UNIVERSITÀ DI PISA

Department of Philology, Literature and Linguistics

Master Degree in Digital Humanities

MASTER THESIS

Probing the linguistic knowledge of word embeddings: A case study on colexification

SUPERVISORS:
Prof. Alessandro Lenci
Dr. Thomas Brochhagen

CANDIDATE:
Sara Bertoldo

Academic Year 2020-21

Abstract

In recent years it has become clear that data is the new resource of power and richness. The companies that are able to manage it to extract useful information are the ones that are expected to last and increase their profits. One of the ways in which data is conveyed is through natural language: every day we produce an enormous amount of linguistic data, in written or spoken forms. Through the help of computational resources, we can manage such a big quantity of information, in an automatized and scaled way. Before being able to do this, we need to find ways to allow computers to represent linguistic knowledge. This is indeed a problem, considering that computers do not have linguistic proficiency as we humans do. For words to be processed by machine models, they are often required to have some form of numeric representation that models can use in their calculations. One method that has become influential in recent years is word embeddings, defined as the representation of terms as real-valued vectors such that the words that are closer in the vector space are expected to be similar in meaning. These techniques are very popular and have shown great success in multiple studies, but it is still not clear what kind of linguistic knowledge they do acquire. Also, it is still an open question exactly in which way some of their parameters affect the knowledge they acquire. The present work is motivated by figuring it out. We are going to test the system on a linguistic problem. The issue under examination is colexification: the phenomenon in which, within a language, multiple meanings are expressed by a single word form. One of the reasons why this circumstance happens has been suggested to be a semantic connection between the meanings. It follows that two similar meanings are more expected to be conveyed through a single term with respect to two meanings pertaining to completely different fields. We assume that there is a relationship between distributional similarity and colexification, in the sense in which the former is informative about the latter. This assumption is more concretely based on the results from Xu et al. (2020). We use this study as a general guide to follow in this investigation. We used some word embedding models, specifically, fastText trained with different window sizes, to obtain the cosine similarity values between pairs of words. Subsequently, we performed two predictive tasks, showing how using a predictive model like logistic regression and nothing else than the cosine similarity values between word vectors, it is possible to predict whether a pair of meanings is a highly frequent colexification or whether it is a colexification at all. The results suggest that the linguistic models in use were able to acquire a certain knowledge as regards word meaning. Additionally, changing the model parameter of window size, we inspected what kind of linguistic knowledge the computational models acquired concerning colexification. The project covered the whole working process. We started from the data collecting, understanding and cleaning, to get to the training of the fastText model, and evaluation of the results obtained by the predictive model. Our findings indicate that a narrow window size value is sufficient to allow the linguistic model to acquire a good level of semantic knowledge in a distributional similarity task. Additionally, the parameter of window size, depending on the task, does not always lead to different results in computation. This raises a broader question: in which tasks does window size matter and what does this tell us about these tasks.

Keywords: Word Embedding, Natural Language Processing, FastText, Window size

Alle scialuppe della mia vita

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"The fish trap exists because of the fish. Once you've gotten the fish you can forget the trap. The rabbit snare exists because of the rabbit. Once you've gotten the rabbit, you can forget the snare. Words exist because of meaning. Once you've gotten the meaning, you can forget the words."

Zhuangzi, Chapter 26

The recent fortune of artificial intelligence and of the technologies that use it has stemmed from its ability to manipulate the vast amounts of data available today.

Data has undoubtedly been established in recent years as the new resource of power and richness. Suffice it to see that the podium of the best global brands is formed by Apple, Amazon, and Microsoft, three companies which work with a huge amount of information every day [1].

One of the ways in which data is conveyed is through natural language. Every day we produce an enormous amount of linguistic data, in written or spoken forms. There is one problem arising: how to harness this information, in other words, how to make this information useful. To explain the problem from another perspective, we have large amounts of linguistic data, but we need ways to capture what this data conveys.

