Exploration of text features representation on applications to the insurance domain

Author: Marc Garcia Ribes
Director: Joel Cantero Priego
Ponent: Javier Béjar

January 2022

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Contents

Abstract

The recent popularization of a more data-centric paradigm in machine learning has turned the study of the quality of feature representation into a very relevant topic. In this document an exploration of different text features representation applied to a Natural Language Processing task and evaluated by its performances.

The method proposed consists in analyzing various factors that can affect to the quality of the text feature and evaluate whether it is relevant or not. Two distinct forms of feature text representation is taken account: Bag of Words and word embeddings.

Resum

La recent popularització d'un paradigma més centrat en les dades en l'àmbit de *machine learning* ha fet que l'estudi de la qualitat de les representacions de les variables s'hagi convertit en un tema molt rellevant. En aquest treball, es realitza una exploració de diferent tipus de formes per representar text en variables aplicades a un problema de *Processat del Llenguatge Natural*, on s'avalua la seva eficàcia.

El mètode proposat consisteix en analitzar diversos factors que poden afectar a la qualitat de les representacions del text en variables i avaluar si són rellevant o no. S'estudien principalment dues formes de representació: model *Bag of Words* i els *word embeddings*.

Resumen

La reciente popularización de un paradigma más centrado en los datos en el ámbito del *machine learning* ha convertido el estudio de la calidad de las representaciones de las variables sean un tópico muy relevante. En este trabajo, se realiza una exploración de diferentes tipos de formas para representar el texto en variables aplicadas a un problema de *Procesado del Lenguaje Natural* y evaluar su rendimiento.

El método propuesto consiste en analizar diversos factores que puedan afectar a la calidad de la represntación del texto en variables y evaluar si són relevantes o no. El objeto de estudio son principalment dos formas de representació: el modelo *Bag of Words* y los *word embeddings*.

1 Introduction

1.1 Contextualization and scope

1.1.1 Context

This document is part of a *Treball de Fi de Grau (TFG)* of modality B from the Computer Science specialization for the insurance company *Grup Catalana Occident (GCO)* that consists in the exploration of text features within documents that belong to said company.

The number of freely available text data is increasing exponentially, and such large company as GCO has lots of texts to extract information from. Therefore, the exploration of possible ways to exploit this data is looked upon.

1.1.2 Concepts

The concepts which the reader needs to be familiar with are the following:

Machine Learning (ML)

Machine learning (ML) is the study of computer algorithms that can improve automatically through experience and by the use of data. A more refined definition would be:

> A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E. [1]

Natural Language Processing (NLP)

As many other concepts surrounding the world of *Computer Science*, NLP doesn't have a single agreed-upon definition. I decided to take [2]'s:

> Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

These concepts will be occurring throughout this document and will be developed into further detail later on.

Deep Learning (DL)

Deep Learning represent a subgroup of machine learning algorithms based on Artificial Neural Networks (ANN). Since the popularization of the ANN in the mid 2000's they have been at the lead of a lot of field within machine learning.

Bag of Words

One of the first approaches to map words into a vector space in the history of Natural Language Processing. In its conception a simple indicator of occurrence of a term in a sentence and later on it was transformed into the count of said term. These were very limited approximations to properly represent a complex vector space from words. Given a term t and a document d:

$$[B_d]_t = \begin{cases} 1 \quad if \quad t \in d \\ \\ 0 \quad if \quad t \notin d \end{cases}$$

Term Frequency - Inverse Document Frequency

Based on the same concept as Bag of Words, but instead of occurrences/counts each position of the vector, thus the value associated with a certain term, is represented by the product of the Term Frequency with the Inverse Document Frequency. These two measures are based on *information theory* and are defined, given a term t, a document d and a corpus D, as:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \qquad\qquad idf(t,D) = log \frac{N}{|d \in D : t \in d|}$$

Transformers

Transformers a novel family of models based in the encoder-decoder architecture. They stand as the state-of-the-art of NLP, as they present excellent results with a significant improvement in computational time due to them being highly parallelizable.

Bidirectional Encoder Representations from Transformers (BERT)

A new language representation model, designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications [3].

1.1.3 Problem formulation

The problem is based on the exploration of the advantages of the representation of text features, studying mainly the differences between the ones generated by a *language model* and others such as Bag of Words. This exploration will be done in the extraction of data from texts revolving accidents where at least one of the participants is insured by the company. The texts consist in the descriptions written by the surveyor of an accident.

1.1.4 Benefactors

The people who benefit from this exploration are mainly the data science department, as this project is in hope to gain a better understanding in the type of information we can extract from diverse texts, as to get a grasp of which approach should be taken in future occasions where NLP is needed.

On the other hand, the other benefactor is the author of the project, since it will help develop as a data scientist as well as gaining knowledge on the company.

1.2 Justification

The use of NLP around numerous sectors including corporations is growing fast due to the possible ways to exploit data that either is registered in form of text or it can be. In order to get a head-start on future projects the data science department has proposed to perform an exploration of text features representation so when a new NLP project arises, have already a throughout inside of which is the best way to tackle the data.

A justification why this exploration is relevant is because its done using texts that are centered around the insurance domain, which is the field that we want to get expertise on. This is a crucial difference between this exploration and others already performed. A lot of the works published around text features, revolve around very specific topics.

In the case of the bag-of-words model, is one of the most popular representation methods for object categorization. The key idea is to quantize each extracted key point into one of visual words, and then represent each image by a histogram of the visual words [4]. On the other hand, word embeddings comparation studies are performed in the medical field.

To sum up, this exploration allows us to compare with precision the models that are considered relevant for our work, in a very specific domain of data, in this case the insurance domain, allowing us to extract more meaningful conclusions about the comparative than the ones that could be provided by previous works on the subject.

1.3 Scope

1.3.1 Objectives

The main objective in this project is to explore different NLP approaches for a given text. As a secondary objective, when performing said techniques, to try to get a better grasp of the information in the various texts as well as a possible exploit of the prediction with the knowledge extracted.

Different targets are being considered. The most likely to be taken into account are the prediction if there is a third-party responsible for the accident, the classification of the type of the accident, a regression of the cost of the accident or to tell whether the writer of the statement is a surveyor or a a person from the technical service based on the text field in which the last operator or the surveyor assigned to the accident describes it.

There are also several partial theoretical objectives, being the main one getting a better understanding the use of NLP in a large company, as well as the research of the diverse techniques to approach texts. Being able to understand the architecture and principles on which are based the *Transformer* models is also a very relevant objective to me.

As for the practical aspects, the main targets are getting experienced in the development of a machine learning project, hone coding skills as well as getting to know new approaches and techniques with *Python* and its libraries.

1.3.2 Requirements

The requirements of this project are the following:

- Powerful computation engines (Cloud)
- Access to the texts in a friendly format

1.3.3 Obstacles

During this project it may be possible to find numerous obstacles, the following might be the most likely:

- Coronavirus: The global pandemic that demands social distancing can be very meddling in business situations and can slow or make a process/transaction more difficult.
- Lack of knowledge: The amount of experience working and studying NLP from the courses from college is very scarce, so it can become a challenge to familiarize with such an extensive branch of *Machine Learning* in a small period of time.
- Deadline: The date in which the project must be finished it is not too far from now so a bad decision or a complication in getting part of the data can be fatal.

1.4 Methodology

Due to the fact that this project must be finished in a short period of time, we will take an *agile* approach. The project will be divided in small tasks, described in further sections, so it is easier to assign a number of hours to be dedicated to each task. This also makes it easier to track the progress of the project in general. There will also be weekly meetings with the tutor of the project to be sure that the project is progressing at the right speed and with the quality that is expected in this kind of project.

We will be using *JIRA*[1] to create and manage these tasks, whereas all the code generated will be stored in a Git[2] repository for a better organization of the code and as a way to improve the control of the version as the project advances. The meeting will be done with *Microsoft Teams*. With this methodology we can ensure that:

- Tasks are properly formulated and they appear in a visual-friendly manner that will help to grasp the amount of progress that has been made
- The code generated will be stored securely and have tools for traceability in case that a mistake is done or a bad turn is taken at some point.
- The project will be carried out reviewed often so it reduces the probability of deviating from the objectives and interests of the company

1.5 Methodology update

There has not been any relevant change to the methodology since the initial proposition was very straightforward and adaptable:

- Planned and unplanned tasks have been divided into smaller subtasks in order to tackle them individually.
- The code generated has been stored in a repository so it could be tracked and easy to access.
- Regular meetings, once per week initially, have been occurring in order to discuss the decisions that have been taken to that point, and the next steps of the project. This may have been the most affected, since a reorganization of the number of hours of the project have been redistributed, more meetings have been scheduled in order to mitigate this event.

Seeing that with the chosen methodology we have been able to control task managing and code storage, we have decided to modify only the number director/developer meetings.

[1] https://en.wikipedia.org/wiki/Jira_(software)
[2] https://git-scm.com

2 Project Planning

2.1 Task definition

In order to set realistic goals and deadlines within the project, it is necessary to divide it in smaller tasks. The tasks that will sum up to the final project are the following:

- GEP deliveries: These are four documents that serve as a starting point for the project. They take up to **four weeks** and consist in the following:
  - Context and scope: Initial definition of the project, the problem to solve, justification and methodology. **25h**
  - Project planning: Finer granularity in the the projects subtasks and a more broad exploration of possible obstacles and alternatives to them. **9h**
  - Budget and sustainability: Analysis of a project's sustainability and the budget surrounding it. **10h**
  - Final project integration: Fixing errors of the previous documents and putting it all together for a final delivery. **19h**
- Research and study: This phase consists on gathering information on the techniques that are available and suited for the problem that we are trying to solve. It will be divided between two big tasks:
  - Research: In this task I will be searching for papers or any relevant information about NLP, Deep Learning and Machine Learning in order to get a better grasp of the main techniques employed currently and which is the state-of-the-art. The time that will be dedicated to this task may depend on numerous factors so it will only be a rough estimation as it may vary as the project advances. **35h**
  - Study: In order to set a solid foundations on Deep Learning and Machine Learning, I will be taking an online course offered by *Coursera*[3] called *Deep Learning Specialization* that consist in five courses on different aspects of DL namely: *Neural Networks and Deep Learning*, *Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization*, *Structuring Machine Learning Projects*, *Convolutional Neural Networks* and *Sequence Models*. **150h**
- Data gathering: In this stage of the project the efforts will be centered around getting the data necessary to apply the techniques explored in the *Research* phase. This will be divided in three tasks:
  - Deciding the data: It is rather obvious that the first point is deciding what type of data we want to explore. **10h**
  - Getting access: After choosing the data we need access to it. In this part there are third-party participants which can slowdown potentially the course of the project. **3 natural weeks**

[3] www.coursera.org

  - Process the data: Once we obtain the data a preprocess must be done in order to be able to *feed it* to the ML models. **30h**

- Model selection: Once we have the data ready to be ingested, different approaches will be tried in order to get the best results. This part will have a cyclic nature since it will consist in a trial-error iteration though several possible options:

  - Deciding the model/method: First, a model or algorithm should be chosen. This will be based from the *research* phase. **10h**
  - Training of the model: After selecting a model, it will be programmed and trained with the data and results will be extracted. **100h**
  - Analysis of the results: Once the model is trained, it will be tested and compared to the performance of the other chosen models. **50h**

- Documentation of the project and oral defense: Finally, once all of the experimental phase of the project is concluded, the documentation of all of the experiments with their results must be written down and start preparing for the oral defense of the project. **50h**

ID	**Name**	**Time**	**Dependencies**
T1	GEP delivieries	**63h**	
T1.1	Context and scope	25h	
T1.2	Project planning	9h	T1.1
T1.3	Budget and sustainability	10h	T1.1, T1.2
T1.4	Final project integration	19h	T1.1, T1.2, T1.3
T2	Research and study	**185h**	
T2.1	Research	35h	
T2.2	Study	150h	
T3	Data gathering	**40h**	
T3.1	Deciding the data	10h	
T3.2	Getting acces to the data	3nw	T3.1
T3.3	Processing the data	30h	T3.1, T3.2
T4	Model selection	**160h**	
T4.1	Deceding the model	10h	T3
T4.2	Training the model	100h	T3, T4.1
T4.3	Analysis of the results	50h	T3, T4.1, T4.2
T5	Documentation of the project	**50h**	T1, T2, T3, T4
Total		**498h**	

Table 1: Summary of the information of the tasks. [Own creation]

2.2 Resources

A fundamental part of a project is the resources that are need to carry it out said project. In the following part we will be detailing each kind of resource that is going to be used:

2.2.1 Human resources

In this project we find five human resources:

- The researcher: Responsible of the development of the project.
- The Data Science team: My coworkers are going o help me with insights and counseling throughout the project, as well as helping with the interdepartmental communication.
- The tutor: Responsible of guiding the researcher so it can achieve a good quality project. He is part of the before mentioned team.
- The data procurer: Responsible of giving access to the data necessary for this project.
- The GEP tutor: Responsible of guiding the researcher in the correct path during the first month, which is critical in order to be able to reach the deadline.

