Anomaly detection in text documents using HTM networks

Zoltán Szoplák, Gabriela Andrejková

Institute of computer science, Faculty of Science
P. J. Šafárik University in Košice
Jesenná 5, 04001 Košice, Slovakia
zoltan.szoplak@student.upjs.sk
gabriela.andrejkova@upjs.sk

CEUR-WS.org/Vol-2962/paper04.pdf

Abstract: Anomalies in texts can be caused mainly by various interventions in texts, such as, by supplementing parts of a text from different authors. Such a type of anomalies can disrupt text that would otherwise be consistent. In order to find anomalies we have combined multiple algorithms, including a non-traditional neural network model - the Hierarchical Temporal Memory (HTM) network. HTM networks are spatiotemporal predictors based on the neocortex that combine the ability to retain memories of time sequences like recurrent neural networks with the spatial representations of convolutional neural networks.

To represent the text inputs for the HTM algortihm we use semantic folding, which encodes text differently than other embedding method: as a collection of contexts they occur in. Alongside such a predictor we use numerous other, more well known metrics, and combine them into a two step algorithm. In the first step we find the division points between the anomalous and non-anomalous parts. In the second step we determine which sections located between two division points are actually anomalous. The algorithm was tested on 40 benchmark texts from PAN plagiarism corpus PAN-PC-11 and the results of determining whether the text contains anomalies or not are 100 %. The percentage of fully detected anomalies is 70.15 %.

Keywords: HTM network; Semantic folding; Text anomaly detection

1 Introduction

Anomalies in texts are certain types of outliers which can be detected in parts of texts different from the rest of the text. Anomaly detection is therefore the task of identifying such parts of text that deviate from the rest of the text to a suspicious degree. In this paper, we are concerned with creating a method capable of detecting anomalous or plagiarized sentences in English language texts.

We have formulated two problems:

T1 *Determining whether the text itself is anomalous (the text contains an anomal part)*

T2 *Determining the number and location of anomalies in text*

Although solving the second problem provides a solution to the first, the first problem is solvable in a shorter time, it is sufficient to detect the first anomaly. For recommender systems, it is often enough to point out that a text does contain anomalous parts and leave the rest of it to manual analysis. Such an approach regarding the dataset in [1] has already been attempted in [2], where the task was to merely determine whether a given text contained any anomalies. The second task is self-explanatory, we aim to detect precisely the chunks of texts that are anomalous. It is important to note that while the algorithm presented does contain a measurement of the degree of anomalousness, we consider sections either fully anomalous or fully anomaly free, as the dataset suggests. The first problem classifies a text as fully anomalous if there is even a single anomaly present, while with the second class we will be labelling individual sections as anomalous or not.

We proposed to experiment with streaming analysis detection taking into account context and narrative progression by analyzing texts sentence by sentence. To implement such analysis, multiple encoding methods were considered. Embedding methods such as Doc2Vec, described in [3] encode the exact word for word composition of the sentence. While useful for many tasks, we wanted a method that would be able to extract and compare the context and topic of a sentence, rather than it's exact wording. We therefore chose to use the Semantic Folding Theory, described in [4] and combined it with the HTM algorithm [5], which is a neural network designed specifically to work with the kinds of representations that the Semantic Folding Theory creates. Since there can be types of anomalies that are not semantic ones or cases where a semantic change is not indicative of an anomaly, we have decided to implement more metrics that are designed to capture syntactic and statistical information as well, supplementing the predictions made by the HTM network as well as comparing their usefulness to the aforementioned method.

The article is conceived as follows: In the second section we discuss the current state of the art. In the third section we describe the "Semantic Folding (SF)" method. In Section 4, we provide a brief description of HTM networks. Section 5 is devoted to the description of our new algorithm for finding anomalies. The results are processed in the sixth section while the seventh section provides a conclusion.

Copyright ©2021 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2 Related works

Anomalies can be considered a kind of outlier. General methods for finding outliers and anomalies such as those in Argavall [6] and Chandola [7] are also useful for finding anomalies in texts, but texts are a special type of data for which special methods can be used.

Zhuang et al. [8] developed a generative model to identify frequent and characteristic semantic regions in the word embedding space to represent the given corpus, and a robust outlierness measure which is resistant to noisy content in documents. Experiments conducted on two real-world textual data sets showed that the method can achieve very strong improvement over outlier ranking.

In Kannan et al. [9] a matrix factorization method is presented, which is naturally able to distinguish the anomalies using low rank approximations of the underlying texts.

Young et al. [16] review significant deep learning related models and methods that have been used for numerous NLP tasks. They also summarize, compare and contrast the various models and put forward a detailed understanding of the past, present and future of deep learning in NLP.

Recently, several articles have been published on the search for anomalies using HTM networks, described in Hawkins et al. [10, 11]. Important publications on the use of HTM networks in finding anomalies include the paper of Ahmad and Purdy [12]. They presented a novel HTM based on-line sequence memory anomaly detection technique for time-series data. They demonstrated impressive results from a live application that detects anomalies in financial metrics in real time.

