string of data it decides the outcome only based on that single string. In case of a text, where all words or characters contribute to the whole, context can be very important for decision making. A solution for this is an Long Short-Term Memory, or LSTM, (Hochreiter & Schmidhuber, 1997), a type of recurrent neural network that remembers what it saw before and can take that knowledge as extra input. Instead of making decisions based on data in this 'run' only, an LSTM can exploit information from earlier runs too (Figure 2).

LSTM's have been used in authorship attribution before at PAN, winning the 2015 edition and beating SVM's that used to win in earlier years. Although the method described in 2.1 did win, it came at great computational cost: Bagnall's 2015 approach took over 10 hours for most of the tasks, where other systems did in in mere minutes. Adding an extra stripping threshold could definitely provide a solution for this disadvantage of the method.
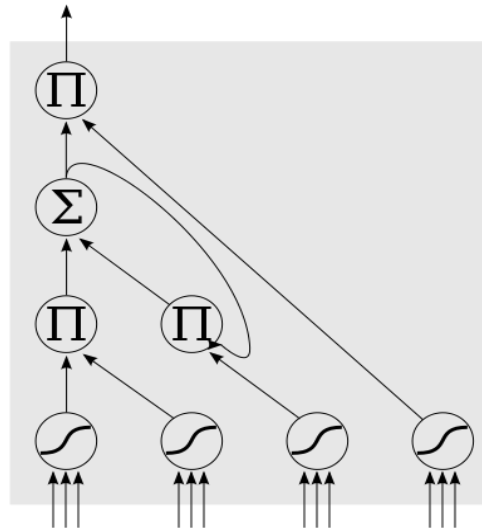

Figure 2: Representation of an LSTM network, where the network can look back to influence the decision. (Source: Wikipedia on LSTM's)

3 Data and approach

This section covers the structure of the data provided by the task committee, and explores how an approach towards solving the task could be constructed. Furthermore, the task itself and the way it should be evaluated are described.

3.1 Data

To properly evaluate and review a method to tackle the authorship clustering problem, it must be applied to a wide variety of data, covering all kinders of texts and multiple languages. The data was provided by the task committee, and consisted of development and test data 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 and Table 2).

Every problem also has an R of either 0.5, 0.7 or 0.9. R in indicative for the number of clusters with multiple authors; the lower the R, the higher the chance that the problem has multi-document 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 (Figure 3), and one with gold data for link ranking (Figure 4).

[[{"document": "document0023.txt"}, {"document": "document0024.txt"},

{"document": "document0047.txt"}],

[{"document": "document0025.txt"}]]
Figure 3: Example of JSON truth file for clustering, one cluster has three documents, the other has one.

[{"document1": "document0023.txt", "document2":"document0024.txt", "score": 0.93},

{"document1": "document0023.txt", "document2": "document0047.txt", "score": 0.84},

{"document1": "document0024.txt", "document2": "document0047.txt", "score": 1}]
Figure 4: Example of JSON truth file for link ranking, with the certainty of the link per document pair in a cluster.

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: Structure of the development dataset

ID	Language	Genre	r	N	k	Links	max C	Words
001	English	articles	0.70	50	35	26	5	752.3
002	English	articles	0.50	50	25	75	9	756.2
003	English	articles	0.86	50	43	8	3	744.7
004	English	reviews	0.69	80	55	36	4	977.8
005	English	reviews	0.88	80	70	12	3	1,089.7
006	English	reviews	0.50	80	40	65	5	1,029.4
007	Dutch	articles	0.89	57	51	7	3	1,074.7
008	Dutch	articles	0.49	57	28	76	7	1,321.9
009	Dutch	articles	0.70	57	40	30	4	1,014.8
010	Dutch	reviews	0.54	100	54	77	4	128.2
011	Dutch	reviews	0.67	100	67	46	4	134.9
012	Dutch	reviews	0.91	100	91	10	3	125.3
013	Greek	articles	0.51	55	28	38	4	748.9
014	Greek	articles	0.69	55	38	25	5	741.6
015	Greek	articles	0.87	55	48	8	3	726.8
016	Greek	reviews	0.91	55	50	6	3	523.4
017	Greek	reviews	0.51	55	28	55	8	633.9
018	Greek	reviews	0.73	55	40	19	3	562.9

r = ratio of clusters with multiple documents, max C = maximum number of documents in a single cluster.