2.2.2 Hardware resources

The hardware that will be used in this project can be classified in two big groups:

- Personal hardware: This group refers to my mobile phone and personal computer. The computer has an AMD Ryzen 5 1400 Quad-Core processor with a freuency of 3.20 GHz, a memory of 16 GB, an NVIDIA GeForce GTX 1070 GPU and is running on a Windows 10 Professional.
- Corporate hardware: This hardware belongs to the company that I am currently working at and will consist in compute instances in Azure[4], which their specification cannot be determined since it will be decided once we know which techniques we will be using and the amount of data that we will be using.

[4] azure.microsoft.com

2.2.3 Software resources

In this section we will list the main software resources that will be used in this project:

- Pyhton[5]: The code will be developed in Python, and numerous libraries will be used such as: pandas, numpy, scikit-learn...
- SQL[6]: The data extraction will be performed with SQL.
- Azure: The experimentation will be carried out in Azure machines.
- JIRA: Planning and a task division will be done in JIRA.
- GIT: All the code of the project will be stored in a GIT repository.
- Microsoft Teams: Meetings with my tutor and my other coworkers will be done with Micorsoft Teams.
- Overleaf/LaTeX[7]: This is the software that will be used to produce the documentation of the project.

2.2.4 Material resources

As this is fundamentally a research project, bibliographic resources will be used to support the development and exploration of the project. These will consist in papers, books, thesis and web pages.

2.3 Risk management: alternative plans

There are multiple possible obstacles that can hinder in the proper development of a project, in this section we will ponder the probability and risk that each of the following possible obstacles can suppose to the project:

- Deadline of the project [Low risk]: A bad estimation or a poor workload balance can lead to not being able to comply with the stated deadline. The risk is low since the planning of the project is being done with enough margin to maneuver in case of any shortcomings. In case of not being able to keep up with the schedule set in the beginning it can be changed. We can also increment the number of hours of daily dedication to the project and narrow the scope of the project to avoid not getting deep enough in the subject.
- Inexperience with NLP [Medium risk]: As this is a new branch of DL for me to explore, it is possible that unforeseen obstacles can appear. This risk is trying to be mitigated with the courses that I am taking. In case of said obstacles appear and a reevaluation of the project is needed it, a realistic adaptation of the current project will be done and downscale it so it es feasible in the amount of time remaining.

[5] www.python.org
[6] en.wikipedia.org/wiki/SQL
[7] overleaf.com

2.4 Gantt chart

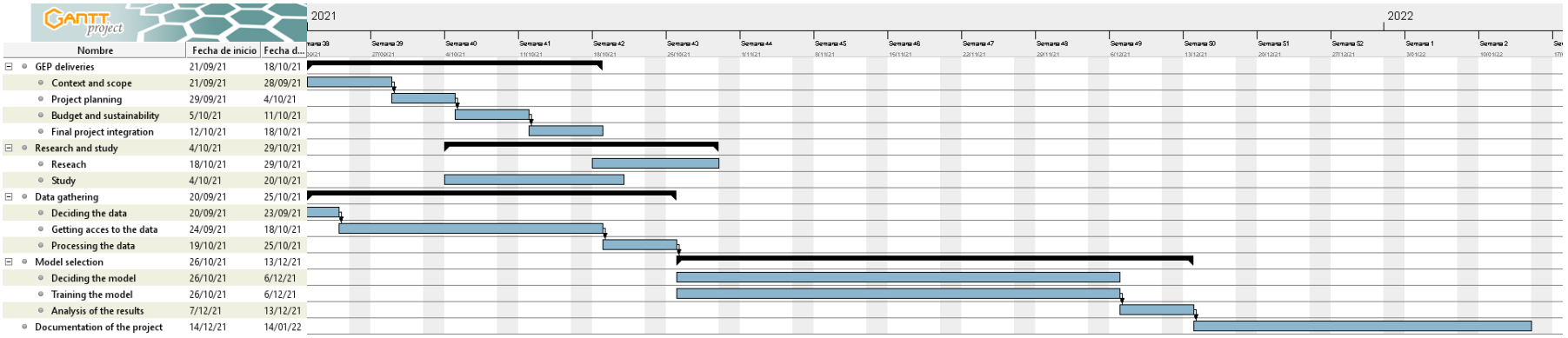


Figure 1: Gantt chart. [Own creation]

2.5 Planning status update

Following the schedule proposed in the Gantt diagram (Figure 1), the course of the project is a bit behind of what was initially proposed. The reason being that due to this project centering around text feature representation, the process of the text was going to take the most amount of time, out of the tasks *Data Gathering* and *Model Selection*. I would redefine this tasks and their respective time assignation the following way:

- Data gathering:
  - Deciding the data: It is rather obvious that the first point is deciding what type of data we want to explore. **10h**
  - Getting access: After choosing the data we need access to it. In this part there are third-party participants which can slowdown potentially the course of the project. **3 natural weeks**
- Model selection:
  - Process the data: Once we obtain the data a preprocess must be done in order to be able to *feed it* to the ML models. This includes trying different approaches to clean texts and extracting the vocabularies. Processing text takes a lot of time and difficulties may rise when exploring different possibilities. **200h** (Increase from 30h)
  - Deciding the model/method: First, a model or algorithm should be chosen. This will be based from the *research* phase. **10h**
  - Training of the model: After selecting a model, it will be programmed and trained with the data and results will be extracted. **50h** (Reduction from 100h)
  - Analysis of the results: Once the model is trained, it will be tested and compared to the performance of the other chosen models. **10h** (Reduction from 50h)

The *Process the data* subtask has been moved from *Data gathering* to *Model selection* task. There are 90h from the *Model selection* task that has been redistributed to the *Process the data* subtask. Since there is an increase of 170h and only 90h have been *compensated*, there are 80h of work that must be added to the schedule. As it was taken into consideration that more time should be dedicated to a certain task given an obstacle or a miscalculation of the time needed, an increase of daily hours is possible to be able to meet the time schedule. This doesn't affect the general scheme of the project nor adds unforeseen costs.

The project is currently in a mix of both tasks mentioned. A model has been chosen for the short texts and a great part of the methodology to process the text has been done.

3 Budget and Sustainability

In this section we will break down the budget for the project, including aspects like personnel's budget as well as other elements like hardware and electricity. Another section that will be analyzed is the sustainability as a concept and how it applies to this project in specific.

3.1 Budget

3.1.1 Personnel costs for activity

The personnel that participates in this project are listed in the human resources section. When accounting for the budget dedicated for each we need to estimate the amount of hours that the dedicate to their task. It is also needed an estimation of their salary so we can expect the result to have a considerable error. In other projects that are developed within a company, when planning the budget it is easier to access this kind of information so it can be more accurate.

Role	Job Position	Annual Salary*	Hours dedicated	Final estimation
Project manager	Data Scientist	38.005€	60	1.260€
Developer	Data Science Intern	17.280€	200	1.800€
Researcher	Date Science Intern	17.280€	185	1.665€
Technical writer	Data Science Intern	17.280€	113	1.017€

Table 2: Annual salary estimations for the different job positions performing different roles. Source for the average annual salary for a Data Scientist in Barcelona[5]. Source for the annual salary of a Data Science Intern [Own]. The final estimation cost is computed dividing the annual salary by 1750 working days multiplied by the number of hours dedicated to the project. Creation of the table [Own] *Including SS taxes

As it shows on Table 1, the salary estimations are based on the job position and not the role that is performed. The project manager role is carried out by the tutor on my data science team, whereas the developer, researcher and technical writer is done by me.

3.1.2 Computing costs

As mentioned before, the execution of the code is going to be done on the cloud platform Azure. This implies certain costs to the project. In one hand there is the subscription for the service. This cost is not going to be take into account since it is used by the whole Data Science team and it is totally independent to the project. On the other hand, there is the cost by the hour for executing an experiment in a specific compute instance. We are going to use a *Standard NC6 with 6 cores, 56GB RAM, 380GB storage and an NVIDIA Tesla K80*. The price for the execution by the hour of this compute instance is $1.17/hr. The number of hours estimated for the training of the models are 100h so approximately $117.

3.1.3 Hardware costs

The amortization formula used for the hardware costs is:

$$Amortization\,(\text{Euro}) = Product\,price\,(\text{Euro}) \times \frac{1}{Lifespan\,(h)} \times Hours\,used\,(h)$$

Therefore, the amortization for the following device is:

iMac (personal computer)

All the project is being done in this computer so the final cost of it is:

$$Amortization\,(\text{Euro}) = 1500 \times \frac{1}{4\,years \times 365\,days \times 24\,hours)} \times 498\,hours = 21,31 Euros$$

3.1.4 Other costs

These are the side costs that have appeared in the making of this project.

Travel

I travel to the office once a week, so in a month I make around 8-10 travels by train. Due to my location and choice of transportation this equals to around 11,35€ a month in travel expenses, since that is the cost of a *T-casual*, a train ticket worth 10 trips.

Food

In my weekly travel to the office I eat out since I am commuting from the university and I have no time to eat at home. It costs me around 6€ the meal, so this add up to 24€ a month in food.

3.1.5 Total costs

Other costs like electricity have not been taken into account since in this moment the price of electricity is very variable during the day thus making it hard to estimate. So the sum of the costs of the above mentioned aspects end up rounding to 5747€ + 101,25€ + 21,31€ + 11,35€ + 24€ = 5904,66€

3.2 Management control

Since the costs and hours dedicated are all estimations some of them if not all are going to differ with reality. To be able to account for these differences we will be proposing the following measures in order to help correct them:

Contingency budget

As we mentioned, estimations are bound to have some error, thus the need to have a contingency budget to be able to account for any unforeseen cost. We decided that the amount representing the 15% of the current budget will be added in a separated one to create this contingency budget. So, if the current budget is 5904,66€ the contingency budget will be 0.15 * 5904.66€ = 885.70€

Deviation when estimating the cost of a task

$$\sum_{i \in Tasks}^{Tasks} (Estimated\_cost\_per\_hour_i - Real\_cost\_per\_hour_i) * total\_real\_hours_i$$

From this formula we can obtain the exceeding or insufficient cost designated for a certain task. If it comes out as negative it means that we need to take the amount that outputs from the contingency budget since we underestimated the cost for that task. If it is positive it means that we overestimated the amount of money required for that task and it can be added to the contingency budget for other tasks that may need it.

Deviation when estimating the amount of hours for a task

$$\sum_{i \in Tasks}^{Tasks} (Estimated\_number\_of\_hours_i - Real\_number\_of\_hours_i) * cost\_per\_hour_i$$

In this case we are calculating the error in the amount of hours needed for a given task. It follows the same logic as the other one, if the result is positive it means that we have overestimated the amount of hours that the task requires and the spare budget budget will be reused for other purposes. On the other hand, if it comes out as negative it means that we underestimated the amount of hours that are required for that task and it will need a deadline extension and more money to be able to finish it.

3.3 Sustainability

3.3.1 Self-assesment

I personally think that sustainability is currently a word that is used around because it helps to come off as environment-friendly but often this 'sustainability' is not explained or justified. So I think it is very important that this term is given the importance that it deserves. I also didn't know the dimensionality of it until I read the poll that we were given. I also find interesting that it is also considered from an economic aspect aside from the more common environmental ones.

3.3.2 Economic dimension

Regarding PPP: Reflection on the cost you have estimated for the completion of the project

The estimation of the cost of this project is mainly based on the cost of the personnel and the Azure on-demand services such as computation clusters. We have considered that the hardware and software used is hardly relevant into the calculations since it is not an investment fully dedicated to this project so it is hard to evaluate its economic impact on a vacuum. Also a lot of side costs like electricity, transportation and other similar costs are either negligible or hard to take into account since it is split in too many factors. As I mentioned before, if this was a project which its entirety was run by a company or the university, it could encapsulate more easily all these factors so it would be possible to produce a faithful estimation.

Regarding Useful Life: How are currently solved economic issues (costs...) related to the problem that you want to address (state of the art)?

Transfer learning[8], as well as pre-trained models, are really useful tools in this cases since it allows to skip a big part of the development and therefore, the costs that imply, so you are able to fine-tune a model centered around to your specific problem without needing to expend larger quantities of money. In case that you develop a model you could also apply the same logic and reuse it later for another project.

In this project, due to the fact that it is centered around the exploration of text features and models that work with them, we pretend to use the results from it so in future projects related to NLP a more narrow exploration will be sufficient, saving time, resources and effort.

How will your solution improve economic issues (costs ...) with respect other existing solutions?

As mentioned in the anterior paragraph, other solutions might have required a from-scratch development, needing a lot of initial investment. This exploration is going to nurture

[8]https://en.wikipedia.org/wiki/Transfer_learning

from these investments and make use of already existent models to cut down the costs of this project.

3.3.3 Environmental dimension

Regarding PPP: Have you estimated the environmental impact of the project?

Machine Learning is estimated to cause a heavy impact to the environment due to the heavy consumption of electricity of GPU's, which are used to train ML models. So it is recommended to choose the right cloud provider. Microsoft Azure is certified carbon neutral and 44% of its electricity consumption directly comes from renewable energy, according to a 2016 estimate[6]. This way the environmental impact is reduced. The estimation of the cost for this project per se has not been done due to the difficulties on accessing the real data of consumption on the Azure computing with its specifics.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources?

