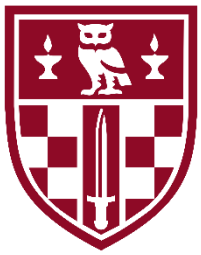
Birkbeck, University of London

MSc Data Science Project Report

Automated Patent Classification by type of Innovation for Clean Energy Storage Technologies

Student:
Roberto Casaluce

[redacted]



DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

Abstract

Around the world, hundreds of thousand patents are granted each year. A single person, group of persons or legal entities can apply to a patent office to gain the legal ownership on their inventions. An invention can refer to a new product, new ways to use a product or new ways to manufacture an existing product. Being able to distinguish among different innovations can help the policymakers to better implement a technological innovation. The project aimed to classify patents automatically and in a consistent way by their type of patented innovation. This end-to end project, where I employed two different word representation methods, TF-IDF and word embeddings methods, had to deal with an extremely unbalanced and limited dataset. Consequently, I used a synthetic minority oversampling technique (SMOTE) to compensate for the imbalance and I investigated alternative ways to increase the performance of the models in response to the lack of labelled data. The results of my classification models showed how well ensemble algorithms such as Random Forest and XGBoost performed for such text classification problem. I trained three classification models, two binary models, to classify both claims and abstracts, and a multiclass model only for the claims. At first, only the two models that classify claims were able to outperform the benchmark models, but, after implementing the keyword search method to increase the amount of labelled data, my third classification model was able to outperform the benchmark for the abstracts too with just a slightly better performance of the benchmark on the negative class.

Table of Contents

List of tables

List of Figures

Amendment from the proposal

There are only two amendments from the original proposal submitted. In the proposal, I wrote that I would conduct the labelling phase with a student hired from the Energy and Policy Group of ETH Zurich. Instead, it was decided that two external research assistants would complete this task. The second amendment refers to the number of classification models that I would train for my project. In the proposal I wrote that I would only train two models, while instead I have trained an additional model, thus a multiclass classification model. This is because it was interesting to investigate how well the model would classify the claims of patents also when adding a third class to the model.

1. Introduction

Around the world, hundreds of thousand patents are granted each year. A single person, group of persons or legal entities can apply to a patent office to gain the legal ownership on their inventions. An invention can refer to a new product, new ways to use a product or new ways to manufacture an existing product. Knowing where the technological innovation resides, thus whether it resides in the design of the product or in the way in which the product is manufactured, can help to understand the complexity of this innovation. The policymakers need to grasp the complexity of this innovation to better implement new policies that can efficiently promote a new technological innovation. Because of the increasing number of patents granted, there is an increasing need to find ways to be able to classify patents according to their type of innovation in a more automated fashion. Previous studies on the classification of patents as product and process relied only on keyword search methods and only very recently a team of researchers employed Natural Language Processing (NLP) methods to automate this classification.

The aim of the present project is to employ Natural Language Processing (NLP) methods to automate the classification of patents according to their type of innovation with a focus on four storage energy technologies. For this end-to-end project, where I used two different word representation methods, TF-IDF and word embeddings methods, I was able to find the best models for this classification problem by choosing them from different learning algorithms with the test harness with can execute multiple models at the same time. This objective was achieved although I had to deal with an extremely unbalanced and very small dataset, whose classes I balanced through a synthetic minority oversampling technique (SMOTE). With regard to the low number of observations, I have investigated alternative ways to increase the performance of the models, such as creating new pseudo labels on an unlabelled dataset, increasing the number of observations using keyword search methods and increasing the accuracy by using data augmentation methods.

The present report is divided into four sections. The first section is dedicated to introducing the project and the previous study, that I used as a benchmark for my models, since it focused on the same classification problem. The second part is dedicated to the data, the sources from which I collected the patents and how the labelling phase has been conducted. The methodology employed for this project is described in the third section. In that section, I first described how many classification models I trained and why I chose the ones I trained. After describing how I pre-processed the text, I illustrated the two word-representation methods employed in the project. Moreover, the methodology section continues with a description of how I split the data and why I did not split them in the same way for all the classification models. In the same section are described the selected learning algorithms and the reason why I chose to train those specifically and not other

algorithms, as well as the evaluation metrics used to evaluate the models. I also illustrated how those metrics were used when I needed them to spot-check the best two models to be trained. The methodology section continues with the description of the tuning of the hyperparameters for both the TF-IDF vectorizer and the learning algorithms. The last part of the methodology section is dedicated to describing the alternative ways I have investigated for this project to increase the performance of a model, when just a few labelled data are available. The fourth section is dedicated to the presentation of the results and to the comparison between them and the benchmark models. At the end of that section, a brief summary of the results in relation to future research directions follows to illustrate how classification models could be further improved.

Aim of the project

The present one is a Natural Language Processing (NLP) project for text classification applied to patents for four clean energy technologies, which is part of a bigger project led by the Energy Politics Group I temporarily joined from the Department of Humanities, Social and Political Sciences of the university ETH Zurich[1]. The main project aims to develop a quantitative approach to support innovation policy in a consistent, replicable and scalable way. Since technological innovation is one of the key ingredients of economic development, the policy makers that implement a technological innovation need to understand which instruments and industrial policy are the most effective to promote it. This can be only achieved by quantifying the complexity of a technology and by determining whether this complexity is in the design of the product or in the manufacturing process. Therefore, my task is to build an automated patent classification capable of classifying patents according to their type of innovation, which can be an innovation in the design of the product or of the process with which a final product is manufactured.

Benchmark

There are many previous works that classify patents by the type of innovation, but those works have focused their attention on a different way to classify patents, which mostly rely on keyword search methods (Bena & Simintzi, 2019). Only very recently a team from the KOF Swiss Economic Institute, while filling in the gap in the previous studies on the importance of process patents as well and their impact on a firm performance, developed a quantitative approach with NLP methods to classify patents in those two categories, process and product patents (Nicolas, et al., 2019). Since

[1] https://epg.ethz.ch/

they were the first to use NLP methods in their work on classification of patents in those two categories, that makes of their model the current state-of-art. They employed a logistic regression with elastic net regularization since they added lasso and ridge penalties to their learning model. The rationale behind the reason to train models similar to the ones trained by the KOF team is that, when selecting the patents to train their models, they did not distinguish them according to the area of technology, to which those patents belong. For the present project, I focused on four energy storage technologies. My results will be compared to those of the KOF team in the section dedicated to the findings. In the next section will be discussed the data sources from which I collected the patent data.

2. Data

Data collection

The sources of patents collected for the project are two patent offices, the European Patent Office (EPO) and United States Patent and Trade Office (USPTO). Only the granted patents have been selected for the project, since those patents went through an examination process prior to be granted, which guarantees that they meet certain standard. The EPO has made available their patents either as bulk data or via RESTful API service. The bulk data set can be requested by paying a subscription or can be download freely using Google Cloud[2]. EPO's RESTful API service, Open Patent Service (OPS), besides a daily limit requests for the free version, does not make available to download the abstracts of the patents. Therefore, I opted for downloading the bulk dataset from the Google Cloud platform. The size of the whole dataset is around 210 GB, divided in 36 txt files in which thousands of patents are stored in XML format. I mainly used regular expressions to find the correct patents, to clean the text from XML tags and to correctly split the claims for each patent.
For the USPTO patents, the office has made available the patents either as a bulk full text data or as TIFF image files. Since I had a limited number of patents to download, I decided to web scrape directly from the USPTO search page[3] the patents of the four technologies I needed. I used Python libraries to retrieve and parse the patents, such as the libraries 'Requests' and 'BeautifulSoup'. As for the EPO's patents, I mainly relied on regular expressions to clean the text data from HTML tags and to split the claims for each patent.

For both patent offices, the correct patent numbers for the four energy store technologies were retrieved using another dataset, a worldwide patent statistical database called PATSTAT Global, which is made available by EPO office. This database includes the biographic information of 100 million patents granted around the World. Some of the information included in this database are, for instance, the classification areas to which a patent belongs, information about the publication of the patents, their family technology IDs, etc[4]. To select the right patents for the four technologies, we selected the patent numbers corresponding to the classification scheme in which they are classified. The patents of the four technologies are classified in the Section Y of the

[2] Google cloud bucket https://console.cloud.google.com/storage/browser/epo-public/
[3] I run a test on 4th September 2020 using the web scraper again before saving the script on the repository and it seems that the US patent office does not allow anymore automatic web scraping.
[4] For a complete list of information included in the PATSTAT database see https://www.epo.org/searching-for-patents/business/patstat.html.

Cooperative Patent Classification (CPC). This section is dedicated to new technological development which includes the Subclass Y02, thus dedicated to applications or technologies directed at contrasting climate change (see Table 1 for the complete classification scheme of the four energy storage technologies)

Table 1 Classification Scheme for the four clean energy technologies

• Y	General tagging of new technological developments;			
	• Y02	Technologies or Applications for Mitigation or Adaptation Against Climate Change		
		• Y02E	Reduction of Greenhouse gas [GHG] Emissions, Related to Energy Generation, Transmission or Distribution	
		• Y02T	Climate Change Mitigation Technologies Related to Transportation	
			• Y02E 60/122 • Y02T 10/7011	Lithium-ion battery
			• Y02E 60/126 • Y02T 10/7016	Lead acid battery
			• Y02E 60/16 • Y02T 10/7033	Fly wheels
			• Y02E 60/13 • Y02T 10/7027	Supercapacitors

Not every patent in those categories was retrieved during the collection phase. For instance, some of the old USPTO patents do not have a HTML version, that can be found through the search page, but they only have a PDF version – the number of those patents are listed in the row Old patents Table 2. I had to exclude other USPTO patents from the final total number of patents that were web scrapped, because when I tried to split their claims using regular expressions, their HTML versions included typos. The most frequent typo was the lack of punctuation between same claims. This prevented me from splitting them correctly[5] – the number of those patents are listed in row Patents deleted in Table 2, but fortunately affected only less than 5% of the total USPTO patents. Some EPO patents also had to be excluded from the final count of the patents retrieved, because their abstracts were not in English.