In another article Ahmad et al. [13], it is proposed a novel anomaly detection algorithm that works on streaming data. The technique is based on an online sequence memory algorithm based on HTM. They presented results using the Numenta Anomaly Benchmark (NAB), a benchmark containing real-world data streams with labeled anomalies.

Cui et al. [14] presented a comparative study of HTM networks, a neurally-inspired model, and other feedforward and recurrent artificial neural network models on both artificial and real-world sequence prediction algorithms. They informed that HTM and long-short term memory (LSTM) networks gave the best prediction accuracy. HTM has many other beneficial properties and features that are desirable for real-world sequence learning.

Hole [15] concentrated on understanding how the HTM learning algorithms can detect anomalies in complex adaptive information and communications technology (ICT) systems. HTM finds anomalies in real-time streaming data. There is no need to store huge amounts of data since HTM builds models representing the properties of the data. They examined anomalies in Amazon Web Services (AWS) streaming data and then studied how HTM detects rogue human behavior.

The aforementioned research has inspired us to make use of these networks. To satisfy the input reuqirements of HTM networks, we needed to find a way to encode text data as Sparse Distributed Representations (SDRs).

In natural language processing (NLP), a method called "Semantic Folding (SF)" [4] is important, which allows to store text data as sparse distributed representations.

3 Semantic Folding

Semantic folding theory creates Sparse Distributed Representations (SDRs) of text data called semantic fingerprints to emulate the structure of the neocortex, the area of the brain that is responsible for several high-level cognitive functions, such as vision, hearing, touch, movement or, most relevant to our case: language. A single fingerprint ideally represents contexts and clusters of contexts that are present in a given text. Such representation is achieved by first gathering a corpus representative of the text that we aim to encode, slicing it into snippets (sequences of words, usually paragraphs) and arranging them into a 2D array using self-organizing maps. As a consequence, similar snippets (those that share a lot of words) end up close together, forming clusters.

After creating the array of our representation, we can obtain the semantic fingerprint of a word by checking every single context of the array whether it contains the input word or not. We set the given index to 1 in case the word is present in the snippets of the context and 0 otherwise. The result is a sparse vector since most words only occur in a handful of contexts, therefore it is preferable to store only the indices that have a value of 1 to save memory.

Since looking through every single context for each input word is very time-consuming, it is preferable to simply create a vocabulary of words from our corpus, calculate the encodings for each word and store their representation in a database. If we want to encode collections or sequences of words, we simply take the fingerprint of each individual word, concatenate the active bits for every index and then activate only the indices where the number of active bits in all the fingerprints exceeds a certain threshold.

Merging the fingerprints in such a way allows us to retain sparsity and prune the representation of all contexts that may be relevant to the individual words, but are irrelevant in the context they occur in. Such a representation does have a few perks of note. First, every individual bit in our representation has its specific individual meaning, unlike word encodings such as Bag-of-words or ASCII code. Furthermore comparing representations can be as easy as calculating the number of overlapping active bits. We can also take advantage of the fact that similar contexts are clustered and compare representations using metrics reliant on geometry such as Euclidean and cosine distance. While such representation does not take into account the word order and thus is unsuitable for language generation, it is very much suitable for topic matching and preventing semantic drift.

4 Hierarchical Temporal Memory

The human neocortex learns by recognizing patterns of information sequences representing sensory "inputs" and predicts following likely likely values based on previous observations. Hierarchical Temporal Memory (HTM) is a type of neural network that tries to reproduce the structure and processes of the neocortex. More detailed description of HTM is given in [5].

4.1 HTM Network Structure

A HTM network has an inherently bottom-up hierarchical structure, as seen in Figure 1, comprised of multiple layers of 3-dimensional arrays of bits, referred to as cells. Interconnected layers form a hierarchy where each layer is connected to the one below it, with the one at the bottom connecting to the input of the network itself. A layer itself is a 3-dimensional array of bits comprised of columns arranged in a 2D array topology, where a column is made up of a single or multiple cells. Each column is connected to a subset of the input/previous layer, and cells in a single column are also connected to the cells above and below. The hierarchy of layers is inspired by biology, with the neocortex consisting of multiple regions that either receive their input directly from sensory organs or from other regions connected to them.

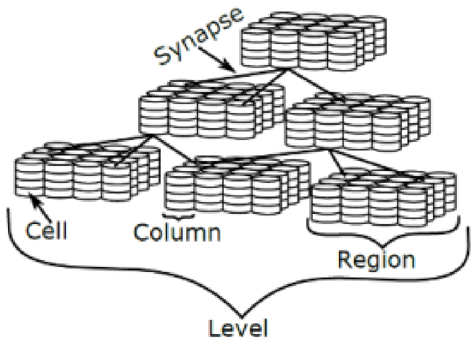

Figure 1: Structure of HTM Network [5].

The learning algorithm of HTM networks can be divided into two steps [17]:

- Spatial Pooler (SP) creates a Sparse Distributed Representation (SDR) based on the input and can be viewed as a mapping function from the input domain to a new feature domain where the meaning of the input is preserved while ensuring that the representation in the feature domain remains sparse. The algorithm is a type of unsupervised competitive learning algorithm that uses a form of vector quantization resembling self-organizing maps (SOMs).

