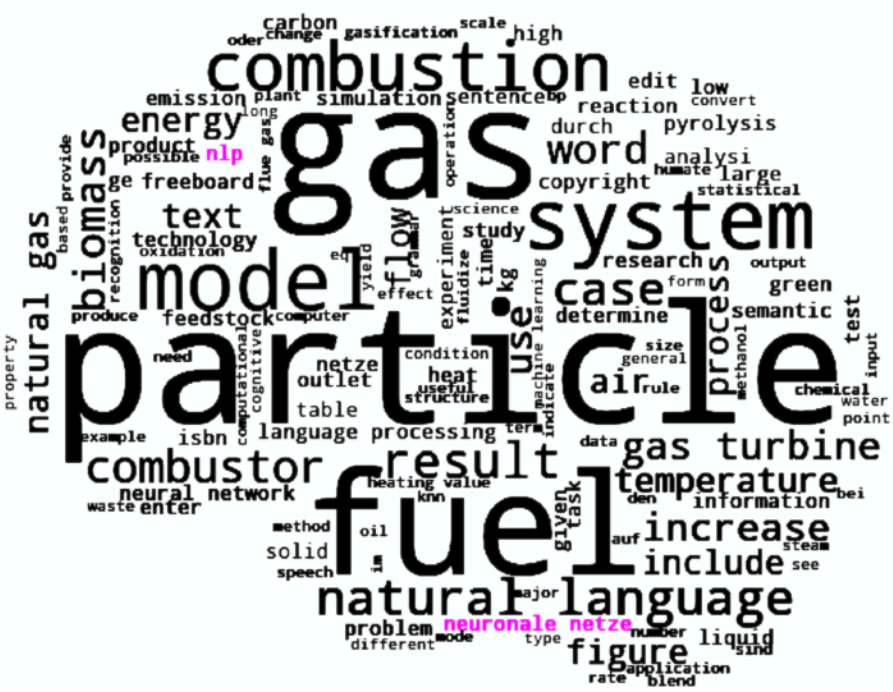
Natural Language Processing based toolbox for detecting novelty in scientific research papers

ME4 Individual Project

Final Report

May 17, 2021



Yousef Nami

Supervisor: Dr Loïc Salles

Imperial College London

Department of Mechanical Engineering

Word count: 11972

Word limit: 12000

Abstract

Determining whether a document contains novel research or not simplifies the otherwise laborious literature review process. This is a largely unsolved problem, both due to the difficulty in defining what constitutes novel research and in extracting document level semantic relationships using statistical methods. In recent years, Deep Learning has boosted Natural Language Processing (NLP), particularly deep representations of text. Despite this, there has been little research into using NLP to detect the *novelty* of long documents (e.g. scholarly articles).

This paper defines novelty as *dissimilarity* provided that the document is *relevant.* Two novelty detection models (pseudo-supervised and unsupervised) are introduced to address this issue. The former, based on a pairwise convolutional neural network, is computationally and memory intensive. It was unable to train. The unsupervised model is based on outlier detection in an embedding space. The model was found to perform well for pre-processing tasks (i.e. filtering out irrelevant documents) but not for novelty detection.

Future researchers are encouraged to find representative document embeddings, including graph based representations.

Table of Contents

Nomenclature

Acronyms	
NLP	Natural Language Processing
UML	Unified Modeling Language. A schematic standard for showing relationships in object oriented programming
SDK	Software Development Kit
SOTA	state-of-the-art
Terminology	
Token	The smallest object in a piece of text. Often this is the words, but could be sentences, paragraphs or characters.
Feature	A variable that contains explanatory information for the point is represents. Typically the columns when data presented in a tabular format. For example, a point on the Cartesian plane contains 2 features, x and y.
One-Hot-Encoding	Vectorial representation of discrete data, whose features are all unique classes set to 0 except for the class that is being represented.
Embedding space	Vectorial representation of text
Tensor	An array whose dimensions are defined by its rank, e.g. Rank 1 is a vector
Fonts	
CMU Classical Serif	This is used to highlight text used in an NLP problem
`CMU Typewriter Text`	This is used for representing Python code, packages or objects and NLP tools
Linear Algebra	
Einstein Notation	The number of non-repeated indices represents the rank of the tensor, e.g. v_i represents components of vector $\underline{v}$ and V_{ij} represents matrix $\underline{\underline{V}}$. Tensors with repeated indices, e.g. $V_{ij}v_j$ imply a summation over j. Abstract Einstein Notation is not used in this report.
$\odot$	Pointwise multiplication of tensors, e.g. the dot product $\underline{a} \cdot \underline{b} = \sum_i a_i \odot b_i$
∇	The gradient operator. Increases the rank of a tensor by 1.
Symbols	
D_i	A single text document
C	A set of text documents, typically stored as a `list` of texts
V_{ij}	Vectorial representation of a document D_i in an embedding space
$\{\}$	Typically represents a set
$P(a \mid b)$	Probability that an event a occurs given b

1. Introduction

1.1. Background

In an age where information is being produced at an accelerating rate, there is an ever increasing need for tools that distinguish relevant subject matter from redundant content in all media, spanning from the news to scholarly articles.

In scientific research, sophisticated publisher search engines streamline this process by returning the most relevant articles based on user-entered keywords. A problem with this is that deep semantic relationships are not captured, and thus a typical search could yield hundreds of potential articles, most of which don't reflect new research trends. The literature review process is therefore highly labor intensive.

The advent of deep learning has boosted Natural Language Processing (NLP) techniques, particularly in topic modelling, Q/A chatbots and sentiment analysis [1]. Much of this is attributed to learned word vectors that capture the semantic meanings of words, but also to modern neural network architectures (some developed as recently as 2018 [2]) that greatly accelerate machine understanding of text documents, dulling out words that aren't informative. Despite these advancements, there has been little research into using NLP to determine the *novelty* of long documents (e.g. scholarly articles).

1.2. Aims and Objectives

Novelty detection is a largely unsolved problem, due to the difficulty in defining what constitutes new research. As this is a first iteration of this project, novelty is defined as finding *dissimilar* documents provided that they are *relevant* to a given research area. The area of focus for this project is Turbomachinery, given the supervisor's research interests and the availability of a large body of data; namely some 26000 research papers in PDF format. The aim of this project is to create a novelty detection toolbox that provides future researchers with methods for collecting, processing and analysing text data.

The objectives of the project are as follows:

- Perform a Literature Review on the state-of-the-art (SOTA) text representation techniques and neural network structures used in NLP
- Provide easy to use tools (i.e. little programming experience) for extracting text data from open access journal articles (from Elsevier, Core, IEEE and Hindawi) and PDFs

- Create pre-trained model[s] to rank papers by novelty (supervised model)
- Create pre-trained model[s] for finding novel papers (unsupervised model)
- Create Python pipelines for training models on custom datasets

1.3. Report Structure

The report starts with a Literature Review that touches upon the theory and history of vectorial representation of text. An account of neural network structures used in NLP is given, followed by a review of SOTA novelty detection methods. The Methodology and Experimental Results section describes data attainment, processing and analysis (and its results). A detailed account of the modelling methods is also given. The Outcomes section describes the results of the project, which are analysed in Discussion and Future Work. This section also outlines bottlenecks in the project workflow and recommendations for future work. Project outcomes and key insights are reported under Conclusion. This is followed by a Reference List.

This project is accompanied by a GitHub repository [3] found at: https://github.com/lsalles23/ContentMining[1].

[1] Note that the repository is private. Please contact yn2217@ic.ac.uk for access.

2. Literature Review

2.1. Vectorial Representation of Text

Before text can be analysed, it must be converted into into a numerical representation. One of the earliest approaches for this is the `Bag-of-Words (BOW)` model created in 1954 [4], where the corpus $C = \{D_1 \cdots D_N\}$ of N documents D_i is represented as a matrix $V_{ij} \in \mathbb{R}^{N \times L}$ where L is the number of unique words in the corpus. Each component of the matrix V_{ij} would thus represent the number of occurrences of word w_j in document D_i. For example, $C = \{$ *"My name is Yousef", "NLP is awesome"*$\}$ would be represented as shown in Table 2.1 below.

Table 2.1: a `BOW` representation of corpus C

	My	*name*	*is*	*Yousef*	*NLP*	*awesome*
My name is Yousef	1	1	1	1	0	0
NLP is awesome	0	0	1	0	1	1

Though this method is simple, it fails to capture information beyond word frequency (such as word interaction or long term connotations). Some interactions can be captured using the `Bag-of-n-grams` model. Here, instead of capturing the words themselves, phrases comprised of n tokens are used instead. For the same corpus as above, a bi-gram ($n = 2$) representation is shown in Table 2.2 below.

Table 2.2: a `bag-of-n-grams` representation of corpus C

	BEG My	*My name*	*name is*	*is Yousef*	*Yousef END*	*BEG NLP*	*NLP is*	*is awesome*	*awesome END*
My name is Yousef	1	1	1	1	1	0	0	0	0
NLP is awesome	0	0	0	0	0	1	1	1	1

Extra tokens *BEG* and *END* are added to signify the beginning and end of a document.

Despite their uses, neither of the methods described above give relative importance to words that are more informative. For example, the word *is* should most appropriately be

thought of as noise, since it would appear in every single document with very high frequency. However, both methods above would incorporate it significantly in the vectorial representation. `TF-IDF` is vector representation method that improves this [5].

`TF-IDF` is defined as the **term frequency** multiplied by the **inverse document frequency**. The former can be calculated using:

$$\text{tf}_{ij} = \frac{\#\text{ occurances of } w_j \text{ in } D_i}{\#\text{ unique tokens in } D_i} = P(w_j \mid D_i) \tag{1}$$

The inverse document frequency is given by:

$$\text{idf}_i = \frac{\#\text{ documents in corpus}}{\#\text{ documents containing word } i} \tag{2}$$

The `TF-IDF` vector V_{ij} representing the documents then becomes:

$$V_{ij} = \text{tf}_{ij} \odot \text{idf}_i \tag{3}$$

Here we see that terms that appear frequently in a given document D_i are rewarded through the term frequency, but punished if they also appear frequently in many other documents through the inverse document frequency. A `TF-IDF` representation for corpus C is given in Table 2.3 below.

Table 2.3: a `TF-IDF` representation of corpus C

	My	*name*	*is*	*Yousef*	*NLP*	*awesome*
My name is Yousef	0.5	0.5	**0.25**	0.5	0	0
NLP is awesome	0	0	**0.33**	0	0.67	0.67

Even though this is a trivial example, we can see that the word *is* in both documents is given less importance to the other words, by virtue of it appearing in both documents. It's also worth noting that the word *is* is more important in the second document than in the first one, since the second document has less words.

There exist a plethora of variations in the methods described thus far, including ways of capturing more context from the words by incorporating interaction terms [4], or adding

activation functions to change the distributions [6]. Although they are useful, these fail to capture deeper semantic information and are also computationally expensive due to the large feature space (which is at least the size of the unique number of words L).

Word embeddings are a way of capturing deeper semantic information with a smaller feature space. Each word is represented by a vector $\underline{v} \in \mathbb{R}^l$ where l is an embedding length chosen by the user (often chosen arbitrarily between 50–300, but can be informed using corpus statistics [7]). These vectors are *learnt* (by means of a neural network for example) empirically given a certain large sized corpus C, and can therefore capture interactions that are otherwise difficult to pick up on.

Words that are similar semantically are also similar mathematically (by use of metrics such as the dot product for example). For example, consider an embedding vector space $\underline{v} \in \mathbb{R}^3$ (where the features are **expensive**, **company**, and **taste**) for a corpus comprised of the set $\{"apple",\ "Apple",\ "IBM",\ "orange"\}$, shown in Table 2.4 below.

Table 2.4: contrived word embeddings for the set $\{"apple",\ "Apple",\ "IBM",\ "orange"\}$

	expensive	company	taste
$\underline{v}_{apple}$	0.02	0.3	**0.9**
$\underline{v}_{Apple}$	**0.9**	0.8	-0.5
$\underline{v}_{IBM}$	0.5	**1.5**	-1.5
$\underline{v}_{orange}$	0.01	0.3	**0.9**

The $\underline{v}_{Apple}$ for the electronics giant vectorially close to IBM, with $\underline{v}_{Apple} \cdot \underline{v}_{IBM} = 2.4$, but far from $\underline{v}_{apple}$ the fruit, with $\underline{v}_{Apple} \cdot \underline{v}_{IBM} = -0.192$.

It's worth noting that these features are contrived for this example to give an intuitive feel for how the vectors of different words might be close in the embedding space. In reality, the features are very difficult to interpret as has been done here.

Beyond closeness, these vectors can capture meaning through direction as well. The classic example is shown below:

$$\underline{v}_{man} + \underline{v}_{queen} - \underline{v}_{woman} \approx \underline{v}_{king}$$

One of the earliest word embedding techniques is `word2vec` by Mikolov et al. at Google in 2013 [8]. This method constructs a contrived classification task using a neural network [2]with the intention of extracting a compressed vectorial representation of the input created in the hidden layers of the network. The classification tasks are either: a) using *context* words to predict a word, known as `Continuous BOW` or b) using *words* to predict their context, known as `skip-gram`. In the `skip-gram` model, the classification outputs are the context words of a word j in the corpus C [9]. So for each word, its neighbouring context words are found and represented vectorially. Table 2.5 below shows this, using a context window of 2.

Table 2.5: showing one-hot-encoded representations of each word j (yellow) and its context words for a context window of length 2 (green)

Corpus	Word j vector	Context words	Classification vectors
Traditional English breakfast contains black pudding	[0, 1, 0, 0, 0, 0]	Traditional breakfast contains	[1, 0, 0, 0, 0, 0] [0, 0, 1, 0, 0, 0] [0, 0, 0, 1, 0, 0]
Traditional English breakfast contains black pudding	[0, 0, 1, 0, 0, 0]	Traditional English contains black	[1, 0, 0, 0, 0, 0] [0, 1, 0, 0, 0, 0] [0, 0, 0, 1, 0, 0] [0, 0, 0, 0, 1, 0]
Traditional English breakfast contains black pudding	[0, 0, 0, 1, 0, 0]	English breakfast black pudding	[0, 1, 0, 0, 0, 0] [0, 0, 1, 0, 0, 0] [0, 0, 0, 0, 1, 0] [0, 0, 0, 0, 0, 1]
Traditional English breakfast contains black pudding	[0, 0, 0, 0, 1, 0]	breakfast, contains, pudding	[0, 0, 1, 0, 0, 0] [0, 0, 0, 1, 0, 0] [0, 0, 0, 0, 0, 1]

Then, *each* word vector is fed into a neural network with one hidden layer and multiple outputs, one for each word in the context window. This is shown in Figure 2.1.