Table 2 Patents collected

USPTO	Supercapacitors	Lithium-ion batteries	Lead acid batteries	Fly wheels
Number of patents PATSTAT	37,277	46,624	13,668	10,601
Old patents	1,485	1,012	1,434	1,400
Patents deleted	1,171	999	146	173
Total USPTO patents retrieved	*34,621*	*4,4613*	*12,088*	*9,029*
EPO				
Number of patents PATSTAT	4,421	7,093	1,389	700
Patents with no abstracts	1,709	2,360	640	432
Total EPO patents retrieved	*2,712*	*4,733*	*749*	*268*
Total per technology	**37,333**	**49,346**	**12837**	**9297**

[5] For instance, between claim 2 and 3 for the patent US 7790600, there is not a punctation that divide the two claims http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-adv.htm&r=1&f=G&l=50&d=PTXT&S1=7790600.PN.&OS=pn/7790600&RS=PN/7790600

Only some of the patents collected were labelled to train the classification models. In the next section will be discussed the labelling phase.

Labelling phase

The labelling of the data was conducted by two research assistants hired by the Department of Energy Politics of ETH Zurich. The assistants were instructed to conduct the labelling with detailed coding guidelines. The team from the Department randomly selected around 50 patents for each technology to be labelled. The patents that were labelled are 194, with a total of 194 abstracts and 4,857 claims. The claims and the abstracts were classified in one of the four classes, namely product, process, use claim and product by process – see Table 3 for a short definition of each of those classes.

Table 3 Labels and their descriptions

Labels	**Definition**[6]
Product	Novel concept or design for a technology.
Process	Novel production process for a technology.
Use claim	Discloses innovative use of a pre-existing technology.
Product by process	Describes a product defined in terms of a process of manufacture.

To check for intercoder reliability, the selected patents were given to both research assistants, so they would be manually labelled twice. After this procedure, the interrater reliability was high. The mean value of the four technologies was 0.9414, which means that around 94% of times the two assistants agreed on the label to assign to a claim or an abstract. Table 4 illustrates the number of observations that fall in one of the 4 classes for the two types of texts – claims and abstracts.

Table 4 Results labelling - Claims and Abstracts

Type of Text	**Product**	**Process**	**Use claim**	**Product by process**	**Total**
Claims	3809	740	307	1	4856
Abstracts	154	13	4	23	194

Another important distinction for the claims, which needs to be brought to the attention of the reader, is the distinction between independent and dependent claims. Dependent are the claims that refer to one or more independent claims that are located before it, whereas those which can stand alone and can be followed by one or more dependent claims are classified as independent – see claims in Table 5 for an example of dependent and independent claims.

[6] 1. WIPO, Patent claim format and types of claims
https://www.wipo.int/edocs/mdocs/aspac/en/wipo_ip_phl_16/wipo_ip_phl_16_t5.pdf

Table 5 Example of independent and dependent claims – Patent nr EP 526903

Type of Claims	**Claim number**	**Text Claim**
Independent	Claim 1	A bearing device having a rotary body (1) and bearing means for supporting the rotary body relative to a fixed portion in a noncontact state, the bearing device comprising a superconducting bearing assembly (2; 20; 44) which comprises an annular permanent magnet portion (11) fixedly mounted on the rotary body (1) concentrically therewith and an annular superconductor portion (8a, 8b) disposed as opposed to the permanent magnet portion, a magnetic bearing assembly (3) being positioned axially away from the superconducting bearing assembly (2; 20; 44) and including electromagnets (A1, A2, B1, B2; A3, A4, B3, B4) for controlling the position of the rotary body (1) with respect to two radial directions (X, Y) orthogonal to each other.
Dependent	Claim 2	A bearing device as defined in claim 1, wherein the magnetic bearing assembly (3) is disposed at one side of the superconducting bearing assembly (2).
Dependent	Claim 3	A bearing device as defined in claim 1, wherein the magnetic bearing assembly (3) is disposed at each of opposite sides of the superconducting bearing assembly (20).

This distinction is particularly relevant when the labelled data need to be split for training and testing purposes.

Since it was very soon apparent that the number of the labelled abstracts was not going to be high enough to have a binary classification with a good performance on both classes, I decided to investigate alternative ways to increase the number of labelled data. The results of this investigation are described in the methodology section.

3. Methodology

Classification models: Introduction

For the present work, I have trained three classification models: two binary classification models for both claims and abstracts and as described in the amendment section, a multiclass classification model of the claims only. In fact, the research team of the department of Energy Policy of ETH Zurich has decided that it would be interesting to train also a multiclass classification for both claims and abstracts since a sizable number of observations are labelled as *use claim*. This was only feasible for the claims and it is not possible for the abstracts, because of the lack of labelled data.

For the binary classification models, I used only two classes, namely *process* and *product*, and I merged into one of these two classes the remaining ones. I merged the observations labelled as *use claim* with the class *process* and the observations labelled *product by process* with the class *product*. Nicholas et al. (2019) did not merge the labels, but, when manually labelling their patents, they only distinguished between *product* or *process* claims, and labelled the claims that were supposed be *use claim* as *process* and *product by process* as *product*. Therefore, my choice to merge the labels is consistent with their strategy. For the multiclass classification for claims, I trained the model only with three labels, namely *product*, *process* and use *claim*. I removed the fourth label, product by process, because there was only one labelled claim in this class – see Table 4 in the previous section.

In the next section will be illustrated how I proceeded to prepare the text before training the classification models.

Data pre-processing

The text data needs to be pre-processed before being used to train a text classification model. This process makes sure that all the unnecessary symbols and words are removed from the text and helps to reduce the noise which is usually responsible for the decrease in the performance of a classification model. In this phase, the first step was to tokenize each of the documents with the NLTK library. I used the NLTK tokenizer because it gives the opportunity to set some rules on how the words have to be tokenized by using regular expressions. This was important because, when pre-processing text patent, it is not unusual to find two words with a specific meaning connected by a hyphen that need to be kept together, such as graphene-enhanced, lithium-ion. Without setting some rules, those words would be split in two separate words and lose their

meaning. In addition, thanks to the rule setting in the tokenizer, I could get rid at once of everything that was not a word or numbers, as for instance punctuations. After that, I converted all the upper-case words to lower-case ones and removed all the numbers. I also removed the stop words, which are the most common words in a language, such as articles or personal pronouns, with the list of stop words of the library NLTK. I could create my own list of stop words so that it could reflect more accurately the type of words in this type of text. Nevertheless, the feature extraction methods chosen here have parameters that could be set to eliminate the most common words even during the word representation phase[7]. The last pre-processing step for the text was to reduce the verbs to their root form, to their lemma. I chose to apply this transformation to the verbs only and not to other parts of the text, because changing words from plural to singular, for instance, would not have reduced the noise much. Figure 1 depicts how a claim is transformed after each of the pre-processing steps and in blue are the numbers of words before and after those steps.

Figure 1 Example of the pre-processing steps on a claim

Original[8]	An apparatus as in claim 4, wherein said non-linear generator means comprises: a plurality of non-linear generators; a drive shaft coupled to said power train for driving at least one of said plurality of non-linear generators; a magnetic clutch coupled to said drive shaft and to at least a second one of said plurality of non-linear generators to permit selective engagement and disengagement thereof. Length 64
Tokenize and **remove punctuation**	['An', 'apparatus', 'as', 'in', 'claim', '4', 'wherein', 'said', 'non-linear', 'generator', 'means', 'comprises', 'a', 'plurality', 'of', 'non-linear', 'generators', 'a', 'drive', 'shaft', 'coupled', 'to', 'said', 'power', 'train', 'for', 'driving', 'at', 'least', 'one', 'of', 'said', 'plurality', 'of', 'non-linear', 'generators', 'a', 'magnetic', 'clutch', 'coupled', 'to', 'said', 'drive', 'shaft', 'and', 'to', 'at', 'least', 'a', 'second', 'one', 'of', 'said', 'plurality', 'of', 'non-linear', 'generators', 'to', 'permit', 'selective', 'engagement', 'and', 'disengagement', 'thereof']
Lower case	an apparatus as in claim 4 wherein said non-linear generator means comprises a plurality of non-linear generators a drive shaft coupled to said power train for driving at least one of said plurality of non-linear generators a magnetic clutch coupled to said drive shaft and to at least a second one of said plurality of non-linear generators to permit selective engagement and disengagement thereof
Remove numbers	an apparatus as in claim wherein said non-linear generator means comprises a plurality of non-linear generators a drive shaft coupled to said power train for driving at least one of said plurality of non-linear generators a magnetic clutch coupled to said drive shaft and to at least a second one of said plurality of non-linear generators to permit selective engagement and disengagement thereof
Remove stop words	apparatus claim wherein said non-linear generator means comprises plurality non-linear generators drive shaft coupled said power train driving least one said plurality non-linear generators magnetic clutch coupled said drive shaft least second one said plurality non-linear generators permit selective engagement disengagement thereof
Lemmatization of the **verbs**	apparatus claim wherein say non-linear generator mean comprise plurality non-linear generators drive shaft couple say power train drive least one say plurality non-linear generators magnetic clutch couple say drive shaft least second one say plurality non-linear generators permit selective engagement disengagement thereof Length 42

In the next section, I will describe how I extracted the features from the plain text and why I chose those methods to complete this phase.

[7] For instance, if it needed it in the Skelarn-TFIDFVectorizer, a parameter can be set to eliminate words with a frequency higher than a threshold. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

[8] Patent publication number US 4204126 claim 5.

Feature Extraction

To be processed by a computer the text data needs to be transformed into meaningful numbers that the machine can understand. For the representation of the words, I chose two different methods. The first one, the Term Frequency Inverse Document Frequency (TF-IDF), is the most commonly used, since it is very fast to implement it and, most of all, it can give good results (Yun & Geum, 2020). The other word representation method is based on word embedding techniques and it is a bit more computationally expensive. Nevertheless, in most of the cases it has given very good results and even when first introduced it achieved the state-of-art results (Mikolov, Chen, Corrado, & Dean, 2013). This technique is known as Word2Vec.

Only one of these two methods was selected for that phase of the project in which the final classification models were trained. Indeed, during the spot-check phase, the best feature extraction method was selected, and it was employed in the final step of training the actual classification models.