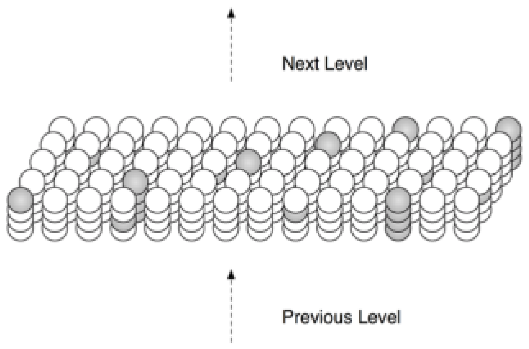

Figure 2: SP - creating SDR of layer based on input [5].

- Temporal Memory (TM) forms connections from the cells active in the current step to cells that were active just prior and makes predictions. The algorithm uses Hebb's rule, where connections are formed between cells that were previously active. Through the formation of those connections a sequence may be learned. The TM can then use its learned knowledge of the sequences to form predictions.

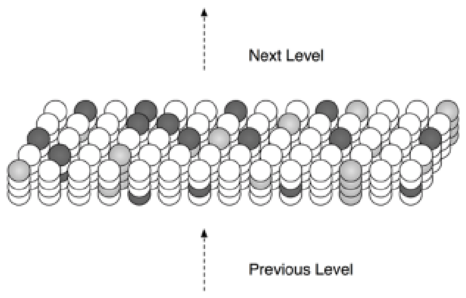

Figure 3: TM - representation of a layer after determining predictive cells (predictive darker cells, white non active cells, grey active cells) in TM. [5].

4.2 Using HTM in our Algorithm

The explicit mathematical description of computations for the HTM network can be found in Mnatzaganian et al. [17]. It describes all aspects of the spatial pooler, a critical learning component in HTM, under a single unifying framework. The primary learning mechanism is explored, where a maximum likelihood estimator for determining the degree of permanence update is proposed.

The HTM algorithm, in essence, allows to take an array of bits as an input and to predict the input of subsequent steps. Such predictions may be enough for some tasks, however, in order to detect anomalies, we need to extract the occurring differences in the patterns presented in the inputs.

Let's define vector x_t as the input generated in our system at time t. Then the sequence of inputs to the algorithm can be defined as $x_1,\ldots,x_{t-1},x_t,x_{t-1},\ldots,x_k,\ldots$ possibly continuing until the detection task is stopped manually.

In general, the goal of streaming anomaly detection is to find abnormalities in inputs as soon as they occur. Such a real time constraint means that for the purposes of an anomaly detection at time t the inputs of earlier times $(1,...,t-1)$ are accessible only, not the input at time $t+1$ or later. The HTM algorithm is capable of detecting such anomalies as well by using a method described in [12].

Let's define two variables to explain the HTM anomaly detection algorithm: Let $a(x_t)$ be the sparse binary representation of the input vector x_t, defined by the binary 2D matrix of all cells within a region, where $a_{i,j}(x_t)$ is set to 1 if the ith cell of the jth column is in the active state and to 0 otherwise. Let $\pi(x_t)$ be the sparse binary representation of the prediction of the next input - $a(x_{t+1})$, defined by the binary 2D matrix of all cells within a region, where $\pi_{i,j}(x_t)$ is set to 1 if the ith cell of the jth column is in the predictive state and to 0 otherwise. The values of the prediction matrix are greatly influenced not just by the input itself, but the context as well.

Therefore the accuracy of the algorithm prediction is largely dependent on its ability to model the data. These two variables are calculated at each step of the algorithm, however, they do not contain sufficient information to find anomalies by themselves. Instead, we use these variables to compute a raw anomaly score for each timestamp, labelled s_t. The raw anomaly score essentially gives us a measurement of a deviation between the predicted and the actual input. The raw anomaly score is given by:

$$s_t = \frac{\pi(x_{t-1})a(x_t)}{|a(x_t)|} \tag{1}$$

Both variables are binary vectors, the multiplication is the inner product, divided by the size of the output. The less the prediction was correct the larger our anomaly score is. The value of s_t is therefore a scalar value between 0 and 1, 0 meaning the prediction was perfect, 1 meaning nothing has been correctly predicted. A weak prediction would therefore be indicative of an anomaly, however it does not take into account the predictive capability of our network on the amount or noise present in the text.

To counteract this, we calculate the distribution of anomaly scores within a certain time window, and therefore find the likelihood instead of simply tresholding the raw anomaly score. The anomaly likelihood metric is designed to measure a change in predictability, rather than change in the input pattern and thus it accounts for the beginning of the text where predictability is low. The metric is ideal for detecting not only the starting points of the anomalies but their ending points as well (since in that case the predictability would suddenly get much more accurate).

To calculate the anomaly likelihood metric, we use a large moving window represented by the vector W that stores the last k raw anomaly scores. In addition, we define the W' window with a size of j which is much smaller than k that is used to calculate a small moving average of the last few anomaly scores. It makes a more accurate comparison metric than a single score. We calculate the anomaly likelihood using the Q-function (the tail distribution function to the Gaussian normal distribution function), where the mean and variance of raw anomaly scores for the normal distribution function are recalculated every time using the values of anomaly scores in our window-sized memory.