As we have mentioned, the selection of the right cloud service can reduce considerably the impact on the environment. Also we will be reusing already trained models.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

In this case, as we are performing an exploration on text features we will be using already conceived text representations.

How will your solution improve the environment with respect other existing solutions?

As this is a comparative study and not a product, so it is not going to improve the environment.

3.3.4 Social dimension

Regarding PPP: What do you think you will achieve in terms of personal growth from doing this project?

These are the list of benefits that I think that I can get from this project:

- Better project planning and managing
- A better grasp of how to research and put across your findings
- Knowledge about ML and DL, specially NLP
- Growth as a relevant member in my work team
- New connections with other departments in the company

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

Through extensive trial and error or with large comparative studies from different sources.

How will your solution improve the quality of life (social dimension) with respect other existing solutions?

I don't think that there is a direct aspect in this project regarding the improvement of quality of life, although we expect to make easier for other people in the department their approach to similar problems.

Regarding Useful Life: Is there a real need for the project?

I think that there is a vast majority of cases where it is debatable whether a project is *necessary* as in my opinion it is a very strong word. I wouldn't consider it *necessary* as in a life or death situation but I believe that it can greatly benefit the level of knowledge regarding NLP on our team and at the same time solve a problem with the data that we will be using.

3.4 Knowledge integration

In this section we are going to mention the different knowledge that has been used for this project, that has been seen in the Computer Engineering degree:

- *Bases de Dades* (BD): Data bases is where the texts that we are using for this project are stored. Using proper SELECT statements has been a relevant part to extract the information needed.
- *Paral·lelisme* (PAR) : In order to process the text faster parallel approaches have been tried. Understanding how threads work and how to manage multiple access to a shared variable and knowing how to put together the processed output has been of paramount importancy.
- *Llenguatges de Programació* (LP): Python is the main language of the project. All the knowledge from how to code in python to comprehension lists and functional programming are key elements to the correct development of the project.
- *Aprenentatge Automàtic* (APA): As we are trying to develop a machine learning project, it is obvious that this subject has set a fundamental background so the project could be conceived.
- *Cerca i Anàlisis d'Informació Massiva* (CAIM): Due to the fact that we are working with text representation, concepts like stemming, lemmatization, tf-idf and text cleaning have been appearing throughout the project.

3.5 Legal framework

In this section we are going to contextualize the legal framework that we are bound to and why we have being lawful too.

Since this project is being developed in a company based in a country member of the European Union (EU), and we are working with data that can be subjected to physical individuals, the *General Data Protection Regulation* (GDPR) comes into matter[7] and we must follow certain conditions.

Although we are working with data related to natural persons, the identification variables are stripped at the beginning of the process. In particular, we are working with accidents and the identification is based on a unique ID of the accident. This can be used to search for the involved people in the accident. We are not using any personal data as we only extract the type of accident and the accident surveyor's description of the accident.

In conclusion, despite the fact that we are working closely with what could be considered sensitive data, the way that is processed and the uses given to models that feed from this data are within the current law, so we can rest assured of the legality of the project.

4 Theoretical framework

NLP has been a relevant topic since the late 1940s and has only gained popularity and relevance throughout the years. Digitization and growth of technology in the corporate industry have made the study of NLP paramount for processes automatization in the Information Retrieval (IR) world such as Document Clustering (DC) and Text Classification (TC). Early studies in the subject consisted in Machine Translation (MT) in order to decipher messages in World War II [8] and as early as the 1960s the first TC problems have been approached [9]. Although being part of the artificial intelligence, it was not until later on with the breakthroughs of machine learning that both subjects started combining to produce the best results.

Machine learning, as mentioned in point 1.1.2, is the study of computer algorithms that can improve automatically through experience by the the use of data. The features extracted with NLP methods can be fed to a ML algorithm to be able to perform numerous tasks. In this document we will focus mainly in text classification tasks. SVM have been proved to be a successful method for categorizing text among other machine learning algorithms [10], although with the the popularization of neural networks and the rise of the Deep Learning field we have chosen to use the latter.

Deep learning (also known as deep structured learning) is a subset of ML algorithms based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised [11]. A network of units called *neurons* are used to find weights to ponder the connections between these neurons and the input features to be able to generate a certain output.

4.1 Previous work

In Song et al. 2005 [12] an explorations of the relevance of the five text representation factors (stop words removal, word stemming, indexing, normalization) to the text classification problem. Within other conclusions, removing the stop-words and using Inverse Document Frequency is highly relevant for the performance of the learning model efficacy.

For text representation involving word embeddings, there is Mascio et al. 2020 [13] where different preprocessing were applied to the text alongside with diverse models such as SVM, Bi-LSTM and RNN among others, also using different embedding methods (Word2Vec, GloVe, BERTbase, ...)

4.2 Machine learning

In order to understand the work done in this project, a first introduction to ML and data science is mandatory. Machine Learning is a multi-disciplinary field closely related to Computational Statistics and Mathematical Optimization. It is based in the Artifical Intelligence paradigm of autonomous learning through data. It has been associated with multiple fields such as Information Theory, Philosophy and Psychology among others. [14]

4.2.1 Data Science problems and Machine Learning

Data Science has been defined by Hayashi and Chikio [15] as "a concept to unify statistics, data analysis, machine learning and their related methods in order to understand and analyze actual phenomena with data". Machine Learning now has laid an extense toolkit to be able to solve the problems that are formulated in the Data Science field, and proven to be successful at it.

Classification Problem

This family of problems consist in classify an input set into a fixed number of classes. It can be binary, e.g. Given a set of clients and their profiles, classify them into solvent or not, or multi-class e.g. Given a set of flowers and their properties classify them into their group/species. Therefore, classification work on the realm of categorical classes. This is a very popular problem and it is seen across both scientific and corporate environment.

Regression Problem

As opposed to the classification problem, regression problems work in the continuous scale and its outputs are numeric. There are some intersections between both problems as some regression problems can be converted into a classification problems through the discretization of the output. So usually the problems revolve around questions such as "how many" or "how much". E.g. Given a set of vehicles and the circumstances of an accident that they have been involved, predict the cost of the repairs. So in this case it would be, "How much does it cost to repair this vehicle given this accident information?"

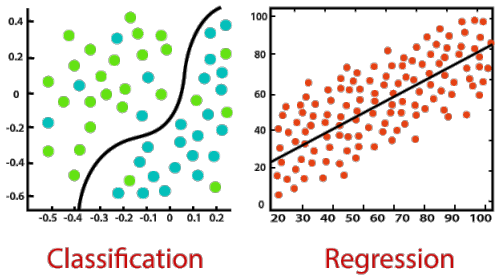


Figure 2: Comparation between classification and regression
Source: javatpoint

Clustering Problem

The Clustering Problem falls in a different category than the both already mentioned problems, as it doesn't rely on a ground truth to learn. Based in the parameters of the algorithm and the measures used, it groups data into *clusters*, which are groups of similar data point, in order to identify distinct types of data among a large pool of instances.

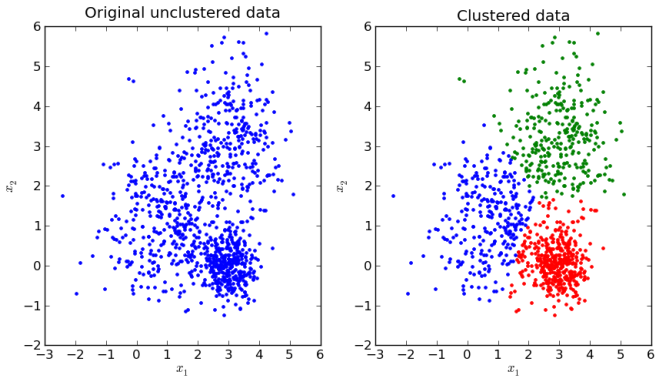


Figure 3: Example of clustering with K-Means
Source: https://i.stack.imgur.com/cIDB3.png

Dimensionality Reduction

Dimensionality Reduction refers to the task of transforming data from a higher dimensional space to a lower one, close to its intrinsic dimension ideally, while keeping its *properties* of the original space. This topic is closely related to feature selection, that consists on retaining the minimum number of variables while getting the best results possible (within a certain margin of tolerance).

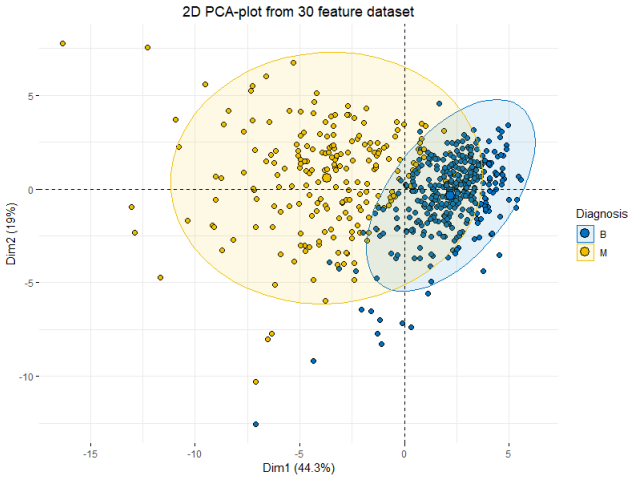


Figure 4: PCA-plot showing clustering of class "B" and "M" across 30 features.
Source: Peter Nistrup, Jan 29, 2019

4.2.2 History of Machine Learning

The concept of artificial intelligence, alongside with the concept of machine learning have been around for a long time. Although a lot of the core of machine learning, such as algebra and statistics had been found a lot prior to it, machine learning started gaining thrust in the late 40's early 50's. Alan Turing's extensive research on computation and the design of the *Turing Test*, which consisted on a conversation between a human and what it could be possible a machine, where if the machine convinced that itself was a human, the machine would be deemed intelligent, set a foundation for what would later on evolve into modern concepts of artificial intelligence.

During the 50's learning algorithms were created to play the popular game *Checkers*. On the following years, games like chess and *Go* will become a medium for machine learning and AI to show its advances, and eventually surpass the current champions in their respective field. During a conference in Dartmouth in 1956 AI was born, two years later the base of deep learning, the *Perceptron* would be conceived by Frank Rosenblatt.

Figure 5: Deep Blue vs Kasparov (1996)

Between the 60's and 80's numerous advances in speech, pattern recognition and robotics were made. And entering the 90's the focus of ML started to shift to a more data-driven perspective. Machine Learning was implemented in order to analyze and process large sets of data as input and extract thoughtful conclusions from it. [16]

After the late 90's, big corporation starting with IBM and followed by Google, Facebook and Amazon would start developing their own products based on machine learning, specially after in 2006 the concept of Deep Learning was settled the solutions started to shift into this new path, as deep learning allowed to receive larger volume of data.

4.2.3 Generic model of Machine Learning

Amongst the different components of the development of a machine learning solution, we are able to generate a general process without relying on the chosen algorithm.

1. Collection and preparation of dataset: This step consists on the gathering of data and its process in order to be able to feed it to a machine learning algorithm. Major decisions are made at this point, since depending on which data you decide to input to your model, it is going to affect highly its performance. The type of data to be used, the normalization, cleaning, feature engineering and numerous other processes are part of this first stage. It is said that alongside with feature selection it ends up being the most important relevant part of a machine learning project.

2. Feature selection: At this second phase of the process, once the data to be used as input is decided and formatted, a selection of the most relevant features can be made in order to limit the dimensionality of the data. This process can be paramount in order to be able to maintain a more efficient model and avoid redundancy on data, correlated variables, or even affecting the performance of the model (garbage-in garbage-out). There are numerous methods for feature selection, but they all fall into one the three following categories:

    - Filter strategy (usually based on information gain)
    - Wrapper strategy (guided by the performance of the chosen model)
    - Embedding strategy (guided by the performance of the chosen model during the training)

3. Choice of the algorithm: Different machine learning algorithms solve better different problems. The most basic example is that linear data can be easily treated with linear model but more complex data require non-linear models to solve the problems related with them. Therefore, a selection of the most suited algorithm is key in order to get the best results.

4. Selection of model and parameters: There is a vast majority of models that require a configuration of its parameters or are of a stochastic nature, thus giving different results in variance of these parameters, so an exploration of the parameter or *hyperparameters* is sometimes crucial for a throughout search for the best version of a model. Some of these parameters can be the random state of initialization in case of stochastic algorithms, regularization in neural networks, the number of cluster in K-means, and so on and so forth...

5. Training: Once the model is set there is still decisions revolving the training of it, in case of neural networks there is the number of epochs and the batch size as an example. The metrics that will evaluate the model are also to be considered at this point e.g accuracy, precision, AUC... The partition of the dataset it is also a factor of relevance. Normaly it is split into training, validation and test sets. The size of each one and if tehre is going to be some type of cross-validation are also going to affect the validity of the results of the model.

6. Performance evaluation: Having the model trained, now it is time to evaluate how well the proposed problem is solved with it. This will be based on the metrics that have been chosen in the training part alongside with the evolution of the model along the training. Also, at this point we are going to evaluate if the model is overfitted or underfitted in order to properly grade the model and if modifications are in place to get better results.