[2] The workings of a neural network are described in greater detail in Section 2.3.1. For now, it suffices to see them as a function with unknown (hidden) parameters that maps tensor inputs X_{ij} to tensor outputs $\hat{y}_i$ that estimate a true value y_i. The function's parameters are optimised through iterations to minimise the error between the predicted outputs $\hat{y}_i$ and the true value y_i. This process is called learning, since the function's parameters (and therefore $\hat{y}_i$) are changing with each iteration.

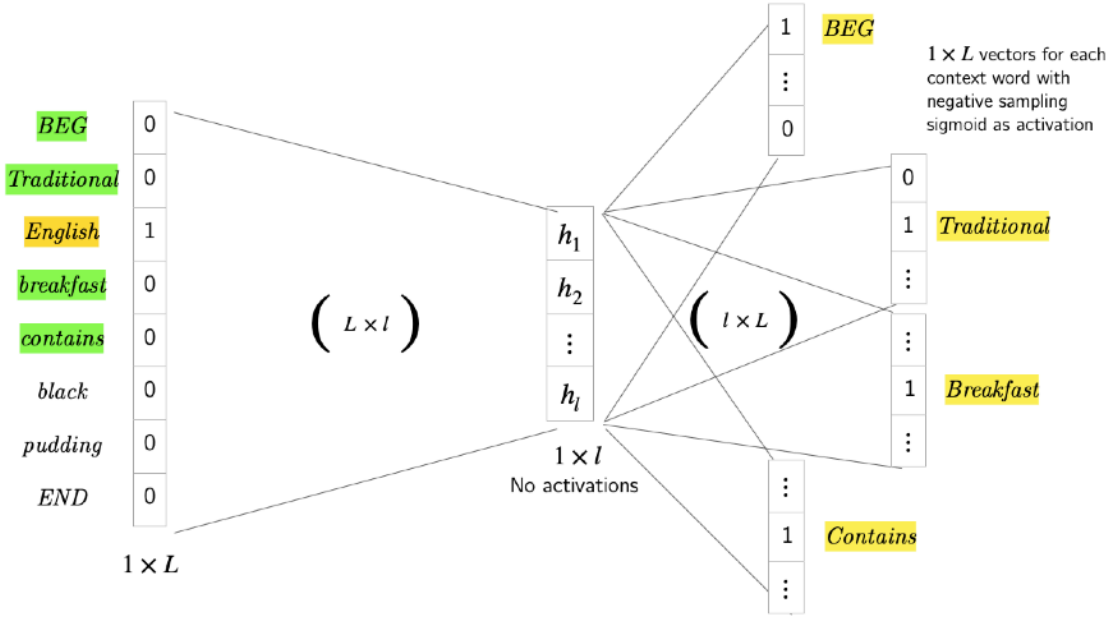


Figure 2.1: a diagram of the skip-gram architecture for the input word *English*

The tokens *BEG* and *END* are added to ensure a uniform embedding size. After training, the embedding vector for each word is effectively the hidden layer of the network. [10] is an excellent resource that delves deep into the computational aspects of `word2vec`, while [11] provides good intuitive understanding of the embedding vectors.

The `Continuous BOW` representation is created using a similar model, but where the inputs and outputs are swapped. So the context words are used to predict the word itself.

Despite its wide success, `word2vec` is unable to capture whole text statistical information [12], since it focuses on context words. Pennington et al. created `GloVe` in 2014 [13], a statistical based model that captures the global context of words.

The `GloVe` algorithm relies on the co-occurrence probability function, defined below:

$$F(w_i, w_j, \tilde{w}_k) = P_{ik}/P_{jk} \tag{4}$$

Where matrix $P_{ij} \equiv P(w_i|w_j)$, i.e. the probability that word j appears in the context of word i. The way this is found is by constructing a co-occurrence matrix X_{ij} (with a pre-defined context window) that tabulates times word j occurs in the context of word[3] i.

[3] Optionally, a multiplicative damping factor $\lambda = 1/\text{offset}$ can be added to give less importance to words that appear further in the context window

F is a function of three words: w_i which is the target word, w_j which is a word in the context of w_i, and finally $\tilde{w}_k$ which are probe words. This metric is useful for extracting relationships between words, as shown in Table 2.6 below.

Table 2.6: the co-occurrence probabilities for ice and $steam$ compared with $\{"solid", "gas", "water", "fashion"\}$ [14]

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Very small or large:
solid is related to ice but not steam, or
gas is related to steam but not ice

close to 1:
water is highly related to ice and steam, or
fashion is not related to ice or steam.

Constraints are then added to F to ensure that it is linear[4] and invariant[5], giving a final expression:

$$J = \sum_{i,j=1}^{L} f(X_{ij})(w_i \cdot \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \tag{5}$$

where b_i, $\tilde{b}_j$ are bias terms, and:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise}. \end{cases} \tag{6}$$

The function f ensures that words that don't appear very frequently (and thus may not be so informative) have a smaller effect on the ensuing word vectors. [13] and [15] are recommended for those interested in the theory behind cost function formulation.

So far, neither of the embedding techniques capture *polysemy*. For example, the word $date$ can refer to the dried fruit, a romantic appointment, the day/month/year or the verb of establishing the age of something (e.g. carbon dating). SOTA word embeddings (derived from generalised language models) are able to account for this. Two of these

[4] This is necessary for additive word vectors, i.e. $\underline{v}_{\text{man}} + \underline{v}_{\text{queen}} - \underline{v}_{\text{woman}} \approx \underline{v}_{\text{king}}$

[5] So that there is no distinction between words and their context (i.e. they can be swapped around)

embeddings, namely Embeddings from Language Model (ELMo) and Bidirectional Encoder Representations from Transformers (BERT) are covered in this literature review. [16] provides a good review of other generalised language models.

ELMo captures this extra context by representing word embeddings as a combination of word-level characteristics and context. This model uses deep bi-LSTM (see Section 2.2.3) layers for language modelling (i.e. predicting the next word in a sentence). The word embeddings are then represented as a weighted combination of the final output of the model and is deep hidden layers; the intuition being that the hidden layers capture syntactic information, while the output layer captures word context.

BERT, developed by Google in 2019 [2], uses bidirectional Transformers (see Section 2.2.5) to generate context-aware word embeddings that have outperformed other SOTA models in various NLP tasks. This is partly due to the use of Transformers that incorporate a mechanism called Attention (see Section 2.2.4), which pick up on trigger words that explain a certain word. For example, in the sentence $dates\ are\ very\ tasty$, the transformer would focus on the word $tasty$ to predict $dates$. In addition to this, BERT shuffles the order of sentences during training, and also randomly masks around 15% of the words for prediction [17]. This helps it capture the context of the words better. It's worth mentioning that BERT does not take words as inputs, but rather **WordPieces**, which are a collection of common subwords that are representative in the given language [18]. So for example, $apples$ might be represented as app and $\#\#les$ ($\#\#$ indicates that it follows another subword). This is highly useful, since out of vocabulary words (OOV) would be formed as a combination of subwords as opposed to being ignored, while keeping the BERT vocabulary quite small (around 30 thousand items) [19].

Many of the above word embedding techniques have been extended to sentence or document level representations [4]. There is also some research in the literature into embedding techniques that capture novel spatial structures in text. These include hyperbolic [20, 21], graph-based [22, 23], hierarchical [21, 24] embeddings.

2.2. Neural Network Models in NLP

This report does not intend to provide a detailed account of neural networks. The purpose of this section is to familiarise readers with network structures that are commonly used in NLP (some of which have been mentioned already). For more theoretical details on neural

networks, the reader is referred to [25], and to 3Blue1Brown's series [26] which provides an excellent visual representation of their inner workings.

2.2.1. Feedforward Neural Networks

As mentioned in Section 2.1, a neural network is a function that takes an input X_{ij} and maps it to an output $\hat{y}_i$, in an attempt to estimate a true value y_i. This function has unknown parameters $\underline{\theta}$ that affect the mapping. The optimal values of $\underline{\theta}$ are those which minimise the the error between $\hat{y}_i$ and y_i.

The network itself is comprised of an input layer, hidden layers and an output layer; a simple network with one hidden layer is shown in Figure 2.2.

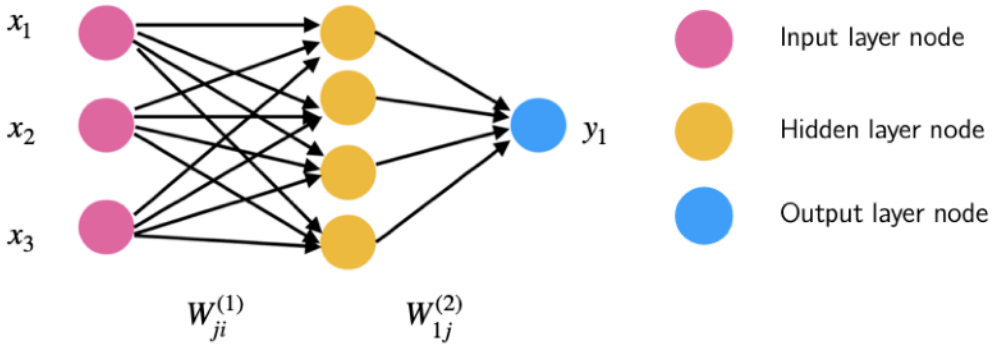


Figure 2.2: a neural network with a single hidden layer and a single vector input [27]

Each layer output acts as the input of the subsequent layer. The layer (L) has i nodes whose values are represented by the vector $a_i^{(L)}$. This is a function of the output nodes of the previous layer $a_j^{(L-1)}$ transformed by a non-linear activation function $\phi^{(L)}$, as defined in Equation (7)[6].

$$a_i^{(L)} = \phi^{(L)}\left(W_{ij}a_j^{(L-1)}\right) \tag{7}$$

W_{ij} is a multiplicative factor (called the weights matrix) that linearly combines the output nodes of the previous layer.

In the simple network shown in Figure 2.2, the predicted output $\hat{y}_1$ is given by:

[6] In the actual implementation, bias terms are also added.

where:

$$a_j^{(1)} = \phi^{(1)}\left(W_{ji}^{(1)} x_i\right) \tag{9}$$

In this case, the unknown parameters of the model that need to be estimated are $\underline{\theta} = \left\{W_{ji}^{(1)}, W_{1j}^{(2)}\right\}$. These are determined by minimising the error between the output and the true value, $J = \text{Error}(\hat{y}_i, y_i)$. Its gradient, $J_{,i} \equiv \underline{\nabla} J$ with respect to each θ_i is used to nudge the values of $\underline{\theta}$ closer to the optimum. The new values of θ_i, namely $\theta_i * = \theta_i - \eta J_{,i}$ (where η is a step size) are then used to compute a new $\hat{y}_i$, and the process is repeated for a number of iterations. This process is known as back-propagation, and what is often referred to as *learning*.

2.2.2. Convolutional Neural Networks

Although feedforward neural networks are very useful, they struggle to extract *spatial* features from highly dimensional data. Convolutional Neural Networks (CNNs) have been designed to do precisely this. For an intuitive example, consider image classification of hand-written letters. Each letter is a picture represented by pixels as shown in Figure 2.3 below. However, since the letters are hand-written, the same letter way be distorted, shifted, shrunk or rotated.

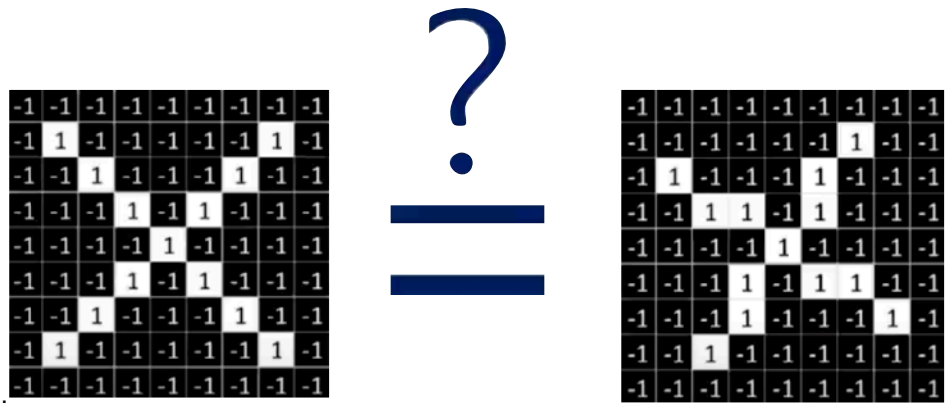


Figure 2.3: pixels of two hand-written X [28]

As humans, we can recognise the spatial features in each image, and recognise that they both represent the letter X. However, a normal feedforward network would treat each pixel as an independent datapoint, and thus see both letters in Figure 2.3 as vastly different. A CNN rectifies this by extracting spatial features. Figure 2.4 shows the features from both images in Figure 2.3 that are representative of the letter X.