Every living creature has a way to communicate. Birds communicate with vocal sounds and body language, trees communicate using pheromones, fishes communicate through various characteristics like color, electrical impulses, and

[1] To see the complete ranking: `https://www.interbrand.com/best-global-brands/`

bioluminescence. But no matter how complete and complex their systems could appear, it seems that the human language overtakes them. Other creatures use language as a means to achieve their primary purposes: eating, reproducing, surviving. Humans, besides these necessities, use language as an entertainment medium (it is the case of poetry or radio), as a way to reach trivial goals (e.g. make friendship), or as a self-studying system (for example, when we learn a new language), just to name some uses [2]. Linguistic competence is so complex that it has been thought to be what distinguishes us as intelligent beings. It is the case of the Turin test where the language was the mean to test whether a person was able to distinguish between human and computer's answers [Turing, 1950]. Little is still known about how language develops or is learned. Theories debate on various linguistics issues, leaving a big amount of questions still unanswered. For example, is language an innate characteristic? How can we know the structure to build a sentence? How to determine if a language allows the lack of the subject pronoun?[3] It is not easy to answer these questions, pointing out how little we know about our linguistic faculty.

Besides these unresolved questions, waiting to have a complete understanding of the subject to start using technology in the linguistic field is not an option. On the contrary, there appears the possibility to use computational approaches to shed a light on human linguistic phenomena; and as already anticipated, the state-of-the-art technologies already use computational techniques to grasp information from language, showing how it is not even necessary to have a full understanding of the language faculty [Storks et al., 2020].

The first thing we will focus on is exactly how it is possible to represent linguistic knowledge in computational forms since computers do not have linguistic proficiency as we humans do. For words to be processed by machine models, they need some form of numeric representation that models can use in their cal-

[2]To deepend the issue, check Hockett [Hockett and Hockett, 1960]

[3]To deepen these questions, some cornerstones are listed. Chomsky and Jackendoff proposed the innate nature of language through Generative Grammar and Universal Grammar, [Chomsky, 1965, Jackendoff and Peruzzi, 1998]. Chomsky proposed the famous X-bar syntax structure [Chomsky, 1957]. Biberauer et al. proposed the existence of a parameter that triggers the absence of the subject pronoun [Biberauer et al., 2010]

culations. Linguists and language engineers tried to solve the problem over the years, providing different ways to represent word meaning and linguistic structures. The meeting between Computer Science and Linguistics was unavoidable: the first studies how to manage information and the second analyzes the most powerful tool humans have to express information (i. e. language). The combination between these two sciences leads to the development of Computational Linguistics: the scientific study of language from a computational perspective [The Association for Computational Linguistics, 2005]. Computational Linguistics has fueled the possibility to represent language in a machine-friendly way, consisting of ruled, ordered, and coherent systems, with the objective of making machines capable of reproducing linguistic competence.

One of the most influential mechanisms in recent years are word embeddings. These models represent each word as a real-valued vector in a multi-dimensional vector space. In this environment, words that are closer in the vector space are expected to be similar in meaning. The paradigm comes from the distributional hypothesis, which states that words that occur in similar contexts tend to have similar meanings [Harris, 1954, Firth, 1957]. The system has been adopted by multiple models among which the most widely known are Word2vec [Mikolov et al., 2013a, Mikolov et al., 2013c], GloVe [Pennington et al., 2014] and fastText [Bojanowski et al., 2017]. These models all rely on a vector space paradigm and would allow performing operations with distributional semantic word meaning. Apart from the problem of word representation, we can perform multiple tasks on linguistic data. To list some, this is the case of sentiment analysis, semantic search, text mining.

Although the popularity and success of these models, it is still not clear what kind of linguistic knowledge they acquire. This is because some of the models rely on a neural network structure, facing the well-known problem of the black box[4]. In short, we can explain what the model does and how it does it, but we cannot explain its results. The present work is motivated by trying to better understand the linguistic knowledge learned by the models, working on a practi-

[4]A model is called black box when input and output are known but its internal behavior is not visible or is unknown; its decisions are not completely understandable or predictable

cal case. We are going to test the system on a linguistic problem and adduce, based on its performance, what kind of linguistic knowledge it acquires and under which circumstances (i.e., parameter settings). The issue under examination is *colexification*: the phenomenon in which, within a language, multiple meanings are expressed by a single word form [Xu et al., 2020a, p. 3]. To make an example, this is the case of the Spanish word "pueblo" which means both "village" and "people".

