Universidad de La Laguna

Máster Universitario en Ciberseguridad e Inteligencia de Datos

# Trabajo de Fin de Máster

# Machine learning and NLP approaches in address matching

Lamine SYNE

La Laguna, 7 de septiembre del 2022

**D. Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

**D. Luz Marina Moreno de Antonio**, con N.I.F. 45.457.492-Q profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

**C E R T I F I C A (N)**

Que la presente memoria titulada:
"Machine learning and NLP approaches in address matching"

Ha sido realizada bajo su dirección por **D. Lamine SYNE**, con N.I.F. Y- 90 77 440 -K.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 septiembre del 2022

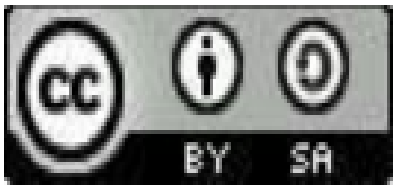
# Acknowledgments

I would like to thank my tutors **Isabel Sanchez Berriel** and **Luz Marina Moreno de Antonio** for their supervision and guidance throughout this project.

I would like to thank the director of the master of Cybersecurity and Data Intelligence master at La Laguna University, **Pino Caballero Gil** for her precious help during the year.

Finally, I would like to thank the Canary Government for giving me the chance to live this experience through the PBCA program.

# Licence



## Abstract

*The object of this project is to explore machine learning and NLP potential to the address matching sub-field of geographic information science. To achieve this a deep study about word and sentence embeddings models was made, how they work and how they can be used to generate numerical representations of an address.*

*For each word or sentence embedding model we generate vector representation of addresses in the database and calculate the cosine similarity between them in order to know which ones represent the same geographic position or not.*

*On the other hand we introduce the confusion matrix for evaluating performance of each model on a dataset of already matched addresses created from ISTAC [1] data sources and make a comparison study between the models.*

*Finally, a use case example will be shown by choosing the most performing model among those one studied above. This last one can be a debut for building a powerful tool for matching address pairs in all Canary Islands.*


**Key words** : machine learning, NLP, language model, address matching, word embedding, similarity

# List of Figures

# List of Tables

# Chapter 1 : Introduction

Address matching, the process of assigning physical location coordinates to addresses in databases, becomes a core function in various location-based businesses like take-out services, express delivery, customers merging, fraud identification, lead outreach etc. This makes the need to match two lists of addresses a common occurrence in many companies, organisations and government bodies.

In particular, the **ISTAC** [1] in their objective to facilitate the obtaining of spatial statistics, as well as the production of multi-source statistics through the **Integrated Data System of Canary Islands Statical Plan 2018-202**, is georeferencing information from different sources in the geostatistical reference of Canary Islands.
A database covering all the georeferenced municipalities of the Canary Islands is used, however the periodical update of the data, as well as the integration of new sources of information into the system requires to match addresses of each registry with the set of those that have already been georeferenced.

However, address matching is an extremely complicated task resulting in a number of challenges such as: address component types, noisy databases, inconsistent and replete with missing values on databases, input error made by users, text-based type … etc.

Despite the importance of matching addresses in the above-mentioned sectors we notice a great lack of open robust solutions. Recent innovations in machine learning, particularly in natural language processing (NLP), have been introduced in the wider area of address matching with significant potential.

In this project, we will focus on bringing solutions about ISTAC's [1] address matching problems exposed above, with machine learning and NLP techniques.

## 1.1 Backgrounds

During years , in the specific field of address matching, notable research has been achieved for resolving address records into “matched” and “not matched”. In particular there has been advanced research into quantitative methods for determining the extent of matching between pairs of text-based records, with numerous string-similarity measures developed, including Levenshtein and Jaro-Winkler.

By contrast a group of researchers developed the concept of ‘similarity join’, whereby two databases are tested by each combination of record pairs against a similarity measure function, with those pairs that exceed a preset threshold being recorded. They acknowledged that despite the availability of numerous similarity or ‘distance’ functions, no one measure excels in every application [2].

The ISTAC [1], in their georeferencing works[22], uses a technique based on Record Linkage that consists of comparing normalised addresses with other records that have already been geo-referenced. Currently, they use the R package “RecordLinkage” to evaluate similarities and assign a weight indicating the similarity between the compared records.

The research reported in [4], [2] shows that machine-learning techniques can be used to either enhance or replace the traditional rule-based solutions that are commonly applied to address matching .

In the first paper [4] it’s about two particular innovations into the address matching workflow: conditional random fields (CRFs) and word (address) embedding.
The second paper [2] introduces a framework called Post Match, the related work is a combination of the open source library “Libpostal” for address-parsing with a post-parse process and the Jaro-Winkler edit distance algorithm together with XGBoost machine learning classification.

In both cases there is an application of bi-class algorithms for several times a bi-class algorithm to decide whether one address matches or not to another. This, applied for each address with respect to the reference address pool, makes the problem very computationally expensive.

## 1.2 Objectives

The main objectives of this project are :

- ➢ Study of machine learning approaches into the field of address matching
- ➢ Study of natural language processing: text analysis and word embeddings
- ➢ Implement differents models for numerical representation of text-based data and introduce a metric to evaluate them
- ➢ Apply those models to ISTAC's address datasets and make a comparative study
- ➢ Build a tool able to resolve address pairs into match and non-match using text similarity

## 1.3 Scope

Due to the lack of power resources caused by the difficulty of processing address components and the time reserved for this work we have to set some limitations .
We will focus only on the addresses of Santa Brígida Municipality located in Gran Canaria where we have a register of 3600 unique addresses (SantaBrigidareferencia set) and another register of 16024 (SantaBrigida set) that have unique addresses with variations of them. For example, below we have a collection of addresses, of which number 1 belongs to SantaBrigidareferencia and 1,2,3,4 are in SantaBrigida with 2,3,4 as variations of 1 .

1. CAMINO ACEQUIA TAFIRA 6 SANTA BRIGIDA
2. CAMINO ACEQUIA TAFIRA MADROÑAL 4 SANTA BRIGIDA
3. CALLE CAMINO ACEQUIA TAFIRA 4 SANTA BRIGIDA
4. CALLE ACEQUIA TAFIRA 15 SANTA BRIGIDA

# Chapter 2 : State of the art

## 2.1 Address matching Challenges

One of the biggest unstructured data points is an address, this makes address matching a downstream challenge. Here are the most likely issues we will run into while trying to match addresses:

- **Input errors made by users** :

In many cases, addresses are input incorrectly by users, including misspellings, missing spaces, incorrect labels (“CALLE” vs “AVENIDA”), abbreviation formats (“C/” and “AV”), synonyms, and more. All of these make it difficult to have standardised data within a single database, let alone across multiple databases.

The following table show the most common input errors:

| **Input Error** | **Example** |
|---|---|
| Misspelling | 29 CALE NAVA |
| Miss Space | 29CALLE Nava |
| Incorrect label | 29 AVENIDA Nava |
| abbreviation | 29 C/ Nava |
| tokenization | Nava CALLE 29 |

**Tabla 2.1** : Address matching most common input errors

While these errors may seem easy to notice at a glance, it is very challenging to program a system to identify each difference. More than that, it requires significant computational power, and will take a lot of time to process. These errors can lead to significant errors when attempting to perform address matching, as the records will not match.

➢ **Problem to link two datasets together**

For all the reasons discussed above, sometimes we can face difficulties to relate two addresses and to connect datasets. When this occurs, we end up with the following disparities between our records :

| **Input address string** | **Reference data to match against (AddressBase)** |
|---|---|
| Unstructured text | Structured, tokenized |
| Messy, containing typos, etc. | Complete & correct (more or less) |
| Incomplete | Snapshot of addresses at a given time |
| Range from historic to very recent addresses, including businesses | Organisation/business names are not always part of the address. Changes due to the Historical Memory Law and other reasons |

**Tabla 2.2** : Input address vs reference data to match [5]

As we can see, the task of matching addresses becomes complicated when we have to compare records that are often formatted and input differently. Because of this, it makes the simple task of matching addresses much more complicated than predicted.
In real business location based these issues with linking datasets will cause major issues with your workflow, slowing up your business and causing errors in delivery, billing, and more [5]

➢ **Data preprocessing Failing**

One of the most common problems in address matching is data preprocessing. In many cases we fail to correctly preprocess our data. However this step is very important and having cleaning data before processing is essential for getting quality results.

➢ **Require of significant computational power processing algorithms**

The automatization of the address matching process through programs still requires a large amount of computational power and time to run. During the process various comparisons and calculations are made. When conventional techniques are used, based on similarity between the strings, each character needs to be compared, and they need to be processed one at a time. Data often needs to be preprocessed beforehand as well.

## 2.2 Introduction to natural language processing

Natural language processing (NLP) is a subfield of Artificial Intelligence that uses algorithms to interpret and manipulate human language. The goal is to make a computer able to understand human language processing and content (text, document ..) in the same way humans can.
NLP can be used in many fields such as speech recognition, knowledge representation, text classification … etc [6],[7]

### 2.2.1 Terminologies

**Corpus**

A corpus is a large, structured set of machine-readable texts produced in a natural communicative setting. If we have a bunch of sentences in our dataset, all the sentences will come into the corpus, and the corpus would be like a paragraph with a mixture of sentences. We just have to know that Corpus is a collection of documents. In our case of study the corresponding corpus is a set of addresses [7].

**Documents**

It is a unique text different from the corpus. If we have 100 sentences, each sentence is a document. Mathematical Representation of Documents is Vector [7]. In this project we will consider each address as a document.