4.2.4 Machine Learning paradigms

Machine learning can be grouped into three big groups of algorithms based in how the data is presented to them and how the algorithm process it. There are other paradigms such as hybrid learning and instance-based learning but we will only consider the three following.

Supervised learning

Algorithms are considered that follow the supervised learning paradigm if they are trained with the labels or *ground truth* they are expected to predict once trained. So if you were building a model to predict if a client is fraudulent or not you would train the algorithm with sets of clients along the chosen characteristics that may be relevant to identify a fraudulent client and then pointing out to the algorithm which of these instances were actually fraudulent or not. Classification and regression algorithms often fall into this type of learning.

Unsupervised learning

If giving the labels or *ground truth* to the training algorithm is considered *Supervised learning*, then not giving it is what *Unsupervised learning* is understood for. This consists on the own algorithm identifying the different labels in the data. The most common example for this is clustering as the same algorithm based on the properties of the data identifies the different groups present in it.

Reinforcement learning

Unlike the two other paradigms, this one distances itself a bit from both as its classification is not based of whether the data has labels or not. Instead, it is formed by algorithms known as *software agents* ought to take actions given an environment and the stimuli that comes from it. This translates into an accumulative learning based on the data incoming from said environment. It often follows a scheme of punishment-reward in function of its response to its environment. This is why its present in numerous disciplines such as game theory, genetic algorithms and much more.

4.3 Deep learning

Deep learning (also known as deep structured learning) is a subset of ML algorithms based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised [11]. A network of units called *neurons* are used to find weights to ponder the connections between these neurons and the input features to be able to generate a certain output.

4.3.1 History

Sources point out that most of the bases for the nowadays neural networks where laid and explored by Frank Rosenblatt [17]. It was him in 1962 to conceive the *Perceptron*, which would become the most stripped down version of the modern ANNs. Computational barriers would stale the advance on the topic until further decades.

The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1967 [18]. Others, more specific architectures, would follow in the next years in different fields such as computer vision. Deep Learning as a concept was introduced to the ML community by Rita Dechter in 1986 [19] and in 1989 Yann LeCun et al. applied the standard backpropagation algorithm that was used for other purposes since the 1970. Then again, computational barriers made not very feasable for an extensive usage since the training took 3 days. [20]

With the progress of Graphics Processing Units (GPU) trainings with Artificial Neural Network became more feasible. ANN gained popularity when they started outperforming other machine learning models in competitions, more notoriously the ImageNet competition and other challenges.

4.3.2 Perceptron

The Perceptron was the beginning of what later would be the modern neural networks. It is an algorithm for supervised learning that performs a binary classification. It belongs to the family of linear classifiers.

The definition of the function that computes the perceptron is called a threshold function, it maps an input real-valued vector x to a single binary value output:

$$f(\boldsymbol{x}) = \begin{cases} 1 & if \quad \boldsymbol{w} \cdot \boldsymbol{x} + b \\ \\ 0 & otherwise \end{cases}$$

4.3.3 Artificial Neural Networks (ANN)

The concept of Perceptron evolved into a series of layers formed by processing units called neurons. The stacking of this layers connected between them would allow complex and deep computation.

Concept and architecture

The concept behind ANN is inspired by the biologic neural networks that are present in the brain of animals. It is essentially formed by the processing units *neurons*, that try to emulate the neurons in a "real brain", and the connection between them called *edges*. Both components have assigned weights that are modified during the learning process in order to solve an optimization problem called *Loss function*.

The neurons are arranged in layers and neurons from different layers can or not be connected to other neurons from different layers. This disposition evokes three different basic types of layers present in an ANN:

- Input layer: The first layer of the network. It receives the feature vector and does not have weights assigned to it. It must have the same number of neurons as features has the vector.
- Output layer: The last layer of the network. It is the responsible of generating the output of the network and is where all the networks learning converges.
- Hidden layer: Any layer that is not the input or output layer is considered a Hidden layer.

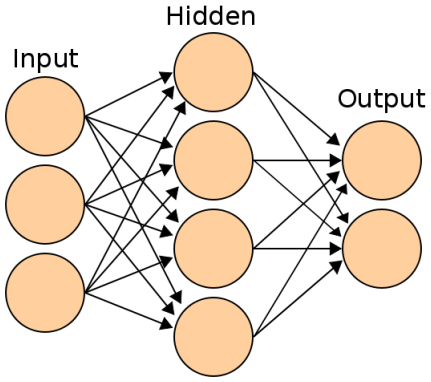


Figure 6: Different kind of layers in an ANN
Source: Wikipedia

Training and *Backpropagation*

The process of modify the weights in order to obtain the expected results when processing an input is called training. In other machine learning models the training is done by formulating a series of explicit derivatives, in the case of ANN the formulation is recursive. Therefore, to be able to compute efficiently at each step of the training the derivatives an algorithm called *backpropagation* [21] is used. It consists of the following steps:

1. Forward propagation: In this step the the input is processed through all the layers and the results of each layer are computed and an output is generated

2. Backward propagation: The output generated in the forward pass is compared to the ground truth and the error is calculated using an *Error function* and propagated from the last layer to the first using the gradients

Regularization

The complexity and high prowess of the ANN tend to do overfitting on the data. To mitigate this there are different ways to *penalize* this tendency thus swaying the training from this direction.

There are two principal forms of regularization, the first to be introduced being the L2 regularization. The idea behind it was to try and pull down the weight values so the complexity of the model would decrease. The formulation being the following:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} E_i(\theta) + \frac{\lambda}{2}||\theta||^2$$

Thus balancing the minimization of the training error and simultaneously the magnitude of its weights. The parameter responsible to ponder how much the weights are regulated is λ which is called *regularization parameter*, with higher values of it reduces the chances to overfit the data and increases the odds of underfit it.

The second form of regularization is widely used and is called *dropout regularization* [22]. This consists on setting to 0 with a given probability the outputs of random neurons so the layers that are connected to them do not see the result. This forces the learning to be more distributed along the network and not use the same *paths*.

Loss function

A *Loss function* is a function that maps values of diverse variables into a real number, thus producing a measurement for the error of a punctual event associated with these variables, in the case of ML or DL maps the predictions and the targets. Intuitively, different problems and different types of data will require different *Loss functions* to be able to produce a meaningful scoring for the error. The *Loss functions* is averaged among all the input, thus it main content is in the inner *error function* $E_i(\theta)$:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} E_i(\theta)$$

For linear data, the most common loss functions are *Mean Squared Error* (MSE) and *binary cross-entropy*. The MSE computes the average squared differences between he estimated values, the prediction, and the actual value, the ground truth.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$$

Where Y_i is the label or ground truth for the i-th sample and $\hat{Y_i}$ is the estimation or prediction for the i-th sample. *Binary cross-entropy* on the other hand is used for binary classifications e.g. Logistic Regression.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Where y is the label or ground truth and $p(y)$ is the predicted probability. This formula implies the existence of it being a *positive class* and a *negative class* since the labels are going to be either 1 or 0.

Other well known losses such as *Mean Absulute Error* MAE, exist or *Categorical cross-entropy*. In conclusion, different problems, data, and network configuration will evoke different *loss functions*, which will be the function that the model will be optimizing in order to minimize the error. In the next section we will describe some of the possible optimizers.

Optimizers

Once the network is designed and the loss function is defined, its time to choose the solver for the proposed optimization problem, typically formulated as $min_\theta L(\theta)$ where L is the chosen loss function.

Usually, the complexity of the problem does not allow to find a global optimal point through analytical solutions, thus an iterative procedure is called for. The most popular technique is *Gradient Descent* (GD) [23]. The weights are updated in the following manner:

$$\theta := \theta - \alpha \frac{\partial L(\theta)}{\partial \theta}$$

Where α is the *learning rate* which is a tuning parameter that represents the step size of each iteration in the process of optimization on a loss function.

Activation functions

Activation functions are part of the neuron, and the are done over the result of the process of that neuron. Alongside with the layer disposition, these components are the responsible for the nonlinearity of the model. Normally the same activation function is done in all the neurons of a same layer.

The sigmoid function was the first activation layer that was used. The funtion is defined as $\sigma(x) = \frac{1}{(1+e^-x)}$ so it maps an input from $(-\infty, \infty)$ to a range of $[0, 1]$. One of the main drawbacks of it are the problems of gradient vanishing around the tails of the function near 0 and 1, where the slope is very small and most of the values fall in those regions. It also is not zero-centered and is computationally more expensive than other activation functions i.e. linear operations.

This function was widely used in the early years of deep learning but it was substituted for the *hyperbolic tangent* which appears to perform strictly better in most cases. Thus the sigmoid function is now mainly used on the output layers. The *hyperbolic tangent* suffers from the same problems as the sigmoid function with exception to it not being zero-centered.

$$tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

With the popularization of convolutional layers for image processing, the *Rectified Linear Unit* (RELU) was conceived. It consists of taking the max between 0 and the input, so it is computationally inexpensive and maps inputs from $(-\infty, \infty)$ to a range of $[0, \infty)$. This was helpful as in a lot of cases it was not desired to have negative values in layers that processed data from images.

There is a wide varieties of activation functions, including variations from those mentioned before i.e. *Leaky ReLU*, but these are the most known and used actually.

Types of ANN

Aside of the typical structure seen previously in this document, there are different configurations of ANN to be able to tackle different types of data, specially unstructured data such as images or text.

Images are processed typically with what is referred as a *Convolutional Neural Network* (CNN). What makes this type of neural network to the ones already seen is the presence of specific layers that help process the data representing images or collections of them. Namely:

- Convolutional layers: The main idea behind this layers is to process the 3-dimensional matrix representing an image or a collection of them with a *convolution kernel*
- Pooling layers: This layers are responsible for synthesising the information after its processed by convolutional kernels, so they are always applied after one.
- Dense layer: This type of layers is not exclusive to CNN but are always present in order to process the transformations made by the convolutional and pooling layers.

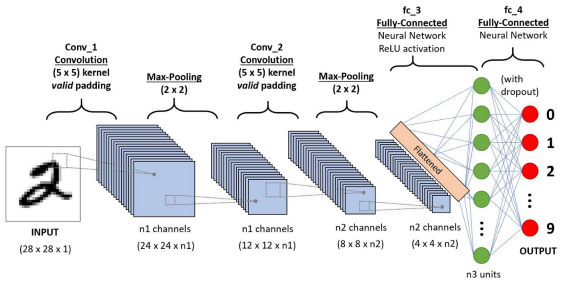


Figure 7: Example of the architecture of a CNN
Source: Towards Data Science

Text and other sequential data on the other hand are processed with *Recurrent Neural Networks* (RNN). As the name suggests, this type of ANN is based on applying *recurrent units* in a sequential fashion for the input of the network.

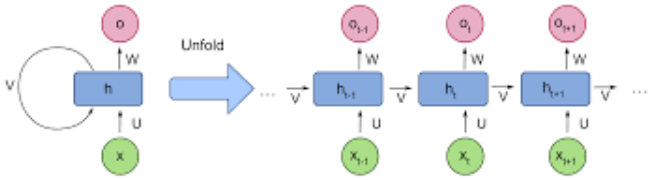


Figure 8: Example of the architecture of a RNN
Source: Wikipedia

4.4 Bag of words

Bag of words can be traced as early as 1954 [24] in a linguistic context defining a set of not contextualized words to map to a text. This kind of representation has been present in most of works in various forms during the evolution of NLP and also in the Computer Vision (CV) world for image classification. It stands today as one of the most intuitive models that can represent text features. Different approaches have been taken to solve its shortcomings such as dimensionality (Latent Semantic Analysis) or relevance (Term Frequency - Inverse Document Frequency) to the problem that is trying to solve.

The Bag of words model belongs to the family of models known as *Vector Space Models*. It consists in generating a set of unique words called often vocabulary or dictionary that contains every single word in a collection of documents, also known as corpus. Document is understood as a series of words, sentences that usually maintain a certain relation between them or it may also consist in the classical definition of a document as in dissertation or an article from a newspaper.

Let these two sentences be documents, d_1 and d_2, forming a *corpus* $D = \{d_1, d_2\}$:

d_1: Marc finds text features exploration fascinating, although he is not sure about classification.
d_2: Núria does not think that text is a good feature for classification.

The vocabulary or dictionary (V) generated from D would be the following:

V = {Marc, finds, text, features, exploration, fascinating, ',', although, he, is, not, sure, about , classification, '.', Núria, does, think, that, a, good, for}

Now that we have every possible word in our vocabulary we can represent each document as a vector of size V where $\{V_i | i \in [0, V)\}$ is 1 if the i-th word in the vocabulary appears in the document, 0 if not. Therefore, d_1 and d_2 are represented as:

$$d_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$$
$$d_2 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1]$$

At this point, d_1 and d_2 are represented as vectors in a V-dimensional space. But the amount of information in this representation may often fall short. It can be improved by instead of having values based on if there is an occurrence of the word or not, count the number of times that said word appears in the document. So instead of a binary vector, its values would range between [0, V].

4.5 Term Frequency - Inverse Document Frequency

Term Frequency - Inverse Document Frequency, for short TF-IDF, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus [25]. It stands as one of the most relevant term-weighting methods and is present in most of the search engines.