TF-IDF

There are many reasons why I chose to use TF-IDF as word representation technique. As I stated above, the first one is that it is easy to implement and gives quite good results, but the most important reason is that also Nicolas and colleague used it (2019). This helps to make my results comparable with theirs.

TF-IDF consists of two elements. The first element is **Term Frequency**, known also as Bag of Words (BOW), which simply counts the frequency of terms in the text[9]. The second element is **Inverse Document Frequency**, which is a way to weigh the words and penalise the words that appear often in the text. idf is the logarithm of the number of the document in the collection – N – over the term frequency in the documents - dft.

Equation 1 $$idf_t = log^N_{df_t}$$

TF-IDF assigns to a term *t* a weight in a document that is highest when *t* occurs many times within a small number of documents, whereas the weight is lower when the term occurs fewer times in a document, or lowest when the term occurs in virtually all documents (Jurafsky & Martin, 2008).

[9] There is also a sublinear version of the Term Frequency value which is done by converting it with 1 + log(tf). During the tuning phase, one of the two versions is chosen.

Equation 2 $TF\ IDF_{t,f} = tf_{t,f} * idf_t$

In Figure 2, I added an example of how the idf weighs the most frequent words. I have highlighted in blue the term frequency of the word 'drive' and how it is converted to TF-IDF floating number.

Figure 2 Example of TF and TF-IDF vectorizations

Example:

Original abstracts patent nr. US 4254843 Fly wheel technology:

['An', 'electrically', 'powered', 'vehicle', 'having', 'a', 'bank', 'of', 'batteries', 'supplying', 'electricity', 'to', 'an', 'electric', 'motor', 'driving', 'wheels', 'of', 'the', 'vehicle', 'includes', 'a', 'whirl', 'ventilator', 'system,', 'a', 'clutch', 'system', 'and', 'an', 'automatically', 'operated,', 'engine-generator', 'unit', 'for', 'recharging', 'the', 'batteries.', 'The', 'whirl', 'ventilator', 'system', 'includes', 'housings', 'to', 'produce', 'a', 'whirling', 'air', 'flow', 'rotating', 'a', 'fan', 'to', 'drive', 'an', 'alternator', 'such', 'that', 'air', 'flow', 'from', 'movement', 'of', 'the', 'vehicle', 'generates', 'electricity', 'to', 'charge', 'the', 'batteries.', 'The', 'clutch', 'system', 'includes', 'a', 'clutch', 'mechanism', 'coupling', 'a', 'drive', 'shaft', 'and', 'a', 'driven', 'shaft', 'to', 'impart', 'a', 'driving', 'force', 'to', 'the', 'wheels', 'when', 'the', 'drive', 'shaft', 'is', 'rotated', 'faster', 'than', 'the', 'driven', 'shaft', 'and', 'to', 'couple', 'the', 'driven', 'shaft', 'with', 'a', 'flywheel', 'when', 'the', 'driven', 'shaft', 'is', 'rotating', 'faster', 'than', 'the', 'drive', 'shaft,', 'the', 'flywheel', 'driving', 'an', 'alternator', 'such', 'that', 'momentum', 'of', 'the', 'vehicle', 'causes', 'the', 'alternator', 'to', 'charge', 'the', 'batteries.', 'The', 'automatically', 'operated,', 'engine-generator', 'unit', 'is', 'started', 'to', 'charge', 'the', 'batteries', 'when', 'the', 'level', 'of', 'charge', 'therein', 'has', 'dropped', 'below', 'a', 'predetermined', 'level', 'and', 'stopped', 'when', 'the', 'charge', 'level', 'reaches', 'a', 'second', 'predetermined', 'level.']

After pre-processing the text:

['electrically', 'power', 'vehicle', 'bank', 'batteries', 'supply', 'electricity', 'electric', 'motor', 'drive', 'wheel', 'vehicle', 'include', 'whirl', 'ventilator', 'system', 'clutch', 'system', 'automatically', 'operate', 'engine-generator', 'unit', 'recharge', 'batteries', 'whirl', 'ventilator', 'system', 'include', 'house', 'produce', 'whirl', 'air', 'flow', 'rotate', 'fan', 'drive', 'alternator', 'air', 'flow', 'movement', 'vehicle', 'generate', 'electricity', 'charge', 'batteries', 'clutch', 'system', 'include', 'clutch', 'mechanism', 'couple', 'drive', 'shaft', 'drive', 'shaft', 'impart', 'drive', 'force', 'wheel', 'drive', 'shaft', 'rotate', 'faster', 'drive', 'shaft', 'couple', 'drive', 'shaft', 'flywheel', 'drive', 'shaft', 'rotate', 'faster', 'drive', 'shaft', 'flywheel', 'drive', 'alternator', 'momentum', 'vehicle', 'cause', 'alternator', 'charge', 'batteries', 'automatically', 'operate', 'engine-generator', 'unit', 'start', 'charge', 'batteries', 'level', 'charge', 'therein', 'drop', 'predetermine', 'level', 'stop', 'charge', 'level', 'reach', 'second', 'predetermine', 'level']

Feature names with dimension of 30:

['air', 'alternator', 'automatically', 'batteries', 'cause', 'charge', 'clutch', 'couple', 'drive', 'electricity', 'engine', 'faster', 'flow', 'flywheel', 'generator', 'include', 'level', 'momentum', 'operate', 'predetermine', 'rotate', 'shaft', 'start', 'system', 'therein', 'unit', 'vehicle', 'ventilator', 'wheel', 'whirl']

The term frequency (TF) vector:

[[2 3 2 5 5 3 ***11*** 2 2 2 2 2 2 3 4 2 2 3 7 4 2 4 2 2 3]]

The TF-IDF vector:

[[0.10355607 0.15533411 0.10355607 0.25889019 0.05177804 0.25889019 0.15533411 0.10355607 ***0.56955841*** 0.10355607 0.10355607 0.10355607 0.10355607 0.10355607 0.10355607 0.15533411 0.20711215 0.05177804 0.10355607 0.10355607 0.15533411 0.36244626 0.05177804 0.20711215 0.05177804 0.10355607 0.20711215 0.10355607 0.10355607 0.15533411]]

This word appeared 11 times in the document but when converted in the TF-IDF, it weighs less than just when counting for repetitions in the text.

Word embedding

The other feature extraction technique is based on word embedding methods. My intention was to build my own domain-specific word embedding using patents from both USPTO and EPO offices, as Risch and colleague (Risch & Krestel, 2019) did by training their model with around 5 million patents only with USPTO patents. I decided to train my word embedding models because I could

add also patents from the European office. In fact, I used around 3.5 million patents from the US office and around 500 thousand patents from the European office. This meant that I had to download more patents to train my domain-specific word embedding. For the EPO patents I did not need to download them because I already had the 36 full-text files downloaded from the Google Cloud platform. However, in this occasion, there was no need to select the patents from a list, but it was enough to select them according to the language in which they were published. Indeed, only patents written in English have been parsed from the text files and saved in a new text file, in which each row corresponds to a new document - claims and abstracts. On the contrary, for the USPTO patents, I had to download new patents. Nevertheless, in this occasion I did not web scrap the 3.5 million patents, because to do so would have been very much time consuming. Instead, I downloaded them from the weekly zip files with full-text patent data[10] that the USPTO has set available for research purposes. I chose the patents from 2005 and up to the end of June 2020 – the last zip file name that I downloaded is *ipg200630.zip.* The challenge was to find a way to download the zip files, extract their XML files and parse each one of them without running out of physical memory. In fact, the number of zip files that I downloaded is 756 with each a size of around 800 MB when uncompressed. This means 600 GB for all the files. I wrote a script that first downloaded the zip file, unzipped them and, from the XML included in it, extracted the abstracts and claims that I needed to store. Before downloading the following zip file, my script deleted both the zip and XML files of the previous passage. Once all the zip files for each year were downloaded and parsed, I saved the information needed in CSV files. After that, for each CVS files, I split the claims and, together with the abstracts, I saved them in text files with every abstract and claim in separate new lines.

For all patents, before saving their abstracts and claims in text files, I pre-processed their texts consistently with the steps followed to pre-process the text data to train the machine learning models. I removed all the XML tags, punctuations, stop-words, numbers, I converted the upper cases to lower cases and lemmatised the verbs. At the end of the collection and pre-processing phases, I merged all the text files from patent offices in a unique and very large text file – 21 GB.

The word embedding technique used in this phase is called Word2Vec and uses a neural networks model. There are two architectures within Word2Vec technique. In the one called Skip-gram (SG) for each word a hidden layer is trained to calculate the probability distribution of its neighbouring words. The number of neighbouring words is decided by the window size that moves across the text (Word2Vec Model - Gensim, 2020). The second architecture – Continuous Bag of Words (CBOW) – also trains a hidden layer but, rather than calculating the neighbouring words, it

[10] Patent Grant Full Text Data (No Images) (JAN 1976 - PRESENT) from https://bulkdata.uspto.gov/

predicts the centre word from them. I trained four different word embedding models – see Table 6 – and, to reduce the amount of time to train them, I added an optimization technique called negative sampling. Negative sampling avoids that the Word2Vec algorithm tweak thousands of weights for every trained hidden layer. It just randomly selects some of them and updates their weights (McCormick C. , 2020).

Table 6 List of Word2Vec models trained

Number of features	Windows size	Architecture model	Negative sampling	Minimum count word	Epoch
300	5	SG	5	2	5
200	5	SG	5	2	5
100	5	SG	5	2	5
100	3	CBOW	5	2	1

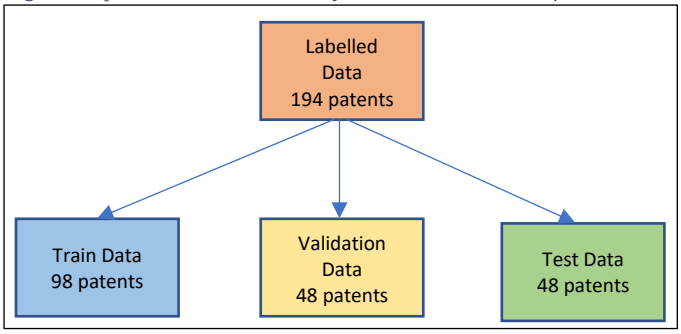
I used the Gensim Python library to train the Word2Vec models and their average training time with only 5 epochs was around twelve hours. There are different ways to extract features using a pre-trained word embedding model. I chose to average the word vectors for each words in the new text, from which I needed to extract features (Text Classification With Word2Vec, 2020).