**Vocabulary**

Vocabulary is the collection of unique words involved in the corpus. Let's take this following example:

sentence 1 = CALLE NAVA Y GRIMÓN
sentence 2 = CALLE EL HAMBRE
Vocabulary = { CALLE, NAVA ,Y, GRIMÓN, EL, HAMBRE }

**Words**

All the words in the corpus .Let's take the previous example
sentence 1 = CALLE NAVA Y GRIMÓN
sentence 2 = CALLE El HAMBRE
words = { CALLE, NAVA, Y, GRIMÓN, CALLE, EL, HAMBRE }

**N-gram**

In the field of computational linguistics, an n-gram is a continuous sequence of n items from a given sample of text or speech [8]. For this given address: CALLE NAVA Y GRIMON, we have:

1-gram set: CALLE, NAVA, Y, GRIMON
2-gram set: CALLE NAVA, NAVA Y, Y GRIMON

**Char N-gram**

Character n-grams are found in text documents by representing the document as a sequence of characters. These n-grams are then extracted from this sequence in order to extract features through a trained model [9]. For this given address: CALLE NAVA Y GRIMON, we have:

Char 1-gram set: C,A, L, L, E, N, A, V, A, Y, G, R, I, M, O, N
Char 2-gram set: CA, AL, LL, LE, EN, NA, …

### 2.2.2 Text preprocessing in NLP

Generally, in natural language preprocessing we have specific techniques for preprocessing and understanding texts. But this depends on the problem we are resolving or the type of text. For example, when we deal with sentiment analysis based on social media content it's important to analyse emoticons and emojis. It has usually been treated as a classification problem etc. Let's describe the key steps of processing text in NLP :

**Preprocessing**

- ➢ Removal of Noise, URLs, Hashtag and User-mentions
- ➢ Lowercasing
- ➢ Replacing Emoticons and Emojis
- ➢ Replacing elongated characters
- ➢ Correction of Spellings
- ➢ Removing the Punctuation
- ➢ …etc

**Stemming**

Stemming is the technique to replace and remove the suffixes and affixes to get the root, base or stem word. We may find similar words in the corpus but with different spellings like having, have, etc. All those are similar in meaning, so to make them into a base word, we use a concept called stemming, which converts words to their base word [7]

**Lemmatization**

Lemmatization is a technique similar to stemming. In stemming root words may or may not have the meaning, but in lemmatization, root word surely would have a meaning, it uses linguistic knowledge to transform words into their base forms [7].

**Parsing**

Parsing refers to the formal analysis of a sentence by a computer into its constituents, which results in a parse tree showing their syntactic relation to one another in visual form, which can be used for further processing and understanding [6].

### 2.2.3 Syntactic and semantic analysis

Syntactic analysis (syntax) and semantic analysis (semantic) are the two primary techniques that lead to the understanding of natural language. Language is a set of valid sentences but, what makes a sentence valid?: syntax and semantics.

**Syntax** is the grammatical structure of a text whereas **semantic** is the meaning being conveyed. A sentence that is syntactically correct, however, is not always semantically correct [6].

## 2.3 Word and sentence embedding techniques

After processing text data the next step is to extract features. To achieve this goal we have to use some techniques for representing text into vectors, so computers can understand the corpus easily. Those are word and sentence embedding techniques.

### 2.3.1 Word embeddings

In natural language processing (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 [10]. Word embeddings can be obtained using various methods, let's deep dive into those methods.

#### 2.3.1.1 One-Hot Encoding & Bag of words

One-Hot Encoding and Bag of Words form part of the most straightforward way to numerically represent words.

For **One-Hot Encoding** , the idea is to create a vector with the size of the total number of unique words in the corpus. Each unique word has a unique feature and will be represented by a 1 with 0s everywhere else. In the case of **Bag of words** representation (also called count vectorizing [11]), each word is represented by its count instead of 1 [12]. Let's look at an easy example to understand the concepts previously explained. We could be interested in analysing the tables 2.3 and 2.4:

| word | Calle | Nava | ….. | …… | …. | Word n |
|---|---|---|---|---|---|---|
| Calle | 1 | 0 | 0 | 0 | …. | 0 |
| Nava | 0 | 1 | 0 | 0 | …… | 0 |

**Tabla 2.3** : One-Hot Encoding illustration

| Address | Calle | Nava | y | Grimon | el | Hambre |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 |

**Tabla 2.4**: Bag of words illustration

#### 2.3.1.2 Term Frequency-Inverse Document Frequency : TF-IDF

TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document (TF), and the inverse document (IDF) of the word across a set of documents [9]. IDF has been used to penalise very commonly used words that do not provide semantic information, such as articles, prepositions, etc.

The TF-IDF value of a term ***t*** in a given document ***d*** from a set of documents **D** is :

$$TF - IDF(t, d, D) = TF(t, D) \times IDF(t, D)$$

Where $TF(t, D)$ is the term count within the document and $IDF(t, D) = log\left(\frac{D}{\{d \epsilon D: t \epsilon D\}}\right)$ ,$\{d \epsilon D: t \epsilon D\}$ is document count across the corpus and $D$ is corpus cardinal.

Let's calculate $TF - IDF\left("Calle" \{d_1, d_2\}, D\right)$ in the following :.

Address 1 (d1) : Calle Nava y Grimon
Address 2 (d2) : Calle el Hambre
Corpus D = [Address 1, Address 2 ]

| document | TF | IDF | TF-IDF |
|---|---|---|---|
| d1 | $\frac{1}{4}$ | $log\left(\frac{2}{1}\right)$ | $\frac{1}{4} \times log\left(\frac{2}{1}\right)$ |
| d2 | $\frac{1}{4}$ | $log\left(\frac{2}{1}\right)$ | $\frac{1}{3} \times log\left(\frac{2}{1}\right)$ |

**Tabla 2.5** : Calculation of $TF - IDF\left("Calle" \{d_1, d_2\},\ D\right)$

#### 2.3.1.3 Word2Vec

**Word2vec** is one of the most popular technique to learn word embeddings based on neural network .The neural network aim to predict the distribution of word contexts in the corpus $p(w|\ Context\ of\ w)$ and simultaneously learn the word representation. A single-layer neural network with a linear activation function is used, the contexts are represented by a succession of previous words of the window size $(n)$ chosen: $p\left(w_i|w_{i-n}, w_{i-n+1}\ ,\ \ldots,\ w_{i-1}\right)$

We have the representation of the words in a continuous multidimensional number space, words with similar contexts will be next to each other in the new space. It takes as input the text corpus and outputs a set of feature vectors that represent words in that corpus. It uses two neural network-based methods :

- ➢ Continuous Bag Of Words (CBOW)
- ➢ Skip-Gram

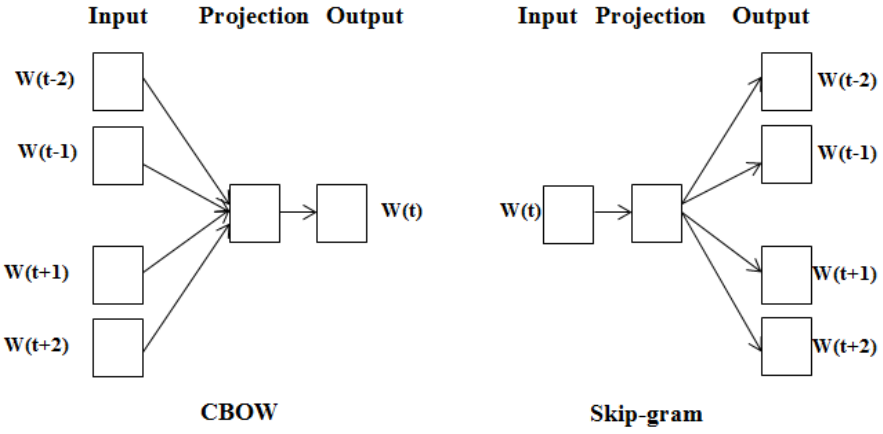



**Figure 2.1** : CBOW & Skip-Gram model [13]

**The CBOW Model** takes the context of each word as the input and tries to predict the word corresponding to the context. Here, context simply means the surrounding words.

**Skip-Gram** uses the target word, the word we want to generate the representation for, to predict the context. In the process of predicting the context words, the model learns the vector representation of the target word .