One of the reasons why this circumstance happens is thought to be a semantic connection between the meanings [François, 2008, Xu et al., 2020a]. Under this assumption, two similar concepts are more expected to be conveyed through a single term with respect to two concepts pertaining to completely different fields. For example, the Italian word "dito" conveys two similar meanings: "finger" and "toe", that is "any of the five separate parts at the end of the foot/hand". At the same time, to avoid incurring in communication's misunderstandings, two concepts that are very related in meaning with a high probability use distinguished lexemes (this is the case, for example, of "brother" and "sister" [Regier et al., 2016]).

In this work, we assume that there is a relationship between distributional similarity and colexification, the former being informative about the latter. This assumption is based on various previous studies that defended a cognitive economy tendency [Geeraerts, 1997, Bloomfield, 1922, Rosch, 1978, Zipf, 1949]; and, more concretely, on the results from Xu et al. [Xu et al., 2020a]. If a computational model was able to correctly represent meaning, we would verify the claim. In short, we expect to find a way to operationalize linguistic similarity (in this case, distributional similarity), and use the obtained values to verify the theory. In addition, we plan to inspect in more detail what kind of linguistic knowledge these models acquire with respect to colexification, by changing the models' parameters. We expect that by changing the parameters used to train the computational model, we should see how the linguistic outcomes are modified.

The results we obtain show how using a computational model of meaning and nothing else than the cosine similarity values between word vectors, it is possible to predict if a pair of meanings is a highly frequent colexification or if it is a colexification at all. This means that the linguistic model was able to acquire a certain knowledge as regards word meaning. Specifically, the model succeeds in representing words in the vector space, with distributional similarity understanding. Furthermore, when changing the parameter of window size in the model training phase, we modify the values of the word vectors, and subsequently both the cosine similarities and the predictions. The window size parameter represents the number of words to consider as context when training the model. Given that we rely on the distributional hypothesis to catch the word meaning, this parameter sounds like the most important one for our task. We aim to obtain the most reliable word similarity schema, and we obtain that with a window size dimension of 3, i.e. we consider three context words on the left and three context words on the right of the target word when training the model. Furthermore, we also noticed that there is a certain invariance with respect to window size depending on the task. This raises broader questions about the kind of tasks in which window size matters; and suggests that highly frequent vs. less frequently colexified meanings are different.

The thesis's structure is composed of six chapters, the first being the ongoing.

Chapter 2 will handle the state of the art of word representations, with a focus on word embedding models.

Chapter 3 will set up the research problem, and explain why we decided to address it. We will study the work of Yang Xu et al. [Xu et al., 2020a], analyzing how they developed their analysis. In this section, we will also describe the development plan to follow in the following chapters and phases of work.

Chapter 4 will explain the implementation of the project, with its evaluation. The chapter will begin with a data management phase, followed by the creation of the colexification data frame, the development of the bootstrap process, and the calculus of probabilities associated with each meaning pair. We will describe how we obtained the cosine similarity values for the colexified meaning pairs using the pre-trained fastText model, and analyze their values. Subsequently, we will use the logistic regression model to answer two questions:

- How well can the cosine similarity values help us in predicting whether two meanings are highly colexified across different languages?

- How well can the cosine similarity values help us in predicting whether two words are a colexification at all?

In Chapter 5 we will modify the process explained above. We will prepare a training corpus and train our fastText models, changing only the value of window size.

To finish, chapter 6 will judge the final outcome, by carrying on the conclusions and the future developments.

It should be noted that the goal of the current work is not to replicate or outperform the results obtained by Xu et al. The focus will rather be to study if a word embedding model is able to acquire linguistic knowledge, specifically the distributional similarity existing between words.

The whole project has been carried on as traineeship experience working remotely at the Computational Linguistics and Linguistics Theory (COLT) group of Universitat Pompeu Fabra of Barcelona [5]. For this reason, besides the purpose of testing linguistic knowledge of word embeddings, there was another aim, consisting of developing work-related skills, to name a few, coding and code management, decision making, reporting, and using existing state-of-the-art computational algorithms.

System specifications

The experiments were performed mainly on the Google Colaboratory service[6]. This platform allows running experiments on the cloud exploiting Google's processing power.

The model training phases were performed remotely on a High Performing Computing (HPC[7]) Cluster of Universitat Pompeu Fabra of Barcelona.