How I decided to split the data for all the classification models is described in the next section.

Splitting the data

This is another important step to take when training a machine learning model, because the data used to train the model needs to be separated from the ones used to evaluate it. For the claim models, since the data for this type of text were more numerous than for the abstracts – with 194 patents and 4857 claims, I have split the labelled data in three separate data sets: train; validation and test data sets. I split the data in three subsets at the patent level to avoid that some claims, that are in the test data set, are also either in the train or in the validation data. As discussed in the section dedicated to the labelling phase, some of the claims in a patent can be either dependent or independent claims and sometimes the wording of the dependent claim could be very similar to other dependent claims for the same patent. Intuitively, this is a problem because some of the data points in the training dataset are very similar to some of the data points in the test dataset and this can overestimate the performance of the model. This means that, when the model tries to predict the class of an unseen patent, the model will perform poorly. Therefore, splitting the labelled dataset into the three datasets at the patent level avoids any data leakage between the datasets.

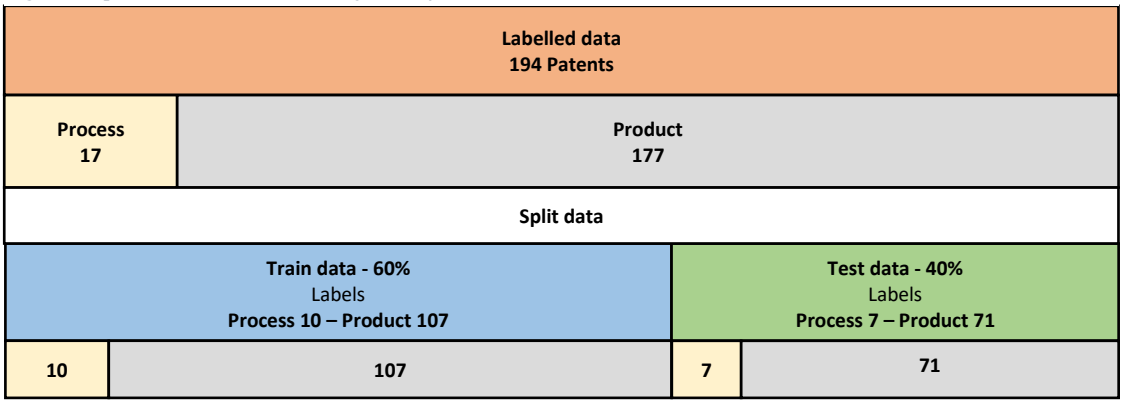
Figure 3 Split Claims data - Binary and Multiclass classification



In total the patents labelled are 194 and, from them, I have randomly selected 48 patents for the validation data, other 48 patents for the test data and the remaining part, thus 98 patents, to train the model – see Figure 3. In percentage that corresponds to assigning 25% to the test data, 25% to the validation data and 50% to the train data.

For the abstracts, where the data points labelled were exactly 194, it was not possible to split the dataset into three sets as for the claims, because of such a low number of labelled observations. Therefore, I split the abstracts data into two groups, 60% for training and 40% for testing.

Figure 4 Split Abstracts data - Binary classification



The reasons why I had to allocate such a high percentage of data for testing is because for the abstracts the data are even more unbalanced and, to have a few more patents from the minority class, I increased the size of the test data. In addition, to preserve the same proportion of labels for the two classes - product and process, I split the dataset by stratifying the data for the class labels[11] -- see Figure 4. This was meant to avoid that only a couple of data points from the minority class

[11] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

be in the train or test datasets. The ratio between the minority and the majority class is 1 to 10.5, so for each abstract labelled as process there are more than 10 labelled as product. The disproportion of abstracts labelled as product patents and such a lack of patents from the other class in addition to a very low number of overall abstracts is a very unfortune problem, which will be discussed further below.

In the next section I will illustrate how I handled the problem of unbalance data.

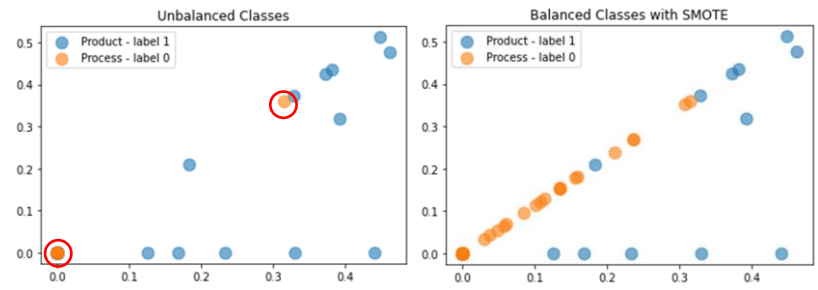
Balancing the data

It is very common to work with an unbalanced dataset and the main problem that arises, when a model is trained with imbalanced classes, is that it will perform very poorly. For a classification model, the main cause of a poor performance is that this model will learn from many examples of the most common class and just a few of the others. Consequently, the model classifies quite well claims or abstracts that belong to the majority class but is less accurate for the minority classes. As mentioned in the previous sections, the classes in my project are extremely imbalanced and for the abstracts out of 11 observations only one belongs to the minority – see Table 4 section dedicate to the Labelling phase.

The misclassification of elements from the minority class due to an unbalanced dataset gets even worse if the dataset only has very few observations. There are different ways to deal with an unbalanced dataset and sometimes the results are quite good. To balance the dataset I have employed a technique that oversamples the minority class called Synthetic Minority Oversampling Technique (SMOTE) (Chawla, Bowyer, Hall, & Kegelmeyer, 2002). I have run a few experiments prior to decide what technique to use but none of them had better results than SMOTE[12]. SMOTE is way to create new data points for the minority class by drawing new synthetic examples of it 'along the line segments joining any/all of the k minority class nearest neighbors' (Chawla, Bowyer, Hall, & Kegelmeyer, 2002, p. 328). Figure 5 illustrates a particularly revealing example of the training set before and after applying SMOTE. In the graph on the left-hand side, the observations of the minority class are marked with red circles and it is between these two positions on feature space that the new synthetic observations are drawn – see results in the graph on the right-hand side.

[12] I ran a few classification models with the class weights parameters set to *balanced* or I constructed directly the weights using the Sklearn class weight function, but all results were inferior to the ones achieved by SMOTE. For the claims, it was also possible to run a under sampling technique for the majority class but, due to the limited number of observations, the results of this technique were the lowest ones achieved in comparison to all other methods.

Figure 5 Graphic representation of SMOTE on the train data



Train data set of the abstracts. The feature extraction used is the TF-IDF vectorization with max number of features equal to 70.

In the section below, I will briefly describe the learning algorithms used in the present work and, in a later section, I will discussed how I investigated ways to increase either the number of labelled data or the performance of the model for the abstracts.

Learning algorithms

Some small preliminary tests were run to select the learning algorithms for the classification models, before deciding which models would go through the tests harness. The tests simply consisted in running a quick training model with a part of the pre-processed data and verifying if the model was worthy to proceed to the test harness. This allowed me to discard learning algorithms, such as Naïve Bayes (NB), K-nearest neighbors (KNN) and Linear discriminant analysis (LDA) and to select for training four learning models: Logistic Regression, Support Vector Machine, Random Forest and Extreme Gradient Boosting (XGBoost).

a. Logistic Regression:

The first model I trained is the logistic regression and was employed also in Nicolas's work (2019). This consists of a supervised learning algorithm and is used for binary classification problems. Its aim is to predict the probability that an observation belongs to a class, where the classes can be 0 or 1. The value is a prediction of the probability and it spams from 0 to 1. The form of a logistic function, also called sigmoid function, is:

Equation 3 Sigmoid Function $$\sigma(x) = \frac{1}{1 + e^{-z}}$$

b. Support Vector Machine:

The Support Vector Machine (SVM) is a supervised learning method, used mostly, but not exclusively, for classification problems. The idea is that, to classify the instances into two classes, one needs to find a hyperplane, that can divide the data points into the two categories. The kernel trick helps to find the right hyperplane and to reduce the complexity required to project our data points in a new dimension space by working on the pairwise similarity values of our instances with the dot product to construct cell entries of the kernel matrix (Zaki & Meira, 2014).

Ensemble models:

The last two models are ensemble models that are employed for both regression and classification problems.

c. Random Forest

Random Forest trains several independent trees by randomly sample each time the features, which help to reduce the correlation between them (Breiman, 2001). Those trees form the forest and the tree that has the majority of votes is selected.

d. XGBoost

XGBoost is currently the most used model because of its scalability and the low time required to be trained. The main difference from Random Forest is that XGBoost does not consider the trees independent from each others and adds them all together (Stein, Jaques, & Valiati, 2019; Chen & Guestrin, 2016).

The next section is dedicated to the description of the metrics utilised to evaluate the models.

Evaluation metrics

The metrics that I used to evaluate the models are the ones most commonly used for text classification tasks. They are also the same evaluation metrics that Nicholas at el (2019) used in their work.

In Figure 6 is depicted a confusion matrix for the binary classification – process and product – of claims and abstracts. A confusion matrix is an important tool used to calculate most of the

classification metrics. It simply summarises the number of times that the two classes, positive and negative, are predicted correctly or misclassified. To be consistent with Nicholas and colleague' work, for the binary classification models, I set as negative the class process and as positive the class product.

Figure 6 Confusion matrix for the binary classification model

		Product		
		Negative Predicted	Positive Predicted	
Process	Negative True	**True Negative (TN)**	**False Positive (FP)**	*Recall*
	Positive True	**False Negative (FN)**	**True Positive (TP)**	
		Precision		Accuracy

Below is illustrated how to calculate the metrics that I used to evaluate the classification models.