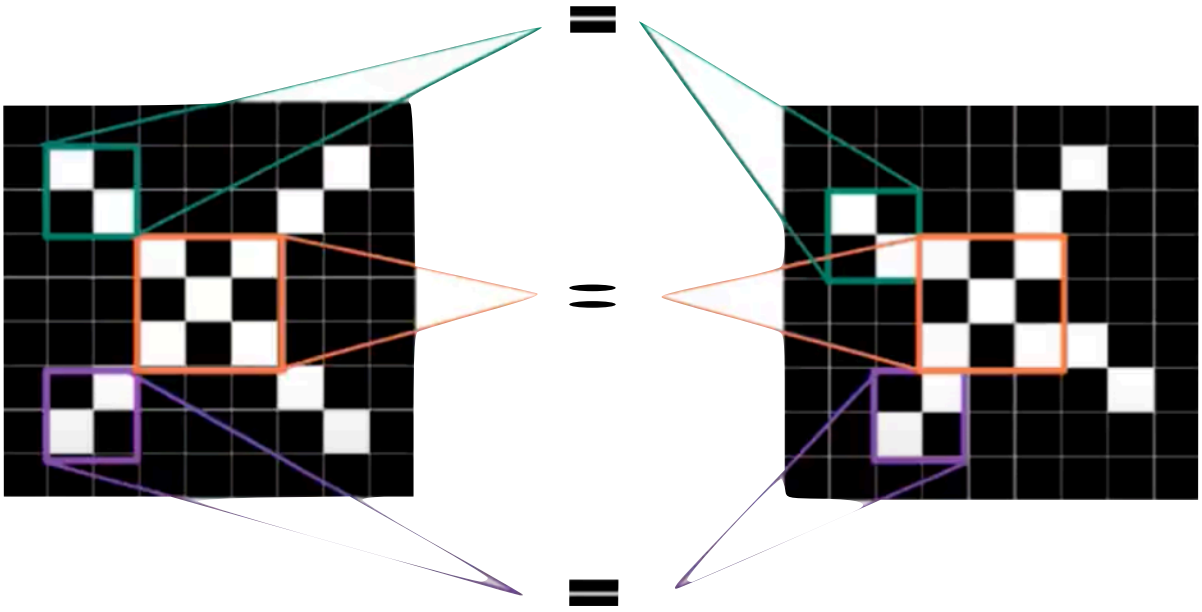


Figure 2.4: spatial features that define the letter X [28]

These features are extracted by means of a sliding window, called a **kernel**, that is initialised with random weights. At each window of the image, the pixel values are convolved (i.e. for example, dot product is taken) with the weights of the kernel. Then a process known as **pooling** is applied, where in each window, the maximum value is taken (effectively extracting the most important features). After the convolutional layers, standard feedforward layers are added to enable classification. An image of the entire process is shown in Figure 2.5 below [29].

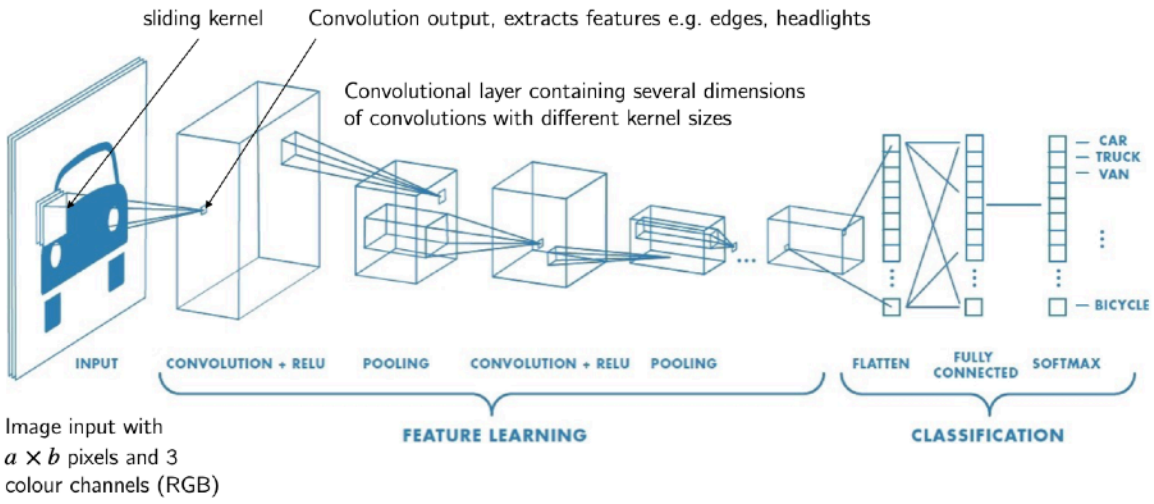


Figure 2.5: a diagram showing the workings of a CNN classifier applied to images [29]

The image example can be extended to NLP, where instead of pixels, you have words and their embedding dimensions, as shown in Figure 2.6 below.

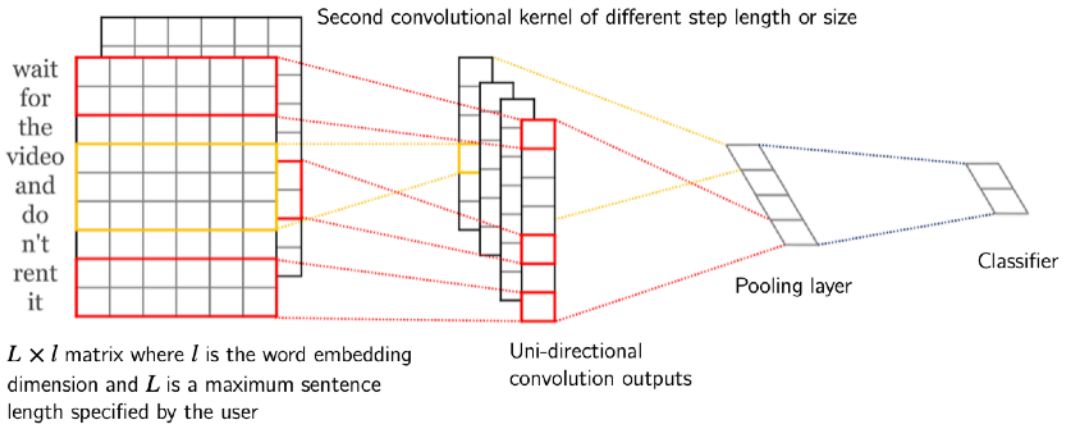


Figure 2.6: a CNN classifier applied to text [30]

In this case, the kernel is uni-directional, and it extracts features from the context window of each word. This idea can be extended to larger segments of text, e.g. sentences where the representation becomes 3 dimensional. One key difference when dealing with text data is that sentences can vary in length, so typically a maximum sentence length is specified. This is explored in more detail in Section 3.4.1.

2.2.3. Recurrent Neural Networks

Unlike images or other data, text contains both spatial and *sequential* data. As a result, neither feedforward neural network nor CNNs provide a *natural* way of reading text data. Recurrent Neural Networks (RNNs) are structures that have feedback loops to enable a sequential reading of data [31]. The RNN structure is shown in Figure 2.7.

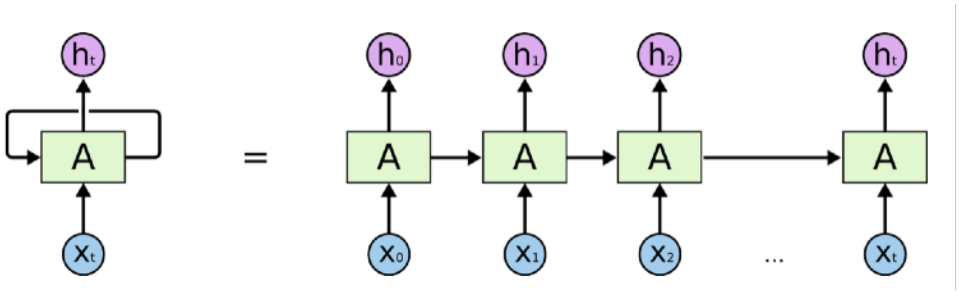


Figure 2.7: RNN structure with feedback loops (Left) and Unrolled RNN (Right)

The output state at each time step, h_t is thus given as a function the input x_t at that time step, and the output state of the previous cell. Therefore, $h_t = f(x_t, h_{t-1})$ [32].

Further, text data has long term dependencies that need to be accounted for. For example, consider the task of predicting the next word in the following sentence:

I grew up in France, but I now live in Boston. I speak fluent ...

In this case, an RNN would have to use the trigger word *France* to predict the next word, *French*. However, since the words are far from one another, during training the impact of *France* vanishes. As a result, certain activations functions that store memory, called Gated Cells, are required. One common Gated Cell is called Long Short Term Memory (LSTM). A schematic is shown in Figure 2.8.

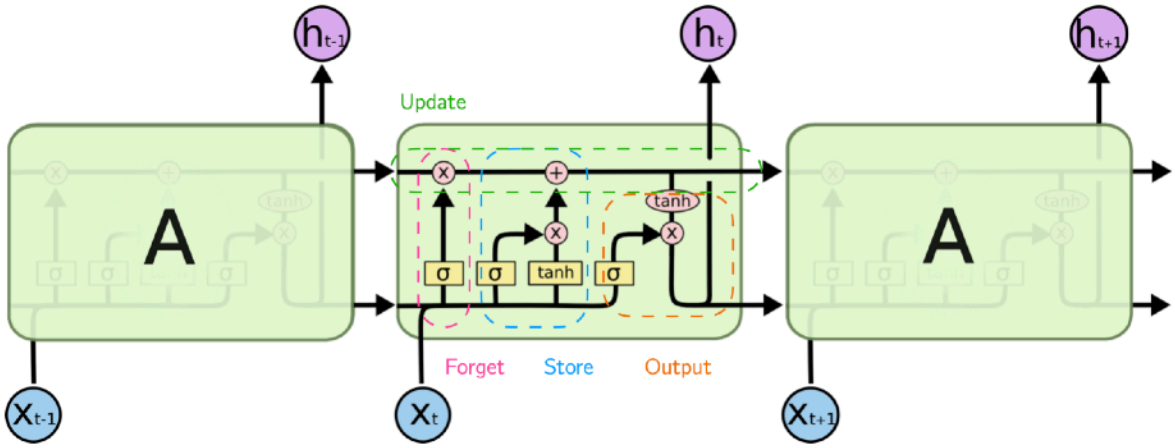


Figure 2.8: a schematic of an LSTM network showing the Gated Cell [31]

For a detailed guide on how these work, the reader is referred to [32].

The important thing to note is that the LSTM cell has four effects: **forgetting** irrelevant information from previous states, **storing** new information, **updating** the cell value (separately), and controlling the **output** information. This regulates the information that flows through the cell, which has the effect of better capturing long term dependencies.

Another consideration in text processing is that information flows two ways. For example, consider now the future context of our example sentence:

I grew up in France, but I now live in Boston. I speak fluent ... because I lived in Frankfurt for 5 years.

A uni-directional RNN will only capture the past information, and thus likely predict *French*. However, when future context is considered, *German* would be more appropriate. Therefore, bidirectional RNN models, for example bi-LSTMs have also been developed and shown to perform better than uni-directional models.

It's worth mentioning that, although RNNs provide a much more natural reading of text, they are typically really difficult to train, and thus CNNs, which only rely on spatial features, typically perform better on a multitude of tasks.

2.2.4. Attention Mechanisms and Transformer Networks

A key limitation of RNN models is that information gets lost in really long sequences. Although an LSTM is able to capture some memory, the structure itself is sequential, and therefore the output is not directly a function of each sequence element. Attention Mechanisms solve this problem by ensuring that each item in the sequence directly affects the output. A high level schematic of this is given in Figure 2.9 below.

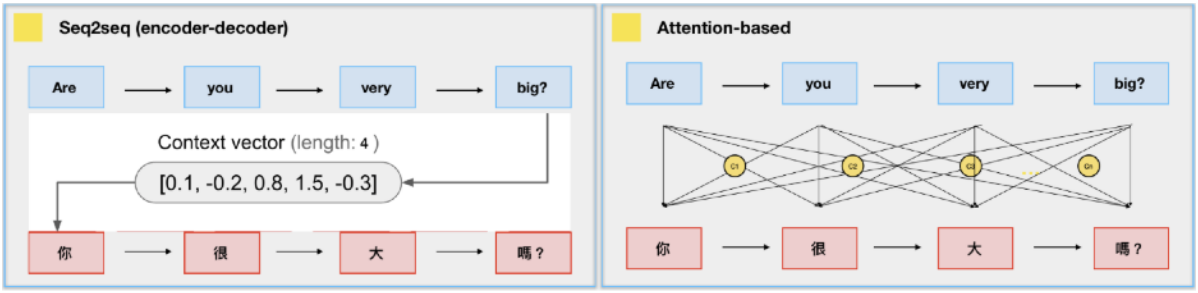


Figure 2.9: a machine translation example using an RNN (Left) and an attention based network (Right) [33]

This has the empirical effect of mimicking human attention, where the machine *attends* to parts of the sequence that are most informative, as shown in Figure 2.10.

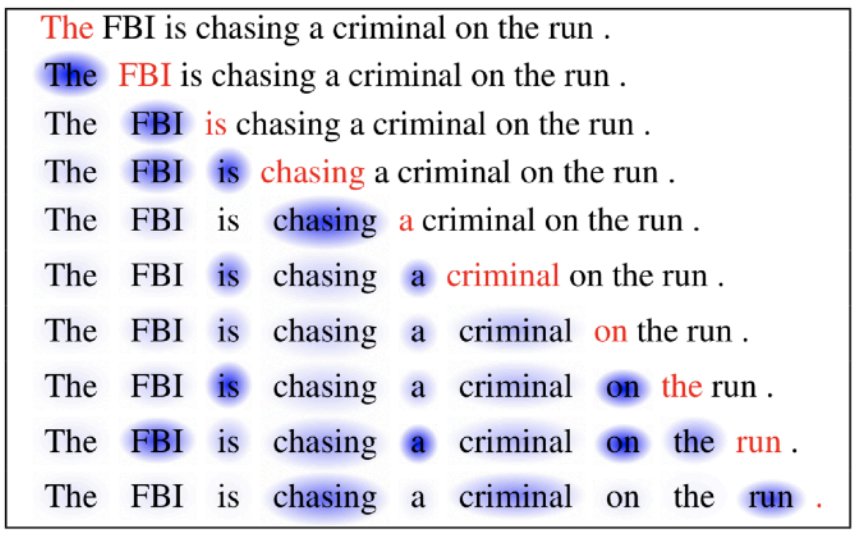


Figure 2.10: A visualisation of the information that the network attends to (in Blue) for each word (Red) [34]

Transformer Networks are models that rely on attention mechanisms and have no recurrent. Their structure enables independent calculations, making GPU parallelisation easy. Transformers are the SOTA model for NLP tasks. Interested readers are referred to [35] and [36].

2.2.5. Other Structures

A large subset of the networks used in NLP are auto-encoders (AEs). These networks consist of an encoder module, which compresses the input X_{ij}, and a decoder module, which reconstructs the same input X_{ij}. Effectively, the model is trying to learn how to re-create the input it is given without a large loss in information. The internals of encoders and decoders are comprised of neural network structures, typically deep layers of bi-LSTMs or CNNs, and more recently Transformers.

Other models leverage specific spatial features found in text, namely hierarchical (Hierarchical NNs) [37] and graph structures (Graph NNs) [38].