4.5.1 Motivation

As mentioned before, TF-IDF is a form of term-weighting that tries to give more weight to the most *relevant* words in a corpus. To do so, the weighting is formed by the *Term Frequency* which is translates to the frequency that a word appears in a document. The other factor is the inverse document frequency which tries to give more importance to words that are not present in every document of the corpus. These two concepts where conceived at different times, the first being by Hans Peter Luhn in 1957, who argued that:

> The weight of a term that occurs in a document is simply proportional to the term frequency.

Later on, Karen Spärck Jones in 1972 gave a statistical interpretation for term-specificity (IDF):

> The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs.

4.5.2 Definition

Now knowing that TF-IDF is the product of two factors and knowing the general concepts of each, let us describe precisely the definition of both factors.

Term frequency

Term frequency is defined as $tf(t, d)$ where t refers to the term that we are evaluating and d the document in which we are evaluating t's frequency. So the definition of tf will be:

$$tf(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Where:

- $f_{t,d}$ is the raw count of a term t in a document d
- $\sum_{t' \in d} f_{t',d}$ is the number of terms in a document d

Inverse document frequency

The inverse document frequency is a measure to ponder the term frequency with how relevant is the word for the rest of the corpus, i.e. how much information provides. An example of a very frequent word in English with no informational value is the pronoun *I*, since appears very frequent, allegedly the most frequent word in English language, so it being present in a document does not give us value of what makes that document distinct or relevant, therefore, its IDF being low. So let us define IDF as:

$$idf(t, D) = log \frac{N}{|d \in D : t \in d|}$$

Where:

- N is the total number of documents in the corpus D
- $|d \in D : t \in d|$ is the number of documents where the term t appears

It is logarithmically scaled to *dampen* the importance of a term that has high frequency.

Term Frequency - Inverse document frequency

Having both TF and IDF defined, let us formulate TF-IDF:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

Therefore, a high tf-idf is achieved with a high frequency term and a low number of occurrences along the corpus, thus being relevant in a document and not overpopulated in the the rest of documents.

It is worth noting that tf is applied given a term t in a certain document d whereas idf *operates* in a *corpus level*. These facts make their computing very different, and interesting since idf can be precalculated and used as a filter for the size of the vocabulary, making easier the rest of the computation.

4.6 Word embedding

In the realm of NLP, word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning [26]. Therefore, by the use of language modeling and feature learning words are mapped to vectors. This can be generated by neural networks or dimesionality reduction on the word co-occurrence matrix amongst others.

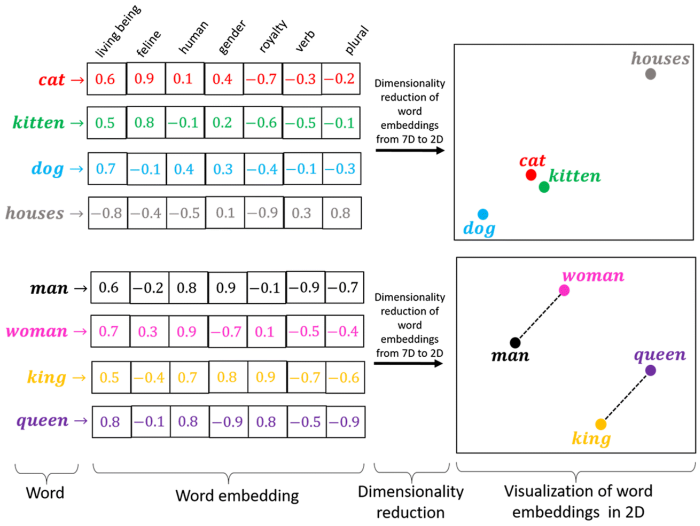


Figure 9: Visual representation of a word embedding
Source: https://medium.com/@hari4om/word-embedding-d816f643140

The first versions of it resulted in sparse vector spaces of high dimensionality and relied on probabilistic models or algebraic methods to reduce the number of dimensions. After 2005 word embeddings started to rely on ANN, this fact and the advances on theoretical work on the quality of vectors alongside with training speed allowed the creation of toolkits that can train vector space models more efficiently. In 2013, Tomas Mikolov led a team at Google to create *word2vec*, and in 2014 GloVe was born. In the last years, with the new concept of attention, contextual embeddings have been created using transformer architectures (e.g. ELMo, BERT).

4.7 Transformers

Transformer models have taken over the NLP lead, becoming the state-of-the-art after years of dominance by Recurrent Neural Networks (RNN). Although similar in manner, encoder-decoder architecture and the use of word embeddings, they attempt to solve of computational bottleneck present in RNN due to their sequential nature. Instead, transformer models are dense models, offering a higher potential parallelization.

The focus of the transformer success is around the use of only attention mechanisms thus not using recurrence to avoid the sequential nature of the problem. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [27].

4.7.1 Architecture

As previously mentioned, transformer models follow in similar fashion the encoder-decoder architecture present in the most competitive RRN models. The original paper proposes two stacks, one being the encoder stack and the other one the decoder stack, of six layers, each layer presenting two sub-layers, the first being a *Multi-Head Attention block* and the other one a fully connected *feed-forward network*. The decoder stack follows the same pattern but instead of two sub-layers, adds a third one at the beginning, which is responsible for performing multi-head attention over the output of the encoder stack.

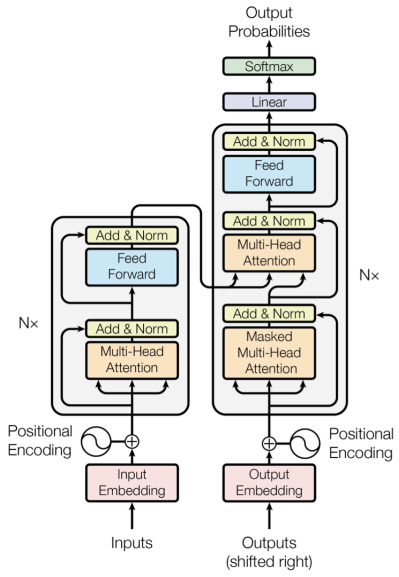
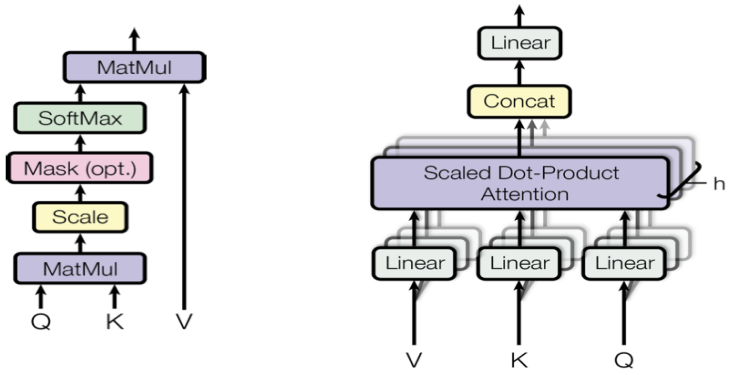


Figure 10: The Transformer - model architecture
Source: Attention Is All You Need [27]

4.7.2 Attention

We have already described the attention function before, now let us go over with detail into the two types of attention blocks proposed.



(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

Figure 11: Attention blocks
Source: Attention Is All You Need [27]

Scaled Dot-Product Attention

The input is formed by queries and keys of dimension d_k, and values of dimension d_v. The dot product is computed of the queries with all the keys and divided each by $\sqrt{d_k}$, then a *softmax function* is applied in order to produce the weights on the values. The computation is performed over a set of queries at the same time that have been formatted into a matrix Q. The keys and values are also layed as matrix K and V.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-Head Attention

Prior to the proper attention block, the queries, keys and values are linearly projected h times. Each h_i projection is different and it is learned. After the projection, the attention function is performed in parallel, yielding d_v-dimensional output values that are concatenated and projected once again.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

Where:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

4.7.3 Pipeline

This section will pretend to describe the overall process that the data goes through until a result is generated. The input consists of a sequence of words and the output must generate another sequence of words, i.e. translated sentence, following word.

1. Embedding algorithm: In order that the model is able to process the data, words must be converted into meaningful features. Word embedding maps each word into a vector in a *vector space* where ideally similar words are represented more close than others. This embedding happens in the first encoder, the other encoders receive as input the output of the previous encoder.

2. Self-attention layers: Here, the vectors generated by the embedding algorithm are processed in five distinguishable steps:

   (a) Three vector are created (q, k and v) for each embedding

   (b) A score is generated, using every other word of the input to determine how important is the word to the rest of the sentence. This score is generated performing the dot product of q and k

   (c) The scores are divided by $\sqrt{d_k}$, in the paper they use 64 dimensions so 8, and then they undergo through a softmax operation

   (d) Each value vector is multiplied by the result generated by the softmax function

   (e) All the weighted value vectors are summed up

3. Addition and normalization: Before the results of the self-attention layer is passed on to the feed-forward neural network, the output is added and normalized with the residual connections of the values before they are processed by the self-attention block.

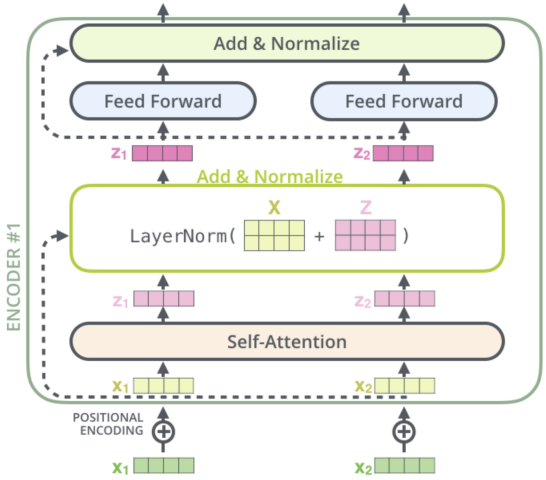


Figure 12: Addition and normalization between residual connection a self-attention results
Source: The Illustrated Transformer [28]

4. Feed-forward neural network: At this stage, a fully connected feed-forward network is applied to each position separately. The network is build with two linear transformation with Rectified Linear Unit (ReLU) activation in between.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

After being processed by each FFN, another step of addition and normalization is made between the results of the previous addition and normalization and the results of the FFN as seen in Figure 12

5. Decoder stack: After the last encoder block outputs its result is then passed on to each decoder block in the form of attention vectors K and V. This will be processed between the self-attention layer and the feed-forward network, in the layer called *Encoder-Decoder Attention*.

6. Linear transformation and softmax: When the last output of the decoder stack is generated in the form of a stack of vectors of floats this is projected by a a fully connected neural network into a larger vector of logits. Each logit represents the score of its corresponding unique word present in the vocabulary, so after applying a softmax we interpret the transformed logits as probabilities, therefore picking the highest one and outputting it as a result with its corresponding word.

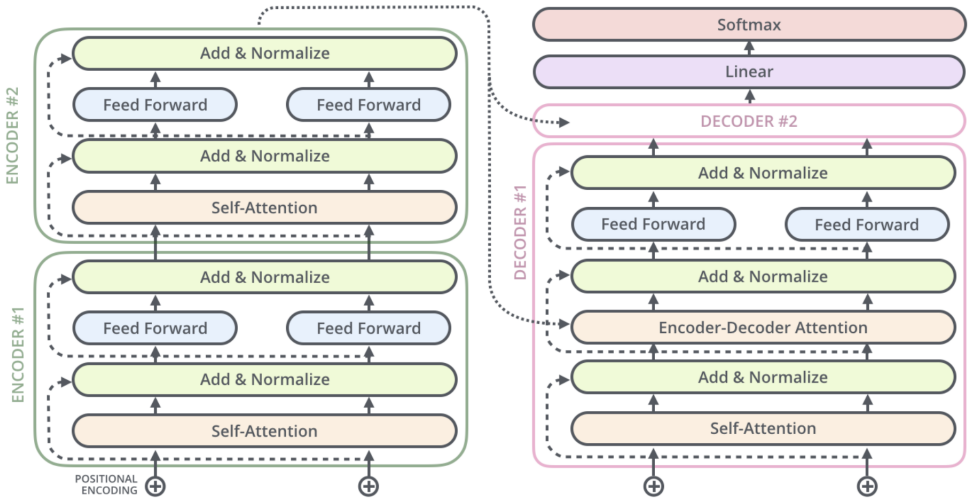


Figure 13: Encoder and decoder stacked - Data flow
Source: The Illustrated Transformer [28]

4.8 Bidirectional Encoder Representations from Transformers (BERT)

A new language representation model, designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications [3].

After the Transformer architecture was proposed, different versions started being published in order to offer improvements to the original concept. One of the first was the BERT, in which they decided to remove the decoder part of the Transformer architecture. This allows the model to either be more deep by stacking more encoding blocks or just perform faster in general. Also, the the training of weights and embeddings is done prior to the main purpose of the model, the task being *Masked Language Model* (MLM) alongside with *Next Sentence Prediction* (NSP).

MLM consists on masking random parts of the training sentences so the model have to guess the missing inputs. This is done so the training can be done bidirectionally without allowing the model see the word that it is processing with itself. The second task NSP consists of a binarized *next sentence prediction* task that consists in guessing which is the next sentence of the input. The training is done so the option are 50% of the time the right option and the other 50% is a random sentence in the corpus. The pre-training of this task is key for tasks such as *Question Answering* (QA) and *Natural Language Inference* (NLI).