Equation 4 $$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy is the sum of the correctly classified elements over the total of the correctly plus the misclassified data points (Equation 1). This is a very unreliable metric when the classes are imbalanced, since its value will be brought up by the majority class without saying almost anything about the other classes.

Equation 5 $$Precision = \frac{TP}{TP + FP}$$

Precision is the total number of positive data points correctly classified over the sum of all the positive data points predicted (Equation 2). Hence, it is the measure of the percentage of positive observations that are correctly classified in the positive class (Jurafsky & Martin, 2008).

Equation 6 $$Recall = \frac{TP}{TP + FN}$$

Recall or *Sensitivity* is the total number of positive data points correctly predicted over the sum of the positive data points correctly classified plus the number of the false positive ones (Equation 3). It is percentage of positive elements correctly classified of the total of all the positive elements, either correctly or wrongly classified.

Equation 7 $$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1-score is simply a harmonic mean of Precision and Recall metrics.

Equation 8 $$Specificity = \frac{TN}{TN + FP}$$

Specificity measures that the number of true negatives divided by the true negative plus the false negative data points (Equation 4). This is an important metrics because it evaluates the performance of a model with regard to the negative class.

Equation 9 $$Balanced\ accuracy = \frac{Specifity + Recall}{2}$$

Balanced accuracy is the sum of Recall and Specificity divided by two.

Area Under the ROC curve (AUC) is the measure of how much the classification model was able to split the elements belonging to the positive class from the ones belonging to the negative class. Its value spams from 0 to 1. The higher the result the better is the model. If the value approaches 0.5, the model struggles to distinguish between the two classes.

The same evaluation metrics were applied to evaluate the performance of the multiclass classification model for the claims with the only difference that, to evaluate a model with three classes, I had to calculate three values for each metric – Precision, Recall and F1-score, see Figure 7.

Figure 7 Confusion matrix for the three classes classification model

	Process	Use claim	Product	
Process	**True Process**			*Recall proc*
Use claim		**True Use claim**		*Recall use cl*
Product			**True Product**	*Recall prod*
	Precision proc	*Precision use cl*	*Precision prod*	Accuracy

Two are the ways to average these metrics to get a single score value for each of them: macro-average or micro-average (Jurafsky & Martin, 2008). I have chosen to macro-average the metrics because all the classes are weighted in the same way.

An example of macro-averaged metrics for the classification model with the three classes:

$$Precision\ macro-averaged\ =\ \frac{\text{Precision proc}\ +\ \text{Precision use cl}\ +\ \text{Precision prod}}{3}$$

The last evaluation metric is a weighted version of the *Mean Absolute Error*. Here this measurement is calculated at a patent level and quantifies the error between the true values and the predicted values (see Equation 7 below).

Equation 10

$$MAE\ weighted\ =\ \frac{\sum_{i=1}^{n}\left|\frac{\acute{y}_i - y_i}{n_{claims}}\right|}{N_{patents}}$$

where $\acute{y}_i$ is the predicted percentage of the *ith* patent classified as a process patent and y_i is its true percentage of the *ith* patent labelled as a process patent.

The percentage of a patent being a process patent is calculated by the number of claims classified as process over the total of claims. In the case of a patent with 10 claims, if six claims out of ten are process, this patent is 60% a process patent. This calculation is conducted on both predicted and true labels and the difference between these two percentages is divided again by the number of claims included in the patents. This makes sure that patents with just a few claims, some of which misclassified, will not drive incorrectly up the value of the error more than a patent with many claims that has the same number of misclassified claims. The example below further illustrates this concept and the reason why I had to weigh the patents with the number of their claims.

Example:

Patent A with 2 claims – true value: 100% process patent – predicted: 50% process patent
Patent B with 20 clams – true value: 80% process patent – predicted: 50% process patent

Error no weighted: $\frac{|0.5-1|+|0.5-0.8|}{2}\ =\ 0.4$

Error weighted: $\frac{\frac{|0.5-1|}{2}+\frac{|0.5-0.8|}{20}}{2}\ =\ 0.1325$

The example shows that without weights the error value would be driven up just because of one misclassified claim, that happens to be part of a patent with few claims.

In the following section I describe how I selected the best models and how I applied the metrics illustrated above.

Spot-check the best models

In order to choose the best learning algorithm for a machine learning model, one strategy could be to run separately many models with different algorithms and select the best one. Nevertheless, there is more reliable way, which consists in creating a test harness which runs all these models at once. A test harness that uses the cross validation technique is employed here to spot-check the best models (Brownlee, 2016). The cross validation technique is a type of resampling method, which helps to get more information on the model by fitting it many times on different samples drawn by the training data (James, Witten, Hastie, & Tibshirani, 2013). I ran a test harness for all the three classification models: two separate binary classification models for the claims and abstracts and a multiclass classification model only for the claims. In this type of test to evaluate the models it is common practice to use the accuracy. By using a loop, I added three important metrics to evaluate my classification models, such as Precision, Recall and F1 for the binary models and their macro-averaged version for the multiclass model.

For all the models, I constructed a pipeline where I added the two feature extraction methods, TF-IDF and Word2Vec. A pipeline helps to make sure that all the models follow the same methods to reach the final step of the pipeline and that they are consistently evaluated on the same samples (Géron, 2017). In addition to the mean values of the classification metrics, I also calculated the standard deviation (Std). The standard deviation is a way to quantify the variance of the scores for each model. The lower values of Std corresponds to a better performance of the model in terms of variance of a specific score metric. For the claims, I used k-fold cross validation to run the test harness for three learning algorithms, namely Supper Vector Classifier, Random forest and XGBoost. This means that a learning algorithm, in turn with one of the two word representation methods, was trained for each of the four metrics k times, while in each run 1/k of the data was retained for testing and the remaining part as training data – k here was set to 10. For the abstracts, I did not use a simple k-fold cross validation method, but I used a stratified version of it – stratified

5-fold cross-validation[13]. As for the splitting phase, by using the stratified version of the cross-validation method, I made sure that each of the split of the training data included the same proportion of the minority class. In addition to the three learning algorithms that I ran for the claims, for the classification of the abstracts I also included a logistic regression into the test harness. This is because Nicolas and colleagues used a logistic regression in their work (2019) and because I ran some preliminary tests before the test harness and the logistic regression gave me some good results.

About the feature extraction methods, for the Word2Vec vectorizer, I used the Skip-gram model trained with a dimension of features of 300, a window size of 5 and a negative sample of 5. For the TF-IDF vectorizer, I selected the feature space of 300 and I set the nrange parameter to 1 and 2 which means that the vectorizer will consider unique words and combination of two words.

In the next section is described what happens after the best two models are spotted and how I decided to tune the hyperparameters of my models.

Tuning hyperparameters

For the claims data, I tuned the hyperparameters of a model by using the validation data, which I left unbalanced. Tuning hyperparameters is an important step to be able to train the best learning model with the best parameters. However, a tuned model does not always perform better than an untuned models. On the contrary, in some occasion the latter performs better with its own default set of parameters (Weerts, Mueller, & Vanschoren, 2020)[14]. This could be attributed to an overfitting problem. The model with the newly tuned parameters could end up fitting really well the data used for the tuning phase, but not so well the holdout data used for testing. In my project the overfitting problem could be caused by the fact that the labelled data are split in three datasets by patents and not by claims, and it could be that some patents from the same technology are more common in the validation and training data than the ones in the testing data.

There are different ways to search for the best parameters and the most common approaches use grid search and random search. Grid search goes through all the combinations of parameters whereas random search randomly samples the combination of those parameters. Therefore, if the number of parameters is low, grid search is the best choice, but would not be a good option if the

[13] I used the Sklearn class called StratifiedKFold - https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

[14] For instance, to see the default parameters for a Sklearn Logistic Regression model it is enough to print LogisticRegression().get_params().

parameters are numerous, because it would be slow and often would not give the best results (Bergstra & Bengio, 2012). On the contrary, random search is a very efficient way to find the parameters, when there are many to sample, since this approach would randomly select the combination of those parameters in every iteration (Géron, 2017). Both approaches use cross a validation technique, which splits the training data in several samples and searches the best parameters on those subsets.

Figure 8 Hyperparameters selected to be tuned

TF-IDF*
The type of normalization applied to the word vectors – norm - default l2.
The number maximum of features - max_features - default None.
This is either to set or not the term frequency to its sublinear version - sublinear_tf - default False.

Random Forest*
The minimum number of samples required to be at a leaf node - min_samples_leaf - - default 1.
The minimum number of samples required to split an internal node - min_samples_split - default 2.
The number maximum of features - max_features - default auto.
The maximum depth of the tree - max_depth - default None.
This sets the number of trees in the forest - n_estimators - default 100.
Whether bootstrap samples are used when building trees - bootstrap - default False.

SVC*
Regularization parameter C – default 1.
Kernel coefficient – gamma – default scale.
Degree of the polynomial kernel function ('poly') - degree – default 3.
Specifies the kernel type to be used in the algorithm - kernel – default radial basis function.
Whether to enable probability estimates - probability – default True.

Logistic Regression
Algorithm to use in the optimization problem – solver – default Limited-memory BFGS.
Inverse of regularization strength - C – default 1
Used to specify the norm used in the penalization – penalty – default l2.

XGBoost#
The maximum depth of the tree - max_depth - default 6.
This is set the number of trees in the forest - n_estimators.
Represents how big the steps are when adjusting the weights for the loss gradient - learning_rate.

*Source description original documentations: a)sklearn.feature_extraction.text, b)TfidfVectorizer, c) sklearn.ensemble.RandomForestClassifier, d) sklearn.svm.SVC and e) sklearn.linear_model.LogisticRegression.
Source description original XGBoost documentation https://xgboost.readthedocs.io/en/latest/parameter.html.

I tuned the hyperparameters only for the best two models – see Figure 8 with the parameters chosen to be tuned for each algorithm. I used both approaches in two different steps of the project. I employed the grid search approach to search for the best parameters for the TF-IDF vectorizer with 2-fold cross-validation and the random search with 3-fold cross-validation to tune the parameters of the best learning algorithm, since for the latter the parameters are often too many for a grid search. Nevertheless, just a few of the possible parameters are tuned and many of them are left to their default values.