Finally, SOTA models typically combine different neural network structures (e.g. CNN-LSTM). These models perform comparatively better in many tasks, but come with high computational cost [1]. For a recent review of neural networks in text classification, the reader is referred to [1].

2.3. Novelty Detection Methods

2.3.1. Statistical Methods

Novelty detection in text has a long history, dating back to First Story Detection (FSD) systems for determining First Story news [39]. These systems rank documents in order of appearance (i.e. as a *stream*) and compare a current document to the previous ones using some statistical metric. If the document is deemed to vary substantially, then it is classified as novel or given a novelty score [40].

One problem with early FSD systems is that they struggle to correctly classify highly dimensional data. As such Petrović et al. used a method known as Local Sensitivity Hashing (LSH) to approximate closeness [40]. This was evaluated on headlines from the social media platform Twitter, and was found to perform well.

Other researchers have found success using named entities (for example, $Apple$ as **organisation**, $Xerxes$ as **person**) to determine the novelty of the given document [39,

41]. The intuition is that the news reports on specific entities (or relationships between entities). Panagiotou et al. proposed a pipeline that has a relation extraction module before using LSH to determine a first story using a novel relation similarity metric.

One key element that makes FSD systems successful is the fact that they can be ordered in a stream logically. This is difficult to justify for research papers, since the time scales are much larger. Further, news stories are short updates on real life events, and thus the intra-topic variation is fairly sequential, whereas this is not the case with scholarly articles. Therefore, FSD systems would not be suitable for document level novelty detection, but may be useful for detection novelty in abstract submissions.

There have been some successful unsupervised novelty detection methods outside of news. Suzuki et al. [42] leverage the hierarchical structures of patent claims to extract features that allow them to successfully find novel claims. Other researchers focus on using classical machine learning models to find novelty [43, 44]. Paweł used a one-class Support Vector Machine (SVM) applied to `GloVe` vectors to determine novelty in discussion forums [44], while Omari et al. used hierarchical clustering on a Yahoo Answers [43].

Overall, all the above methods work best on small sized documents (e.g. sentences) and not so much on larger documents [45], making them largely unsuitable for finding novelty in research papers.

2.3.2. Neural Network Methods

Ghosal et al. were one of the first to use deep learning for document novelty detection [45]. They did so on a supervised dataset, where each document is labelled as novel or not. Their method uses a recent sentence word embedding paradigm [46] to create document vector, which are fed into a CNN classifier based on [47]. This model performed comparatively well to bi-LSTM models and SOTA statistical document representation techniques, but required a high computational cost. Ghosal et al. has since produced an attention mechanism based novelty detection model that outperformed baselines by 3% [48]. Despite the success of both models, they are not appropriate for the novelty task in this project, since they rely on strictly defined supervised data. Further, both models are binary classifiers, whereas for this project, a continuous novelty value is desired (in order to rank papers).

Qin et al. [49] proposed an open world CNN classifier for detecting novel sentences. Their approach predicts the probability that a pair of sentences fall in the same class. Sentences from the test document are then paired with K sentences from each class m of the original training corpus. For each class m, the average probability that the test sentence belongs in that class is determined. If all the averages are low, then the test sentence is deemed novel, otherwise it is classified as the class with the highest average probability.

The research on deep unsupervised novelty detection in text is relatively thin, especially when considering large documents.

2.3.3. Other Methods

Some promising novelty detection research comes from non-NLP domains [50-52]. In image recognition, Tian et al. [52] improved auto-encoder performance for novelty detection using attention mechanisms and entropy minimisation, whereas Hendrycks et al. [53] did so through outlier injection. To date, none of the above methods have been applied successfully to text data.

Finally, Wang et al. [54] use one-class GNNs to detect novelty in graph structures. This is promising because other research [55] has shown that graphical representations of text can extract semantic information well. A combination of these methods could provide interesting results, in particular if the definition of novelty is concerned with finding new links in research, where graph theory metrics can be used to determine the importance of edges and links.

2.3.4. Conclusive Remarks

Statistical novelty detection methods rely on detecting anomalies in data, but are typically unable to capture novel information at document level. The vast majority of neural network driven novelty detection techniques in the literature rely on supervised techniques, which incidentally show the best performance. The research presented by Qin et al. [49] is the most promising for document level novelty detection proposed by this project, provided that a document's sentences could be clustered into distinct classes. That way, the percentage of novel sentences in each test document could be used to determine document level novelty. ¶

3. Methodology and Experimental Results

Given that this is a data driven project, the John Rollins' data science methodology [56] was used to manage the workflow. This section is organised with that in mind.

3.1. Data Collection

The data sources for this project, and their extraction methods are described in Table 3.1.

Table 3.1: Text sources and their extraction methods

Source Name	Details	Extraction Method
ASME Turbomachinery papers	26030 files in remote Linux computer. Older PDFs are comprised of images, while newer ones are encoded	PDF extraction libraries using OCR
Hindawi XML Corpus	All Hindawi papers from 2008—2020 saved as XML files. Relevant journals are: *Shock and Vibration* and *International Journal of Rotating Machinery.*	XML parser
Elsevier API	Open Access or Institutional Access papers found with user query.	API toolbox
IEEE API	Full Access papers found with user query.	
Core API	Full Access papers found with user query.	

The Hindawi corpus was extracted using Python's `xml.etree.ElementTree` class. An implementation can be found on GitHub under *PythonFiles/text_extraction*.

For the publisher APIs, an object oriented approach was used to enable identical behaviour for extracting and saving research papers across all publishers, but also to add specific features for publishers where appropriate (such as Elsevier's Institutional Access papers). The file was written such that it could be run with the Python shell (for integration into a pipeline on the College's HPC Cluster) but also with callable methods for it to be used by future researchers like a Python library. Figure 3.1 summarises the current features of the API module[7].

[7] For IEEE, there already exists an SDK online for Python 2. However, writing the glue code required to convert it to Python 3 and allow similar behaviour to the other APIs was deemed inconvenient, so new methods were developed.

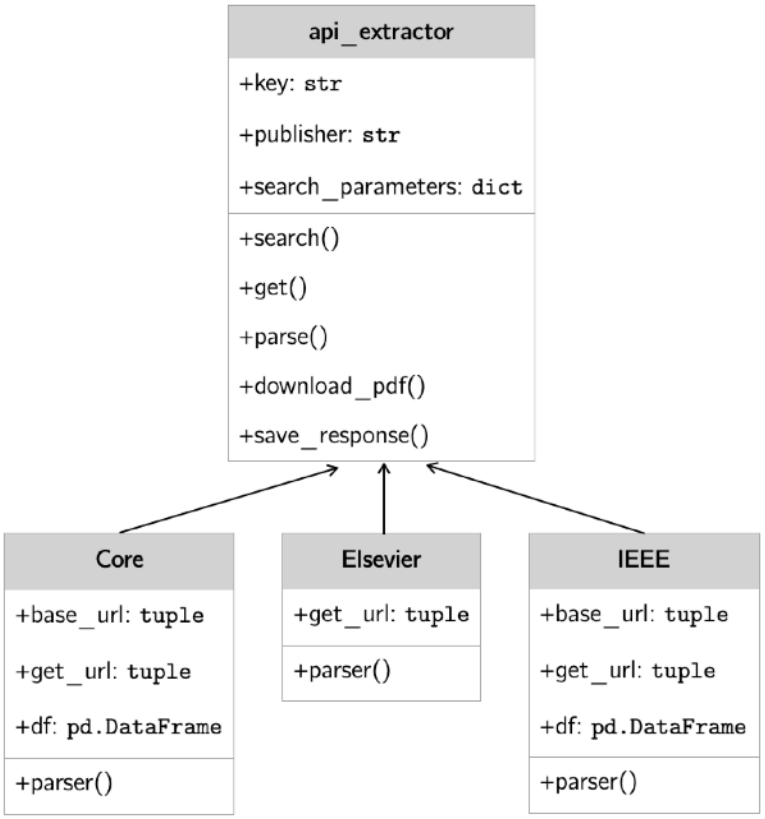


Figure 3.1: a UML diagram for the API module

The PDF extraction library proved to be the most interesting to design, since it needs to extract documents from either a remote or local device. Previously [57], the module was designed for remote extraction only. This was updated to enable dual function. The initial PDF to text extraction was done using `textract`, but this was subsequently changed to `pdfminer` (see Section 3.3 for more detail). The current implementation also uses an object oriented approach to enable future researchers to utilise their own text extractors.

The main data source in this project is the ASME papers. It was decided that they would be used for training and validation, while the other sources would be used for testing.

3.2. Data Processing

Once the data had been saved into a text (or binary) format, it had to be cleaned. In NLP, this is an involved and highly iterative step unlike in other areas of data science. The NLP library `spaCy` was used to accelerate this process; its standard pipeline is shown below in Figure 3.2.

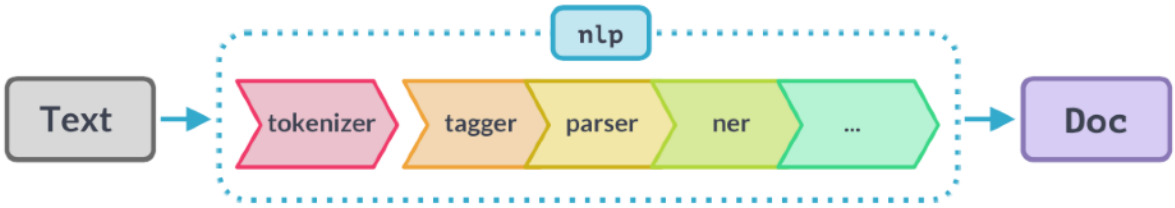


Figure 3.2: standard `spaCy` pipeline [58]

The NLP pipeline components can be modified using `spaCy`'s interface. This means that custom parsers and taggers (for example, using SOTA models [59]. See issue #14 on GitHub) can be used instead. In this project, the standard pipeline was used for efficiency.

Raw text is converted into `Doc` objects, which store words in a tokenised format (as `Token` objects) with their attributes[8]. These attributes capture rich semantic and syntactic information which can be used for cleaning, as shown in Figure 3.3 below.

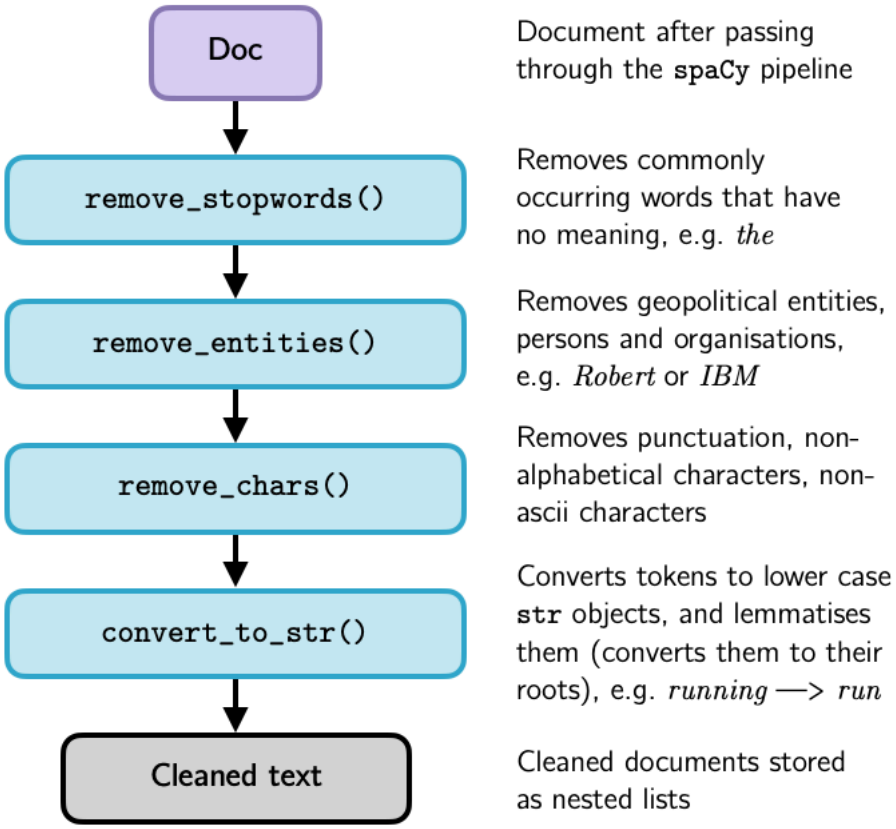


Figure 3.3: an annotated diagram of the cleaning pipeline used

[8] A full list of these can be found here: https://spacy.io/api/token

The overall pipeline was tested on a small subset of the dataset to ensure that it worked well before it was applied to the entire dataset. A sample output, showing the distribution of the most common words is shown below in Figure 3.4.

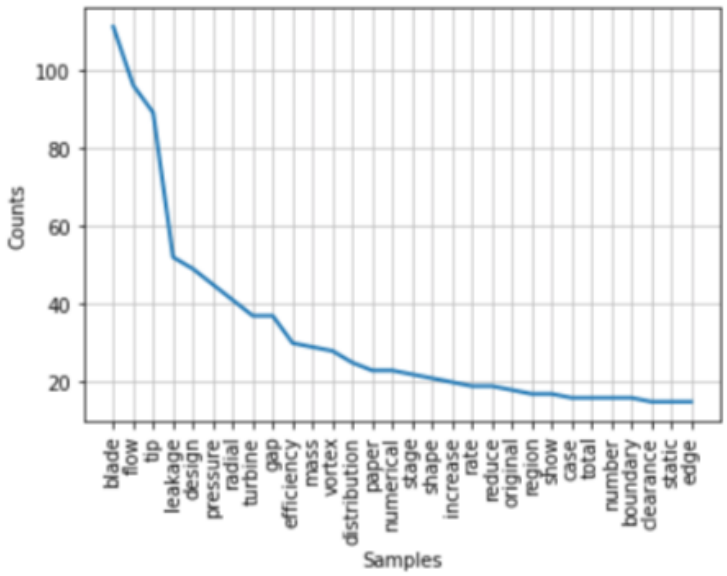


Figure 3.4: distribution of the most common words after passing through the `spaCy` and cleaning pipelines

When applied to a large number of papers, the entire process both computationally and memory intensive (can lead to crashes). As a result, the pipeline was written with parallel processing capabilities as a runnable script that would take user parameters such as the number of computer cores to use and batch size. This enables researchers to use the College's HPC services well.