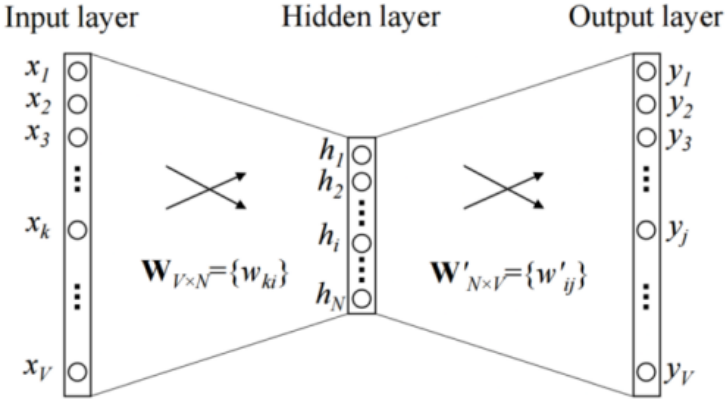



**Figure 2.2** : CBOW model with one word in the context [12]

Considering the following address : Address : “Plaza de la paz”

Let’s say we use the word ‘Plaza’ as the input to the neural network and we are trying to predict the word ‘paz’. We will use the one-hot encoding of the input word ‘plaza’, then measure and optimise for the output error of the target word ‘paz”.In this process of trying to predict the target word,this shallow network learns its vector representation. As the same way the model used a single word to predict the target, it can use multiple context to do the like the in the architecture in figure :

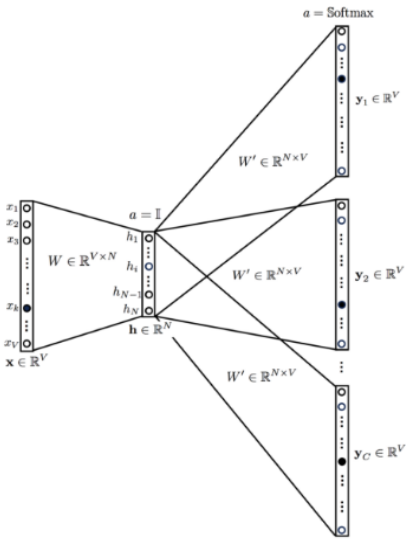



**Figure 2.3**: CBOW model with multiple words in the context [12]

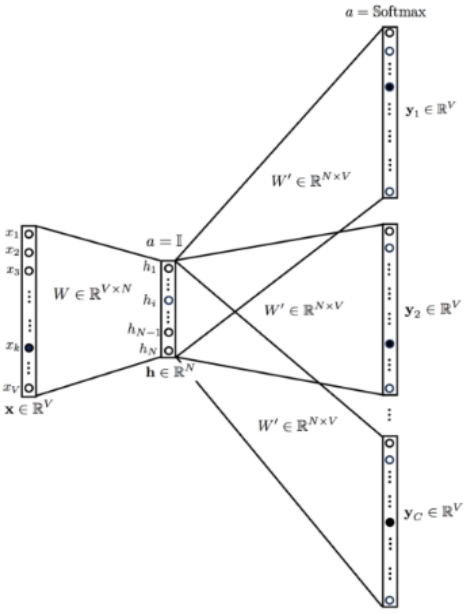



**Figure 2.4** : Skip-Gram model using target words [12]

The choice of using CBOW or Skip-Gram when training a **word2vec** model will depend on the case we intend to resolve. CBOW is better at learning syntactic relationships between words while skip-gram is better at understanding the semantic relationships. However Skip-gram works better when working with a small amount of data, focuses on semantic similarity of words, and represents rare words well. On the other hand, CBOW is faster, focuses more on the morphological similarity of words, and needs more data to achieve similar performance.

#### 2.3.1.4 FastText

FastText is a proposal model by **Facebook AI Research(FAIR)** for learning word embeddings and text classifications. This model allows creating unsupervised learning or supervised learning algorithms for obtaining vector representations for words. FastText supports both **CBOW** and **Skip-gram** models.

Fastext tries to include the morphological structure of words because this carries importance about the meaning and such structure is not taken into account by traditional word embeddings like word2vec, which train unique word embedding for every individual word. FastText attempts to solve this by treating each word as the aggregation of its subwords. For the sake of simplicity and language-independence, subwords are taken to be the character n-grams of the word. The vector for a word is simply taken to be the sum of all vectors of its component char-ngrams [14]. For example, the fastText representation of the word "CALLE" when using 3-grams corresponds to the collection of trigrams of the string <CALLE>: <CA, CAL, ALL, LLE, LE>.

The algorithm always starts the string of each word with "<" and ends them with ">". This representation helps to extract morphological information from the words such as suffixes and prefixes. With the generated n-grams a skip-gram model is trained to create the word representations [15].

### 2.3.2 Sentence embeddings

So far we have discussed how word embeddings represent the meaning of the words in a text document. But sometimes we need to go a step further and encode the meaning of the **whole sentence** to readily understand the context in which the words are used.
A straightforward approach for creating sentence embeddings is to use a word embedding model to encode all words of the given sentence and **take the average of all the word vectors**. While this provides a strong baseline, it falls short of capturing information related to word order and other aspects of overall sentence semantics.

#### 2.3.2.1 Doc2vec

Doc2vec is a model for creating numerical representation of a document, it extends the idea of word2vec and as this last one, doc2vec has two variants :

- **Distributed Memory model (IDM)**

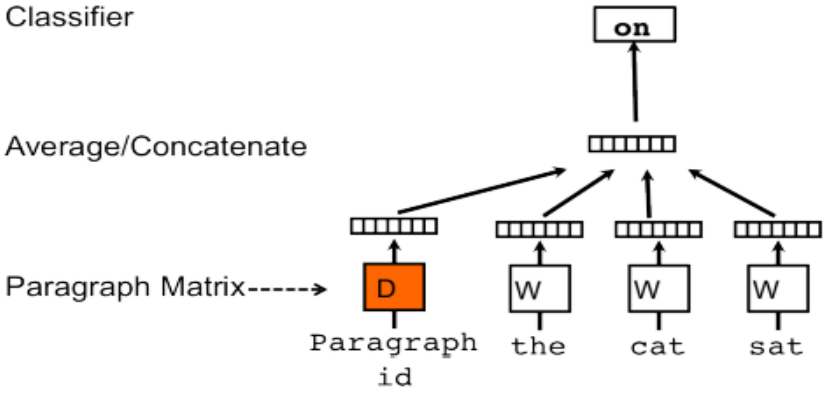



**Figure 2.5** : Doc2vec Distributed Memory model [16]

Each word and sentence of the training corpus are **one-hot encoded** and stored in matrices D and W, respectively. The training process involves passing a sliding window over the sentence, trying to predict the next word based on the previous words and the sentence vector (or Paragraph Matrix in the figure above). This prediction of the next word is done by concatenating the sentence and word vectors and passing the result into a softmax layer.

The sentence vectors change with sentences, while the word vectors remain the same. Both are updated during training.
The inference process also involves the same sliding window approach. The difference is that all the vectors of the models are fixed except the sentence vector. After all the predictions of the next word are computed for a sentence, the sentence embedding is the resultant sentence vector [12].

➢ **Distributed Bag of Words (DBOW) model**

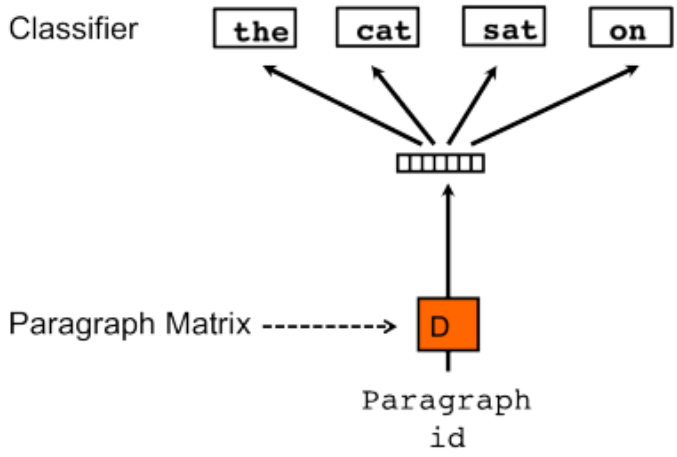



**Figure 2.6** : Doc2vec distributed bag of words model [16]

The DBOW model ignores the word order and has a simpler architecture. Each sentence in the training corpus is converted into a one-hot representation. During training, a random sentence is selected from the corpus, and from the sentence, a random number of words. The model tries to predict these words using only the sentence ID, and the sentence vector is updated (Paragraph ID and paragraph matrix in the figure). During inference, a new sentence ID is trained with random words from the sentence. The sentence vector is updated in each step, and the resulting sentence vector is the embedding for that sentence [12].

As a comparison between the two doc2vec models we can follow the direction of the authors in the original paper [16] who affirm that the DM model "is consistently better than" DBOW . However other studies [17] showed that the DBOW approach is better for more tasks. In other ways we have to know that the DM model takes into account the word order, the DBOW model doesn't. Also, the DBOW model doesn't use word vectors so the semantics of the words are not preserved.

### 2.3.2.2 BERT : Bidirectional Encoder Representations from Transformers

BERT is a transformers-based language representation model pre-training developed by Google. It's designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers [18].

BERT provides a way to pre-train models that consider contexts both to the right and left of words using the Masked LM (MLM) technique. In BERT, MLM instead of using pre- or post-word sequences, the entire sequence is used, from which a percentage of words to be predicted is removed. The algorithm works on pairs of sentences, once the words have been predicted, BERT uses the prediction of the next sentence. This part of the algorithm predicts whether the second sentence is the next sentence according to the original text.
The algorithm embeds metadata to indicate start and end of segments, separation between sentences, the masked words, etc. as can be seen in the example:

[CLS] the [MASK] has blue spots [SEP] it rolls [MASK] the parking lot [SEP] [19]

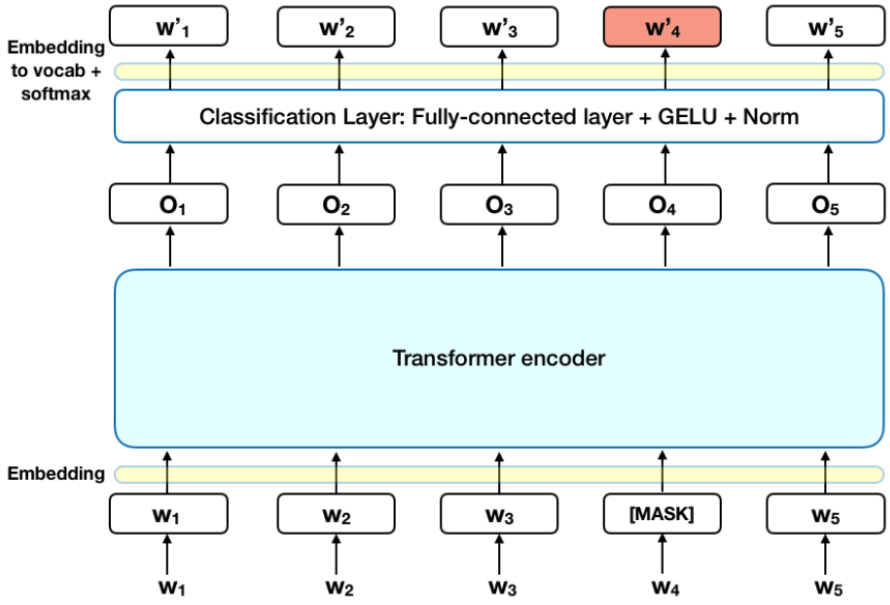



**Figure 2.7** : BERT mask LM [19]

Within the implementation of BERT we have two steps : **pre-training** and **fine-tuning** . During pre-training the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialised with the pre-trained parameters, and all of the parameters are fine-tuned using labelled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialised with the same pre-trained parameters [18].