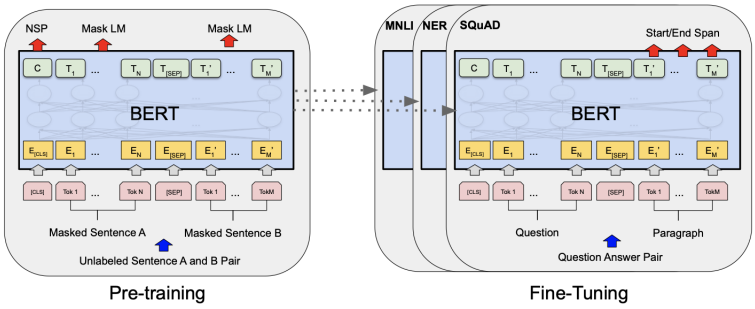


Figure 14: Overall pre-training and fine-tuning procedures for BERT
Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [3]

There is a second aspect to the BERT, that is the fine-tuning. For each task, it is plugged into the model the task-specific inputs and outputs into the model and the parameters are fine-tuned end-to-end. This results in a relative inexpensive process compared to the pre-training section.

5 Experimental setup

This section will document the intentions of this exploration and how the comparative is going to be performed. The dataset is composed of the descriptions of accident surveyors on the circumstances of a given accident. These descriptions average 400 hundred words and pretends to give a precise description of the events.

5.1 Objective

The main objective of this study is to compare the performances between two types of text features representation, namely: Bag of Words and word embeddings. This will be done by exploring a set of different configuration for both representations and then perform two classification tasks with each configuration to analyze and compare their performance.

The two tasks consist on classifying using the features extracted form the dataset the type of accident: *Aguas*, *Incendios*, *Extensivos* or *Eléctricos*. The other task is to try to predict whether the one writing the description of the events is a surveyor or technical personnel, therefore one being a multi-class classification and the latter a binary classification.

5.2 Proposed method

The proposed strategy is to make a custom preprocess for each type of configuration and identifying and taking into account the different factors that can influence in the quality of the features to be extracted. Therefore, the process is going to be the following:

1. Preprocess text: The first step consists on cleaning the raw text and normalizing it to be able to extract the features homogeneously

2. Feature generation: Once the text is clean, depending on which text representation we are going to work with the text features are generated

3. Training model: At this point, the texts have been turned into vectors of features, then a model is selected to be able to perform the classification. The target, size of the feature vector and sample size will affect to the configuration and training of the model

4. Results evaluation: Finally, when all models are trained each one of them is to be evaluated in order to compare the results. Appropiate metrics must have been chosen, as well as similar sample sizes with the same distribution.

5.2.1 Bag of Words

Based on previous works, such as the ones mentioned in section 4.1, and through visual inspection while navigating across the preprocess of the text various factors which could affect the quality of the text features representation appeared. The main ones studied in this document are the following: stop words removal, word stemming, indexing, vocabulary size and vocabulary reduction.

Stop words

Stop words are the most frequent words in a given languages often prepositions or pronouns. These words usually offer no relevant information to a sentence, especially when there is no *attention* mechanism or a contextual representation of a sentence. So when working with individual word representation the most logical thing to do is remove them. The removal of the stop words in a corpus makes its processing more efficient and favours a better representation by being able to focus on the relevant parts of the text. For these reasons, the *stop words removal* is a fixed variable and is not going to taken into account for the study of the quality of the representation, in other word, all configurations are going to have the stop words removed.

Stemming

Stemming is the process to reduce a word to its *stem* or *root* which consists on a shorter version of a word without the suffixes. A stemmer is an algorithm or an engine that performs the mentioned stemming. This allows to reduce significantly the size of a vocabulary by collapsing various entries of different conjugations or derivatives of a same word into a single entry based on the stem of these words.

An example that occurred during a visual inspection of the vocabulary generation with and without stemming was with the word *abandonar*. Without stemming there were 23 different derivatives of this verb, that once stemmed it collapsed all into a single entry, *aban*. It also helps with misspelled words, as the stemmer may *cut off* the typo from the original word. Intuitively, this process may be able to affect the quality of the text feature representation and the efficiency of the processing altogether, therfore it is a factor that has been taken into account. We will use the Snowball stemming algorithm built into Python's Natural Language Toolkit (NLTK) [29] to reduce each word to its stem.

Term Frequency - Inverse document Frequency

Due to the fact that the initial proposition of the BoW model is very simplistic, and upon manual experimenting wasn't able to produce a very good representation of the relevant aspects of the text, a form of indexing has been chosen. So in order to give a more informative representation to the feature vectors I have used TF-IDF. As the representation without any form of indexing was fairly poor, the process of indexing will not be studied, and as so, all texts will be represented using TF-IDF.

Vocabulary size

The vocabulary generated from the raw text is massive, 129161 entries without stemming the text and removing entries with two or less apparitions in the text; and 78363 entries with stemming also removing the entries with two or less apparitions in the text. It is expected that the vocabulary size will affect the amount of information that is represented and therefore, be relevant to the final text representation. Given this fact and that the original size of the vocabulary without making any kind of selection makes the processing and the training of the models with the generated features, the size of the vocabulary will be considered as a meaningful factor for the quality of the features and will be studied. The sizes that have been selected have been 2048 and 4096.

Vocabulary selection

In order to reduce the vocabulary size to the measures that have been mentioned in the previous section, a proper way of selecting the parts of the vocabulary that will be used must be devised. As the indexing method of choice is TF-IDF I decided to use the IDF score of each entry level to choose the most suitable entries. Intuitively, there are 4 ways to do this partition. All of them start by sorting the entries of the vocabulary by IDF value, thus the first words are the most common and the last words are the less commons. Given this sorted list, and being N the size to which we want to reduce the vocabulary, the four approaches are:

- Beginning: This approach would fetch the most common words in the corpus as it would yield the N first entries of the sorted by IDF vocabulary. This approach doesn't have much sense since all the distinct information would be lost and all feature vectors would look fairly similar.

- End: This approach would fetch the least common words in the corpus as it would yield the N last entries of the sorted by IDF vocabulary. This approach doesn't have much sense either since all the distinct information would be selected and all feature vectors would be really sparse. Upon a quick test, trying this method a lot of the feature vectors where empty (all zeros), thus being useless. This is due to that theoretically these words are the most informative, they also comprehend misspelled words and really rare occurrences, so it may up being not helpful.

- Beginning-End: This approach would fetch the $N/2$ most common words and the $N/2$ least common words in the corpus. I like this approach since it is meant to produce fairly dense vectors with some specificity sprinkled into them. Although several approaches are being mentioned, only this representation will be used since the other couldn't produce a proper feature vector.

- Middle: This approach would fetch the range between $[|V| - N/2, |V| + N/2]$ where $|V|$ is the size of the original vocabulary. The intuition to this approach is to discard the most common entries so the feature vectors are specific enough, and remove also the outliers that are found at the end of the vocabulary. Upon a test performed with these method, it also yielded a vast majority of empty feature vectors, thus being not a feasible way to select the vocabulary entries. I firmly believe that this is due to the reduced size of the vocabulary, but increasing its size enough for it to be viable would render almost impossible to preprocess and operate with the feature vectors in a reasonable time.

5.2.2 BERT

The BERT model is one of the most updated versions for the *Transformer* architecture, able to produce contextual embeddings. The BERT model that we have chosen to generate the embeddings of this dataset is the BERT multilingual base model (uncased) build and trained by *Hugging Face Co* [30]. The model is referred to as `bert-base-multilingual-uncased` is a pretrained model on the top 102 languages with the largest Wikipedia entries using a masked language modeling (MLM) objective [31].

5.2.3 fastText

FastText is an open-source, free, lightweight library developed by *Facebook AI Research* (FAIR) lab that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices [32].

In order to generate the embeddings for our dataset we use pre-trained word vectors for 157 languages, trained on *Common Crawl* and *Wikipedia* using fastText. These models were trained using CBOW with position-weights, in dimension 300, with character n-grams of length 5, a window of size 5 and 10 negatives [33].

As the text are mostly in Spanish, we use the model that is trained with a Spanish corpus. All models by default generate embeddings of 300 features, so they offer the possibility to reduce it to a smaller vector size, although we decide to keep the original 300 dimensions. Also, the model is able to generate an embedding for a single word or for a whole sentence. We decided to use the whole sentence as input, given that the BERT embeddings are generated this way also.

5.3 Data overview

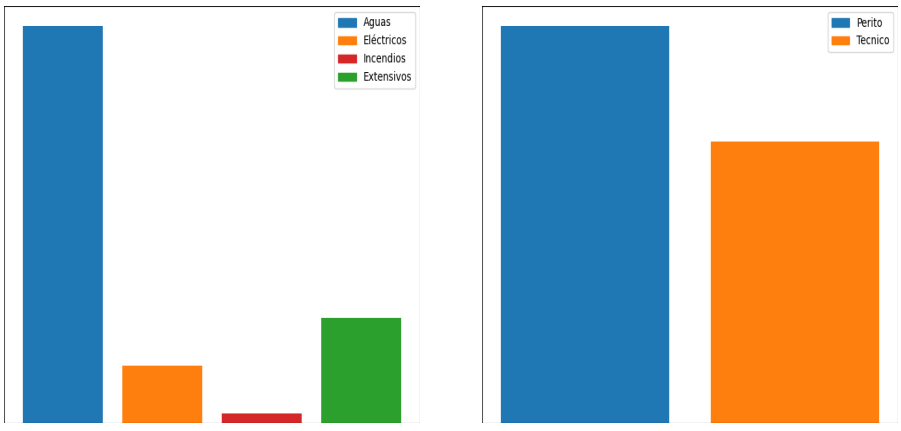
The data consists on three columns, one columns with all the text to be processed, *DesInfCirucnstancias*, and the two target columns *VarSiniestro* and *IndTienePerito*.

- *DesInfCirucnstancias*: Text column with the description of the circumstances of an accident. This text can be written by a surveyor or a member of the techical personnel. The length of these texts average 400 words and there are misspells and typos, so it has to be processed before using it as features. Most of the text is written in Spanish but there is a minority of the text that are written in Catalan.

- *VarSiniestro*: Is a categorical column with four possible values. Each one encompasses a different type of accident:

  - *Aguas*: This category groups accidents in which water is the responsible of the damages, except from the ones caused by rain and climate phenomenons. This includes broken pipes, malfunction of a dishwasher, and so on and so forth.
  - *Eléctricos*: This category groups accidents caused by electric surges or lightning.
  - *Incendios*: This category gathers accidents caused by fire. Examples of this would be a gas leak that catches fire, an unatended kitchen stove, ...
  - *Extensivos*: This category comprehends accidents caused by certain types of climate inclemencies such as blizzard or heavy winds.

- *IndTienePerito*: A binary column where the value is 1 if a surveyor has been involved in the inspection of the accident and therefore the text of *DesInfCircunstancias* is written by himself, and 0 if only members of the technical personnel have been to the scene where the incident has occurred, thus the text *DesInfCircunstancias* being written by the last person that witnessed the circumstances of the place.



(a) Distribution of accidents by *VarSiniestro*

(b) Distribution of accidents by *IndTienePerito*

Figure 15: Data distribution based on the two targets

5.4 Preprocess and feature generation

To be able to generate each representation a custom preprocess was performed for each one. In this section the methods and libraries used will be described with precision. Although there is a specific part of the preprocess for each configuration, there is also a common section.

5.4.1 Common part

The dataset is in the form of a *Coma Separated Values* (CSV) file in which the three columns of interest appear alongside with others that are not relevant to our purpose. To read this CSV file and load the data into *Python* we will use the *pandas* library [34]. With this library we are also able to sample a subset of 500000 instances (the original is well above 3M rows), with the same distribution of the original dataset. From now on we only work with this sample. Once the reduced dataset is loaded a generic preprocess is done along the corpus:

1. All the text is uncased in order to avoid repetition between cased and uncased instances of the same word.
2. A translation is done to remove all accents and special characters from the vowels. Also the *ç* character is converted into a *z*.
3. Then all words containing characters that are not [a-z] or a white space are removed. This is done with the *re* library [35]
4. And finally a translation from *ny* to *ñ* is also performed also with the *re* library.

5.4.2 Bag of Words

The preprocess for the BoW representation includes the common part mentioned in the above section and the factors described in the 5.2.1 section. So in each process of the step will be identifying the considered factors and highlighting the ones that finally will be studied:

1. A stop word removal is performed using the Spanish stop words set in the NLTK library. No factor as it is the same for every configuration.
2. Word tokenization is done to be able to generate the vocabulary and later calculate the TF-IDF. It is done with the `word_tokenize` function in the NLTK library. No factor is generated as every configuration is tokenized the same way.
3. At this point stemming is performed. So now the first factor materializes and divides the experiment between `stem` and `nostem` preprocessed text.
4. The vocabulary is generated and its IDF is calculated. The `stem` and `nostem` vocabulary are saved into a *Yet Another Markup Language* (YAML) file.
5. In this part, the size of the vocabulary is selected by sorting it by its IDF values. The second factor comes into dividing once again the experiment between `2048` and `4096`. Thus having four possible combinations at this point: `stem|2048`, `stem|4096`, `nostem|2048` and `nostem|4096`

6. Having the size of the vocabulary set, now the part of the vocabulary that will be used must be chosen. As mentioned in the 5.2.1 section, we found out that the only viable one was using the the $N/2$ most common words and the $N/2$ least common words in the corpus, being N the size of the vocabulary. We intended to analyze other selections within the vocabulary but unusable vectors were produced.