Table 2: Structure of the testing dataset (data is not freely accesible)

ID	Language	Genre	r	N	k	Links	max C	Words
001	English	articles	0.71	70	50	33	5	582.4
002	English	articles	0.50	70	35	113	8	587.3
003	English	articles	0.91	70	64	7	3	579.8
004	English	reviews	0.73	80	58	30	4	1,011.2
005	English	reviews	0.90	80	72	10	3	1,030.4
006	English	reviews	0.53	80	42	68	5	1,003.7
007	Dutch	articles	0.74	57	42	24	4	1,172.1
008	Dutch	articles	0.88	57	50	8	3	1,178.4
009	Dutch	articles	0.53	57	30	65	7	945.2
010	Dutch	reviews	0.88	100	88	16	4	151.7
011	Dutch	reviews	0.51	100	51	76	4	150.3
012	Dutch	reviews	0.71	100	71	37	4	155.9
013	Greek	articles	0.71	70	50	24	4	720.5
014	Greek	articles	0.50	70	35	52	4	750.3
015	Greek	articles	0.89	70	62	9	3	737.6
016	Greek	reviews	0.73	70	51	24	4	434.8
017	Greek	reviews	0.91	70	64	7	3	428.0
018	Greek	reviews	0.53	70	37	44	4	536.9

3.2 Task

The task at hand can be seen as two separate, but related subtasks:
  1. Clustering documents by author (Figure 5).
  2. Ranking links between documents within clusters (Figure 6).
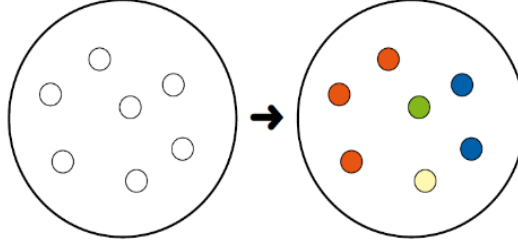
Figure 5: Subtask 1; Clustering documents by author.
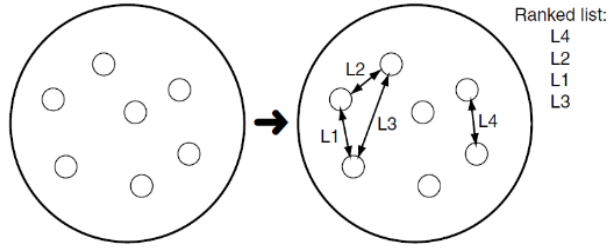
Figure 6: Subtask 2; Ranking links between documents within clusters.

Despite similiraties between the two tasks, both ask for a different approach as will be explained in the next section. The output of both tasks will be of a comparative format, leading to the idea that the output of step 2 might be higly informative for step 1. The two tasks ordered by the task committee do not have to be executed in that order, so a similarity score for every document pair would make a great feature for clustering. Another advantage of designing and building the system as a pipeline, is that loading and preprocessing the data only has to happen once. This leads to a very clear dataflow, as will be explained later in this chapter.

3.3 Method

Based on the earlier work and proven concepts, a combined approach will be investigated. 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[3] provides an excellent API for several clustering algorithms and preprocessing steps, and Keras[4] allows for building an artificial neural network on either the TensorFlow or the Theano backend.

3.3.1 Pipeline

With the two subtasks being closely related, the assumption is that clustering and an artificial neural network can inform each other in order to achieve better results on the data. In Figure 8 a pipeline is reported that shows the exact dataflow throughout the software. In this section, a step by step guide for Figure 8, starting when all preprocessing has been done, will be provided.

The idea of Bagnall (Bagnall, 2015) to use a neural network for ranking document similarity is very promising but also very time consuming on this number of document pairs. For this PAN task, 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 (6). 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 (7). That particular pair can then be removed from the full set (8).

Figure 7 shows the output of two iterations, indicating for every of the 54 documents in what cluster it belongs. The clusters are numbered randomly, so cluster 1 can be a different cluster every iteration. In Figure 7, cluster 1 in the top is suddenly cluster 4 in the bottom, but still contains most of the same documents (except for the third document that now belongs to the new formed cluster 11). Using this approach, some very frequent clusters appeared. The most common clustered combination appeared in 80% of the iterations as cluster, while the number 20 and number 50 appeared together in 60% and 40%, respectively.