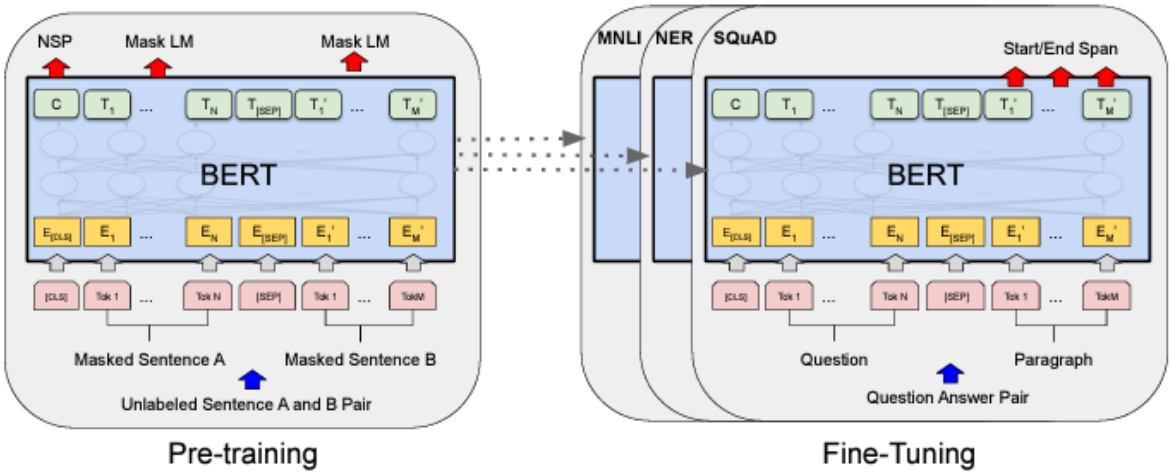



**Figure 2.8** : pre-training and fine-tuning procedures for BERT [18]

This is just the initial part of BERT implementation and whole steps are described in the original paper[18] but we have to keep in mind that BERT is one of the best general language models and produces good results on sentence embeddings.

## 2.4 Text Similarity and measures

Similarity is the distance between two vectors where the vector dimensions represent the features of two objects. In simple terms, similarity is the measure of how different or alike two data objects are. If the distance is small, the objects are said to have a high degree of similarity and vice versa. Generally, it is measured in the range 0 to 1. This score in the range of [0, 1] is called the similarity score [12].

As the same **text similarity** is how different or alike two texts or sentences are. However as humans it is very obvious to us that two sentences mean the same thing despite being written in completely different formats. But algorithms and to come to that same conclusion we have first to solve the problem of text representation by converting it into feature vectors using a suitable text embedding technique above. Once we have the text representation, we can compute the similarity score using one of the many distance/similarity measures [12]. Let's dive deeper into the text **similarity measures**.

**Jaccard Index**

Jaccard index, also known as Jaccard similarity coefficient, treats the data objects like sets. It is defined as the size of the intersection of two sets divided by the size of the union.

In the case in figure 2.9 :

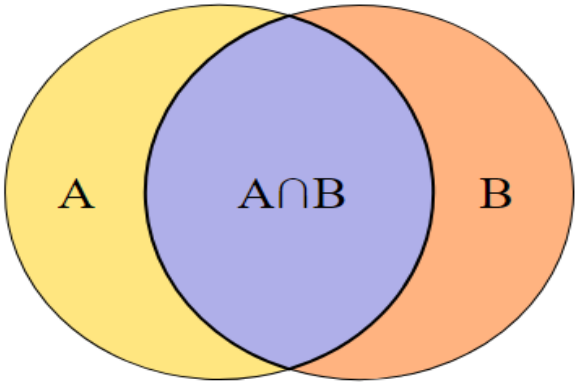



**Figure 2.9** : Jaccard distance on two sets [20]

The jaccard distance as : $J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$

**Euclidean Distance**

Euclidean distance, or L2 norm, uses the Pythagoras theorem to calculate the distance between two points as indicated in the figure 2.10. Generally speaking, when people talk about distance, they refer to Euclidean distance. It below :

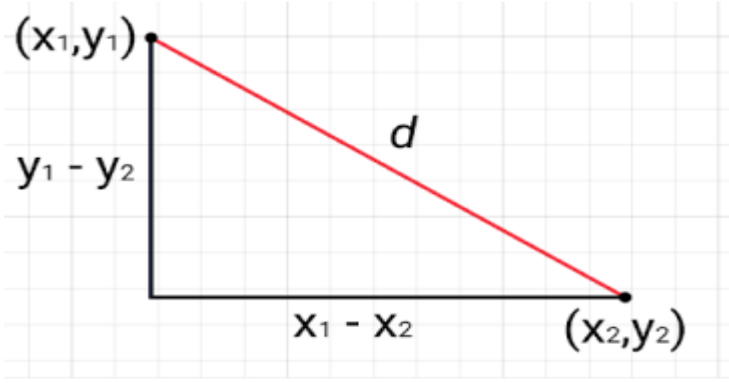



**Figure 2.10** : Euclidean distance representation [12]

The larger the distance **d** between two vectors, the lower the similarity score and vice versa .The distances can vary from 0 to infinity, we need to use some way to normalise them to the range of 0 to 1.

Although we have our typical normalisation formula that uses mean and standard deviation, it is sensitive to outliers. That means if there are a few extremely large distances, every other distance will become smaller as a consequence of the normalisation operation. So the best option here is to use something like the Euler's constant $\left(\frac{1}{e^d}\right)$ [12].

**Levenshtein distance**

The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other [21].

Mathematically, the Levenshtein distance between two strings a, b (of length |a| and |b| respectively) is given by the formula below:

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

**Figure 2.11** Levenshtein distance formula [21]

**Cosine Similarity**

Cosine Similarity computes the similarity of two vectors as the cosine of the angle between two vectors. It determines whether two vectors are pointing in roughly the same direction. So if the angle between the vectors is 0 degrees, then the cosine similarity is 1 [12].

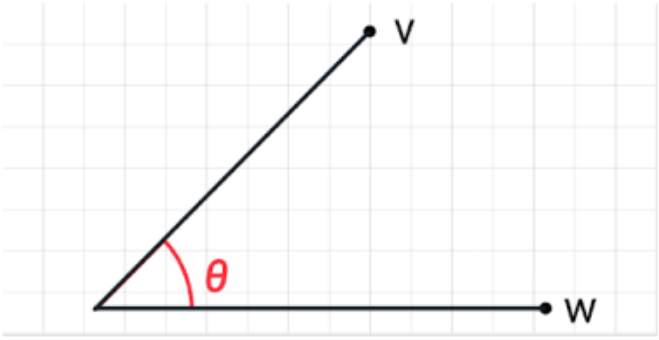



**Figure 2.12** : θ angle of vectors $(\overline{v}, \overline{w})$ [12]

It is given as : $cos(\overline{v}, \overline{w}) = \frac{\overline{v} \bullet \overline{w}}{||\overline{v}|| \times ||\overline{w}||}$. Where $||\overline{v}||$ represents the length of the vector $\overline{v}$, $||\overline{w}||$represents the length of the vector $\overline{w}$ and '•' denotes the dot product operator.

# Chapter 3 : Methodology

In order to achieve the objectives set for the realisation of this project it's necessary to find the right methodologies.
Several meetings were held with the tutor and the co-tutor. At first an explanation of the topic was made, secondly we defined the process and the necessary tasks to achieve for producing results. During the development of the project we frequently held meetings for checking tasks progression, raising doubts and verifying that the steps taken were the right ones. The main idea is that at each review the project should show some evolution with respect to the previous check, which is in line with the Scrum planning model.

## 3-1 Process definition

In the articles studied ([2], [4]), they apply machine learning techniques over two sub-tasks : address normalisation and address classification into matched and not matched. For each address to be georeference (or match) they generate a classification problem for each of the addresses that serve as a reference in this case.

During our investigations in order to find machine learning opportunities in the address matching field a lot of approaches were tested but the most promising one remains the use of word or sentence embedding coupled to text similarity measure.

Our proposal in this work consists of determining the similarity of each address with the reference addresses through the embeddings generated using the different algorithms exposed above. We consider an existing matching between two addresses when the similarity in the representation space exceeds a threshold.

Our approach is to measure the distances between addresses, but by using language models, complex relationships in words such as semantics and morphology are considered and not only similarities at the character level.

However, there are different techniques of word embedding, so our process will naturally be in the first time a study of each one, in a second time implement them using address dataset and finally make a practical comparison.
In other hands, we define a performance evaluation procedure similar to those applied in machine learning classification,we set a confusion matrix and evaluate the metrics accuracy, precision and recall .