The anomaly likelihood metric at time t is defined as the complement of the tail probability:

$$L_t = 1 - Q\left(\frac{\bar{\mu}_t - \mu_t}{\sigma_t}\right) \tag{2}$$

where:

$$Q(x) = \frac{1}{\sqrt{2\pi}} - \int_x^\infty \exp\left(-\frac{u^2}{2}\right) du \tag{3}$$

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{k}, \qquad \bar{\mu}_t = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{j} \tag{4}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{k-1} \tag{5}$$

Naturally, the greater the likelihood score the more probable it is that we have found the offset of an anomaly.

5 Anomaly Detection in Texts

The new developed system combines the previously described methods and works in two steps:

1. **Finding the locations of the text changes.** The first step is to create an algorithm that can find the exact locations where the style of the text changes, whether from non-plagiarized to plagiarized or vice-versa. We consider sentences the smallest unit of text to be analyzed this way as we do not expect sentences that have both anomalous and non-anomalous parts within them. Therefore finding the locations of text changes in practical terms means finding the offsets of the first sentence of the anomalous part and the first sentence of the non-anomalous part.
2. **Filtering out the non-anomalous potential sequences.** The second step is to take the offsets of the first step and filter out the sections located between two offsets that are not actually anomalous.

5.1 Finding the Locations of Text Changes

We have created the following features for the purpose of determining where such points of change lie:

F1. The anomaly likelihood score of each sentence.

F2. The cosine similarity of the Doc2Vec vectors and their autoencoder predictions.

F3. The Euclidean distance between the current fingerprint and the fingerprint of preceding sentences.

F4. The cosine similarity between the current fingerprint and the fingerprint of preceding sentences.

F5. The Jaccard index between the current fingerprint and the fingerprint of preceding sentences.

F6. The average relational frequency of the words contained within a sentence.

F7. The lowest relational frequency of the words contained within a sentence.

F8. The highest relational frequency of the words contained within a sentence.

F9. The difference of the mean relational word frequency of the sentence and the text.

To calculate F1, we use the semantic fingerprint method to create a fingerprint of each sentence and use them as inputs to the HTM network described in Section 3 and then calculate the anomaly likelihood score.

To calculate F2, we use Doc2Vec, an embedding method described in [3] to create a vector representation of each sentence of our text. We then use it as inputs to train an autoencoder to first create an encoding of lesser dimensionality and then reconstruct an output from the code that best matches the input. After training the autoencoder, we calculate the cosine similarity between the input embedding and the reconstructed output embedding. Since most of the text is expected to come from a single source with occasional anomalous parts inserted, the network, through training, creates generalized reconstructions that have more success reconstructing the inputs from the original text than from the anomalous parts.

To calculate F3, F4 and F5, we use semantic fingerprints, comparing the fingerprint of each sentence to the merged fingerprint of the preceding 5 sentences in a window (determined to be the average anomaly length). We use 3 different metrics of comparison: Euclidean distance, cosine similarity and Jaccard distance. The metrics are suited to detecting the first sentence of an anomalous section as well as the first sentence of a non-anomalous section that follows an anomalous section.

To calculate F6, F7 and F8, we use relational frequency metrics described in [18], which measure how specific a current word is to a given segment. We calculate the frequency of the most and least frequent word as well as the mean frequency of all words in a sentence. Anomalous sentences are presumed to have low relational frequency scores due to having words atypical for the rest of the text. A low mean relational frequency means that a sentence contains many unique words. A low highest relational frequency means that unique stopwords are used. A low lowest relational frequency means that a sentence unique word is present (which might not be by itself indicative of an anomaly).

To calculate F9, we use the the mean relative frequency of the author style metric described in [19]. We calculate the frequency of each word in the sentence as well as in the document. We average these values across all of the words and compare the mean value of the sentence frequencies and the document frequencies to get the difference in author styles.

We use a Gradient Boosting Classifer (GBC), described in [20], to combine the predictive capability of the aforementioned features as well as evaluate their relative importance. The GBC is an ensemble of multiple models, specifically decision trees, which has superior prediction capability compare to the individual models. It does so by iteratively adding models to the combined model that minimize the residual loss obtained by the combined model in the previous step.

To label the dataset, we choose a rather simplistic method: We label all sentences as zeroes except the first sentence in each anomaly and the first sentence after the anomaly ends. Our prediction of these division points is then used to make the prediction about where the potential anomalous sections might be.

5.2 Filtering out Non-anomalous Potential Sequences

Now that we have our division points, we still need to identify which sections lying between two points are anomalous and which are not. If we merely tagged them in an alternating fashion, it would lead to a large number of errors, as a single faulty division point could mean that we misclassify our entire dataset. We need some way of determining the anomalous nature of individual sections. We can assume that most anomalous sections are relatively short compared to non-anomalous sections. We can pair up indices of division points that are located close to one another, specifically 50 sentences from each other. While we can create possible pairings of anomalous parts that are located close to one another, there are individual division points that cannot be paired. They might be a false positive or they corresponds to beginning or ending that has not been found yet. For each isolated index, we construct multiple artificial sections that are created varying distances before or after it. The exact process is described in algorithm 1.