---

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

```
[7 3 4 4 8 1 2 1 1 6 0 1 2 5 7 0 6 6 2 5 7 3 8 1 6 3 0 8 4 10
1 0 3 1 0 6 8 2 8 6 2 1 5 9 6 7 2 2 4 3 6 8 8 7 0]

[5 8 11 5 1 4 5 4 4 0 6 4 0 10 5 6 7 7 7 10 5 8 1 4 7 8 6 1
11 9 4 6 3 4 7 7 10 2 1 0 1 7 8 7 7 5 6 0 11 4 0 1 1 2 6]
```
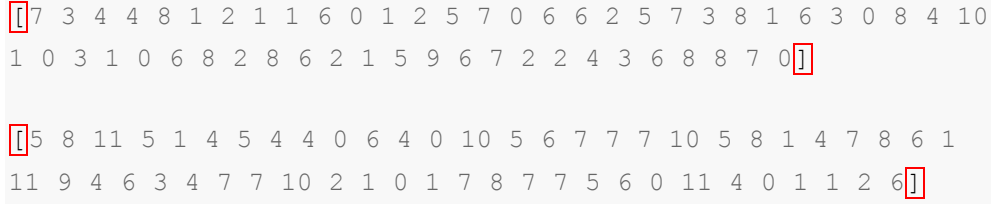
Figure 7: Result of two iterations of K-Means clustering. 54 Documents are grouped first in 10 clusters, then in 11. Note how cluster 1 in the first exampled is labeled cluster 4 in the second.

Once the set of document pairs has been trimmed, the remaining texts can be processed (9) to be fed into Keras' s Long Short-Term Memory, or LSTM, neural network (Hochreiter & Schmidhuber, 1997) (10). 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 past correct pair it has seen 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˙steps, input˙dim]. Sequences is defined by the total number of sequences (sentences, words or characters), steps by the length you want the network to 'remember' and input dimension by the total of different characters in the vocabulary (the total alphabet for a language and some special characters). Both nb˙sequences and input˙dim are defined by the data, while nb˙steps is an arbitrary number that can be experimented with for better results. A too small nb˙steps might result in the network forgetting about the former, while a too large nb˙steps could result in generalization due to a too large amount of data in the memory of the network.