The figures 3.1 shows the key steps of our work process

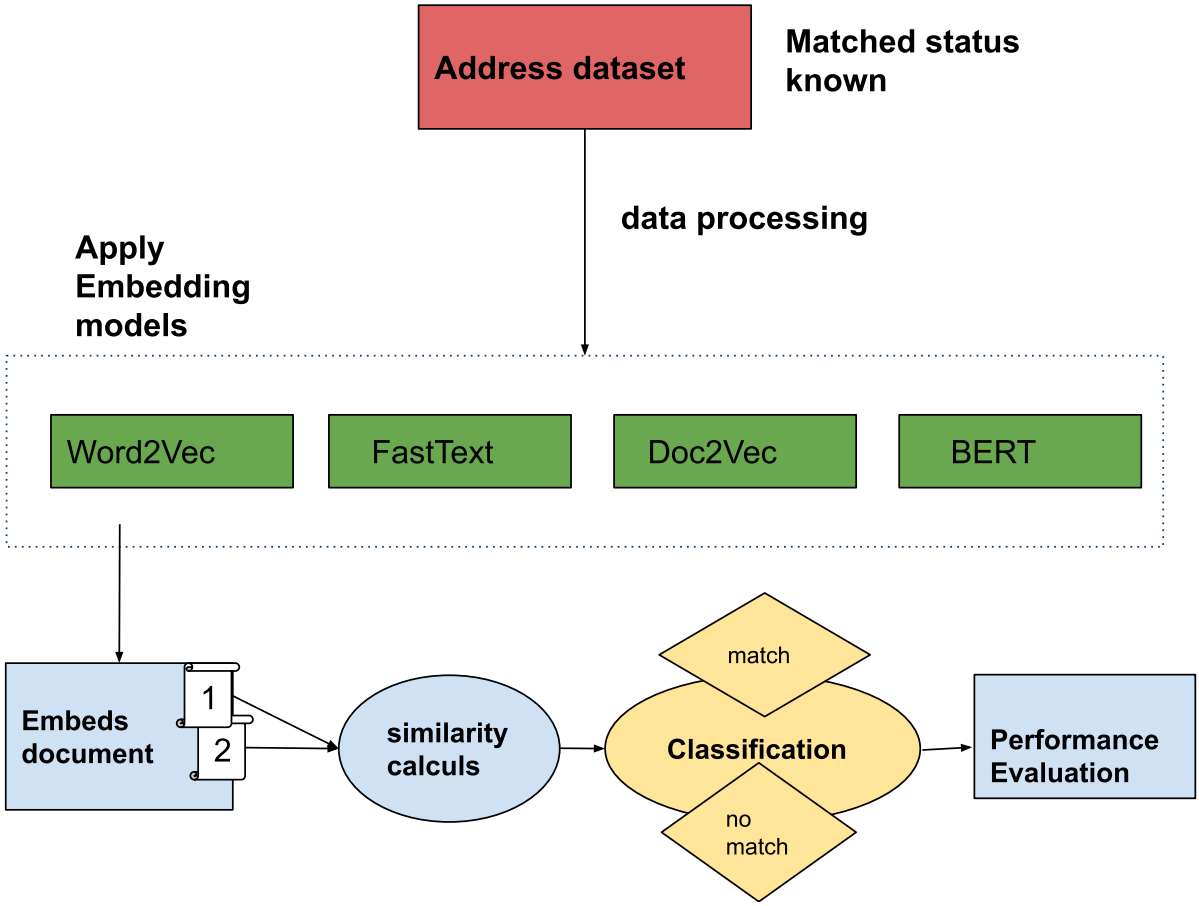



**Figure 3.1** : Project process

As shown in the figure we firstly generated address embeddings for each model Wor2vec, Fastext, Doc2vec and BERT, and secondly classify addresses into match or no match through the similarity. Finally we evaluate the performance of each model in the objective to make a comparison.

**Similarity**

After getting the vector representation(embedding) of each address we introduce the cosine similarity as a measure of the similarity between addresses.

**Classification**

For classifying addresses into matched or no matched we will compare the result of the similarity calculation to a fixed threshold value.

**Performance Evaluation**

Our initial dataset provides the status of matching for addresses by an identification number( uuid_idt) so a classical method would be the use of a machine learning classification algorithm and extract the performance. But a lack of variation on our dataset motivates us to do a manual evaluation calculating true positives, false positives, true negative and false negative from classification results and known status of addresses

## 3.2 Implementation planning

In order to correctly implement the defined process for this project a planning was made the table 3.1 gives the details of needed tasks to implement en-to-end the drescripted process .

| Task | Start date | End date |
|---|---|---|
| Data set creation | 10/06/2022 | 14/06/2022 |
| Implement wor2vec models | 15/06/2022 | 17/06/2022 |
| Performance and operation analysis | 18/06/2022 | 20/06/2022 |
| Improvements | 21/06/2022 | 26/06/2022 |
| Implement fastext models | 27/06/2022 | 30/06/20022 |
| Performance and operation analysis | 1/07/2022 | 3/07/2022 |
| Implement doc2vec model | 04/07/2022 | 9/07/2022 |
| Performance and operation analysis | 10/07/2022 | 15/07/2022 |
| Improvements | 16/07/2022 | 20/07/2022 |
| Implement BERT model | 21/07/2022 | 31/07/2022 |
| Performance and operation analysis | 1/08/2022 | 8/08/2022 |
| Improvements | 9/08/2022 | 14/08/2022 |
| Comparison study | 15/08/2022 | 28/08/2022 |

**Table 3.1** : Implementation planning

# Chapter 4 : Development

## 4.1 Dataset Creation

The ISTAC [1] provides a csv file with normalised and georeferenced addresses that currently appear in their Integrated Data System.
The variables included in the file correspond to the different elements that make up an address, as well as an identification code(uuid_idt) shared by all the addresses for which the matching has been positive according to their algorithms. Among the variables we find territory codes("códigos de territorio"), the normalised and unnormalised type of road("tipo de via"), the normalised and unnormalised road name("nombre de via"), road code("código de la vía"), normalised and unnormalised portal number.
Also we can find information about the technique used to generate the matching and a categorical variable with values: AVERAGE, HIGH or VERY_HIGH, which indicates the quality of the link .
This dataset is from all the municipalities in the Canary Islands but in our case we will extract a part from one municipality called Santa Brígida.We will train our models using this dataset of 16024 addresses in order to build word embeddings.

```
santaBrigida = pd.read_csv('/content/sample_data/direcciones_SantaBrigida.csv', sep=",")
print(santaBrigida.shape)
```

```
(16024, 24)
```

**Figure 4.1** : Dataset registers

From this dataset we select relevant columns that we will need in the rest of the work

```
santaBrigida = santaBrigida[['uuid_idt','tvia','nvia','numer','codmun','nommun','direccion']]
santaBrigida.head(3)
```

| | uuid_idt | tvia | nvia | numer | codmun | nommun | direccion |
|---|---|---|---|---|---|---|---|
| 0 | 6072464c-3251-11e8-b2a5-480fcf5217b3 | NaN | SILOS 58 | 0 | 35021 | Santa Brígida | SILOS 58 SANTA BRIGIDA |
| 1 | 69f2a3ab-3251-11e8-bd93-480fcf5217b3 | CALLE | TEJAR | 10 | 35021 | Santa Brígida | CALLE TEJAR 10 SANTA BRIGIDA |
| 2 | 6fd61eba-3251-11e8-b6de-480fcf5217b3 | CALLE | RASO | 23 | 35021 | Santa Brígida | CALLE RASO 23 SANTA BRIGIDA |

**Figure 4.2** : selecting columns forming an address at Santabrigida

| Column | description |
|---|---|
| uuid_idt | Identificador compartido entre las direcciones que causan match |
| tvia | Tipo de vía |
| nvia | Nombre de vía normalizado |
| numer | Número de portal normalizado |
| codnum | Código de municipio |
| nommun | Nombre de municipio |
| direccion | Unión de los campos: tvia+nvia+numer+nommun, en caso de disponer de todos ellos |

**Table 4.1** : Columns descriptions (spanish) [22]

**Create dataset** : once we have a good sample of our dataset we can export it in csv format for evaluating performance of our models

```
name='muestra'
select.to_csv('/content/sample_data/'+name+'.csv')
```

**Figure 4.3** : Dataset creation

Finally, we have the dataset “Santabrigida” (16024 addresses) that we will use to train our word and sentence embedding models.
To evaluate the performance of the models and compare them, we will use the dataset “muestra” which is a fraction of this dataset from “Santabrigida”.
We make this reduction of the data because of a lack of resources and as exposed in chapter 3 the performance evaluation is very costly .

## 4.2 Libraries: Gensim, NLTK and Sentence Transformers

In addition to the basic libraries for data analysis we used some special libraries during the project with specifics roles for each one :

**Gensim**

Gensim is an open source python library for topic modelling able to train large-scale semantic NLP models , represent text as vectors and find related documents.
Gensim runs on Linux, Windows and Mac OS X, and should run on any other platform that supports Python 3.6+ and NumPy[23].
We can install it by running this command : ***pip install gensim***

In this project we use gensim to train wor2vec , FastText and doc2vec models for generating vector representation of addresses and calculate the similarity.