Such pairings inevitably lead to false positives, therefore it is vital to devise a method that can recognize genuine anomalous parts.

We use a gradient boosting regressor, to predict the anomalousness of the potential section (the percentage of anomalous sentences from the section).

There are multiple parameters that we use as inputs, some from the previous step and others from modified algorithms that deal with sections instead of sentences. Input values for the regressor are the following:

Algorithm 1: Algorithm for creating potential anomalous sections from division points

```
Data: $P$ - Sorted list of division points, where the
      value of each point is the index of the
      sentence within the document
Result: $S$ - List of potential anomalous sections
$i \leftarrow 1$
while $i \leq length(P)$ do
    if $P[i] - P[i-1] \leq 50$ then
        $S.addSection(P[i-1], P[i])$
    else
        if $P[i+1] - P[i] \leq 50$ then
            $S.addSection(P[i], P[i+1])$
        else
            for $j$ in range(1,10) do
                $S.addSection(P[i] - 5 \times j, P[i])$
                $S.addSection(P[i], P[i] + 5 \times j)$
            end
        end
    end
end
```

- The mean value of F2, F6-F9 for every sentence of the section (metrics using Semantic Folding are not used due to them having high values in using division points).
- The Euclidean distance between the fingerprint of the section and the fingerprint of the entire document.
- The cosine similarity between the fingerprint of the section and the fingerprint of the entire document.
- The Jaccard index between the fingerprint of the section and the fingerprint of the entire document.
- The average relational frequency of the words contained within the entire section.
- The lowest relational frequency of the words contained within the entire section.
- The highest relational frequency of the words contained within the entire section.
- The difference of the mean relational word frequency of the section and the text.

After training our regressor, we treshold the anomalousness values to obtain the list of anomalous sections. Due to the artificial creation of sections, overlaps may be possible. We describe a way to merge these section in algorithm 2.

6 Results

6.1 Data preparation

We worked with the PAN intrinsic anomaly detection plagiarism corpus 2011 [1] as experimental text data. It is a collection of 4753 larger text bodies in the English language, made up of various topics, where some texts have plagiarised parts artificially inserted into them. We have chosen 40 texts from the corpus to test our system.

Algorithm 2: Algorithm for merging overlapping sections

```
Data: $S$ - List of potential anomalous sections that
      are defined by the index of their starting
      sentence and the index of their last sentence
Result: $S$ - List of potential anomalous sections
        without overlaps
$OverlappingSections \leftarrow TRUE$
while $OverlappingSections$ do
    $OverlappingSections \leftarrow FALSE$
    for $i$ in range(0, $length(S) - 1$) do
        for $j$ in range(i, length(S)) do
            if overlaps($S[i]$, $S[j]$) then
                $OverlappingSections \leftarrow TRUE$
                $first = \min(S[i].first, S[j].first)$
                $last = \max(S[i].last, S[j].last)$
                $S.addSection(first, last)$
                $S.removeSection(S[i])$
                $S.removeSection(S[j])$
            end
        end
    end
end
```

For each text, we have two files: `a.txt` file containing the texts themselves and `an.xml` file containing various metadata such as the source of the base text and the author and most importantly: the list of anomalous parts defined by the division points and the length of such a part. The number of plagiarized parts in texts is not constant and can contain 0, with no plagiarism inserted, up to 10 anomalous parts. Anomalous parts are whole sentences or entire paragraphs. Thus we only consider entire sentences or larger collections of sentences as possible anomalies.

First we remove all stop words from the text, such as a, the, is, at, which, on etc., since these characters have little relevance upon the meaning or authorial style of the text. We solved two tasks as follows:

Task 1: To find out if the suspicious text is anomalous: Here, we are only trying to determine whether a text contains any anomalies or not, ignoring the location or the number of anomalies present.

Task 2: To identify the individual anomalous parts within the text correctly, taking into consideration their precise locations.

We first split the sentences using the NLTK (Natural Language Toolkit) tokenizer, described in [21], marking certain words ending with a punctuation mark as exceptions when splitting (such as st., mr., mrs., dr. etc.). We removed also all non-text characters, all stop words as well as words shorter than 3 letters. Finally, we also merged sentences that consist of only a single word with the first non-single word sentence that comes after it. Such a way of merging allows us to avoid a lot of false positives which would occur by having sentences that have very little meaning on their own. We also store which sentences they were merged from and apply the features calculated to all of the sentences.

We then calculate the features for each sentence. For the merged fingerprints of the preceding sentences used for F1, F3, F4 and F5, we have chosen to merge the fingerprints of five sentences that came before the current sentence. In case of the first sentence, we do not have a merged fingerprint, thus we use the current fingerprint as the input to the Temporal Pooler as is. In a case where there are less than 5 sentences available we use a merged fingerprint consisting from sentences that are available. Then, in case of F1, as was described, we create a fingerprint where only those bits have a value of 1 that are active in both the merged print and the current print for each sentence. The fingerprints arrays have a size of 128×128 and use the standard English associative dictionary of `cortical.io`. As the fingerprints are encoded as a collection of active bits, we have a list of sorted indices that are between 1 and 16384.