The output of the LSTM, as ratings of one document versus every other (11), one of the two subtasks, will be used as an additional feature (12) for the Meanshift algorithm (13). 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 (14) as is requested for the first subtask, that can be transformed in the same JSON format the task committee provides the truth data in (15).
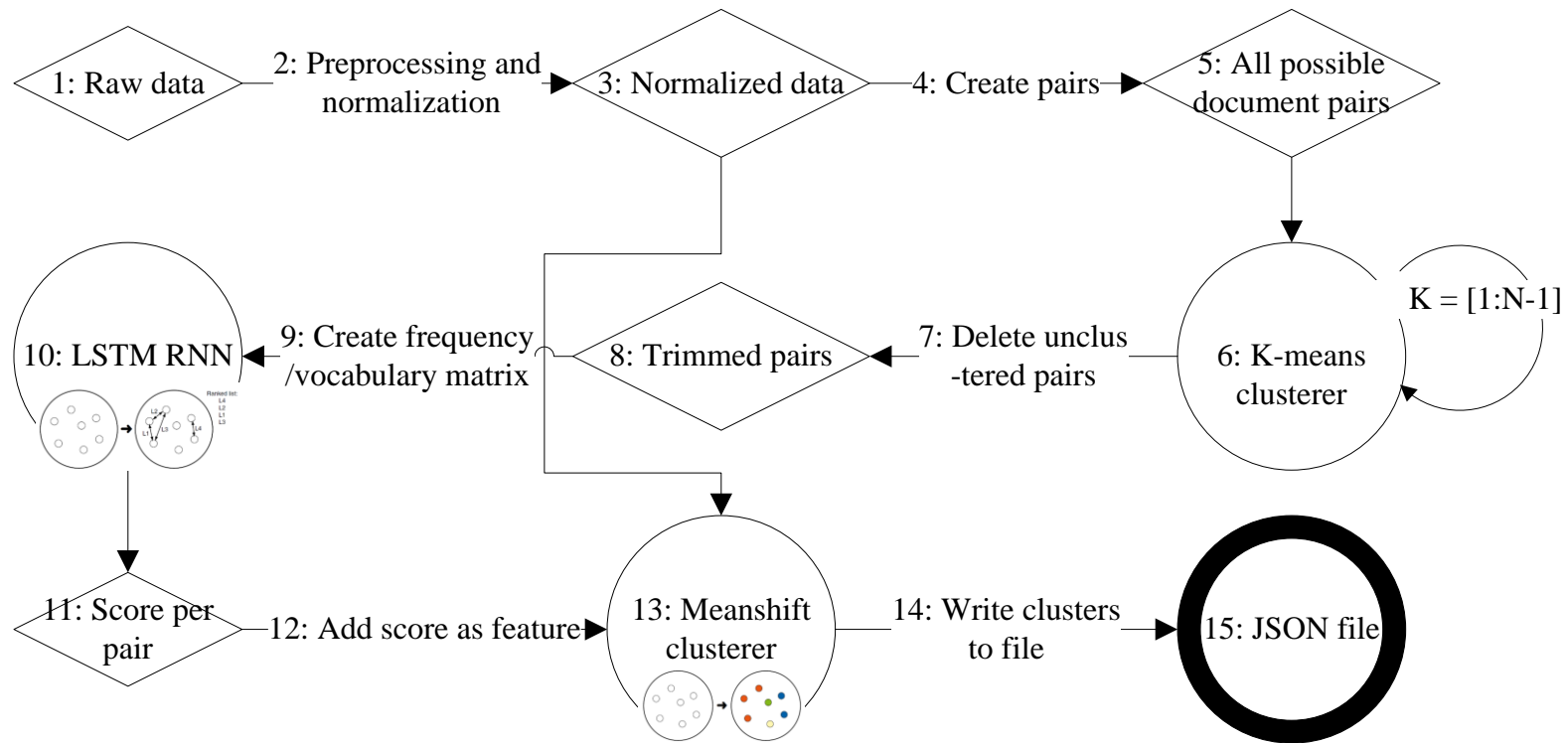
Figure 8: Pipeline of the system with the clustering and ranking diagrams from Figure 1.

3.3.2 Preprocessing and features

In order to train the system, the most promising features must be selected and applied. As the approach will be based on Bagnall 2015's, his feature selection also has been investigated 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.

At the character level one can apply almost all techniques used on words, like the relative and absolute frequency of characters per document and the full corpus (tf-idf). Additional information 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 (2).

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 (3). 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 9).

Figure 9: Different unicode forms and their output.

At this moment, the data is ready for usage in the two clustering algorithms, but not yet for the neural network. For this, the data 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 final feature, character embeddings will be constructed using skipgrams (Mikolov et al., 2013). Embeddings of either word, characters or even bytes are a way of representing a datastring as a vector, and placing it near , or embedding it inbetween, similar datastrings in a multidimensional space. For example, a word like 'cat' can be placed near 'dog', as they are both pets, in one dimension, but next to 'lion' in another dimension. This way the system knows what words are like 'cat', but it also knows that 'lion' and 'dog' are not that close to each other.

Skipgrams is a way of constructing embeddings making use of the context of every datastring. 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.

3.3.3 Evaluation

Using the online review environment Tira (Potthast et al., 2014), automatic evaluation of the answers given versus the gold standard data will be done.
Tira is a system for mass-evaluation of software that offers participants of shared tasks an online platform to upload their program to. The platform runs the software on a dataset chosen by but invisible to the user. This way, test datasets do not have to be published and can be used again with no possibility of cheating. Evaluation scripts are provided by the the task committee, and can not be altered or seen by participants. Although shared tasks mostly have no prize money, it is only fair to give the organizers as much control over the equality of submissions as possible.