Once this step was complete, the text was further cleaned using a bigram model. This effectively pairs words that frequently appear as pairs to decrease the size of the dataset. For example, the tokens New and $York$ would likely be paired to form New_York, which is more informative than either New (which could mean new) or $York$ (referring to the city of $York$). This step was repeated to form trigrams (i.e. 3 words connected to each other).

3.3. Exploratory Data Analysis

3.3.1. Data Quality

After cleaning, the text was scrutinised for ways it could be improved. The data was first visually inspected for problems. This was compared to the work done by Matthew

Warner[9] who used `pypdf2` instead of `textract`. It was found that the former would ignore whitespace for a large subset of the papers, as shown in Figure 3.5.

Figure 3.5: an example of text extraction by `pypdf2` where whitespace is ignored

Despite this, `textract` has its own problems. For example, it is unable to read 1842 of the papers on the Linux device due to certain characters that it can't decode. This prompted the author to use `pdfminer` instead of `textract`. The former behaves almost identically to `textract` (except for minor differences in how non-ASCII characters are processed), and is able to extract the 1842 papers that `textract` could not. A detailed account of this is given in issues #9 and #26 on GitHub.

3.3.2. Topic Distribution

Once this was complete, the data was analysed using Latent Dirichlet Allocation (LDA) to explore the topic distribution. This method is one of the earliest document embedding techniques [4]. The algorithm for this is given below [60]:

Algorithm 1 Determine topic distribution

1: **Initialise** Randomly assign each word in C to one of m topics
   - m chosen by user
   - C is the corpus
2: **Foreach** D_i in C
   - D_i is a document

[9] A student working on a different NLP project with Dr Salles

3:	**Forach** w_j in D_i	▸ w_j is a word in document D_i
4:	Calculate $P(m \mid D_i)$	▸ $P(m \mid D_i)$ is the proportion of topic m in document D_i
5:	Calculate $P(w_j \mid m)$	▸ $P(w_j \mid m)$ is the proportion of word w_j in topic m
6:	Calculate $P(m \mid w_j)$ using 4: and 5:	▸ $P(m \mid w_j)$ is the conditional probability of w_j belonging in m
7:	Re-assign w_j to the topic with the highest value in 6:	

This process created a model that could estimate the probabilities of m topics belonging in a document D_i, allowing for individual documents to be analysed. LDA is discussed in detail by Jonathan Hui [61]. The NLP library `gensim` was used for this. Figure 3.6 shows the topic distribution across the whole corpus, for $m = 5$ topics, whose meanings are shown in Figure 3.7.

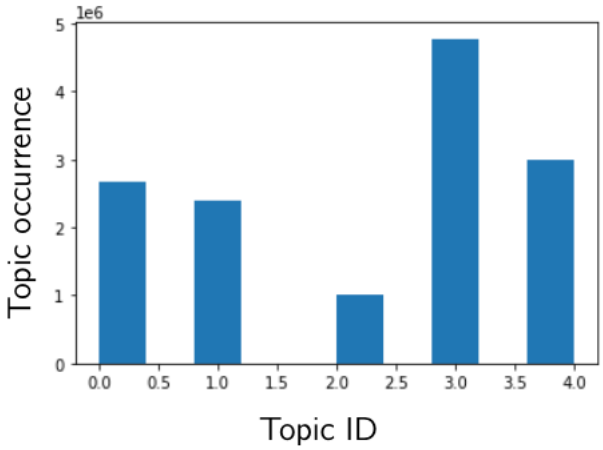


Figure 3.6: a distribution of the topics present in the entire ASME corpus

```
[(0,
  '0.016*"flow" + 0.016*"flow" + 0.010*"case" + 0.009*"pressure" + 0.009*"high" + 0.009
*"figure" + 0.008*"show" + 0.008*"wall" + 0.008*"surface" + 0.007*"result"'),
 (1,
  '0.022*"blade" + 0.020*"rotor" + 0.018*"flow" + 0.015*"pressure" + 0.011*"design" +
0.011*"figure" + 0.010*"compressor" + 0.008*"show" + 0.008*"stage" + 0.008*"speed"'),
 (2,
  '0.015*"fuel" + 0.013*"combustor" + 0.013*"combustion" + 0.012*"temperature" + 0.012
*"flame" + 0.011*"air" + 0.009*"flame" + 0.009*"pressure" + 0.009*"high" + 0.009*"veloc
ity"'),
 (3,
  '0.015*"model" + 0.008*"method" + 0.008*"figure" + 0.006*"result" + 0.006*"time" + 0
.006*"value" + 0.006*"base" + 0.005*"analysis" + 0.005*"show" + 0.005*"system"'),
 (4,
  '0.016*"temperature" + 0.013*"turbine" + 0.012*"engine" + 0.012*"system" + 0.011*"po
wer" + 0.010*"pressure" + 0.010*"design" + 0.009*"high" + 0.008*"figure" + 0.008*"perf
ormance"')]
```

Figure 3.7: the meanings of each of the $m = 5$ topics

The choice of m in the LDA model can be optimised by trying to go for topics that have little overlap with one another, but also how semantically similar the top words in a topic are[10]. The former is defined by the **Jaccard** score,

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{10}$$

whereas latter is defined by the Coherence score (this is calculated in `gensim`). A good topic would thus minimise the Jaccard and maximise the Coherence. Figure 3.8 below shows the optimal m in the range $\{0,14\}$.

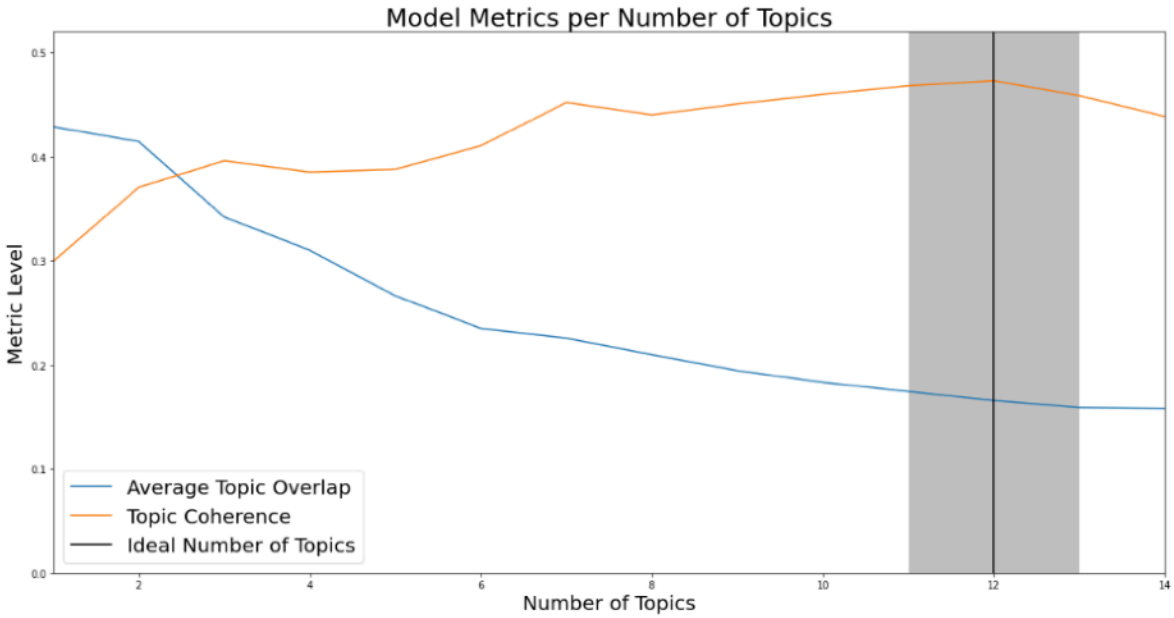


Figure 3.8: the distribution of the Jaccard and Coherence scores for $m \in [1,14]$

3.3.3. Benchmarks for Novelty Detection

Before creating the main novelty detection models, two benchmarks results were created to get a point of comparison.

The first used the pre-trained LDA model to extract document embeddings vectors for each document, v_i, and the average corpus embedding was found:

$$\bar{v} = \frac{\sum_i^N v_i}{N} \tag{11}$$

[10] For example, {"Baby", "Turbomachinery", "Mortal Kombat"} would be incoherent, whereas {"Turbomachinery", "Blade", "Cooling"} would be more coherent.

The distance of each document from the mean, $d_i(v_i, \bar{v})$ was found, and the papers were ranked in order of closest to furthest from the mean. Figure 3.8 below shows the 5 closest and furthest documents.

	doc_id	distance_from_avg	doc_names
363	363.0	0.068030	GT2007-27909.pdf
466	466.0	0.071759	GT2007-28115.pdf
395	395.0	0.076959	GT2007-27960.pdf
53	53.0	0.086042	GT2007-27387.pdf
527	527.0	0.094285	GT2007-28248.pdf
...	...	...	...
76	76.0	0.728559	GT2007-27419.pdf
52	52.0	0.739405	GT2007-27386.pdf
54	54.0	0.743983	GT2007-27388.pdf
127	127.0	0.757386	GT2007-27491.pdf
975	975.0	0.765162	GT2008-50606.pdf

Figure 3.9: Red indicates 5 most dissimilar documents (furthest) and Green indicates the 5 most similar documents (closest)

These results were poor, as might be expected since the method only considers high level semantics. To understand the results a bit further, the documents were taken and vectored using `TF-IDF`. A similarity matrix was plotted to examine the document relationships further, as shown in Figure 3.9.

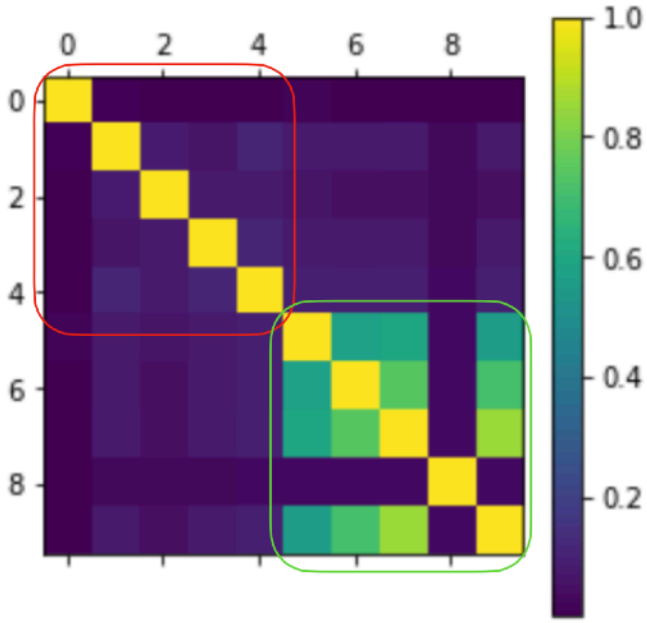


Figure 3.10: a similarity matrix for the 5 most dissimilar documents (red) and the 5 most similar documents (green)

The results indicate that the documents close to the mean (green) are fairly similar to one another, whereas the documents far from the mean (red) are vastly different. This was more insightful and agreed with a visual inspection of the papers[11].

The second method of defining novelty slightly more promising results. Instead of finding the difference compared to the mean, the cosine similarity between the `TF-IDF` vectors of the corpus C and a query document D_{test} were found. This was tested on a sample text from Harry Potter and also on sample documents from the corpus C. It was found that the variation within C was negligible, whereas Harry Potter was easily discernible. Though this method cannot finding novelty within the main corpus, it can at least act as a filtering stage to remove any documents that are *irrelevant*.

3.4. Pseudo-Supervised Novelty Detection

The data for this project is unsupervised. However, successful novelty detection models rely on supervised data [49]. As a result, a pseudo-supervised approach was used for the first model. The model used is based on an open world classification problem presented by Qin et al. [49]. This type of problem tries to predict **unseen** classes from a set of

[11] The 5 most similar documents were about ceramics, whereas the 5 most dissimilar were about different topics each, and very different from ceramics, e.g. jet engine

previously **seen** classes. In this context, we can split our documents into sentences[12], and for each we predict a topic[13] using the pre-trained LDA model from Section 3.3.2. The ensuing dataset is pseudo-supervised, since the labels were not pre-labeled. Given this formulation, the open classification problem seeks to determine if a new sentence would belong to one of the topics already seen, or a completely new one.

To do this, the unique combinations of sentence pairs are found, giving a matrix $\underline{\underline{S}} \in \mathbb{R}^{\mathcal{N} \times 2}$, which each datapoint representing a pair of sentences. A new feature $\underline{c} \in \mathbb{R}^{\mathcal{N}}$ is constructed, such that:

$$c_k = \begin{cases} 1 & \text{if topic}\left(S_{k1}\right) = \text{topic}\left(S_{k2}\right) \\ 0 & \text{otherwise}. \end{cases} \tag{12}$$

A neural network (see Section 3.4.1) is trained to perform a binary classification, i.e. predicting for a pair of sentences $\{S_{k1}, S_{k2}\}$ the probability that they both have the same topic. A summary of the training pipeline is given in Figure 3.10 below.

[12] Instead of tokenizing documents by words, this tokenizes them by sentence, e.g. sentencize. `nltk` (manual) , `stanza` and `spacy` sentencizers were used and it was found that spacy had the best combination of accuracy and speed. See #12 for more detail.