To increase computational efficiency, we split the list into four parts, each having a portion of the values in the following way: the first list contains all of the values between 1 and 4096, the second between 4097 and 8192, the third between 8193 and 12228 and the fourth between 12229 and 16384. We then create sparse vectors from these lists, by having 1 in an index that is present in the list and 0 everywhere else. Such a division does not pose much of a problem for the prediction capabilities of HTM algorithm, as it has a feature that allows it to separate individual patterns that represents a sequence of inputs that still belongs to a single object.

6.2 HTM Network and Gradient Boosting Classifier Training

We first fit the HTM network using the entire text once as the training set and then run the same inputs through the network to get our predictions. We calculated the anomaly scores and from them the likelihood scores for each sentence of the document. For the moving window of mean anomaly scores, we used a window size of 5 that stores the previous 5 anomaly scores. For F2, we have used Doc2Vec to create 100 dimensional vectors from each sentence. We have chosen a five layered feed-forward perceptron as our autoencoder network, with the layer sizes being (experimentally chosen): 100, 50, 20, 50, 100 with the first and last layer being the same size as the Doc2Vec inputs. We trained the network on the text in 5 epochs (experimentally chosen), then performed the reconstruction for each sentence and calculated the cosine similarity between the input and the reconstruction.

Features F3-F5 are calculated just as described in Section 5, using the five sentences preceding the current sentence to form our merged fingerprint and comparing it to the fingerprint of the current sentence. Features F6-F9 are calculated just as described in Section 5, with calculating the relational frequencies and author style separately for every sentence on a text that did not have short words or stop words removed.

We used these features as the input parameters to our Gradient Boosting Classifier with 200 estimators and a maximum depth of 4. After training our classifier, we can get the positive examples of sentence offsets and construct the potential sections from them. To filter them out, we train a Gradient Bossting regressor with 200 estimators and a depth of 4 to predict their relevance. We then threshold the prediction to get the potential anomalous parts. We experimented with multiple thresholds and found the threshold value of 0.7 as sufficient. Finally, we merge the potential sections and then compare them with the actual anomalous sections using multiple metrics, such as precision, recall and accuracy to measure our success. For evaluation, we consider the anomalies as positive examples and label every single character of the full unmodified text, therefore taking into account the length of sentences as well.

6.3 Experimental results

We have evaluated the predictions of anomalies made by our model and organized the results into Table 1. For comparison, we have implemented the Author style algorithm proposed by Kuznetsov et al. [19] and evaluated it on our dataset as well. The results have also been organized into Table 1, alongside the results of our own algorithm.

The columns of the table are arranged in the following manner: The Txt column corresponds to the id number of the text. The Plag det column tells us how many anomalies did our algorithm detect fully out of the number of plagiarisms present. The T1 (Task 1) column tells us the conclusion our algorithm reached when determining whether the text is anomalous or not (N for non anomalous, Y for anomalous). Columns Pre, Rec and Acc refer respectively to the precision, recall and accuracy values achieved by our algorithm with classifying anomalies. Columns Pre A2, Rec A2 and Acc A2 are the results achieved by [19]. If there are no predicted anomalous parts/sentences, the precision metric has a values N/A since calculating the metric becomes meaningless. Likewise if there are no actual anomalous parts/sentences present in the text, the recall metric similarly has a value of N/A.

As we can see in Table 1, our algorithm has achieved an accuracy of 100 % in T1. Regarding T2, the overall percentage of fully found plagiarisms is 70.15 %. The results show high accuracy due to the far greater number of non-anomalous sections in the actual text as well as high precision due to the number of false positives being further by filtration. The downside of such extensive filtering can be seen in the recall values however, where the results vary. Such a discrepancy between the precision and recall values can be observed with both algorithms. When it comes to accuracy, the results of the author style algorithm have only been better in case of text 11, equal in multiple cases and inferior to ours in most.

When looking at the metrics of precision and recall, they tend to achieve a better, sometimes perfect precision score, but this usually comes at the cost of a significantly worse recall and accuracy score. While precision is important, we believe that for recommender systems, it is better to create a few false positives in order to flag most of the anomalous sections, rather than being more sure with the anomalousness of fewer sections.

As for the performance of the author style algorithm on T1, it failed to achieve 100 % accuracy, due to often predicting sentences to be anomalous when there are no anomalies to be found (such as texts 5 or 10 in Tale 1) or not finding any anomalies despite the text containing them (such as texts 23 or 28). It appears that our results mostly surpass the results generated by the author style algortihm, therefore showing that sequential anomaly detection is worth consideration.

However the disparity in precision is still not ideal, and lovering our threshold did not give us much improvement in recall in comparison to the decrease in precision. We believe this is occuring because of short anomalous sections, because our artificially constructed anomalous sections are at least 5 sentences long, which might be more than the minimum length of said anomalies. We can see that a lot more depends on the division point detection part of the algorithm, therefore we have decided to plot the importance of features for our classifier, showed in Figure 4.