Evaluation results will be according to the Bcubed score (Rosales-Méndez & Ramírez-Cruz, 2013), a measure that combines the scores within a cluster with the ones across clusters for computing precision and recall (Figure 10). 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. Bcubed was chosen as evaluation because it has some advantages over other measures: it calculates for cluster homogeneity, cluster completeness, and the rag bag criterion (where multiple unrelated items are merged into a single cluster)(Stamatatos et al., 2016).
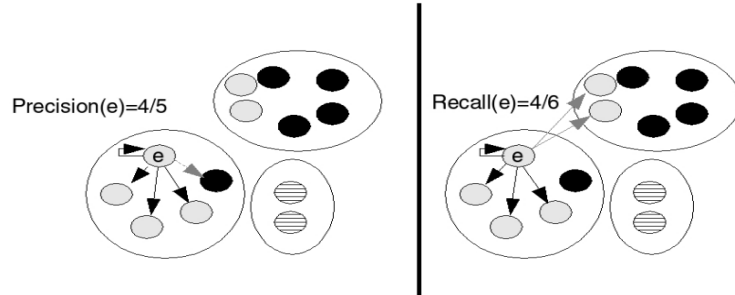
Figure 10: Calculation of Bcubed precision and recall.

Baselines

In the evaluation results will be compared to three baselines set by the task committee, two for clustering and one for ranking. Because most of the authors have only one document in their set, a Singleton baseline that puts every document in its own cluster is fairly high, but only for a (very) similar dataset. Next, a random baseline guesses both the number of authors and the number of documents per author. This baseline also has to deal with ranking links, as there can be more documents per author. This is also done randomly. The metric is iterated 50 times over a problem, and the average of those scores is taken as baseline.Finally, the baseline for ranking is calculating cosine similarity for document pairs. Cosine similarity is a measure for the difference between two vectorized texts, representing the text in numbers. This baseline looks only at structural similarity, and is therefore sensitive for documents with the same topic, and not the same author.

4 Experiment

This section covers all sides of the experiment, from challenges during development to the actual results of the software, and a reflection on those results.

4.1 Development

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.

4.2 Results

Implementing K-Means with K=2/3(N) worked out well in F-score (Table 3) It is a highly unstable method on unseen testdata however, as it assumes that the shape will always be the same as that of the development data. The good results in Table 3 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 3: 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 (Stamatatos et al., 2016) of the 2016 shared task, the evaluation script and other results also were made available for comparison (Table 4). Note that the results in the table are presumably on the test set, while our results are on the development set. The organizers have not released the test set for evaluation, as it can be used in future tasks. As seen before the two datasets are very alike (Table 1 and Table 2), 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 described on page 14.

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 4: Final results of the PAN 2016 shared task for author clustering. (Stamatatos et al., 2016)

Participant	Complete clustering			Link ranking	Runtime
	B3 F	B3 rec.	B3 prec.	MAP	
Bagnall	**0.822**	0.726	0.977	**0.169**	63:03:59
Gobeill	0.706	0.767	0.737	0.115	00:00:39
Kocher	**0.822**	0.722	**0.982**	0.054	00:01:51
Kuttichira	0.588	0.720	0.512	0.001	00:00:42
Mansoorizadeh *et al.*	0.401	0.822	0.280	0.009	00:00:17
Sari & Stevenson	0.795	0.733	0.893	0.040	00:07:48
Vartapetiance & Gillam	0.234	**0.935**	0.195	0.012	03:03:13
Zmiycharov *et al.*	0.768	0.716	0.852	0.003	01:22:56
BASELINE-Random	0.667	0.714	0.641	0.002	–
BASELINE-Singleton	0.821	0.711	**1.000**	–	–
BASELINE-Cosine	–	–	–	0.060	–
Louwaars MS[5]	0.376	0.877	0.256	0.014	00:03:28

Detailed results per genre and language for clustering can be found in Table 5. 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 5: Evaluation results (mean BCubed F-score) for the complete author clustering task. (Stamatatos et al., 2016)

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
Vartapetiance & Gillam	0.234	0.284	0.183	0.057	0.595	0.049	0.230	0.241	0.230
Louwaars MS[5]	0.376	0.386	0.367	0.270	0.465	0.394	0.356	0.376	0.397

Table 6 shows the results for authorship ranking 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 6 also shows the same rising scores as Table 5 with a descending r, but for the ranking all other teams show the same behavior.