[13] These would be our classes

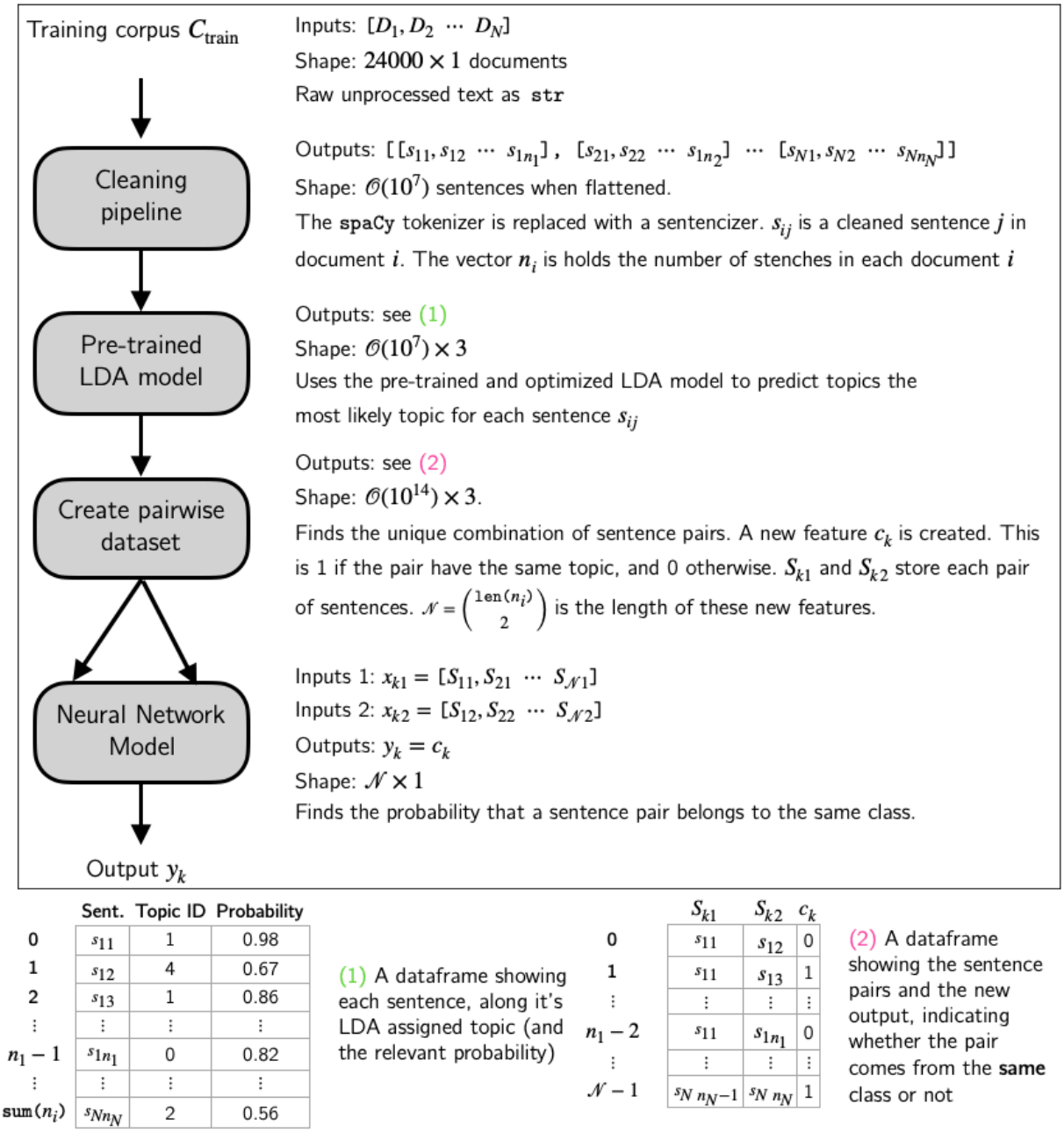


Figure 3.11: a diagram of the training procedure

In order to predict whether a test sentence, s_{test} is from an unseen topic (i.e. it is *novel*), one would select K samples from each topic and pair them with s_{test}. Each pair would be fed into the trained network, giving the probability that s_{test} has the same topic as the other sentence. The probabilities are averaged across each topic, giving $P_m = P(s_{\text{test}}, m)$, the probability that s_{test} belongs in topic m for each m. The novelty of the sentence is then determined by the following:

$$\varphi(s_{\text{test}}) = \begin{cases} 1 & \text{if } \max(P_m) < \lambda \\ 0 & \text{otherwise}. \end{cases} \tag{13}$$

Where λ is a user chosen threshold. The test procedure is summarised in Algorithm 2.

Algorithm 2 Determine document novelty

1: **Foreach** D_i in C_{test}
2: **Forach** s_{ij} in D_i
3: Calculate $P_m = \frac{\sum_p^K \hat{y}_p}{K}$ for each m
4: Calculate φ_j
5: Calculate document novelty $\sum_j \varphi_j$
6: **Rank** documents based on document novelty

- D_i is a document
- s_{ij} is a sentence in document D_i
- $\hat{y}_p$ is a subset of $\hat{y}_k$ where the topic is m. P_m is a vector
- 1 if the sentence s_{ij} is novel, 0 otherwise
- $P(w_j|m)$ is the proportion of word w_j in topic m
- $P(m|w_j)$ is the conditional probability of w_j belonging in m

3.4.1. Neural Network Structure and Performance

The neural network chosen for the binary classification task is based on Yoon Kim's 2014 CNN network [47], but modified to take a pairwise input. This model was chosen because despite its simplicity, it has superior performance in classification tasks. Figure 3.11 below shows a schematic of the model developed in `keras` and `tensorflow`.

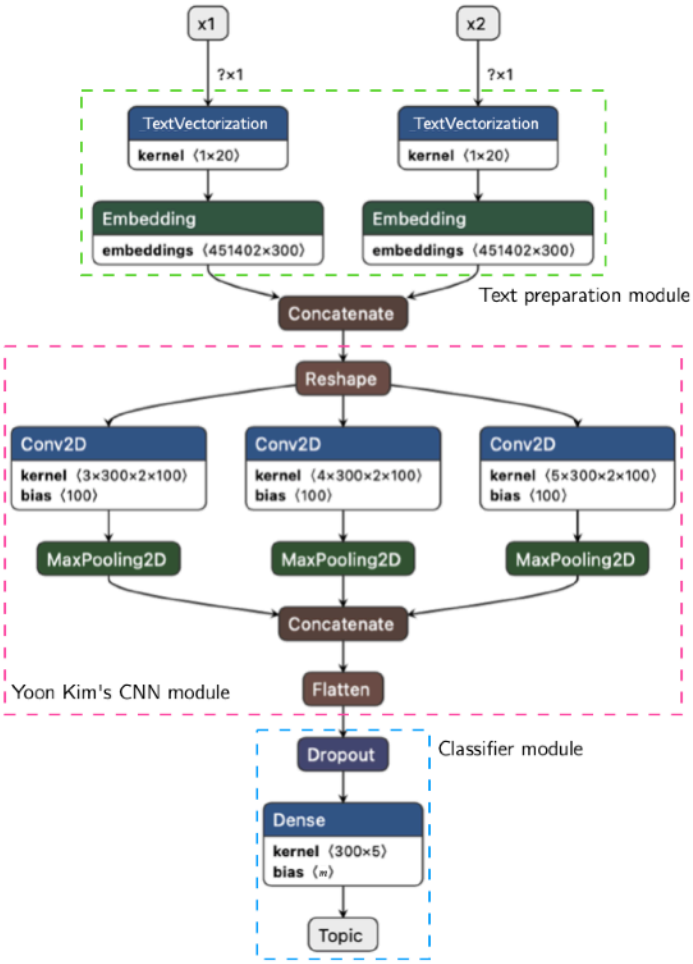


Figure 3.12: a schematic of the pairwise CNN module

The vectorizer and embedding layers were added to ensure that the model is end-to-end[14]. These layers are pre-trained for the ASME corpus. Python files were written for creating these layers based on user inputs. This enables users to create similar embedding and vectoriser layers for different corpora or word embeddings. The current implementation uses the `GloVe` word embeddings, since these were found to capture document level semantic well.

It's important to note that the `TextVectorization` layer converts sentences into fixed length vectors. This is necessary, as otherwise the sizes of the inputs into the model would differ, which would make matrix multiplication difficult. A cutoff length is chosen by the user, which sets a cap for the maximum sentence length that can be captured. This implementation chose a cutoff length of 20 as a reasonable, since it covers the vast majority of sentences (see Figure 3.13).

[14] So that the user need not worry about using external vectorisers

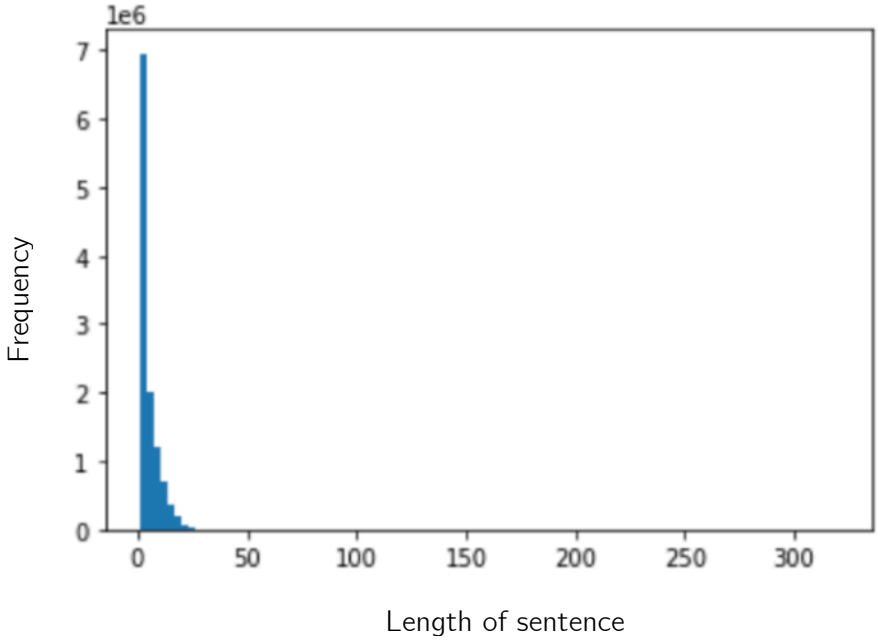


Figure 3.13: distribution of sentence lengths in C

3.4.2. Computational Considerations

The training dataset contains about 14 million sentences, approximated by $\mathcal{O}(10^7)$. The pairing operation would be:

$$\binom{\mathcal{O}(10^7)}{2} \approx \mathcal{O}(10^{14})$$

Even with a liberal estimation of each input being 1 byte, that means a total memory requirement of 200 terabytes, rendering pre-built combination algorithms useless. As a result, a combination algorithm that takes batched inputs[15] was designed. This was designed to be as efficient as possible. The process was negligible compared to to the model training time. The dataset size per batch is given by:

$$\mathcal{N}_{\text{batch}} = \frac{b}{2}\left(2n - b - 1\right) \tag{14}$$

Where b is the batch size, and n is the number of total sentences, approx $\mathcal{O}(10^7)$.

The size of the dataset increases exponentially with the batch size, as shown in Figure 3.12 below.

[15] Note that this is different to the `batch_size` that is given to the neural network during training.

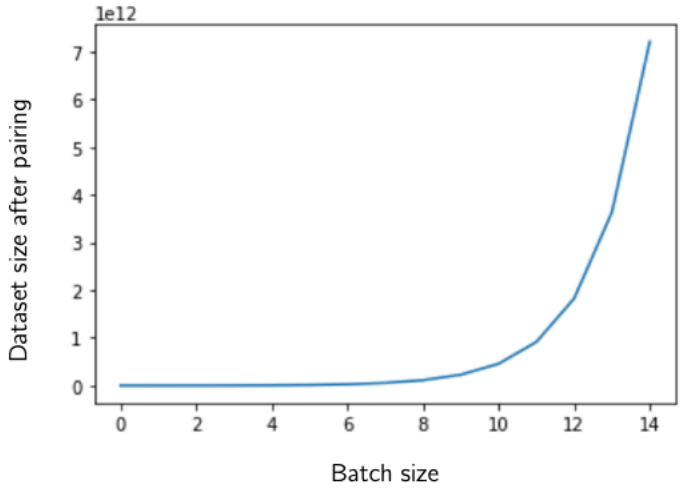


Figure 3.14: dataset size as a function of batch size

So, although the plot above suggests using the tiniest batch size, i.e. $b = 1$, one must also consider that the training process will be interrupted more frequently, and that a balance is required.

3.5. Unsupervised Novelty Detection

The unsupervised method proposed in this project for detecting novel documents is `IsolationForest` from `sklearn`. This is in principle, an outlier detection method that creates boundaries between normal and abnormal observations. Its behaviour is mainly controlled by three hyper parameters, namely:

- `contamination`, which represents the proportion of outliers in the sample
- `max_features`, which defines the number of features to use for detecting outliers
- `n_estimators`, which is the number of weak learners used to make predictions

The author has successfully used this algorithm for outlier detection in a highly imbalanced dataset [62]. The intuition behind using this model for novelty detection is that novel documents will be underrepresented, and should thus more appropriately be treated as outliers (or anomalies). `IsolationForest` will be used on document embeddings comprised of the average word embeddings created by `GloVe` and `TF-IDF`.

Optimising unsupervised algorithms is always hard because there is no 'ground truth'. However, this can often be emulated by using datasets where one expects some difference.

As a result, four datasets are used to examine the performance of the isolation forest algorithm. These are described in Table 3.2.

Table 3.2: Testing dataset descriptions

Dataset	Details	Method
Hindawi XML Corpus	All Hindawi corpus papers	Visual inspection
IEEE "Deep Learning" Query Outcomes	Sample "Deep Learning" abstracts using the API extractor	Labelled as 'Novel'. Prediction accuracy measured.
Sample NLP papers cited in literature review	Sample NLP papers extracted using the PDF extractor module	Labelled as 'Novel'. Prediction accuracy measured.
Core "Turbomachinery" Query Outcomes	Sample "Turbomachinery" papers query using the API extractor	Visual inspection

4. Outcomes

4.1. Pseudo-Supervised Model

The pseudo-supervised model could not train on the entire dataset because of a walltime restriction on the College's HPC service[16], and therefore it has no outcomes. The only results are training curves for the model trained with a tiny subset of the data. This is shown in Figures 4.1 and 4.2.