[5]COLT: `https://www.upf.edu/web/colt`

[6]Google Colab: `https://research.google.com/colaboratory/`

[7]HPC UPF: `https://guiesbibtic.upf.edu/recerca/hpc`

Script availability

All the scripts to inspect and replicate the analysis are available in the following GitHub repository:

`https://github.com/sarabert96/Colexification`

Chapter 2

Representing word meaning

This chapter will retrace the steps of the computational word representation, starting from the most naive methods and reaching developed word embedding mechanisms, able to better grasp word meaning.

Computers, for the way they are designed, can only work with electronic signals, processed at the lowest level in the processor. The most straightforward way to represent an electronic signal is through digits, specifically zeros, and ones. These digits constitute the binary code which, in turn, is the elaboration of numbers, operations, and instructions. Just like images, processed in tiny squares called pixels where each pixel is stored in memory as a combination of colors; or songs, designed as slices of audio signals, represented through binary values; text also needs its representation.

ASCII (American Standard Code for Information Interchange)[1], ISO-8859, Unicode[2] are good ways to store text data. In these coding systems, every digit is saved as one or more bytes. But unfortunately, the systems do not allow to have any awareness of the words' meanings, merely creating a convention between a code and a graphic symbol.

To try to solve this problem, there have been explored various possibilities. In the following sections, we will see different ways to represent words, looking at how they work and tracing the development of the field.

[1]ASCII: `https://www.ascii-code.com/`

[2]Unicode: `https://home.unicode.org/`

2.1 Bag of Words

Bag of Words (BoW) is a technique to extract features from a text. In the model, the features are the unique words in the document, represented as a vector (aka a list) of numbers. The length of the vector, that is the number of dimensions, is the length of the vocabulary (list of unique words). It relies on the idea that similar documents contain similar words. Further, from the content of a document, it is possible to understand the meaning of the document. Any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document, this is why is called *bag*. The representation involves two things: a vocabulary and a word frequency measure. Depending on the way to calculate the frequency measure, we can identify different kinds of Bag of Words, explained as follows.

2.1.1 One-Hot Encoding

Each document will be represented as a vector of words, where the frequency measure is given by a boolean value: 0 if the word is absent, 1 if it is present. To better understand the mechanism, let's consider an example. Citing Brownlee [Brownlee, 2017, p. 64], here are the first lines of the book "A tale of two cities" by Charles Dickens.

It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness

We will treat each line as it was a separate document. The vocabulary is made of unique words, lowercased and without punctuation:

[it, was, the, best, of, times, worst, age, wisdom, foolishness]

The corpus contains 24 words but our vocabulary is only size 10. The size of the vocabulary will be the dimension of the vector. To mark the presence of a word we will use boolean values: 0 is absent, 1 if present. The resulting vectors are shown in table 2.1.

Table 2.1: One-Hot documents' vectors

Vectors	it	was	the	best	of	times	worst	age	wisdom	foolishness
1	1	1	1	1	1	1	0	0	0	0
2	1	1	1	0	1	1	1	0	0	0
3	1	1	1	0	1	0	0	1	1	0
4	1	1	1	0	1	0	0	1	0	1

2.1.2 N-gram model

A way to simplify the vectors is to group a certain number of words in a single dimension. In this case, a word is called a *gram* and so, if we consider two words at once, we are using bigrams. The general approach is called the n-gram model, where n refers to the number of grouped words. For example, the bigrams of the first sentence in the previous section are

[it was, was the, the best, best of, of times]

A n-gram approach has been shown to works better than a 1-gram bag of words approach given its improven ability to capture structure [Goldberg, 2017, p. 75].

2.1.3 Frequency vectors

This representation works like the one-hot encoding, but instead of boolean values, we use discrete numbers. In this case, 0 stands for a word not in the document, while a specific number represents the number of times that word is repeated in the document. To understand the approach with an example, let's consider the following sentences:

Alice likes watching TV. Bob doesn't like watching TV.

Bob loves watching movies.

The vector representation of the sentence is shown in table 2.2.

Table 2.2: Frequency vectors

Sentence	alice	like	watch	tv	bob	do	not	movie	love
1	1	2	2	2	1	1	1	0	0
2	0	0	1	0	1	0	0	1	1

One problem with this representation is that language contains a big amount of words that are very common but that do not contain much information. It is the case of words like *the, do, not, a, some*, among others.