7. Now the feature vectors are generated by calculating the TF-IDF of each word of the vocabulary present in a given instance of the dataset in a *numpy* object [36]. Once all the dataset is processed, the empty feature vectors are removed and the rest of the processed dataset is saved in a *npy* file with the corresponding vectorized form of the target *VarSiniestro* and *IndTienePerito*.

5.4.3 BERT

For the BERT preprocess only the common part was used. Thus, only lower casing, strange characters removal, and translation of some of them was done. On the other hand, to generate the embeddings a series of steps had to be taken in order to be able to process the text:

1. Download tokenizer and model from the pretrained `bert-base-multilingual-uncased`

2. After the text was cleaned, it all was fed into the tokenizer with the padding option which padded the sentences that have less tokens than the longest sentence in the dataset.

3. In order to process the embeddings, an iterative apporach had to be taken since the computer could not compute them "at the same time". The vectors generated were of the shape $N \times T \times E$ where N is the number of instances fed to the BERT model, T the number of subtokens generated by the model for the instance N_i and E a constant size 768 which is the size of the embeddings generated by the model. The size of the collection of generated embeddings goes up very quickly, so generating and reducing these embeddings in batches yielded the best results.

   (a) The result of the tokenizer, a *python* dictionary, is received and a the first batch is processed by the model. Upon different tries, the batch size was decided on to 250, since it was the fastest without giving an allocation error for the RAM.

   (b) Then, the last hidden states of the output of the model are accessed. These are the embeddings of size $N \times T \times E$, where in this process $N = 250$ since its the batch size, T is the length of the longest subtoken list of a given instance in the 250 batch that is currently processing and $E = 768$.

   (c) Once the embeddings are generated, they are reduced using the *tensorflow* [37] function `mean_reduce()` which performs the mean of the embeddings across a given axis, in this case we wanted to collapse the T dimension into one, in order to have individual embeddings for each instance of the data set with fixed size 768.

   (d) Once reduced, the embeddings stored in *tensors* are concatenated with the previous batches, generating a stack of embeddings of size $N \times E$.

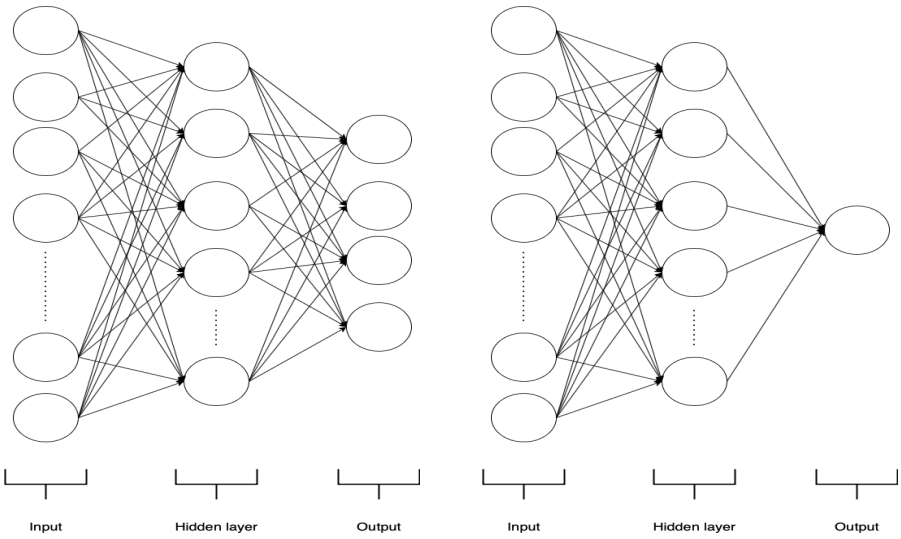
5.4.4 fastText

To be able to generate the embeddings with *fastText* two extra steps had to be made to the common preprocess. After the text was cleaned, extra white spaces had to be removed along with any $\backslash n$ character as it produced errors when computing the `get_sentence_vector()` function. In order to call to this function the model had to be loaded. Then, feature vectors of 300 features were generated and stored in a *numpy* file.

5.5 Models and training

To be able to have a fair comparison between the text features, the same architecture has been used for all representations. As there are two tasks, multi-class classification for *VarSiniestro* and binary classification for *IndTienePerito*, two main architectures were conceived.

5.5.1 Models



(a) Multi-class ANN (b) Binary class ANN

Figure 16: Two main architectures used
Source: Own

The specifications of these networks are:

- N neuron on the input layer, where N is the size of the feature vector; A 64 neuron fully connected hidden layer; A 4 neuron output layer in cas of the multi-class ANN and 1 neuron output layer in the case of the binary ANN
- All activation functions are *hyperbolic tangent* (tanh) except from the output layer which has a *softmax* function in the case of the multi-class ANN and a *sigmoid* function in the binary classification ANN.

5.5.2 Training configuration

In order to minimize the differences between trainings, an effort to limit the configuration disparity has been done. Regarding the dataset, all training have been done with the same data, and the same split between train and test. This has been ensured using the `scikit-learn` [38] function `train_test_split()`. The parameters of the function are eighty percent of the data is destined to the training set and the remaining twenty percent is used for the test set. As we won't be performing an exploration of hyperparameters thus not doing a fine-tuning of them, a validation set is not required. The same random seed (42) is used in order to ensure that the splits are all the same. Regarding the training configuration of the models we also tried to keep it the most homogeneous possible:

- Optimizer: The optimizer for all trainings is the *adam* [39] optimizer.
- Batch size: The batch size for all trainings is 64, except for the trainings with the BERT embeddings which is 16.
- Loss function: All trainings with the multi-class ANN have the *categorical crossentropy* loss function, whereas all trainings done with the binary classification ANN have the *binary crossentropy* loss function.
- Epochs: All models are trained with 10 epochs and the input is shuffled

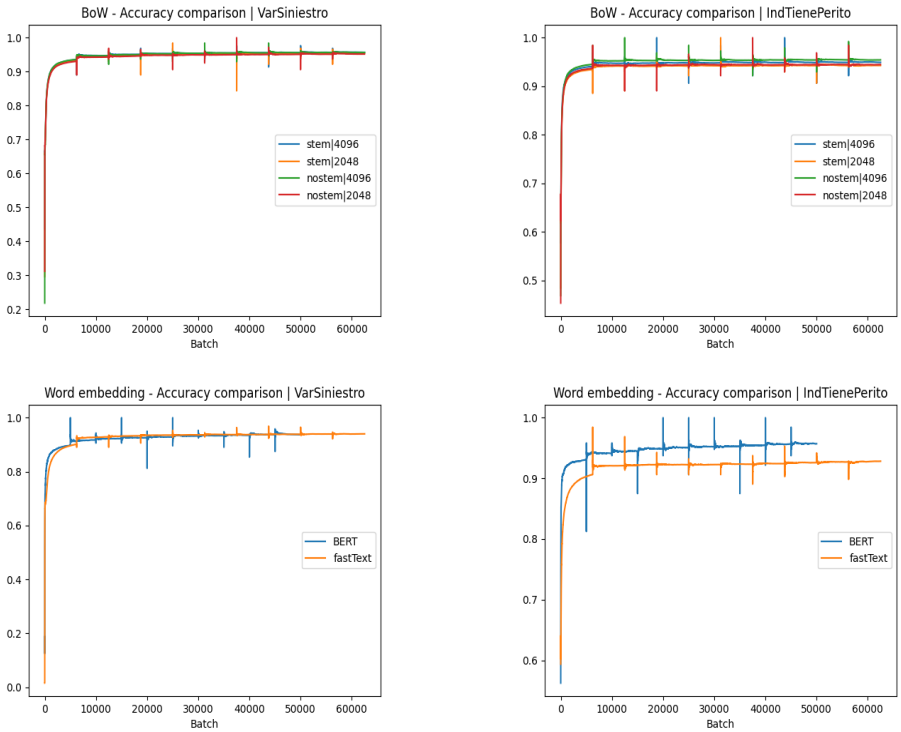


Figure 17: Training plots picturing the progress of accuracy in all models. In the bottom right we can observe that us the only one with a clear distinction between training results.

5.5.3 Bag of Words

In order to visualize the training progress of all models the *matplotlib* [40] library has been used.

Stem with 4096 features

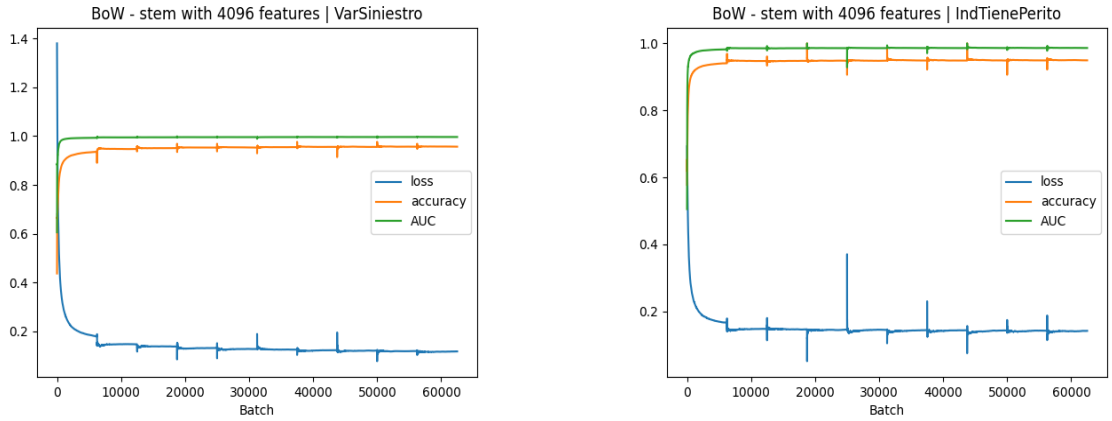


Figure 18: Training plots for the BoW representation using stemming and a vocabulary of 4096 words. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

Nostem with 4096 features

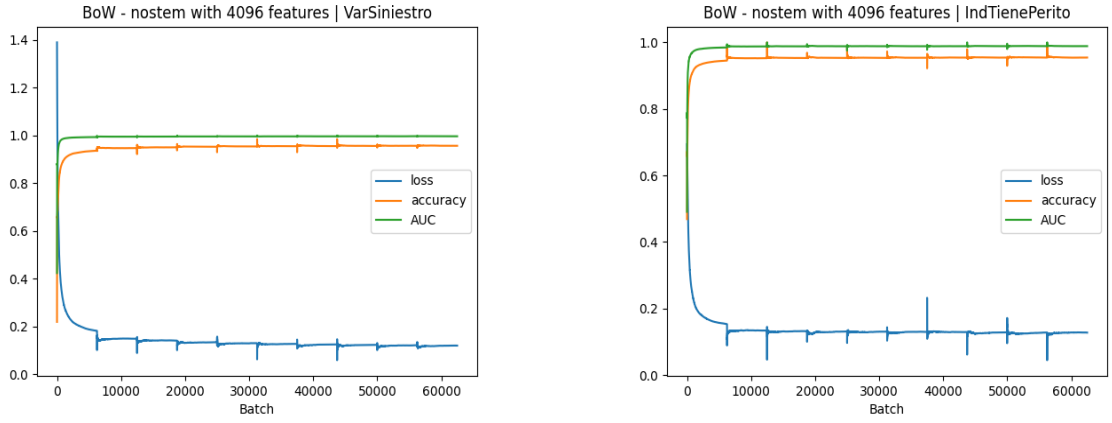


Figure 19: Training plots for the BoW representation without using stemming and a vocabulary of 4096 words. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

Stem with 2048 features

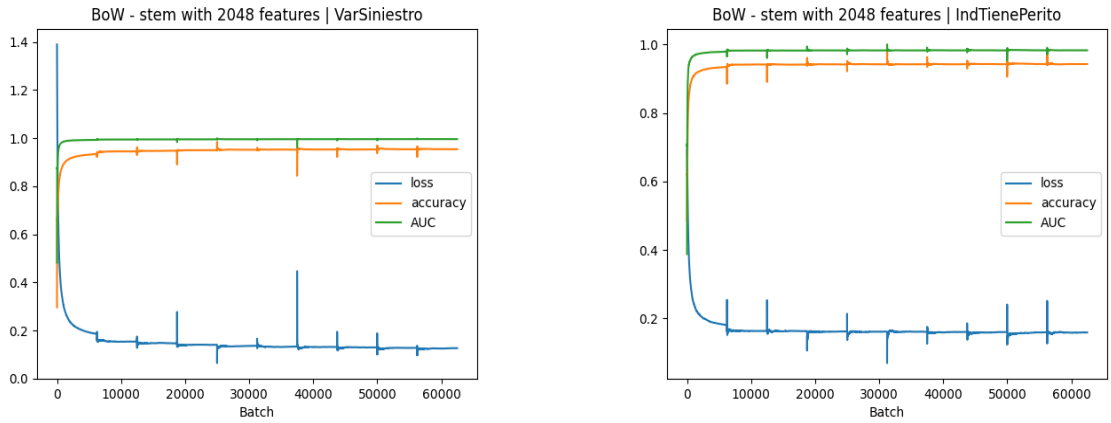


Figure 20: Training plots for the BoW representation using stemming and a vocabulary of 2048 words. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