The figure below shows a basic syntax of importing and training gensim models

```
from gensim.models import Word2Vec,fasttext,doc2vec
addresses=["CALLE NAVA Y GRIMON","PLAZA DEL ADELANTO"]
model= Word2Vec(sentences=addresses, min_count=1,size= 100,workers=3, window =3)
```

**Figure 4.4** : Gensim training models example

**NLTK : Natural language processing Tool-kit**

NLTK is a leading platform for building python programs to work with human language data. It provides a suite of text processing utilities for classification, tokenization, stemming, tagging, parsing .. etc.
In this project we use NLTK to preprocess our address dataset and in particular to tokenize data before passing it to models.

The figure 4.5 shows a basic syntax of tokenisation addresses with NLTK

```
import nltk
address = """PLAZA DEL ADELANTO """
tokens = nltk.word_tokenize(address)
tokens
```

```
['PLAZA', 'DEL', 'ADELANTO']
```

**Figure 4.5** : NLTK tokenization example usage

**Sentence Transformers**

SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings. It can be used to compute sentence / text embeddings for more than 100 languages. These embeddings can then be compared e.g. with cosine-similarity to find sentences with a similar meaning. This can be useful for semantic textual similar, semantic research or paraphase mining [24].

Sentence Transformers can be installed by running this command:
***pip install -U sentence-transformers***
It recommended to have python 3.6 or higher, and at least Pytorch 1.3.6 remain that sentence-transformers are based on Pytorch and transformers.

In this project we use sentence transformers for implementing the BERT model. The figure shows an example of sentence-transformers implementing a BERT model

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('small_bert/bert_en_uncased_L-4_H-512_A-8')

#Our sentences we like to encode
sentences = ['CALLE NAVA Y GRIMON',
                  'PLAZA DEL ADELENTADO']

#Sentences are encoded by calling model.encode()
embeddings = model.encode(sentences)
```

**Figure 4.6** : Example Usage Sentence-Transformers

## 4.2 Implementation

### 4.2.1 Data description and preprocessing

From the data creation process above we generate this present data on which one we will build our word and sentence embedding models and evaluate performance.

| uuid_idt | tvia | nvia | numer | codmun | nommun | direccion | r |
|---|---|---|---|---|---|---|---|
| 74ed2308-3251-11e8-a29c-480fcf5217b3 | CAMINO | TEJAR | 28 | 35021 | Santa Brígida | CAMINO TEJAR 28 SANTA BRIGIDA | |
| 28ef08aa-bdfc-4ac4-9a21-372fad2c7621 | PASEO | GUINIGUADA | 44 | 35021 | Santa Brígida | PASEO GUINIGUADA 44 SANTA BRIGIDA | |
| 6a9815e7-3251-11e8-ba6c-480fcf5217b3 | CALLE | ALFEREZ VICENTE MONZON BARBER | 10 | 35021 | Santa Brígida | CALLE ALFEREZ VICENTE MONZON BARBER 10 SANTA B... | |
| 67ab5779-3251-11e8-bb6f-480fcf5217b3 | CALLE | SAN JOSE VEGA | 86 | 35021 | Santa Brígida | CALLE SAN JOSE VEGA 86 SANTA BRIGIDA | |
| 6739cf67-3251-11e8-9c16-480fcf5217b3 | CALLE | BUENAVISTA | 61 | 35021 | Santa Brígida | CALLE BUENAVISTA 61 SANTA BRIGIDA | |
| ... | ... | ... | ... | ... | ... | ... | |

**Figure 4.7** : Created dataset

Before starting work with the data let's check missing values and make various descriptions through graphics and statistics.

```
print("Distribution of missing values for each variable: ")
print(muestra.isnull().sum())
print(".....................")
print("Registers: ",muestra.shape)
```

```
Distribution of missing values for each variable:
index        0
uuid_idt     0
tvia         0
nvia         0
numer        0
codmun       0
nommun       0
direccion    0
DIRECC       0
dtype: int64
.....................
Registers:  (145, 9)
```

**Figure 4.8** : missing values and registers numbers

As we can see the dataset doesn't present missing values, so we don't need to apply a technique to fill out missing values. Our next step is to take a look at the variable nvia : "nombre de via ".

The figure 4.9 shows the count word of nvia in each address row.

| | nvia | nvia_count |
|---|---|---|
| 0 | TEJAR | 1 |
| 1 | GUINIGUADA | 1 |
| 2 | ALFEREZ VICENTE MONZON BARBER | 4 |
| 3 | SAN JOSE VEGA | 3 |
| 4 | BUENAVISTA | 1 |

**Figure 4.9** : nvia word count

Once we have for each nvia the number of words we can represent the distribution graphic of words

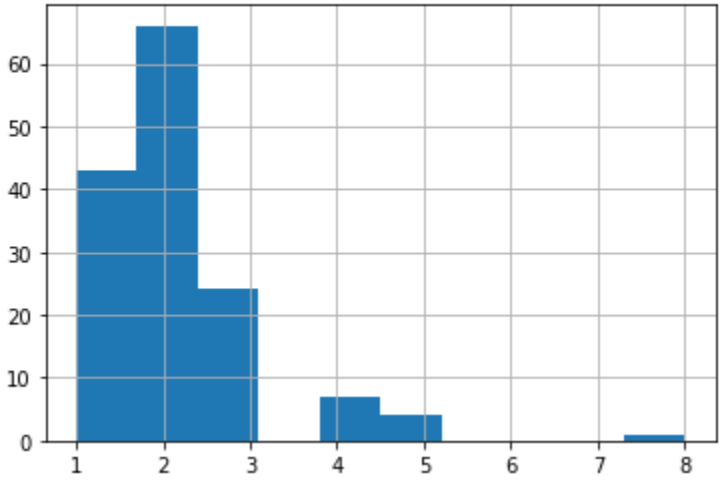



**Figure 4.10** : distribution word count of nvia variable

In our last part of describting data we will show the most used words of the variables nvia and tvia on addresses. The words that appear the most are the higher dimensions the most and vice versa.


**Figure 4.11** : Most commonly used word on addresses (nvia)


**Figure 4.12** : Most commonly used word on addresses (tvia)

Once we understand the data the next step is preprocessing, firstly we format addresses. In principle an address is an concentenation of the variables tvia, nvia, nume, codmun and nommun but we have to remember that we are working with data from one municipality (Santa Brígida). It means that all our addresses have the same value on the variable codmun(35021) and nommun (Santa Brígida). That's why it will be necessary to remove them from the address in order to get the root of an address.
So, we are going to create a address column without the two variables cited above

```
muestra = pd.read_csv('/content/sample_data/muestra.csv', sep=",")
muestra.drop('Unnamed: 0', inplace=True, axis=1)
muestra['DIRECC']= muestra['tvia']+" " + muestra['nvia']+" " + muestra['numer'].map(str)
muestra.head(3)
```

| | index | uuid_idt | tvia | nvia | numer | codmun | nommun | direccion | DIRECC |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 10893 | 74ed2308-3251-11e8-a29c-480fcf5217b3 | CAMINO | TEJAR | 28 | 35021 | Santa Brígida | CAMINO TEJAR 28 SANTA BRIGIDA | CAMINO TEJAR 28 |
| 1 | 134 | 28ef08aa-bdfc-4ac4-9a21-372fad2c7621 | PASEO | GUINIGUADA | 44 | 35021 | Santa Brígida | PASEO GUINIGUADA 44 SANTA BRIGIDA | PASEO GUINIGUADA 44 |

**Figure 4.13** : Creation of new address column

Now we have address data almost ready to be trained but for some models like wor2vec and fasttext it's preferable to pass them the data in a certain format that's why we will tokenize the data before the training phase.

```
#Before training the model  we have to put adresses dataset  rightformat format
direcciones=datos['direccion'].to_list()
data = []
# iterate through each address
for i in direcciones:
    temp = []
    # tokenize the address into words
    for j in word_tokenize(i):
        temp=[t for t in temp if len(t) > 1]
        temp.append(j)

    data.append(temp)
data
```

**Figure 4.14** : Tokenization addresses

### 4.2.2 Modelling

In NLP instead of always training your own model it is recommended in some cases to use pre-trained models.The advantage of these models is that they have been trained in a larger corpus of words so they gain in maturity. We can find these models in differents public repositories or research publications.In this project, in addition to our own trained we a word2vec , fastext and BERT pre-trained model for the Spanish language

As already mentioned, in this project we work with wor2vec, fastText, doc2vec and BERT model. For word2vec and fastText, first we train our own model and second we load pre-trained models. In the case of doc2vec we also train our own model but for BERT we load a spanish BERT model.

➢ word2vec

**Train model**

A word2vec model uses a set of parameters that affect both the training speed and the quality. During our training phase we adjust the parameters several times in order to have a good model. The figure below shows the way to train the model.

```
#Training  model
w2v_model1=  Word2Vec(data, min_count=1,size= 100,workers=1)
```

**Figure 4.15** : Training wor2vec  model

It's important to notice that word2vec sets some parameters by  default,  in our case  we use the CBOW variant which  is the default variant implemented by wor2vec. The model receives the dataset in the right format as the first parameter here this last one is **data** which is the result of our tokenized addresses. The parameters **min_count** is for ignoring the word that does not appear a certain number of times in the corpus. By default the value is 5  but this can pose a problem in our case that's why  we put the minimum value 1. The **size** determines  the number of dimensions (N) of the N-dimensional space that gensim Word2Vec maps the words onto; we chose a 100-dimensional space. The **workers** parameters determine the number of cores to use for the training. It takes effect when we set it to 1. If we put another value we have to install some tool like Cytron.

**Load pretrained  Model**

To use a pre trained  word2vec model we just need to download the corresponding  model In most cases they are in vector or bin format and we can find a lot of pre-trained model from the communities or IT companies  like Facebook , Google …etc
In this project we use a pre-trained word2vec for Spanish Language [25], the figure below the code to execute for loading this model.