2.1.4 Term Frequency/Inverse Document Frequency

As we saw before some words like articles, negative particles, auxiliary verbs, are very frequent but convey low information. In addition, it is highly probable that long documents contain more repetitions of the same word, compared to short documents. To solve the situation, Term Frequency/Inverse Document Frequency [Jones, 1972, Ramos, 2003] is a method to normalize the frequency of tokens. In the method, the importance of a word is directly proportional to the number of times a word appears in the document but is offset by the number of documents in the corpus that contain that word and the length of each document. The mechanism works with two definitions:

- Term Frequency (TF): is a scoring of the frequency of the word in the current document.

$$TF(t) = \frac{number\ of\ times\ term\ t\ appears\ in\ document}{total\ number\ of\ terms\ in\ the\ document} \tag{2.1}$$

- Inverse Document Frequency (IDF): is a scoring of how rare the word is across documents. The rarer the term, the higher is the IDF score.

$$IDF(t) = \log_e \frac{total\ number\ of\ documents}{number\ of\ documents\ with\ term\ t\ in\ it} \tag{2.2}$$

It results that:

*TF-IDF = TF * IDF*

This time, in the document vector, instead of the boolean or counted value, for every term we insert the TF-IDF score.

In all these representations, every new word added to the vocabulary becomes a new dimension in the vector. We can foresee that the bigger the vocabulary, the higher the number of dimensions. If we take into account a large corpus, it results that we will have vectors of thousands or millions of dimensions. As a logical consequence, the higher the number of dimensions, the more challenging their management. This is what is called the "curse of dimensionality". Furthermore, the vectors contain lots of zero scores, called a sparse vector or sparse representation. Sparse vectors require a lot of memory and computational resources, with a very low optimization.

To decrease the number of dimensions to improve the performances, there exist various techniques: ignoring case, ignoring punctuation, ignoring very frequent words that do not contain much information (called *stop words*), fixing misspelled words, reducing words to their stem [Brownlee, 2017, p. 65]. For example, the sentence *Yesterday, we were eating some delicious cookies.* will result in the following vocabulary:

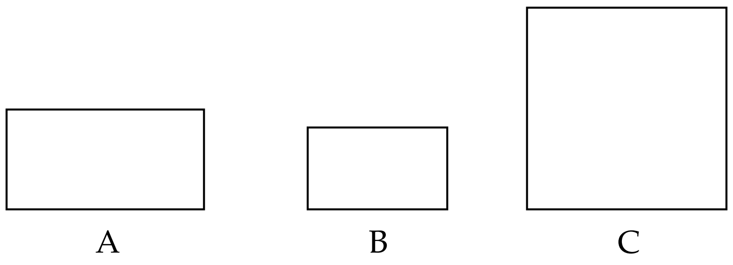
[yesterday, eat, delicious, cookie]

In addition, an obvious limitation of these approaches is that they do not encode any idea of meaning or word similarity into the vectors. For this reason, we will move on with other representational models, seeking if they can solve our problem.

2.2 Understanding Word Embeddings

2.2.1 The mechanism

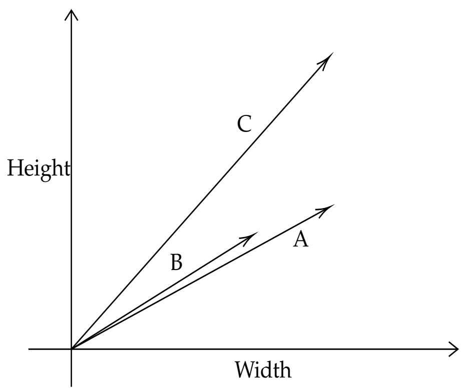
To prepare the ground for a full understanding, let's start with a simple example. Image we are trying to describe three rectangles as in this figure:



We could describe each rectangle with two values: height and width.