Nostem with 2048 features

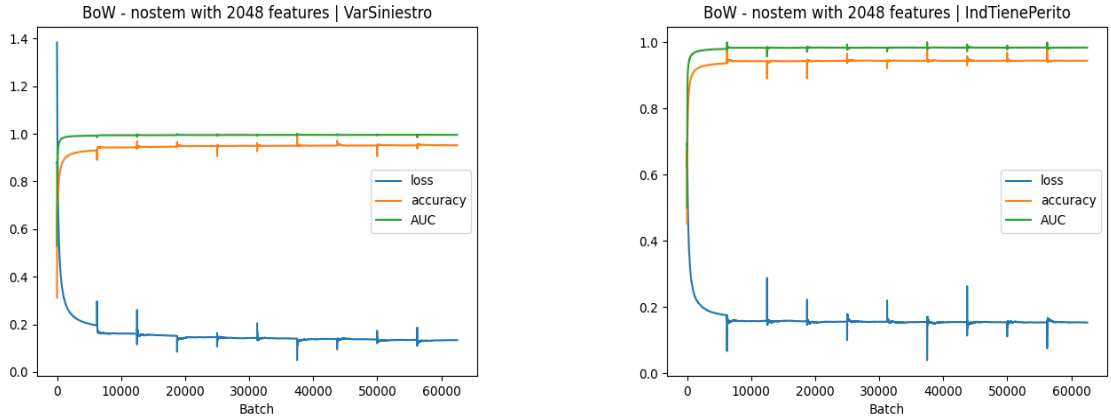


Figure 21: Training plots for the BoW representation without using stemming and a vocabulary of 2048 words. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

5.5.4 BERT

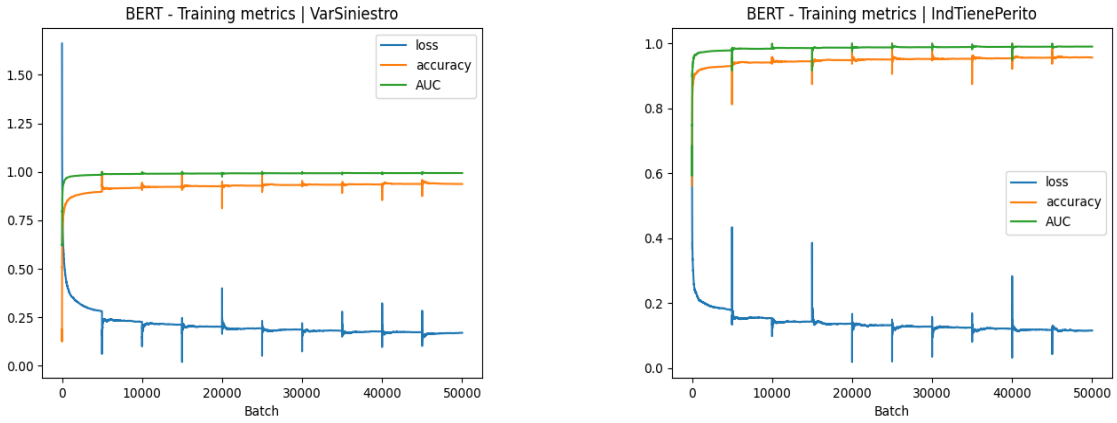


Figure 22: Training plots of the models using BERT embeddings. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

5.5.5 fastText

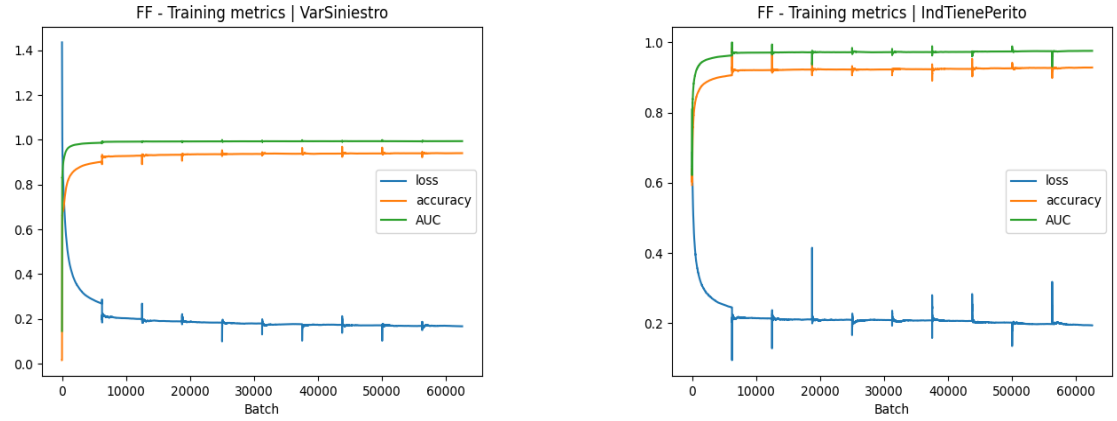


Figure 23: Training plots of the models using fastText embeddings. On the left the training progress on the multi-class classification can be seen, on the right it showcases the progress for the binary classification. In both models we can see an early convergence

5.6 Results

The results obtained follow our expectations in both tasks. The metrics that have been used is accuracy and Area Under the ROCurve (AUC). The first being a standard metric for classification and the latter another popular option but more sensitive to unbalanced datasets. In Tables 3 and 4 we can observe that although by not a large margin, in the classification for the type of accident the `stem|4096` performed the best out of all the others. This was to be expected due to the problem being prone to be really sensitive to specific words easily detected by a bag of words model.

	Stem		Nostem		BERT	fastText
Target	2048	4096	2048	4096	768	300
VarSiniestro	0.9513	**0.9547**	0.9491	0.9531	0.9354	0.9404
IndTienePerito	0.9470	0.9423	0.9434	0.9522	**0.9538**	0.9278

Table 3: Accuracy values for the test set

For the task of whether the text was written by a surveyor or a member of the technical personnel the best performer is the one using the BERT embeddings. This result was also expected since intuitively the problem would not be centered around key word, although it could have been, but more about *undersanding* the meaning of the text and generating a faithful representation.

	Stem		Nostem		BERT	fastText
Target	2048	4096	2048	4096	768	300
VarSiniestro	0.9957	**0.9960**	0.9953	0.9957	0.9931	0.9937
IndTienePerito	0.9828	0.9852	0.9835	0.9879	**0.9889**	0.9770

Table 4: AUC values for the test set

As seen in Table 4 the results match with the accuracy values in Table 3 so we can rest assured that any possible skew in the dataset representation is not affecting the evaluation of the performance of the models.

6 Conclusions and future work

Going over the objectives and intuitions for this exploration we can see that the results are very satisfactory. We have been able to explore different factors that can influence the quality of the feature vectors extracted from text. Each one of this factor has been either evaluated upon visual inspection or thoroughly compared within its possibilities and the results yield a positive answer on the expected outcome.

The principal points of this comparative study can be expressed in the following hypothesis:

- Hypothesis A: Bag of words model is better for the classification task for the target *VarSiniestro*.
- Hypothesis B: BERT contextual embeddings is the better representation for text in the classification task for the target *IndTienePerito*
- Hypothesis C: Stemming will aid to provide a more concise vocabulary and therefore a better representation of the features of the text.
- Hypothesis D: Vocabulary size affects the quality of the feature vector

With the results obtained, we can answer with a certain confidence that for this text, contextualized in the insurance domain, the following:

- Hypothesis A: Bag of words model it is a better model for the classification task for the target *VarSiniestro*.
- Hypothesis B: BERT contextual embeddings are the best representation for the text in the classification task for the target *IndTienePerito*
- Hypothesis C: Stemming has proven to be useful in the reduction of the vocabulary and removing irrelevant entries, providing a better feature vectors.
- Hypothesis D: Vocabulary size indeed affects the quality of the feature vector, at least in the bag of words model.

Future work could include:

- Lemmatization as the reduction method of a word into its root.
- A bigger vocabulary could also be explored using *Support Vector Machines* and using a different approach for culling the irrelevant entries of the vocabulary.
- Exploring other forms of word embedding as well as different ways to combine them to represent the sentence formed by it
- Using a custom trained BERT instead of using a pretrained one

7 References

[1] Tom Mitchell. *Machine Learning*. New York: McGraw Hill., 1997. ISBN: 0-07-042807-7.

[2] Liddy E.D. *Natural Language Processing*. Encyclopedia of Library and Information Science 2nd Ed. NY. Marcel Decker, Inc., 2001.

[3] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[4] Y. Zhang, R. Jin, and Z. Zhou. "Understanding bag-of-words model: a statistical framework". en. In: *International Journal of Machine Learning and Cybernetics* 1.1-4 (2010), pp. 43–52.

[5] *Glassdoor*. URL: http://www.glassdoor.es. (accessed: 11.10.2021).

[6] Microsoft. "Beyond carbon neutral". In: *white paper* (2018).

[7] *Your Europe. 2021. Data protection under GDPR.* URL: https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_en.htm. (accessed: 01.12.2021).

[8] John Hutchins. "From First Conception to First Demonstration: the Nascent Years of Machine Translation, 1947-1954. A Chronology". In: *Machine Translation* 12.3 (1997), pp. 195–252. ISSN: 09226567, 15730573. URL: http://www.jstor.org/stable/40027329.

[9] M. E. Maron. "Automatic Indexing: An Experimental Inquiry". In: *J. ACM* 8 (1961), pp. 404–417.

[10] Thorsten Joachims. "Text categorization with Support Vector Machines: Learning with many relevant features". In: *Machine Learning: ECML-98*. Ed. by Claire Nédellec and Céline Rouveirol. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142. ISBN: 978-3-540-69781-7.

[11] Jürgen Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *CoRR* abs/1404.7828 (2014). arXiv: 1404.7828. URL: http://arxiv.org/abs/1404.7828.

[12] Fengxi Song, Shuhai Liu, and Jingyu Yang. "A comparative study on text representation schemes in text categorization". In: *Pattern analysis and applications* 8.1 (2005), pp. 199–209.

[13] Aurelie Mascio et al. *Comparative Analysis of Text Classification Approaches in Electronic Health Records*. 2020. arXiv: 2005.06624 [cs.CL].

[14] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. "Machine Learning from Theory to Algorithms: An Overview". In: *Journal of Physics: Conference Series* 1142 (Nov. 2018), p. 012012. DOI: 10.1088/1742-6596/1142/1/012012. URL: https://doi.org/10.1088/1742-6596/1142/1/012012.

[15] Chikio Hayashi. "What is data science? Fundamental concepts and a heuristic example". In: *Data science, classification, and related methods*. Springer, 1998, pp. 40–51.

[16] J. Han, Y. Cai, and N. Cercone. "Data-driven discovery of quantitative rules in relational databases". In: *IEEE Transactions on Knowledge and Data Engineering* 5.1 (1993), pp. 29–40. DOI: 10.1109/69.204089.

[17] Charles C. Tappert. "Who Is the Father of Deep Learning?" In: *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. 2019, pp. 343–348. DOI: 10.1109/CSCI49370.2019.00067.

[18] A.G. Ivakhnenko et al. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967. ISBN: 9780444000200. URL: https://books.google.es/books?id=rGFgAAAAMAAJ.

[19] Rina Dechter. "Learning While Searching in Constraint-Satisfaction-Problems." In: Jan. 1986, pp. 178–185.

[20] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[22] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

[23] Haskell B. Curry. "The method of steepest descent for non-linear minimization problems". In: *Quarterly of Applied Mathematics* 2 (1944), pp. 258–261.

[24] Zellig S. Harris. "Distributional Structure". In: *WORD* 10.2-3 (1954), pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

[25] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[26] Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

[27] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[28] J Alammar. *The Illustrated Transformer [Blog post]*. 2018. URL: https://jalammar.github.io/illustrated-transformer/. (accessed: 07.01.2022).

[29] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[30] Hugging Face. *Hugging Face - The AI community building the future*. URL: https://huggingface.co. (accessed: 14.01.2022).

[31] Hugging Face. *Hugging Face - bert-base-multilingual-uncased*. URL: https://huggingface.co/bert-base-multilingual-uncased. (accessed: 14.01.2022).

[32] Facebook Inc. *fastText: Library for efficient text classification and representation learning*. URL: https://fasttext.cc. (accessed: 14.01.2022).

[33] Edouard Grave et al. “Learning Word Vectors for 157 Languages”. In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[34] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[35] Guido Van Rossum. *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.

[36] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[37] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[38] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[39] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. 2014. URL: http://arxiv.org/abs/1412.6980.

[40] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.