```
#Loadingg pre-trained model
w2v_model2 = Word2Vec.load("/content/drive/MyDrive/models/complete_model/complete.model")
```

**Figure 4.16** : Loading word2vec  pre-trained model

➢ FastText

**Train Model**

Like word2vec , fastText model uses a set of parameters that affect both the training speed and the quality**.** We train the fastText model in the same way we did with wor2vec
The model receives the dataset in the right format as the first parameter here this last one is **data** which is the result of our tokenized addresses. The parameters **min_count** is for ignoring the word that does not appear a certain number of times in the corpus.
The **size** determines the number of dimensions (N) of the N-dimensional space that gensim Word2Vec maps the words onto; we chose a 100-dimensional space. The **workers** parameters determine the number of cores to use for the training.

```
#Training Model
fast_Text_model1 = FastText(data,size=100, min_count=1  workers = 1)
```

**Figure 4.17** : Training word2vec model

**Load Model**

For the pre-trained fasText model [26], depending on the format that we have downloaded the model (vec or bin) there is a way to load it. In our case we download a pre-trained model for Spanish language in vector format because the bin format needs complex transformations.

```
fastText_model_2 = KeyedVectors.load_word2vec_format("/content/drive/MyDrive/models/fastText/cc.es.300.vec")
```

**Figure 4.18** : Loading FastText pre-trained model

➢ doc2vec

The process of training a doc2vec model is similar to word2vec but here we have some additional steps. Below we have the step to train the model

**Model initialization**

```
#initialise the model
doc2vec_model = gensim.models.doc2vec.Doc2Vec(vector_size=100, min_count=1, epochs=30)
```

**Figure 4.19 :** Initialise doc2vec model

The parameters **vector_size** and **min_count** represent the same as on wor2vec and fastText but here we use additional parameters **epochs** for setting the number of iteration over the corpus into 10.

**Tagged documents**

```
#tag Document
def tagged_document(list_of_list_of_words):
   for i, list_of_words in enumerate(list_of_list_of_words):
      yield gensim.models.doc2vec.TaggedDocument(list_of_words, [i])
training_data = list(tagged_document(addr))
```

**Figure 4.20** : tagged document

Different to wor2vec where the model directly receives addresses(documents) in a format of list of addresses ( addr variable in figure 4.20 ). A tag has been added to each address (document) before passing to the model for building vocabulary and training . The vocabulary is just a list of all of the unique words extracted from the training corpus.

**Training model**

```
#building vocabulary
doc2vec_model.build_vocab(training_data)
#Train the model
doc2vec_model.train(training_data, total_examples=doc2vec_model.corpus_count, epochs=doc2vec_model.epochs)
```

**Figure 4.21** : Training doc2vec model

➢ BERT

In this project we will use a BERT model for Spanish language [27]. Below we have the steps to follow :

**Install and import Transformers**

```
#Installing Transformers
pip install -U sentence-transformers

#Loading Tranformers
from sentence_transformers import SentenceTransformer
```

**Figure 4.22** : Installing and loading Transformer

**Load the model**

```
bert_model = SentenceTransformer('hackathon-pln-es/paraphrase-spanish-distilroberta')
```

**Figure 4.23** : Loading BERT model

### 4.2.3 Vectorization and similarity calculs

Once we have the trained and pre-trained models we can vectorize each address in the dataset and calculate the similarities in order to classify into matched and not matched. For word2vec and FastTex we cannot directly get the vector representation of a whole address, we have a vector per word, so we will average the word vectors.
The function below will receive a list of addresses and a model to generate the corresponding vector representation of each address .

```
def vectorize(list_of_docs, model):
    features = []

    for tokens in list_of_docs:
        zero_vector = np.zeros(model.vector_size)
        vectors = []
        for token in tokens:
            if token in model.wv:
                try:
                    vectors.append(model.wv[token])
                except KeyError:
                    continue
        if vectors:
            vectors = np.asarray(vectors)
            avg_vec = vectors.mean(axis=0)
            features.append(avg_vec)
        else:
            features.append(zero_vector)
    return features
```

**Figure 4.24 :** Function for generating vector per document(address)

```
w2v_model_2_vectors = vectorize(datas, model=w2v_model2)
print(w2v_model_2_vectors[0])
```

**Figure 4.25 :** Applying the function with the trained wor2vec model

We can see the corresponding vector of the address : CAMINO TEJAR 28

```
[ 0.32487175 -0.06983898 -0.28631088 -0.07769249  0.18696737  0.14068598
  0.1528792   0.11050671 -0.42816576 -0.21292378 -0.20988278  0.13287395
 -0.04321728  0.1939742   0.09000826 -0.07008602  0.00350338 -0.26867136
  0.25565034 -0.01817125 -0.06040456 -0.40466762 -0.09120344 -0.03716366
  0.00863494  0.28039432  0.32198122  0.14744426 -0.23559588 -0.25140622
  0.00496901 -0.11078467  0.07221968 -0.1807878   0.05796105  0.05160815
 -0.1925986   0.05685646  0.01226194 -0.02997083 -0.16425471  0.10060788
 -0.02159311 -0.20773236  0.28445613  0.23026954 -0.31417975 -0.04507994
 -0.10413143 -0.08262124  0.02941904 -0.01481266 -0.13296624  0.24946253
  0.11183866  0.04785863 -0.07550734  0.0510516  -0.15613616  0.08149436
 -0.19185121  0.01418037 -0.14817716  0.16204579 -0.17794545 -0.06325928
 -0.23334403 -0.01964824 -0.091258   -0.09305105 -0.14677541 -0.18551378
 -0.09972171  0.12538408  0.18113041  0.35706174  0.09967774  0.09975817
  0.13189076 -0.15870689 -0.05590704  0.02495185 -0.06690349  0.02773413
  0.06630064 -0.06057267 -0.05382111 -0.1306314   0.00786229  0.10132403
  0.10147706 -0.1254906   0.16279471 -0.10182753  0.13775833 -0.07452685
  0.19163716 -0.16254686  0.13291274  0.16646305] CAMINO TEJAR 28
```

**Figure 4.26** : Vector representation of an address with word2vec

With doc2vec and BERT we directly generate the vector representation of the whole address.

In order to measure the similarity between addresses we evaluate the cosine similarity between their represented vectors. However, we will introduce a heatmap to represent the similarities.

Beforehand we will define two functions respectively for similarities calculations and heatmap creation

```
def calcul_similarity(vects,adr):
  similarity = []
  for i in range(len(adr)):
      row = []
      for j in range(len(adr)):
          row.append(cos_similarity(vects[i],vects[j]))
      similarity.append(row)
  return similarity
```

**Figure 4.27** : function for similarities calculations

```
def create_heatmap(similarity,adr,cmap = "YlGnBu"):
  labels = [headline for headline in adr]
  df = pd.DataFrame(similarity)
  df.columns = labels
  df.index = labels
  fig, ax = plt.subplots(figsize=(5,5))
  sns.heatmap(df, cmap=cmap)
```

**Figure 4.28** : Function for heatmaps creation

The figure 4.29 represents the heatmap of the similarity between addresses using our trained word2vec model .
The graphics for each models is available in the appendices 6.1

```
create_heatmap(calcul_similarity(w2v_model_1_vectors,addresses),addresses)
```

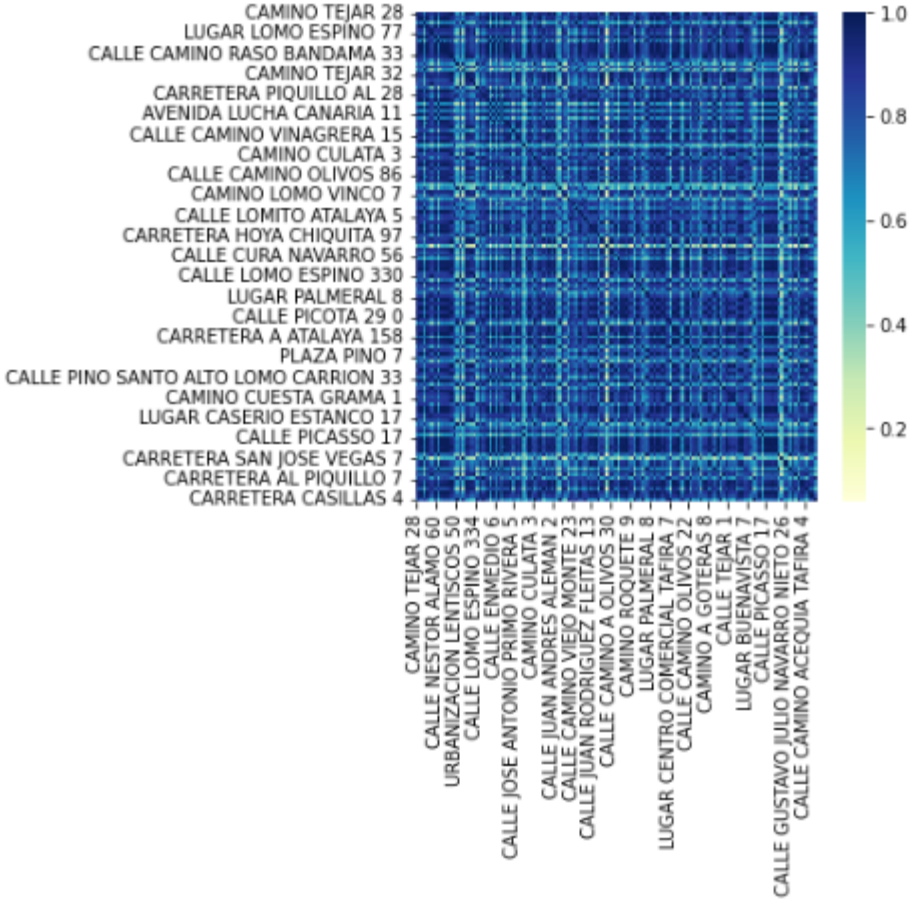



**Figure 4.29** Heatmap of similarity using wor2vec trained model

## 4.3 Results

In order to compare the models we will define, as discussed in chapter 3, a procedure for performance evaluation similar to those applied in machine learning classification algorithms. A confusion matrix is defined and the metrics accuracy, precision and recall are evaluated.

The function in the figure 4.30 shows the different steps for creating the confusion matrix