Table 6: Evaluation results (MAP) for the authorship-link ranking task. (Stamatatos et al., 2016)

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
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
Louwaars MS[5]	0.014	0.015	0.013	0.016	0.007	0.018	0.004	0.01	0.027

[5] Results were based on the development set instead of the test set

18

4.3   Reflection

The results presented above 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 could 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. Whether or not this is of any influence is highly speculative however, and should be determined by another study to make an actual conclusion.

R values, the probability of a problem having clusters with multiple documents, 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 number of clusters (K). Finding a way to increase the number of 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 noted, 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.

5   Conclusion

Looking back at working on this shared task, one of the major downsides was that, at the deadline for software submission, the system was not yet finished. 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 the shape of the final system.

Approaching the problem with two clustering methods, K-Means and Meanshift, as input and output of an LSTM recurrent neural network seemed very promising based on earlier results. Meanshift is designed exactly for the purpose of clustering without knowing the number of desired clusters, making it an ideal candidate for this assignment.

K-Means is a much used clustering method, but needs a number of clusters as argument to be able to work. By iterating K-Means over all possible cluster numbers, some popular clusters should stand out to the others, while pairs of documents that never appear together would not be likely to form a cluster, and could be removed.

Using an LSTM for document structure comparison is a new method, used only once (Bagnall, 2015), but was an instant success on PAN 2015. This is underlined by the fact that Bagnall's (winning) system of this year (Stamatatos et al., 2016) is based on his submission and suggestions of last year. Unfortunately, at the time of writing this thesis, the shared tasks individual papers were not released yet, so it is not possible to read the details of 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. Note that the winning system of PAN 2016 also used an LSTM, with a processing time of over 60 hours (Stamatatos et al., 2016).

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.

Finally, the main contribution of this thesis can be found in the development pipeline. The idea of chaining multiple machine learning algorithms together is not new ofcourse, but usually such a pipeline is only used to add more and more features towards the end. The addition of a threshold to allow less data in the system works well for both making the data less skewed and pushing back processing time.

Bibliography

Bagnall, D. (2015). Author identification using multi-headed recurrent neural networks. *arXiv preprint arXiv:1506.04891.*

Berry, D., & Sazonov, E. (2015). *Clustering technical documents by stylistic features for authorship analysis.* Paper presented at the SoutheastCon 2015.

Chollet, F. (2015). Keras. *GitHub repository: https://github. com/fchollet/keras.*

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence, 24*(5), 603-619.

Fitch, F. B. (1944). McCulloch Warren S. and Pitts Walter. A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic, 9*(02), 49-50.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

Holmes, D. I., & Forsyth, R. S. (1995). The Federalist revisited: New directions in authorship attribution. *Literary and Linguistic Computing, 10*(2), 111-127.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Potthast, M., Gollub, T., Rangel, F., Rosso, P., Stamatatos, E., & Stein, B. (2014). *Improving the Reproducibility of PAN' s Shared Tasks.* Paper presented at the International Conference of the Cross-Language Evaluation Forum for European Languages.

Rosales-Méndez, H., & Ramírez-Cruz, Y. (2013). *CICE-BCubed: A new evaluation measure for overlapping clustering algorithms.* Paper presented at the Iberoamerican Congress on Pattern Recognition.

Stamatatos, E., Daelemans, W., Verhoeven, B., Juola, P., López-López, A., Potthast, M., & Stein, B. Overview of the Author Identification Task at PAN 2015.

Stamatatos, E., Daelemans, W., Verhoeven, B., Stein, B., Potthast, M., Juola, P., Barrón-Cedeño, A. (2014). *Overview of the Author Identification Task at PAN 2014.* Paper presented at the CLEF (Working Notes).

Stamatatos, E., Tschuggnall, M., Verhoeven, B., Daelemans, W., Specht, G., Stein, B., & Potthast, M. (2016). Clustering by Authorship Within and Across Documents. *CEUR Workshop Proceedings, Working Notes Papers of the CLEF 2016 Evaluation Labs.*

Appendix I: Scripts

Processor.py

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

```python
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)
```

```python
        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
```

```python
                                                    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
```

```python
                    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)
```

```python
                    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.

```python
# !/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]
```

```python
        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)
```