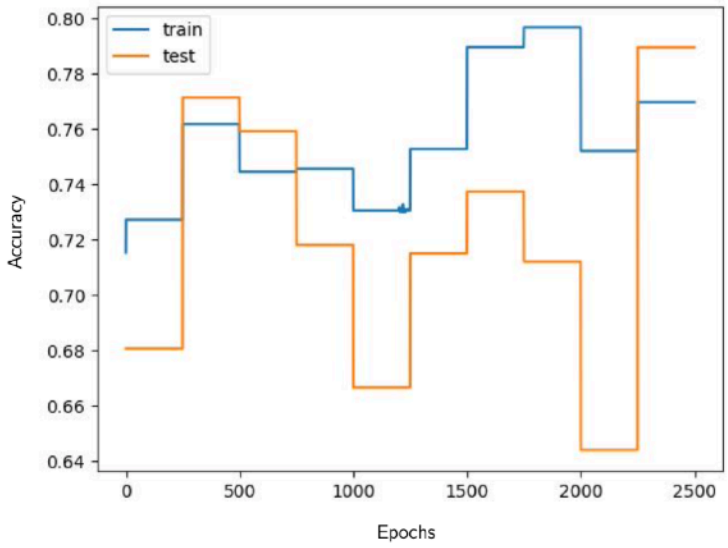


Figure 4.1: accuracy curves for a dataset size of $n = 200$ and a batch size $b = 20$

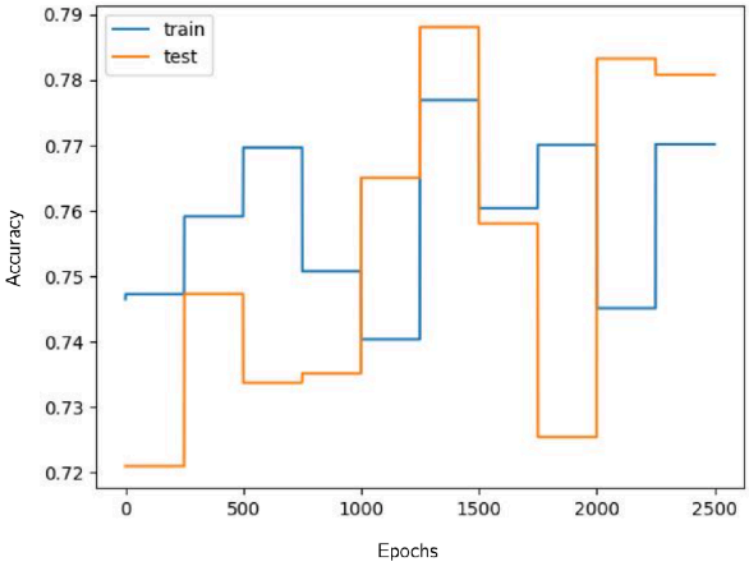


Figure 4.2: accuracy curves for a dataset size of $n = 1000$ and a batch size $b = 100$

[16] This is a cutoff time after which the training process is killed.

4.2. Unsupervised Models

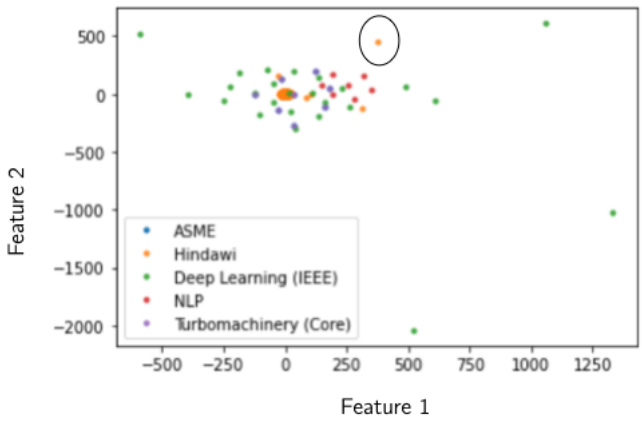


Figure 4.3: a t-SNE[17] plot of the `GloVe` word embeddings for an ASME sample, as well as the four test datasets. The black circle indicates one possible case of novelty.

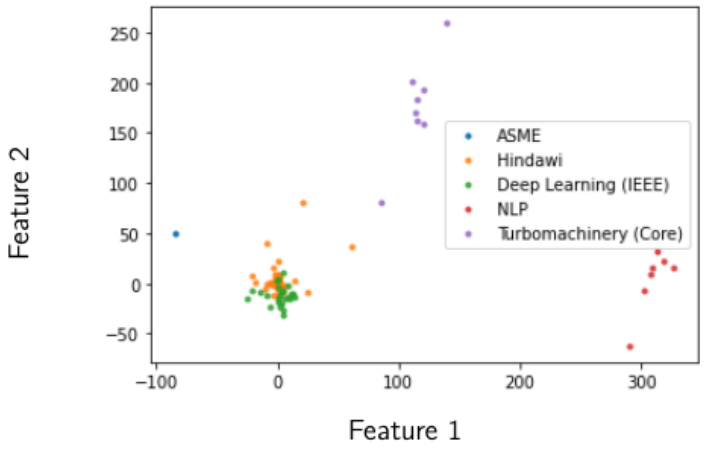


Figure 4.4: a t-SNE plot of the `TF-IDF` vectors for an ASME sample, as well as the four test datasets

[17] This is a dimensionality reduction algorithm that reduces the embedding vector to 2 dimensions for visualisation

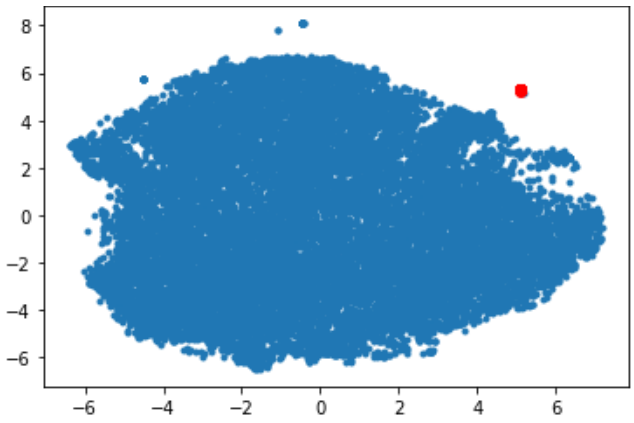


Figure 4.5: a t-SNE plot of the `GloVe` vectors for the entire ASME corpus, showing the novel documents predicted by `IsolationForest` with `n_estimators=50`, `contamination=0.001`, `max_features=1000`

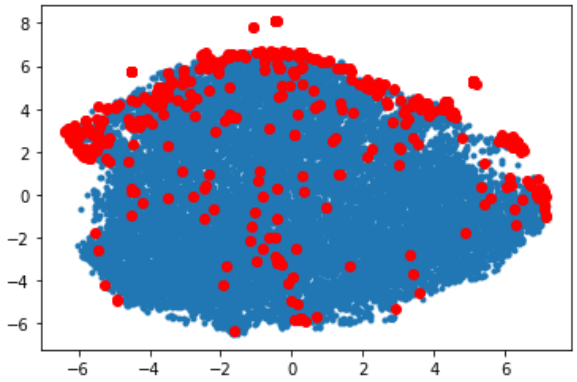


Figure 4.6: a t-SNE plot of the GloVe vectors fo the entire ASME corpus, showing novel documents predicted `IsolationForest` with `n_estimators=50`, `contamination=0.03`, `max_features=1000`

5. Discussion and Future Work

5.1. Data Quality and Limitations

The quality of the data extracted from the PDFs was not great. This was apparent from the size of the corpus vocabulary which is around 400000; a number significantly higher than the size of the English dictionary (around 170000 [63]). Further, around half of the words in the corpus were not found in the `GloVe` embeddings; a full list of these words can be found in the file nonexistent.txt on GitHub. A visual inspection of these words found a number of distinct classes, the main ones are:

- Human made spelling mistakes, e.g. $cap\underline{a}turing$ instead of $capturing$
- Incorrectly read characters, e.g. $\underline{c}apyright$ instead of $copyright$
- Words from foreign languages, etc. $brennkammersystem$
- Mathematical symbols and equations, e.g. $bsiny$ instead of $b \sin y$
- Phrases that were captured without whitespace, e.g. $burnerloudspeakersetup$
- Named entities that weren't removed, e.g. $mohammadpour$ (Persian surname)
- Technical terms, e.g. $eulerian$
- Different fonts, e.g. see Figure 5.1 below.

Figure 5.1: words without GloVe embeddings due different fonts

The most concerning of the above are the latter two points, because they risk ignoring important information relevant to the research paper. This is of particular importance for the task of novelty detection in scientific research papers, where often times the relationships between technical terms are sought out.

Another thing to note is that the despite the cleaning pipeline, the data had elements of systematic noise. For instance, the words {$"proceedings"$, $"figure"$, $"diagram"$} were exceedingly common. This makes sense, since almost all papers have figures and diagrams, and many are conference papers. Be that as it may, the words contain no semantic information about the research conducted.

Further to this, the Hindawi corpus was fairly limited, as many of the documents are incomplete or mostly written in `MathML`, rendering them useless for NLP.

Finally, the average sentence length is around 4.5 words, which sounds short. More research needs to be done into the cleaning mechanism to make sure that informative words are not deleted.

5.2. Pseudo-Supervised Model

5.2.1. Performance

The Pseudo-Supervised model did not train because the training time would exceed the maximum allowed walltime. Thus the model was only tested with $n = 200$ and $n = 1000$ datapoints (before the pairwise operation). The training curves for both these results are difficult to interpret. For both cases, the model perform well when judged by accuracy. However, they have concerning training curves that are difficult to analyse (see Figure 4.1 and 4.2). These curves are characterised by jumps and phases of constant accuracy; behaviour that is uncommon in training curves. This might be attributed to the fact that many of the vectors are sparse, due to the large number of words that exist outside the `GloVe` embeddings. The models were trained for a second time on a dataset stripped of all the words in **nonexistent.txt**, and with a new embedding matrix whose sparse rows were removed; these have unfortunately been training on the HPC for over a week. The fact that the validation accuracy exceeds the training accuracy in both is also suspicious. This is typically indicative of unrepresentative (or little) training data. Unfortunately, test cases with $n > 1000$ have consistently failed to converge within the allowed walltime.

Another consideration with this model is that its Pseudo-Supervised nature was not rigorously analysed. It was assumed that the LDA, which generates topics at a corpus level would also be able to accurately predict topics at a sentence level. In principle this is possible [64], however it depends on the hyper parameters of the model which affect the degree of topic overlap. Further, when the pseudo-supervised dataset was created, the degree of confidence with which the LDA model classified sentence topics was not considered. A quick distribution of the probabilities used with $m = 5$ topics shows that the degree of confidence is not very high (see Figure 5.2).

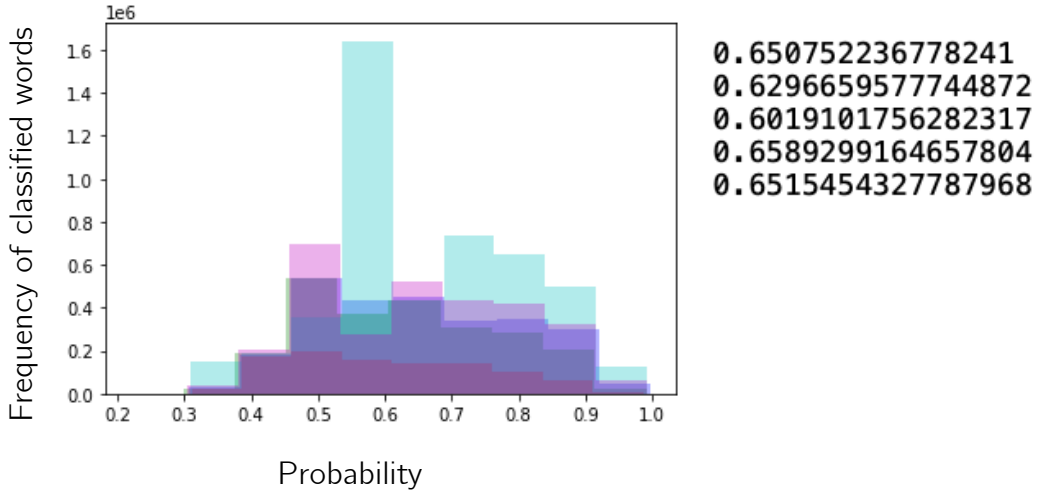


Figure 5.2: the distribution of the probabilities with which the LDA classified the topics, as well as the mean probabilities per topic

It's worth noting that a substantial number of the sentences were classified with very low probabilities $\mathcal{O}(0.3)$. The average probabilities are also fairly low. A solution to this would be setting a higher threshold (for example, 0.8) required to classify a sentence as a topic. This would also have an effect of reducing the computational costs of this model, which is discussed in the next subsection.

It is difficult to conclude without further testing whether this model works as intended to determine document novelty.

5.2.2. Computational Considerations

The weakest aspect of this model is that it is highly resource intensive. Unlike in the original implementation [49], the model here is being applied at a much larger scale. The $n = 1000$ model would take 4700 seconds to train using formidable computation settings[18]. The College does have options for faster computation, but then a human cost is incurred, because the models get queued for much longer.

It's also worth noting that the model struggled to train after the text had been processed and prepared. In a fully end-to-end pipeline, text cleaning and labelling using the LDA would also be part of the pipeline. The latter does not support GPU acceleration, and may act as a bottleneck.

[18] 4 RTX GPUs and 1 XLA enabled processor

There are obvious and necessary improvements that need to be made to the model before it is to be considered a serious novelty detection algorithm, irrespective of the accuracy of the open world classifier. These will be discussed in greater detail under Section 5.4.

5.3. Unsupervised Models

5.3.1. Performance

The performance of these models was not much better than that of the benchmarks. However, the visualisations using t-TSNE provided interesting insight. From Figure 4.3, one can notice that the `GloVe` embeddings (despite their wide use in the literature) were not able to discriminate between different datasets, whereas `TF-IDF`, an antiquated statistical method was able to do so somewhat well. However, it's worth mentioning that with regards to novelty detection, the `GloVe` plot is more reassuring as because it correctly discriminated a document that could be considered novel.

These plots suggest that `TF-IDF` is best as a pre-processing feature that can remove irrelevant documents, and that neither `TF-IDF` nor `GloVe` embeddings can consistently point to novel papers.

The `IsolationForest` algorithm was found to perform poorly. The results would either: a) point to obvious clusters that could be seen by the human eye, as seen in Figure 4.5, while ignoring other possibly novel documents, or b) label many non-novel documents as novel.

An important conclusion to draw is that unsupervised novelty detection is difficult, and that robust ground truth datasets are needed to verify the outcomes.

5.3.2. Limitations

The limitations of this method are likely due to the quality of the data (much of which was corrupted) and the lack of highly informative document vectors. The method used here simply averages word vectors in order to a get a document representation, which does not capture richer semantic information.