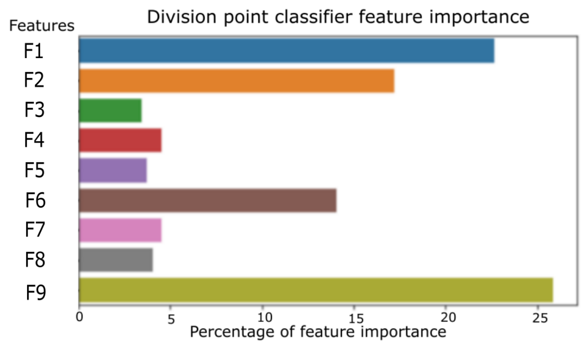

Figure 4: Feature importance of the gradient boosting classifier

As we can see, the most successful feature used was F9, the mean relational word frequency comparison between the whole text and the sentence. The anomaly likelihood is very close, being designed specifically to detect points of change. We can see that using Semantic Folding without a strong predictor like HTM decreases the performance as evidenced by the relative uselessness of the fingerprint comparison metrics (Euclidean, Cosine and Jaccard).

Table 1: *Information about 40 English texts from corpus [1].*

Txt	Plag det	T1	Pre	Rec	Acc	Pre A2	Rec A2	Acc A2
1	0/0	N	N/A	N/A	1.0	0.0	N/A	1.0
2	0/0	N	N/A	N/A	1.0	0.0	N/A	1.0
3	6/8	Y	0.86	0.64	0.98	0.73	0.66	0.97
4	3/4	Y	0.97	0.44	0.98	0.70	0.15	0.96
5	0	N	N/A	N/A	1.0	0.0	N/A	0.98
6	4/5	Y	0.96	0.70	0.98	1.0	0.24	0.96
7	11/16	Y	0.96	0.57	0.94	0.97	0.32	0.90
8	0	N	N/A	N/A	1.0	0.0	N/A	1.0
9	7/13	Y	0.97	0.74	0.94	0.85	0.67	0.91
10	0	N	N/A	N/A	1.0	0.0	N/A	0.99
11	5/11	Y	0.97	0.50	0.90	0.91	0.57	0.91
12	0	N	N/A	N/A	1.0	N/A	N/A	1.0
13	6/10	Y	0.97	0.73	0.93	1.0	0.07	0.77
14	6/6	Y	0.85	1.0	0.99	1.0	0.02	0.96
15	0	N	N/A	N/A	1.0	N/A	N/A	1.0
16	8/13	Y	0.87	0.60	0.90	0.75	0.05	0.78
17	2/2	Y	0.81	1.0	0.99	1.0	0.08	0.96
18	4/4	Y	0.92	1.0	0.98	1.0	0.87	0.96
19	6/9	Y	0.97	0.54	0.93	0.98	0.11	0.87
20	13/16	Y	0.91	0.81	0.97	0.89	0.73	0.95
21	8/12	Y	0.96	0.65	0.97	0.90	0.69	0.96
22	6/6	Y	0.83	1.0	0.99	0.88	0.27	0.96
23	6/7	Y	0.93	0.94	0.99	N/A	0.0	0.95
24	0/0	N	N/A	N/A	1.0	N/A	N/A	1.0
25	4/5	Y	0.87	0.98	0.99	1.0	0.21	0.97
26	5/12	Y	0.98	0.52	0.88	0.94	0.53	0.88
27	0/0	N	N/A	N/A	1.0	N/A	N/A	1.0
28	4/5	Y	0.69	1.0	0.98	N/A	0.0	0.95
29	6/8	Y	0.88	0.72	0.99	1.0	0.07	0.96
30	3/4	Y	0.94	0.94	0.99	N/A	0.0	0.95
31	7/7	Y	0.94	1.0	0.99	1.0	0.18	0.90
32	1/3	Y	0.99	0.15	0.96	0.82	0.09	0.95
33	0/0	N	N/A	N/A	1.0	N/A	0.0	1.0
34	0/0	N	N/A	N/A	1.0	N/A	0.0	1.0
35	2/6	Y	0.94	0.40	0.86	0.99	0.20	0.82
36	0/0	N	N/A	N/A	1.0	N/A	N/A	1.0
37	8/9	Y	0.88	0.97	0.98	1.0	0.07	0.87
38	0/0	N	N/A	N/A	1.0	0.0	N/A	1.0
39	0/0	N	N/A	N/A	1.0	N/A	N/A	1.0
40	0/0	N	N/A	N/A	1.0	N/A	N/A	1.0

The metrics that determine the most unique sentences instead of division points are also quite important in the prediction as demonstrated by the usefulness of the cosine similarity of the Doc2Vec vectors as well as the mean relational frequency of words. In case of the lowest frequency metric, there may easily be sentence-unique words in sentences that are from non-anomalous parts and sentences from anomalous parts that do not have sentence-unique words. As for the highest frequency, while some anomalous sentences might not have the same stop words as the rest of the sentences, others might, making it not all that reliable.

7 Conclusion

We have developed a method capable of detecting intrinsic anomalies in natural text based on sequential analysis of the syntactic and semantic patterns exhibited by potentially anomalous text. The algorithm consists of the two step process of identifying the location of the starting and ending sentences of anomalies and determining whether a section located between two such points is anomalous. For both of these steps we use gradient boosting to form a prediction out of various metrics extracted from the text using the Semantic Folding algorithm, HTM network, Doc2Vec, autoencoders and metrics based on word frequencies.