The procedure is as follows. After selecting the best models from the test harness, I trained both the models separately. I first ran the models without any tuning and for the TF-IDF vectorizer I set only the ngram parameter as unigram and bigram. After training and evaluating the default version of the two models, I tuned the hyperparameters of the TF-IDF vectorizer and I trained and evaluated the models with the TF-IDF tuned. Only then, I tuned the hyperparameters for the learning algorithms and I ran the models with both TF-IDF and the tuned learning algorithm. At the end of this procedure, I chose the best model on the basis of its performance across all the metrics.

Before presenting the results of the models, I will illustrate my attempts to increase a poor performance due to a low number of labelled observations.

Attempts to increase performance of the classification models of the abstracts

I tried to increase the number of labelled abstracts by finding alternative ways to label new set of them. The first investigation involved the use of the keywords search method to find and label the abstracts as either process or product. I conducted two experiments with this method. The first one, a very conservative strategy of labelling, aimed to label an abstract as process if it had a process keyword among its first three words and as product otherwise – see Table 7 below with a list of keywords used by Nicolas and colleagues (2019)). The second experiment aimed to label an abstract as process when there were at least four process keywords within its text and as product if there were less than four process keywords. Once I labelled the new abstracts with these two experiments, I ran a test harness again with this new data. I selected the best model to be trained and I tested it with the test data split from data manually labelled.

Table 7 Product and Process Keyword for extra labelling (Nicolas, et al., 2019, p. 6)

Type	**Keywords**
Product keywords	Device, Machine, Material, Tool, Apparatus, Vehicle, Compound, Composition and Substance
Process Keywords	Method, Process, Use, Utilisation, Utilization and Usage

A second type of investigation was led with a pseudo labelling method, which consists in the labelling of a set of unlabelled data with the predictions obtained by a previously trained classification model. This is a plain semi-supervise learning method because it uses labelled data to train the classification model and with it it predicts the labels of an unlabelled dataset. Once, the unlabelled data have been labelled, I trained again the classification model and tested on the same test data that had been manually labelled. A variant of this method is used to increase the performance in deep learning models (Lee, 2013). For the two methods illustrated above, keyword

search and pseudo labelling ones, I made sure that the patents, that had been manually labelled and used to train the classification model in the first place, were not included in the unlabelled dataset. This is to avoid having to train a model with patents that are also in test data set.

The last method I used does not attempt to label an unlabelled dataset as in the previous cases, but rather creates new data by changing some of the words included in the text. This method is a data augmentation technique called Easy Data Augmentation (EDA)[15] (Wei & Zou, 2019). This method randomly chooses to performance on a document one of the following tasks:

> '*1. Synonym Replacement (SR): Randomly choose n words from the sentence that are not stop words. Replace each of these words with one of its synonyms chosen at random.*
> *2. Random Insertion (RI): Find a random synonym of a random word in the sentence that is not a stop word. Insert that synonym into a random position in the sentence. Do this n times.*
> *3. Random Swap (RS): Randomly choose two words in the sentence and swap their positions. Do this n times.*
> *4. Random Deletion (RD): Randomly remove each word in the sentence with probability p*' (Wei & Zou, 2019, p. 6383).

To run this technique, the algorithm needs a text file, in whose lines the documents preceded by their labels are stored separately. There are two parameters to set before running the algorithm: n_{aug} and α. n_{aug} simply corresponds to the number of new documents created for each of the original ones, whereas α is the percentage of the words in the document that would be changed using one of the four tasks listed above. My strategy was to boost the performance of the minority class by data augmenting the documents from this class. By using one of these two techniques, I was not expecting to have a life changing boost of the performance of a model, but one of these three methods could slightly increase its performance.

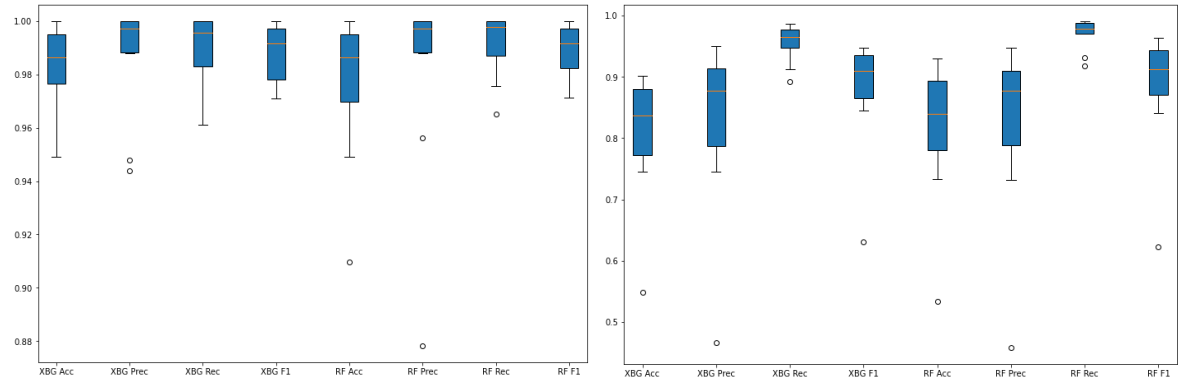
[15] The repository of EDA can be cloned from https://github.com/jasonwei20/eda_nlp. I had to do a very small adjustment to make the algorithm to work because of a Unicode error in line 37 of the *augment.py* file where I change it from "lines = open(train_orig, 'r') readlines()" to "lines = open(train_orig, 'r', encoding="utf-8").readlines()".

4. Results

Test harness results

The results of the tests harness conducted on both claims and abstracts have showed that the ensemble learning models are the most reliable models, since they are the best at classifying these types of text. For the binary classification of claims the best two performances have been achieved by the Random Forest and XGBoost algorithms. The most relevant information the tests harness provided is that for all cases the best results are achieved by the TF-IDF word representation method. Both the best two models have excellent results across all the metrics implemented. Figure 9 depicts box plots for each of the metrics for both learning algorithms. The variance of each metric is quite stable with certain outliners – graph (a) Figure 9. This proves that the models are very promising.

Figure 9 Box Plots metrics scores Spot-check Binary classification Claims - XGBoost and Random Forest



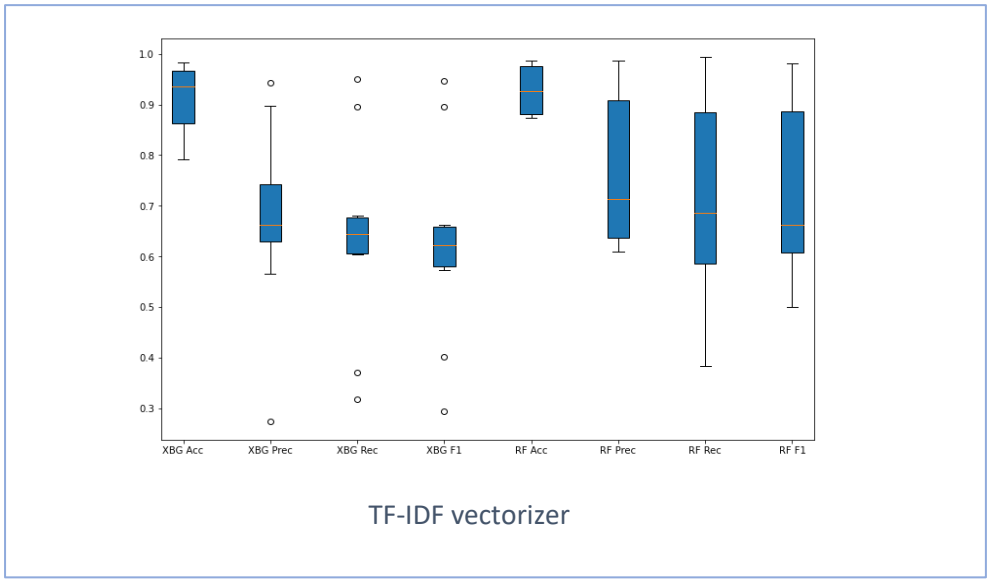
(a) TF-IDF (b) Word2Vec

The word embedding model did achieve a fair performance but was, nevertheless, outperformed by the models with TF-IDF word representation - see graph (b) Figure 9. The numeric results can be found in Appendix A.

The test harness conducted to spot-check the best two models for the multiclass classification of claims has showed too a slightly better performance for the two ensemble models, Random Forest and XGBoost. From Figure 9 one can see that in this occasion the models have not

performed as well as for the binary classification models. There is a lot of variance displayed by the score metrics, but, nevertheless, the results are still reasonable.

Figure 10 Box Plots metrics scores Spot-check Multiclass classification Claims - XGBoost and Random Forest



One reason for the underperformance of the multiclass classification model could be that all the test harnesses have been conducted on data not yet balanced by SMOTE. With more classes and an unbalanced dataset, a multiclass model will inevitable perform worse than a model with just two classes.

For the test harness to spot-check the best two models of the abstracts, the results are not very informative since many models achieved the same performance in all the four metrics. This is again due to the lack of data, which makes the test harness not very helpful to select the best two models – see results in Appendix A. Consequently, I chose one of the two best models between those that had the same performance – the Logistic Regression model, and I chose XGBoost as the second model. For the abstracts, the models trained with Word2Vec had pretty much the same performance in comparison to the ones where I used TF-IDF vectorizer. Therefore, I trained also the best two models with both word representation methods to be able to compare their results.

In the next section I will go through the results of the best two models selected for the final phase.

The results of best two models

Claims:

The best performance for the binary classification model of claims was achieved by the Random Forest model with both TF-IDF and learning algorithm tuned. The results are very impressive in all the metrics measured on the test data. In Figure 11 are displayed the results and hyperparameters for both TF-IDF vectorizer and the learning algorithm.

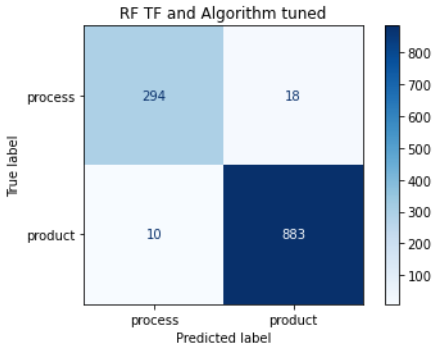
Figure 11 Hyperparameters and results of the binary classification model for the Claims.