A (5,10); B (4,7); C (10,10)

Let's symbolize each rectangle as a vector in two dimensions:



The vector space (a collection of vectors) allows us to represent the vectors' features (but this is something we could do even with the previous methods). The usefulness of such representation comes when we want to compare two vectors. If we would want to find the most similar rectangle to A, we could use the cosine measure of the angle between two vectors, defined as follows:
let a and b two vectors, θ their angle, then:

$$cos(\theta) = \frac{a \cdot b}{||a|| \cdot ||b||} \tag{2.3}$$

The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for orthogonal vectors, to -1 for vectors pointing in opposite directions [Jurafsky and Martin, 2000, p. 11]. In our case, the vector with the higher cosine would be B, as we could notice graphically.

2.2.2 Representing words

Likewise, we can represent a word via a real-valued vector, but this time the number of dimensions would not be two. We would have to deal with a high-dimensional space, of about hundreds of dimensions, barely impossible to imagine in a vector space. A good thing is, though, that cosine similarity still works, since it works with any number of dimensions. In addition, we are dealing with vectors of hundreds of dimensions in contrast to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding [Brownlee, 2017]. Computationally speaking, it has been proven that the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors and benefit of dense representations [Goldberg, 2017]. Representing words as low-dimensional dense vectors requires the classifiers to learn far fewer weights than if we represented words as high-dimensional vectors, and the smaller parameter space possibly helps with generalization and avoiding overfitting [Jurafsky and Martin, 2000, p. 17].

The distributed representation is learned based on the usage of words. The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distributions of word neighbors [Jurafsky and Martin, 2000]. This allows words that are used in similar ways to result in having similar representations. This can be contrasted with the representation in a bag of words model where different words have different representations, regardless of how they are used [Brownlee, 2017].

The theory is the distributional hypothesis proposed by Harris in 1954. The paradigm states that "words that occur in similar contexts tend to have similar meanings" [Harris, 1954]. The theory has been revived by the idea of Firth (1957) by which:

> You shall know a word by the company it keeps [Firth, 1957, p. 11]

To explain the concept, let's consider an example:

The dog is eating the food.

The cat is eating the food.

Looking at these two sentences, we can say that *dog* and *cat* are two words that convey similar meanings, given the fact that they can appear in the same context. Moreover, if the model could have the notion of *is eating*, it would understand that *dog* and *cat* are two living creatures, just because they are followed by an action verb.

Let's see another example.

Let's go get a xxxxx or something.

If you don't know how to make a homemade xxxxx, you've come to the right place.

I'll take a xxxxx and fries.

In this case, we do not know which is the missing word. Perhaps, xxxxx could be a new English word. Nevertheless, we can for sure guess that this new word is a food.

2.2.3 Obtaining word vectors

But, how do we obtain word vectors? We first need input data, which in our case will be a big quantity of text data. Fortunately, the only constraint in the input data is that we need plain text. This allows us to dispose of huge quantities of data effortlessly and is the main reason why models relying on word embedding have such great success. The specific steps to obtain word vectors are the distinctive features by which we differentiate models.

There are two main categories to distinguish: static embeddings and dynamic or contextualized embeddings. With static embeddings, methods learn one fixed embedding for each word in the vocabulary. It is called context independence, in the sense that if a word assumes a different meaning depending on the context, the model combines all the different senses of the word into one vector. So, for example, in the sentence "He went to the prison cell with his cell phone to extract blood cell samples from inmates", where the word "cell" has different meanings based on the sentence context, the model just collapses them all into one vector. With dynamic embeddings, on the other hand, the vector for each word is different in different contexts. In the previous example, we would have three different "cell" vectors depending on the context [Jurafsky and Martin, 2000]. In the first case, we only need the word embeddings to perform our tasks. In the second case, instead, it is usual to take into consideration the model itself to be fine-tuned for the specific task it is needed [Rajasekharan, 2020].

Given the task we will perform and explain in the following chapters, we will only need a word embedding for a single word, collapsing all word meanings into a single representation. For this reason, the models we will explain subsequently belong only to static embeddings.

2.3 Word2vec

The Word2vec model was developed by Tomas Mikolov, et al. at Google in 2013 [Mikolov et al., 2013a]. The intuition of Word2vec is that instead of counting how often each word w occurs near a certain word c, we will train a classifier on a binary prediction task: "Is word w likely to show up near c?" We do not actually care about this prediction task; instead, we will take the learned classifier weights as the word embeddings. As anticipated, the revolutionary intuition is that we can just use running text as implicitly supervised training data for such a classifier; a word w that occurs near the target word c acts as gold "correct answer". This method, called *self-supervision*, avoids the need for any sort of hand-labeled supervision signal [Jurafsky and Martin, 2000, p. 17-18].