The algorithm was tested and evaluated on 40 English texts with artificially inserted plagiarisms from the PAN intrinsic anomaly detection plagiarism corpus 2011. Our objective was twofold, the first, to determine whether the text contains any anomalies at all and the second, to determine the exact number and position of the anomalous passages. The algorithm achieved an accuracy of Task 1 was 100 % and better-than-expected results in Task 2, depending on the text with relatively high values of precision and accuracy, but varying recall. The overall percentage of fully found plagiarisms within the text was 70.15 %. We found that our algorithm was able to improve on the results of Kuznetsov et al. [19]. While there is much room for improvement, we believe the paradigm of continuous analysis in plagiarism detection to be an interesting and valid one that provides non-dismissible results and might be a step in the right direction when solving similar problems.

Acknowledgements. The research is supported by the Slovak Scientific Grant Agency VEGA, Grant No. 1/0177/21 “Descriptional and Computational Complexity of Automata and Algorithms”.

References

[1] Potthast, M., Stein, B., Eiselt, A., Barrón-Cedeño, A. and Rosso, P.: PAN Plagiarism Corpus 2011 (PAN-PC-11), DOI 10.5281/zenodo.3250095, (2011) http://www.uniweimar.de/en/media/chairs/webis/corpora/pan-pc-11/

[2] Almarimi, A., Andrejková, G., Salem, A.: Anomaly Searching in Text Sequences CEUR, ISSN 1613-0073, Vol-2046 urn:nbn:de:0074-2046-8, Proceedings of the 11th Joint Conference on Mathematics and Computer Science Eger, Hungary, May 20-22, 2016.

[3] Le, Q. and Mikolov, T.: Distributed Representations of Sentences and Documents. Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2):1188-1196, (2014)

[4] De Sousa Weber, F.: Semantic Folding (Theory and its Application in Semantic Fingerprinting. White paper, **Version 1.2**, 1–59 (2016)

[5] Hawkins, J., Ahmad, S., Purdy, S. and Lavin, A: Biological and Machine Intelligence (BAMI), `http://numenta.com/business-strategy-and-ip/`. Release 0.4, (2017)

[6] Aggarwal, C. C.: Outlier analysis. Springer Science+Business Media New York, (2013). https://doi.org/10.1007/978-3-319-47578-3

[7] Chandola, V., Banerjee, A. and Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv., 41(3), (2009)

[8] Zhuang, H., Wang, Ch., Tao, F., Kaplan, L. and Han, J.: Identifying Semantically Deviating Outlier Documents. Proceedings of the conference on Empirical Methods in Natural Language Processing, pages 2748–2757, Copenhagen, Denmark, September 7–11, 2017. ©2017 Association for Computational Linguistics

[9] Kannan, R., Woo, H., Aggarwal, C. C. and Park, H.: Outlier Detection for Text Data: An Extended Version. arXiv:1701.01325v1, (2017)

[10] Hawkins, J. and Ahmad, S.: Why neurons have thousands of synapses, a theory of sequence memory in neocortex. Frontiers in Neural Circuits **10**(23), 1–13 (2016)

[11] Hawkins, J., Lewis, M., Klukas, M., Purdy, S., Ahmad, S.: A framework for intelligence and cortical function based on grid cells in the neocortex. Frontiers in Neural Circuits, Vol. 12,

[12] Ahmad, S. F. and Purdy, S.: Real-Time Anomaly Detection for Streaming Analytics. arXiv:1607.02480v1, (2016)

[13] Ahmad, S, Lavina, A., Purdy, S., Agha, Z.: Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 267 (2017), 134-147

[14] Cui, Y., Surpur, Ch., Ahmad, S., Hawkins, J.: A comparative study of HTM and other neural network models for online sequence learning with streaming data, 2016 International Joint Conference on Neural Networks (IJCNN), 2016, pp. 1530-1538, doi: 10.1109/IJCNN.2016.772738

[15] Hole, K. J.: Anomaly Detection with HTM. Chapter 12 in the book: Anti-fragile ICT Systems, Springer 2016

[16] Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent Trends in Deep Learning Based Natural Language Processing. 2018, arXiv:1708.02709v8 [cs.CL]

[17] Mnatzaganian, J., Fokoué, E. and Kudithipudi, D.: A Mathematical Formalization of Hierarchical Temporal Memory's Spatial Pooler. arXiv:1601.06116v3, (2016)

[18] Oberreuter, G., L'Huillier, G., A. Ríos, S., D. Velásquez, J.: Approaches for Intrinsic and External Plagiarism Detection, Notebook for PAN at CLEF 2011, (2011)

[19] Kuznetsov, M., Motrenko A., Kuznetsova, R. and Strijov, V.: Methods for intrinsic plagiarism detection and author diarization, Notebook for PAN at CLEF 2016, (2016)

[20] Natekin, A. and Knoll, A.: Gradient boosting machines, a tutorial, Frontiers in Neurorobotics www.frontiersin.org December 2013, Volume 7, Article 21, DOI 10.3389/fnbot.2013.00021, (2013) Article 121, (2019)

[21] Loper, E., Bird, S.: NLTK: The Natural Language Toolkit. CoRR, cs.CL/0205028., (2002)