```
def MatrixConfusion(vectors,uuid):
  TP=0
  FP=0
  TN=0
  FN=0
  for k in range(len(uuid)):
    row=[]
    for j in range(len(uuid)):
        if(cos_similarity(vectors[k],vectors[j])>0.9 and uuid[k]==uuid[j]):
          TP=TP+1
        if(cos_similarity(vectors[k],vectors[j])>0.9 and uuid[k]!=uuid[j]):
          FP=FP+1
        if(cos_similarity(vectors[k],vectors[j])<0.9 and uuid[k]==uuid[j]):
          FN=FN+1
        if(cos_similarity(vectors[k],vectors[j])<0.9 and uuid[k]!=uuid[j]):
          TN=TN+1
  Accuracy = (TP+TN)/(TP+FP+TN+FN)
  Precision = TP/(TP+FP)
  Recall=TP/(TP+FN)
  F1=(2*TP)/((2*TP)+FP+FN)
  print("****Confusion Matrix******")
  print("Accuracy : ",Accuracy)
  print("Precision : ",Precision)
  print("Recall : ",Recall)
  print("F1-Score : ",F1)
  print("**************************")
```

**Figure 4.30** function for creating confusion matrix

The function receives as parameters address vectors and the list of address uuids. The function process by calculating the cosine similarity between addresses one by one and comparing the result with the fixed threshold (0.9 ). The threshold has been fixed to this value after testing the performance of the models with several values between 0.7, 0.8 and 0.9 .

The table 4.2 resume the results :

| Model | Accuracy | Precision | Recall |
|---|---|---|---|
| Trained wor2vec | 0.675 | 0.023 | 0.93 |
| Pre-trained wor2vec | 0.789 | 0.034 | 0.92 |
| Trained FastText | 0.552 | 0.017 | 0.94 |
| Pre-trained FastText | 0.976 | 0.233 | 0.848 |
| Trained doc2vec | 0.996 | 0.694 | 0.848 |
| BERT | 0.997 | 0.80 | 0.848 |

**Tabla 4.2** : Results resuming

In general the model trained with address dataset presents acceptable performance. In the case of pre-trained models only BERT is giving acceptable results. The accuracy obtained with word2vec and fastText is very low, taking a high value of recall. This means that these models predict as “match” addresses that “no match” (precision), however, they predict as “match” addresses that match (recall). On the other hand, doc2vec outperforms the two previous models and finally, the results obtained with BERT improves the performance reaching promising values.

Doc2vec with BERT form the most performing models, the trained word2vec and fastText present almost the same results while their pre-trained fail to perform.

# Chapter 5 : Conclusions and future development

## 5.1 Conclusions

In this project we explore the use of machine learning techniques in the field of address matching.I n particular, with addresses in text-based format, we introduce natural language processing approaches and generate numerical representations for each pair of addresses.

In order to generate numerical representations of addresses we study several word and sentence embedding models such as wor2vec , fastText , doc2vec and BERT.
In a first time we train these models with real address datasets from ISTAC [1], in a second time we use pre-trained models for the Spanish language pre-trained by other communities and with a large corpus of data.

We introduce the cosine similarity as a metric for resolving address records into match and not match and finally evaluate the performances.

The specific studies during this project show great potential for the use of machine learning and NLP in the field of address matching but it's really important in the implementation processes to accurate data and choose the right models.

The results obtained lead us to the conclusion that it is promising to solve the address matching problem through the similarity of the vectors that generate the language models. They also reveal the need for models generated with large numbers of documents, in our tests the guarantees are offered by the BERT model for the Spanish used, but they also suggest that generating a doc2vec model with a much larger volume of addresses can lead to good system performance.

## 5.2 Future developments

- This study has been restricted to the municipality of Santa Brigida, but in future works, with more available resources, we plan to extend it into all municipalities of the Canary Islands in order to confirm our generic method.

- As we did with the different models comparing their performance using cosine similarity it should be realised as a comparison, a study based on the different similarity metrics such as euclidean distance, levenshtein distance ….etc

- In the case of having a dataset with a great pool of variations for each address, it should consider each pool of addresses as a machine learning classification problem. In this case, we will use machine learning classification algorithms like XGBoost, random forest ….etc

- Generate language models with the data for all Canary Islands addresses in the Canary Islands Integrated Data System

- Explore algorithms to improve efficiency in the comparison of the similarity of all addresses in order to extend the results to datasets that include a significant volume of addresses, for example for the whole of the Canary Islands.

# Chapter 6 : Appendices

### 6.1 Dataset creation code source

At the following Colab notebook we have the python code to created the dataset that will be use in the project

Google Colab Link

### 6.2 Project code source

Below is a link to the project on Google Collaboratory where you can view and test the Python code that has been shown throughout this project.

Google colab Link

### 6.3 Data sources

At the following link we have a shared google drive folder that contains all the data files in csv format used for this project.We can find the “Santabrigida.csv” file used for creating the base dataset “muestra.csv”.

Google drive Link

### 6.4 Pre-trained models

At the following link we have a shared google drive folder that contains all the pre-trained models used on this project.

Google drive Link

# Bibliography

**[1]** . ISTAC ( Instituto Canario de Estadística) official website
http://www.gobiernodecanarias.org/istac/

**[2]**. PostMatch : A Framework for Efficient Address Matching
Springer Nature Singapore Pte Ltd. 2021 Y.Xu et al. (Eds) : AusDM 2021, CCIS 1504,pp. 136-151,2021.
https://doi.org/10.1007/978-981-16-8531-6_10

**[3]** ISTAC Sistema-georreferenciación
https://jecas.es/wp-content/uploads/2021/11/21.4.ISTAC_Sistema-georreferenciacion.pdf

**[4]**. Comber S , Arribas-Bel D. Machine learning innovations in address matching : A practical comparison of wor2vec and CRFs.*Transaction in GIS* . 2019;23:334-348.
https://doi.org/10.1111/tgis.12522

**[5]**. The ultimate guide to address matching (online)
https://www.placekey.io/blog

**[6].** Introduction to NLP
https://builtin.com/data-science/introduction-nlp

**[7]**. Theory behind the basics of NLP
https://www.analyticsvidhya.com/blog/2022/08/theory-behind-the-basics-of-nlp/

**[8]**. Wikipedia : n-gram
https://en.wikipedia.org/wiki/N-gram

**[9]** Char n-gram
https://subscription.packtpub.com/book/big-data-and-business-intelligence/9781787126787/9/ch09lvl1sec56/character-n-grams#:~:text=An%20n%2Dgram%20is%20a,high%20quality%20for%20authorship%20attribution.

**[10]**. Word Embedding, Wikipedia
https://en.wikipedia.org/wiki/Word_embedding

**[11]**. Countvectorizer, scikit-learn
http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

**[12]**. Ultimate guide to Text similarity with python
https://newscatcherapi.com/blog/ultimate-guide-to-text-similarity-with-python

**[13]**: Efficient Estimation of Word Representations in Vector Space (Research paper)
https://arxiv.org/pdf/1301.3781.pdf

**[14]** Fastext Model
https://radimrehurek.com/gensim/auto_examples/tutorials/run_fasttext.html#:~:text=The%20main%20principle%20behind%20fastText,embedding%20for%20every%20individual%20word.

**[ 15 ]** Fastext
https://blogs.sap.com/2019/07/03/glove-and-fasttext-two-popular-word-vector-models-in-nlp/

**[16]** Distributed Representations of Sentences and Documents (Research paper)
https://cs.stanford.edu/~quocle/paragraph_vector.pdf

**[17]** An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation
https://arxiv.org/abs/1607.05368

**[18]** BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
https://arxiv.org/pdf/1810.04805.pdf

**[19 ]** BERT Explained State of the art language model for NLP
https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

**[20]**. Jaccard Index ,Wikipedia
https://en.wikipedia.org/wiki/Jaccard_index

**[21]** The Levenshtein Algorithm

https://www.cuelogic.com/blog/the-levenshtein-algorithm#:~:text=The%20Levenshtein%20distance%20is%20a,one%20word%20into%20the%20other.

**[22]** González Yanes, A., Betancor Villalba, R., Hernández García, M.S. (2021). Título. XXI Jornadas de Estadística de las Comunidades Autónomas, JECAS. Las Palmas de Gran Canaria: ISTAC.
Retrieved from https://jecas.es/sistema-de-georreferenciacion-para-fines-estadisticos/

**[23]** Gensim documentation
https://radimrehurek.com/gensim/

**[24]** Sentence transformers Documentation
https://www.sbert.net

**[25]** Wor2vec pretrained model for spanish language
https://github.com/aitoralmeida/spanish_word2vec

**[26]** FastText pretrained Models
https://fasttext.cc/docs/en/crawl-vectors.html

**[27]** BERT Spanish Model
https://huggingface.co/hackathon-pln-es/paraphrase-spanish-distilroberta