The model is in the form of a shallow neural network, with one input, one output, and one hidden layer. Each word in the vocabulary has two different representations of dimension equal to the number of neurons in the hidden layer. The input words are fed into the input layer, and the network is trained using a regular back-propagation algorithm to output the target word. We will study this process deeply below.

The first thing to prepare is the corpus. Once we have the input corpus, we will set a window, which is a number of considered words. Through all the corpus, we will have to slide this window, considering step by step different words. To better understand the mechanism, let's look at the example proposed by Alammar [Alammar, 2019].
Let's consider the following sentence:

Thou shalt not make a machine in the likeness of a human mind

If the window size is 3, we will consider the first three words, where the two initial terms will be labeled as input and the third as output. The mechanism is shown in figure 2.1.

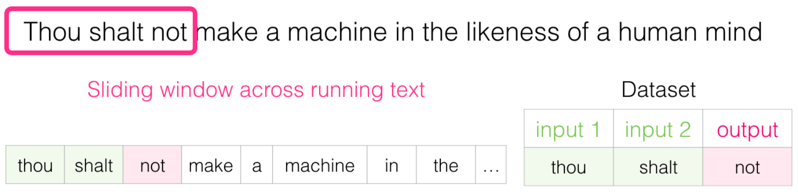

Thou shalt not make a machine in the likeness of a human mind
Sliding window across running text
Dataset
input 1
input 2
output
thou
shalt
not
make
a
machine
in
the
...
thou
shalt
not

Figure 2.1: Mechanism of the sliding window, part 1 [Alammar, 2019]

Then, we will move to the second set of three words, as shown in figure 2.2. Subsequently, we will continue until the end of the text, obtaining a big amount of inputs and outputs words. These labeled words will later be used to train the language model.

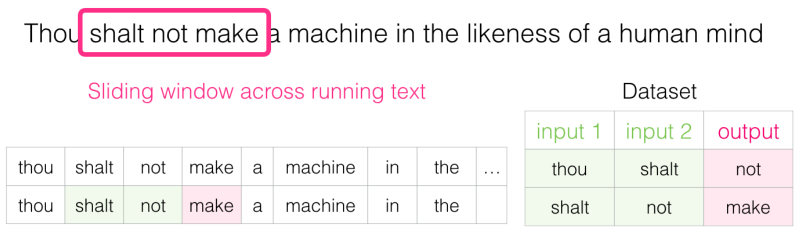

Thou shalt not make a machine in the likeness of a human mind
Sliding window across running text
Dataset
thou shalt not make a machine in the …
thou shalt not make a machine in the
input 1 input 2 output
thou shalt not
shalt not make

Figure 2.2: Mechanism of the sliding window, part 2 [Alammar, 2019]

In the example, we considered a three words window, looking at the target word and at its two preceding words. So, for example, in the sentence:

John was taking a _____

where the target word is *photo*, we could propose the words *nap, shower, photo*, and more. What if we took into account the following words also?

John was taking a _____ photo

in this case, we would guess that the missing word is likely to be *beautiful*, and we know for sure that the missing word cannot be *photo*. Taking into consideration the following word lets us improve the knowledge we previously had. Accounting for both directions leads to better results in comparison to only one direction.

There exist two training architectures to approach the problem: Continuous Bag of Words (CBOW) and Skip-gram.

2.3.1 CBOW

In the CBOW architecture, given a set of context words, we try to predict the target word. Figure 2.3 well represents the mechanism.

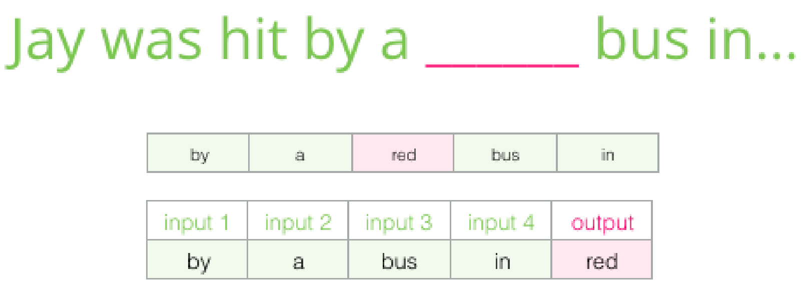

Jay was hit by a ______ bus in...
by a red bus in
input 1 input 2 input 3 input 4 output
by a bus in red

Figure 2.3: CBOW architecture example [Alammar, 2019]

As explained above, every row of inputs and outputs will form the dataset to train the model. In this example, we have four input words for every output.