The hyperparameters tuned for the TF-IDF are:

Maximum number of features: **300**,
Normalization: **l2**,
Sublinear version of TF: **True**.

The hyperparameters tuned for the learning algorithm are:

Maximum number of features: **sqrt**,
The maximum depth of the tree: **100**,
The number of trees in the forest: **600**,
The minimum number of samples of a leaf node: **1**,
The minimum number of samples to split an internal: **2**.



The MAE-weighted is 0.003498269118210207

Model	Data	Precision	Recall	Specificity	F1-score	Accuracy	Bal Accuracy	AUC
RF TF-IDF and Algorithm tuned	Test data	0.980022	0.988802	0.942308	0.984392	0.976763	0.965555	0.965555
Test data 48 patents with 312 process and 893 product claims								

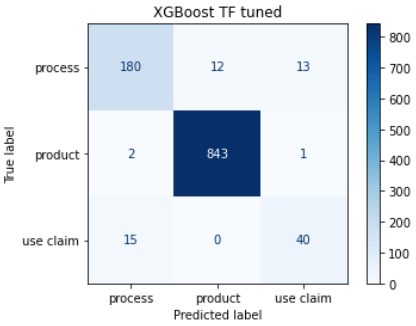
The best performance for the multiclass classification model of the claims was achieved by the XGBoost model with TF-IDF tuned and learning algorithm with the default parameters. The results displayed in Figure 12 are still very good but not as good as for the binary model, since there are slightly more cases of misclassification than for the binary classification.

Figure 12 Hyperparameters and results of the multiclass classification model for the Claims.

The hyperparameters tuned for the TF-IDF are:

Maximum number of features: **500**,
Normalization: **l2**,
Sublinear version of TF: **True**.

MAE-weighted is 0.005204097443644654



Model	Data	Precision	Recall	Specificity	F1-score	Accuracy	Bal Accuracy	AUC
XGBoost TF tuned	Test data	0.880137	0.867258	0.973886	0.873550	0.961121	0.867258	0.920572
Test data 48 patents with 55 use claim, 205 process and 846 product claims.								

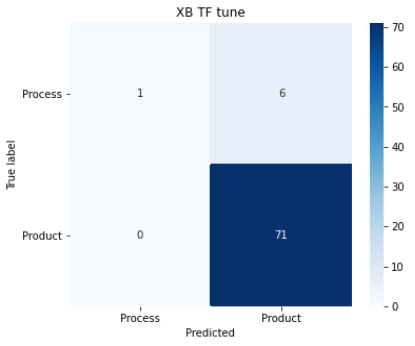
Abstracts:

The best performance for the binary classification model of the abstracts was achieved with the XGBoost with the TF-IDF tuned. As expected, the results in this case are quite disappointing, since the model achieves an extraordinary performance for the positive class displayed by Precision and Recall, but not for the Specificity – see Figure 13. This metrics shows how the model performs with the negative class.

Figure 13 Hyperparameters and results of the Binary classification model for the Abstracts.

The hyperparameters tuned for the TF-IDF are:

Maximum number of features: **300**,
Normalization: **l2**,
Sublinear version of TF: **False**.



Model	Data	Precision	Recall	Specificity	F1-score	Accuracy	Bal Accuracy	AUC
XGBoost with TF tuned	Test data	0.922078	1.0	0.142857	0.959459	0.923077	0.571429	0.571429
Test data 78 abstracts with 7 process and 71 product claims.								

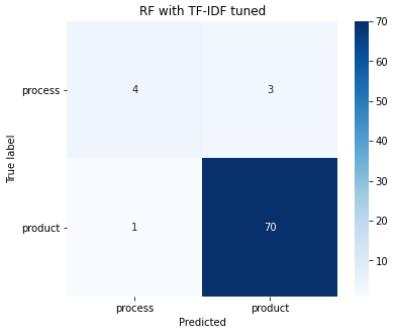
In fact, the model classified correctly for the minority class – process abstracts – 1 out of 7 abstracts and 71 out of 71 for the positive class – product abstracts. Another hint to how poorly the model performed, is suggested by the Area Under the Curve (AUC) value. which is slightly above 0.5. This means that the model is almost as bad as a random model, thus a model that assigns randomly a class to an observation.

As I mentioned above, I tried to find ways to increase the performance of the binary classification model of the abstracts by increasing 'synthetically' the number of observations. For the experiments with data augmentation techniques, the results were quite disappointing, since the models run with the augmented observations did not improve the performance on the minority class, but only worsened that on the majority class – see results in Appendix A. Another method I implemented, entailed the labelling of new data with the predicted model from the original binary classification model trained with the labelled abstracts. Also in this occasion the performance of binary classification of the abstracts did not improve the results of the original binary model. Fortunately, the keyword search experiments, thus another of the additional investigations conducted to increase the performance of the binary classification for the abstracts, provided extremely satisfactory results – see Figure 14.

Figure 14 Hyperparameters and results of the Keyword search experiment - 1 out 3 process keywords - Binary classification model for the Abstracts.

The hyperparameters tuned for the TF-IDF are:

Maximum number of features: **400**,
Normalization: **l2**,
Sublinear version of TF: **True**.



Model	Data	Precision	Recall	Specificity	F1-score	Accuracy	Bal Accuracy	AUC
Rf with TF tuned	Original Test data	0.958904	0.985915	0.571429	0.972222	0.948718	0.778672	0.778672
Test data 78 abstracts with 7 process and 71 product abstracts.								

The conservative experiment, where I searched within the first three words in the text for process keywords, gave good results with the Random Forest algorithm with the TF-IDF vectorizer tuned.

The model trained on the data labelled with the keyword search method and tested using the manually labelled test data showed a clear improvement on the minority class since 4 out of 7 abstracts were correctly classified. This was corroborated by a net increase of the Specificity metric that amounted to 0.57 in comparison to 0.14, that was achieved by the original binary model. Also, the value of the Area Under the ROC Curve showed a net improvement by going from 0.57 to 0.77. The second experiment where I searched in text for at least four process keywords to classify the abstracts as process did not improve the original model and they are not reported here.

Below is a comparison of my models' performance with the benchmark models on the same classification problem. Table 8 is divided into two sub-tables. In the top sub-table are listed the results of the benchmark model, which is a binary classification model, and my two models trained to classify claims. The results show that my binary classification model has outperformed the results of the previous study for all the selected metrics. In addition, even though the benchmark model and my multiclass model for the claims are not directly comparable, because they have a different number of categories to classify, my model, with exception of the F1-score, has also outperformed the benchmark model.

Table 8 Comparison Results of the three models with the benchmark

CLAIMS	**Precision**	**Sensitivity**	**Specificity**	**F1-score**	**Accuracy**	**Bal. Accuracy**	**AUC**
Benchmark – Binary*	0.88	0.69	0.92	**0.90**	0.85	0.81	0.81
Binary (form Figure 11)	**0.98**	**0.99**	**0.94**	**0.98**	**0.98**	**0.96**	**0.96**
Multiclass (form Figure 12)	**0.88**	**0.87**	**0.97**	0.87	**0.96**	**0.87**	**0.92**
ABSTRACTS							
Benchmark – Binary[#]	0.94	0.82	**0.68**	0.79	0.71	0.75	0.75
Binary Original Model (form Figure 13)	0.92	**1.0**	0.14	**0.96**	**0.92**	0.57	0.57
Binary with Keyword search (form Figure 14)	**0.96**	**0.97**	0.57	**0.96**	**0.96**	**0.77**	**0.77**

*Benchmark model trained with 4,897 and tested with 2,097 claims.

[#]Benchmark model trained with 632 and tested with 269 abstracts.

The situation is different for the abstracts. My model, trained only with manually labelled data, has some of the metrics slightly above the metrics of the benchmark model. Indeed, my first binary classification model better classifies abstracts that belong to the positive class, but does not do well for the other class, for which the evaluation metrics are very low. On the other hand, the model trained with new abstracts labelled with the keywork search method improved most of the metrics and only Specificity is still slightly below the benchmark – 0.57 against 0.68.

I saved all the three best models so they can be loaded again in other scripts using the library Pickle.

In the next section, will be briefly discussed the findings and some possible future research directions, that could lead to further improvements of this classification task.

Discussion and future research directions

The models trained within this project to classify a patent based on the basis of the type of innovation patented, have improved the current state-of-art. The ensemble models, Random Forest and XGBoost algorithms, have showed their strenght even when the data available for training and testing purposes were not nearly enough for other learning algorithms. All the models have showed, at least for the claims, outstanding results with little room left for improvement. This is more true for the binary classification model of claims, where the results achieved are excellent with just a few misclassifications. The situation is slightly different for the multiclass model for the claims. Even though its results are still good, in this case there is still room for improvemet, since the misclassified claims were various.

For the abstracts a classic training and testing strategy did not give good results, at least on the negative class. Some experiments were conducted to improve this model but only labelling new data by using process keyword search method improved significantly the results of the initial classification model. Almost all the metrics after employing the keyword search method surpassed the benchmark model, with just a slightly better performance of the latter model for the negative class measured by the Specificity value.

To improve the perfomance of the models, there are a few further steps that should be considered for future research. For instance, for the multiclass model, the best results were achieved by the model untuned and this would not happen if more labelled data were available. Besides working on more complex learning algorithm to improve the performance of the multiclass model, only after adding more labelled data, it would be helpful to split data in a stratified fashion by considering also the fourth type of technology. This would avoid overfitting on the validation data when tuning the hyperparameters, since also the testing data would have the same amount of patents from the different technologies.

Another direction for future research could focus on keyword extraction algorithms on labelled data (Hu, et al., 2018), This could help to refine the set of keywords of both product and process and could be used to increase the numbers of data available to train and test the models.

The last direction to improve the models could be to train a word embedding model with more refined parameters, such as increasing the number of epochs. Another improvement could be achieved by changing the way in which the single word in text are transformed in vectors of numbers uisng the Word2Vec model. In the present work, I transformed the words in the text in numbers by averaging the vectors calculated with Word2Vec for each of the words but, instead, I could average the Word2Vec vectors by documents instead of single words in the text data.