5.4. Further Work

For those wishing to continue this work, here are a few recommendations:

- **Data Enhancement:** the cleaning process should be enhanced to improve a `TF-IDF` filtration function to remove corpus specific noise (e.g. $proceedings$); spell checking algorithms can be used to improve data quality; custom word vectors can be trained (or SOTA models such as `BERT` can be used)
- **Improved document representations:** in addition to using better word level representations, future researchers should look into document representations that capture content better
- **Memory and Computation:** the pipeline can be re-written to use `tensorflow`'s `tf.Dataset` module to enable better GPU acceleration (this includes creating the LDA model within `tensorflow`, and parallelizing the PDF extraction method)
- **Other ideas:** researchers are encouraged to explore better methods for document representation for the unsupervised model; particular emphasis is given on exploring new definitions of novelty that leverage the graphical structure of documents; other ideas include using paper bibliography to determine links between different papers as an extra feature for analysis; researchers are also encouraged to follow the research presented in Section 2.3.3.

The GitHub page for this project discusses further work with regards to implementation.

6. Conclusion

Two novelty detection models (one pseudo-supervised and one unsupervised) have been created for detecting novelty in research papers. The latter is pseudo-supervised because the data is labeled by means of a statistical topic modelling tool known as Latent Dirichlet Allocation (LDA). The unsupervised model uses `IsolationForest` to look for anomalies (i.e. novel documents) in an embedding space.

The unsupervised model showed predictable performance but nothing spectacular, as it simply picked up on isolated clusters which were divisible to the human eye as well. The pseudo-supervised model produced no results, due to bottlenecks in the training process. This model, based on the pairwise implementation by [49] is memory and computationally intensive.

Future developments should prioritise **Data Enhancement** (better text extraction, processing and correction), **Document Representation** (use of state-of-the-art document embedding techniques) and re-thinking the **definition** of novelty as dissimilarity given a document's relevant. Researchers are referred in the direction of graphical representations of text in order to extract inter-document links that could define novel research areas.

7. Reference List

[1] Minaee S, Kalchbrenner N, Cambria E, Nikad N, Chenaghlu M, Gao J. Deep Learning Based Text Classification: A Comprehensive Review. *Arxiv* [Preprint] 2020. Available from: 10.1145/nnnnnnn.nnnnnnn

[2] Devlin J, Chang M-W, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Computation and Language Arxiv* [Preprint] 2019. Available from: 1810.04805v2

[3] lsalles23, namiyousef, MWarnerImperal, MatthewRWarner, jordanpatel97. ContentMining. 2020. https://github.com/lsalles23/ContentMining.

[4] Palachi S. *Document Embedding Techniques*. Available from: https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#fea6. [Accessed 25th May 2021]

[5] Madan R. *TF-IDF/Term Frequency Technique: Easiest explanation for Text classification in NLP using Python (Chatbot training on words)*. Available from: https://medium.com/analytics-vidhya/tf-idf-term-frequency-technique-easiest-explanation-for-text-classification-in-nlp-with-code-8ca3912e58c3. [Accessed 20th May 2021]

[6] Wikipedia. *tf-idf*. Available from: https://en.wikipedia.org/wiki/Tf–idf. [Accessed 31st October 2020]

[7] Patel K, Bhattacharyya P. Towards Lower Bounds on Number of Dimensions for Word Embeddings. In: *Proceedings of the The 8th International Joint Conference on Natural Language Processing*, *November 27 – December 1, 2017*, *Taipei, Taiwan*. 2017. p.31-36.

[8] Mikolov T, Chen K, Corrado G, Dean J. Efficient Estimation of Word Representations in Vector Space. *Computation and Language Arxiv* [Preprint] 2013. Available from: 1301.3781v3

[9] Kulshrestha R. *NLP 101: Word2Vec — Skip-gram and CBOW*. Available from: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314. [Accessed 20th April 2021]

[10] Kim E. *Demystifying Neural Network in Skip-Gram Language Modeling*. Available from: https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling. [Accessed 1st May 2021]

[11] Alammar J. *The Illustrated Word2vec*. Available from: https://jalammar.github.io/illustrated-word2vec/. [Accessed 15th May 2021]

[12] Ganegedara T. *Intuitive Guide to Understanding GloVe Embeddings*. Available from: https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010. [Accessed 20th May 2021]

[13] Pennington J, Socher R, Manning CD. GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); October 25-29; Doha, Qatar2014. p. 1532-1543.

[14] Hui J. *NLP — Word Embedding & GloVe*. Available from: https://jonathan-hui.medium.com/nlp-word-embedding-glove-5e7f523999f6. [Accessed 20th May 2021]

[15] Gauthier J. *A GloVe implementation in Python*. Available from: http://www.foldl.me/2014/glove-python/. [Accessed 21st May 2021]

[16] Weng L. *Generalized Language Models*. Available from: https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html#bert. [Accessed 25th May 2021]

[17] Rizvi MSZ. *Demystifying BERT: A Comprehensive Guide to the Gorundbreaking NLP Framework*. Available from: https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/. [Accessed 15th May 2021]

[18] Huggingface. *Summary of the tokenizers*. Available from: https://huggingface.co/transformers/tokenizer_summary.html. [Accessed 25th May 2021]

[19] Gupta L. *Differences Between Word2Vec and BERT*. Available from: https://medium.com/swlh/differences-between-word2vec-and-bert-c08a3326b5d1. [Accessed 29th May 2021]

[20] Tifrea A, Bécigneul G, Ganea O-E. POINCARE´ GLOVE: HYPERBOLIC WORD EMBEDDINGS. *Computation and Language Arxiv* [Preprint] 2018. Available from: 1810.06546v2

[21] Zhang C, Gao J. Hype-HAN: Hyperbolic Hierarchical Attention Network for Semantic Embedding. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*. 2020.

[22] Amin F, Rafi M, Shaikh MS. Document clustering using graph based document representation with constraints. *Arxiv* [Preprint]. Available from: 1412.1888

[23] Malliaros FD, Meladianos P, Vazirgiannis M. Graph-based Text Representations: Boosting Text Mining, NLP and Information Retrieval with Graphs. In: *The WEB Conference*, *April 23, 2018*, *Lyon, France*. 2018.

[24] Karlsson V. Introducing a Hierarchical Attention Transformer for document embeddings [Masters]: KTH Royal Institute of Technology; 2019.

[25] Goodfellow I, Bengio Y, Courville A. Deep Learning: MIT Press; 2016.

[26] 3Blue1Brown. *Neural Networks*. Available from: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi. [Accessed 30th July 2020]

[27] Logan S. *Understanding the Structure of Neural Networks*. Available from: https://becominghuman.ai/understanding-the-structure-of-neural-networks-1fa5bd17fef0. [Accessed

[28] Amini A. *MIT 6.S191: Convolutional Neural Networks*. Available from: https://www.youtube.com/watch?v=AjtX1N_VT9E. [Accessed 15th March 2021]

[29] Prabhu. *Understanding of Convolutional Neural Network (CNN) — Deep Learning*. Available from: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed 20th May 2021]

[30] Britz D. *Implementing a CNN for Text Classification in TensorFlow*. Available from: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/. [Accessed 20th May 2021]

[31] Olah C. *Understanding LSTM Networks*. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 29th May 2021]

[32] Amini A. *MIT 6.S191: Recurrent Neural Networks*. Available from: https://www.youtube.com/watch?v=qjrad0V0uJE. [Accessed 10th May 2021]

[33] arvindpdmn. *Attention Mechanism in Neural Networks*. Available from: https://devopedia.org/attention-mechanism-in-neural-networks. [Accessed 20th May 2021]

[34] Weng L. *Attention? Attention!* Available from: https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html. [Accessed 20th May 2021]

[35] Bloem P. *Transformers from Scratch*. Available from: http://peterbloem.nl/blog/transformers. [Accessed 15th May 2021]

[36] Alammar J. *The Illustrated Transformer*. Available from: https://jalammar.github.io/illustrated-transformer/. [Accessed 15th May 2021]

[37] Kränkel M, Lee H-E. *Text Classification with Hierarchical Attention Networks*. Available from: https://humboldt-wi.github.io/blog/research/information_systems_1819/group5_han/#text-classification-with-hierarchical-attention-networks. [Accessed

[38] Yao L, Mao C, Luo Y. Graph Convolutional Networks for Text Classification. *Computation and Language Arxiv* [Preprint] 2018. Available from, https://arxiv.org/pdf/1809.05679.pdf [Accessed

[39] Panagiotou N, Akkaya C, Tsioutsiouliklis K, Kalogeraki V, Gunopulos D. First Story Detection using Entities and Relations. In: Matsumoto Y, Prasad R. (eds.) *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics:Technical Papers*, *December 2016*, *Osaka, Japan*. The COLING 2016 Organizing Committee; 2016. p.3237–3244.

[40] Petrović S, Osborne M, Lavrenko V. Streaming First Story Detection with application to Twitter. In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*, *June 2010*, *Los Angeles, California*. 2010. p.181—189.

[41] Ng KW, Tsai FS, Chen L, Goh KC. Novelty detection for text documents using named entity recognition. In: *2007 6th International Conference on Information, Communications & Signal Processing*, *10–13 Dec 2017*, *Singapore*. 2007.

[42] Suzuki S, Takatsuka H. Extraction of Keywords of Novelties From Patent Claims. In: Matsumoto Y, Prasad R. (eds.) *Proceedings of COLING 2016, the 26th International*

Conference on Computational Linguistics:Technical Papers, *December 2016*, *Osaka, Japan*. The COLING 2016 Organizing Committee; 2016. p.1192–1200.

[43] Omari A, Carmel D, Rokhlenko O. Novelty based Ranking of Human Answers for Community Questions. In: *SIGIR '16*, *17–21 July 2016*, *Pisa, Italy*. 2016.

[44] Cichosz P. Unsupervised modeling anomaly detection in discussion forums posts using global vectors for text representation. *Natural Language Engineering*. 2020;26(5). Available from: 10.1017/S1351324920000066

[45] Ghosal T, Edithal V, Ekbal A, Bhattacharyya P, Tsatsaronis G, Chivukula SSSK. Novelty Goes Deep. A Deep Neural Solution To Document Level Novelty Detection. In: Bender EM, Derczynski L, Isabelle P. (eds.) *Proceedings of the 27th International Conference on Computational Linguistics*, *August 2018*, *Santa Fe, New Mexico, USA*. Association for Computational Linguistics; 2018. p.2802–2813.

[46] Conneau A, Kiela D, Schwenk H, Barrault L, Bordes A. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, *7–11 Sep 2017*, *Copenhagen, Denmark*. 2017. p.670–680.

[47] Kim Y. Convolutional Neural Networks for Sentence Classification. *Computation and Language Arxiv* [Preprint] 2014. Available from: 1408.5882v2

[48] Ghosal T, Edithal V, Ekbal A, Bhattacharyya P, Satya S, Chivukula SK, et al. Is your document novel? Let attention guide you. An attention-based model for document-level novelty detection. *Natural Language Engineering*. 2020: 1—28. Available from: 10.1017/S1351324920000194

[49] Qin Q, Hu W, Liu B. Text Classification with Novelty Detection. *Arxiv* [Preprint] 2020. Available from, https://arxiv.org/abs/2009.11119 [Accessed 21st Jan 2021]

[50] Amorim M, Bortoloti FD, Ciarelli PM, Salles EOT, Cavalieri DC. Novelty Detection in Social Media by Fusing Text and Image Into a Single Structure. *IEEE Access*. 2019;7. Available from: 10.1109/ACCESS.2019.2939736

[51] Wang X, Jin B, Du Y, Cui P, Yang Y. One-Class Graph Neural Networks for Anomaly Detection in Attributed Networks. *Machine Learning Arxiv* [Preprint] 2020. Available from: 2002.09594v2

[52] Tian M, Guo D, Cui Y, Pan X, Chen S. Improving auto-encoder novelty detection using channel attention and entropy minimization. *Computer Vision Arxiv* [Preprint] 2021. Available from: 2007.01682v2

[53] Hendrycks D, Mazeika M, Dietterich T. Deep Anomaly Detection with Outlier Exposure. *Machine Learning Arxiv* [Preprint] 2019. Available from: 1812.04606v3

[54] Wang Q, Huang L, Jiang Z, Knight K, Heng J, Bansal M, et al. PaperRobot: Incremental Draft Generation of Scientific Ideas. *Arxiv* [Preprint] 2019. Available from, https://arxiv.org/abs/1905.07870 [Accessed 21st January 2021]

[55] Shi F, Chen L, Han J, Childs P. A Data-Driven Text Mining and Semantic Network Analysis for Design Information Retrieval. *Journal of Mechanical Design*. 2017;139(11). Available from: 10.1115/1.4037649

[56] Hammar Y. *Data Science Methodology (Or The Master Plan To Succeed Your DS Project)*. Available from: https://medium.com/@yassine.hammar1/data-science-methodology-or-the-master-plan-to-succeed-your-ds-project-d21397cf52b2. [Accessed 20th May 2021]

[57] Nami Y. Natural Language Processing based toolbox for detecting novelty in scientific research papers. Imperial College London. Report number: Progress Report, 2021.

[58] spaCy. *Language Processing Pipelines*. Available from: https://spacy.io/usage/processing-pipelines. [Accessed 20th May 2021]

[59] Qi P, Zhang Y, Zhang Y, Bolton J, Manning CD. Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. *Computation and Language Arxiv* [Preprint] 2020. Available from: 2003.07082v2

[60] Kulshrestha R. *A Beginner's Guide to Latent Dirichlet Allocation(LDA)*. Available from: https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2. [Accessed 21st May 2021]

[61] Hui J. *Machine Learning — Latent Dirichlet Allocation LDA*. Available from: https://jonathan-hui.medium.com/machine-learning-latent-dirichlet-allocation-lda-1d9d148f13a4. [Accessed 20th May 2021]

[62] Nami Y. *Anomaly Detection in Significant Movements*. Available from: https://github.com/namiyousef/Kin-Keepers/blob/master/Anomaly%20Detection%20in%20Significant%20Movements/Summary.ipynb. [Accessed

[63] Interpreters and Translators I. *Which Language Is Richest In Words?* Available from: https://blog.ititranslates.com/2018/03/07/which-language-is-richest-in-words/. [Accessed 3rd Jun 2021]

[64] user0. *Can LDA model be useful for sentences (not documents) clustering / classification?* Available from: https://stackoverflow.com/questions/46900543/can-lda-model-be-useful-for-sentences-not-documents-clustering-classificatio. [Accessed 3rd June 2021]