References

Bena, J., & Simintzi, E. (2019). Machines Could Not Compete with Chinese Labor: Evidence from U.S. Firms' Innovation. *Available at SSRN: https://ssrn.com/abstract=2613248 or http://dx.doi.org/10.2139/ssrn.2613248*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research, 13*(1), 281-305.

Breiman, L. (2001). Random Forests. *Machine Learning, 45*, 5–32.

Brownlee, J. (2016). *Machine learning mastery with Python: understand your data, create accurate models, and work projects end-to-end.* Machine Learning Mastery.

Chawla, N., Bowyer, K., Hall, L., & Kegelmeyer, W. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research, 16*, 321-357.

Chen, T., & Guestrin, C. (2016). August. Xgboost: A scalable tree boosting system. (pp.). *In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* , (pp. 785-794).

Géron, A. (2017). *Hands-on Machine Learning With Scikit-Learn and TensorFlow: Concepts Tools and Techniques to Build Intelligent Systems.* Sebastopol, CA, USA: O'Reilly,.

Hu, J., Li, S., Yao, Y., Yu, L., Yang, G., & Hu, J. .. (2018). Patent keyword extraction algorithm based on distributed representation for patent classification. . *Entropy, 20*(2), 104.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning.* New York: Springer.

Jurafsky, D., & Martin, J. H. (2008). *Speech and Language Processing* (2nd ed.). International Edition.

Lee, D. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In Workshop on challenges in representation learning, ICML (Vol. 3, No. 2). *ICML, 3*(2).

McCormick, C. (2020, July 5). *Word2Vec Tutorial Part 2 - Negative Sampling*. Retrieved from http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/.

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

Nicolas, B., Behrens, V., Feuerriegel, S., Heinrich, S., Rammer, C., Schmoch, U., . . . Wörter, M. (2019). *Knowledge Spillovers from Product and Process Inventions in Patents and their Impact on Firm Performance.* Zürich: European Patent Office.

Risch, J., & Krestel, R. (2019). Domain-specific word embeddings for patent classification. *Data Technologies and Applications, 53*(1), 108-122.

Stein, R., Jaques, P., & Valiati, J. (2019). An analysis of hierarchical text classification using word embeddings. *Information Sciences, 471*, 216-232.

Text Classification With Word2Vec. (2020, July 30). Retrieved from DS Lore: http://nadbordrozd.github.io/blog/2016/05/20/text-classification-with-word2vec/

Weerts, H., Mueller, A., & Vanschoren, J. (2020). . Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv, 07588*, 2007.

Wei, J., & Zou, K. (2019). EDA: Easy data augmentation techniques for boosting performance on text classification tasks. *In Proceedings of the Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing.* (pp. 6383-6389). Hong Kong, China: Association for Computational Linguistics.

Word2Vec Model - Gensim. (2020, July 10). Retrieved from Gensim: adimrehurek.com/gensim/auto_examples/tutorials/run_word2vec

Yun, J., & Geum, Y. (2020). Automated classification of patents: A topic modeling approach. *Computers & Industrial Engineering, 147*, 106636.

Zaki, M., & Meira, W. (2014). *Data mining and analysis: fundamental concepts and algorithms.* Cambridge University Press.

Appendix A

Results test harness to Spot-check best models - Binary Claims				
Models	**Scores**	**Mean Scores**	**Standard Deviation**	**Time**
Logistic TF-IDF	Accuracy	0.93961	0.04949	5.95907
	Precision	0.92522	0.08515	2.11139
	Recall	0.99600	0.00768	1.50395
	F1-Score	0.95701	0.05150	1.32250
Logistic Wor2Vec	Accuracy	0.80980	0.15693	79.23438
	Precision	0.80980	0.15693	80.47450
	Recall	1.00000	0.00000	82.22772
	F1-Score	0.88499	0.11496	81.86665
XGBoost TF-IDF	Accuracy	**0.98235**	0.01610	4.83840
	Precision	**0.98640**	0.02073	3.92257
	Recall	**0.98994**	0.01307	3.88163
	F1-Score	**0.98797**	0.01095	3.79187
XGBoost Word2Vec	Accuracy	0.80784	0.09987	95.24144
	Precision	0.82862	0.13682	93.67431
	Recall	0.95576	0.03003	93.80878
	F1-Score	0.87956	0.08942	92.95249
SVM TF-IDF	Accuracy	0.96157	0.02875	2.91021
	Precision	0.95408	0.04436	2.88431
	Recall	0.99503	0.00832	2.88330
	F1-Score	0.97356	0.02448	2.89724
SVM Word2Vec	Accuracy	0.80980	0.15693	90.81453
	Precision	0.80980	0.15693	90.02387
	Recall	1.00000	0.00000	90.47688
	F1-Score	0.88499	0.11496	89.67098
Random Forest TF_IDF	Accuracy	**0.97608**	0.02685	5.82551
	Precision	**0.98068**	0.03645	4.19621
	Recall	**0.99160**	0.01183	4.19612
	F1-Score	**0.98849**	0.01013	4.12927
Random Forest Word2Vec	Accuracy	0.81647	0.11202	86.48091
	Precision	0.82662	0.13981	92.02235
	Recall	0.97033	0.02383	91.93391
	F1-Score	0.88496	0.09542	89.69107
Results from the script D_Spot-Check\Claims\Spot_check_Binary_Classification.ipynb				

Results test harness to Spot-check best two models - Multiclass Abstracts				
Models	**Scores**	**Mean Scores**	**Standard Deviation**	**Time**
XGBoost TF-IDF	Accuracy	**0.91311**	0.06231	8.55874
	Precision	**0.67302**	0.17445	7.36731
	Recall	**0.63797**	0.18511	7.51093
	F1-Score	**0.62705**	0.18463	7.56578
XGBoost Word2Vec	Accuracy	0.77879	0.09090	117.84614
	Precision	0.49916	0.10807	121.80696
	Recall	0.38704	0.06922	125.54688
	F1-Score	0.38240	0.07579	126.77967
SVM TF-IDF	Accuracy	0.90964	0.04233	3.24633
	Precision	0.75574	0.20005	3.23432
	Recall	0.64130	0.19200	3.24636
	F1-Score	0.66118	0.20345	3.23535
SVM Word2Vec	Accuracy	0.78354	0.11803	90.04591
	Precision	0.34155	0.09299	93.35307
	Recall	0.43333	0.08165	92.76617
	F1-Score	0.37946	0.08554	93.21798
Random Forest TF_IDF	Accuracy	**0.92832**	0.04662	6.42495
	Precision	**0.76384**	0.14097	4.34690
	Recall	**0.70767**	0.19640	4.31575

	F1-Score	**0.71916**	0.16634	4.39527
Random Forest Word2Vec	Accuracy	0.77443	0.11360	92.50002
	Precision	0.52543	0.17514	91.76488
	Recall	0.38267	0.07481	94.70404
	F1-Score	0.36214	0.08639	99.19329
Results from the script D_Spot-Check\Claims\Spot_check_Multicalss_Classification.ipynb				

Results test harness to Spot-check best models - Binary Abstracts

Models	Scores	Mean Scores	Standard Deviation	Time
Logistic TF-IDF	Accuracy	0.91377	0.00145	2.51451
	Precision	0.91377	0.00145	1.81618
	Recall	1.00000	0.00000	0.79388
	F1-Score	0.95494	0.00079	0.12168
Logistic Wor2Vec	Accuracy	0.91377	0.00145	34.86831
	Precision	0.91377	0.00145	35.81436
	Recall	1.00000	0.00000	37.58057
	F1-Score	0.95494	0.00079	38.35905
XGBoost TF-IDF	Accuracy	0.89638	0.02193	1.59075
	Precision	0.91219	0.00285	0.61336
	Recall	0.98095	0.02333	0.62932
	F1-Score	0.94522	0.01225	0.52958
XGBoost Word2Vec	Accuracy	0.92246	0.01709	38.97333
	Precision	0.92207	0.01630	39.24076
	Recall	1.00000	0.00000	39.48931
	F1-Score	0.95938	0.00872	39.33130
SVM TF-IDF	Accuracy	0.91377	0.00145	0.08976
	Precision	0.91377	0.00145	0.08976
	Recall	1.00000	0.00000	0.09076
	F1-Score	0.95494	0.00079	0.10168
SVM Word2Vec	Accuracy	0.91377	0.00145	37.45109
	Precision	0.91377	0.00145	37.75157
	Recall	1.00000	0.00000	38.53895
	F1-Score	0.95494	0.00079	38.93471
Random Forest TF_IDF	Accuracy	0.91377	0.00145	3.20009
	Precision	0.91377	0.00145	1.44985
	Recall	1.00000	0.00000	1.52966
	F1-Score	0.95494	0.00079	1.41242
Random Forest Word2Vec	Accuracy	0.90507	0.01781	40.39152
	Precision	0.91298	0.00240	40.40804
	Recall	0.99048	0.01905	40.66048
	F1-Score	0.95008	0.00995	39.44437
Results from the script D_Spot-Check\Abstracts\Spot_check_Binary_Abstracts.ipynb				

Data augmented results – Binary Abstracts

	Class*	Precision	Recall	Specificity	F1-score	Accuracy	Bal Accuracy	AUC
#Orig. XGBoost fitted	1 class	0.921053	0.985915	0.142857	0.952381	0.910256	0.564386	0.564386
Orig. XGBoost fitted Bal	1 class	0.920000	0.971831	0.142857	0.945205	0.897436	0.557344	0.557344
XGBoost TF tune	1 class	0.904110	0.929577	0.000000	0.916667	0.846154	0.464789	0.464789
XGBoost not tune	2 class	0.917808	0.943662	0.142857	0.930556	0.871795	0.543260	0.543260
XGBoost TF tune	2 class	0.917808	0.943662	0.142857	0.930556	0.871795	0.543260	0.543260

The number of new documents $n_{aug} = 11$
The percentage of the words changed $\alpha = 0.05$
*Number of classes augmented
trained model with the original labelled data
Results from the script F_Extra_Experiments_Abstracts/Data_Augmentation.ipynb