2.3.2 Skip-gram

In the Skip-gram architecture, given an input word, we try to predict the surrounding context. This time, in the example, we have four input words and four output words. As result, with this mechanism, we obtain a much bigger dataset, that will slow the training process but hopefully improve the results.

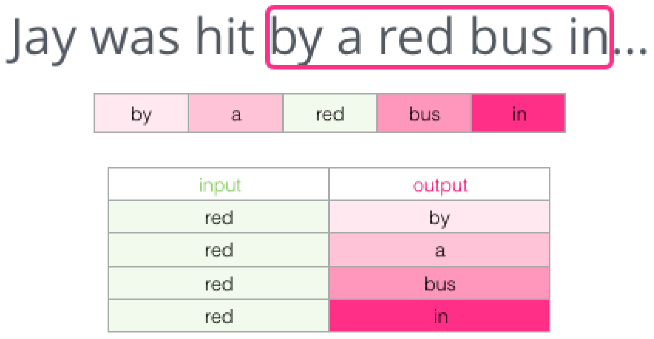


Figure 2.4: Skip-gram architecture example. The pink boxes are in different shades because this sliding window actually creates four separate samples in the training dataset [Alammar, 2019]

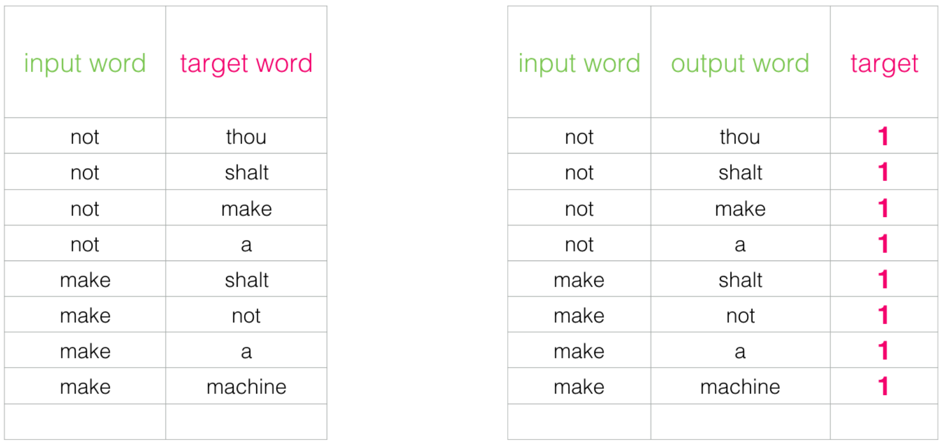
A side effect of this architecture is that when training the model with the pair [In: "red", Out: "by"] we are saying that the output "bus" is wrong. Conversely, when training the model with the pair [In: "red", Out: "bus"] we are saying that the output "by" is wrong. We are penalizing the model unfairly, telling it to adjust for words that actually appear as if they did not. To solve the issue, we should have enough data in order to make up for those incorrect adjustments, converging to a trade-off probability [Norris, 2018].

2.3.3 From Neural Network to Logistic regression

So far, we described the mechanism as a neural language model where the purpose is to predict the target of the context words. The idea of Word2vec is to switch to a logistic regression model, simplifying and speeding up the calculation.

This switch requires we change the structure of the dataset: the label is now a new column with values 0 or 1. They will be all 1 since all the words we added are neighbors as we can see from figure 2.5.

To avoid obtaining a model that always returns 1, showing zero learning, we need to introduce some negative samples, meaning, words that are not neighbors.



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Figure 2.5: Changed dataset for logistic regression. [Alammar, 2019]

To add negative samples, we keep the same input word, putting a random word labeled as 0 as a new entry in the dataset.

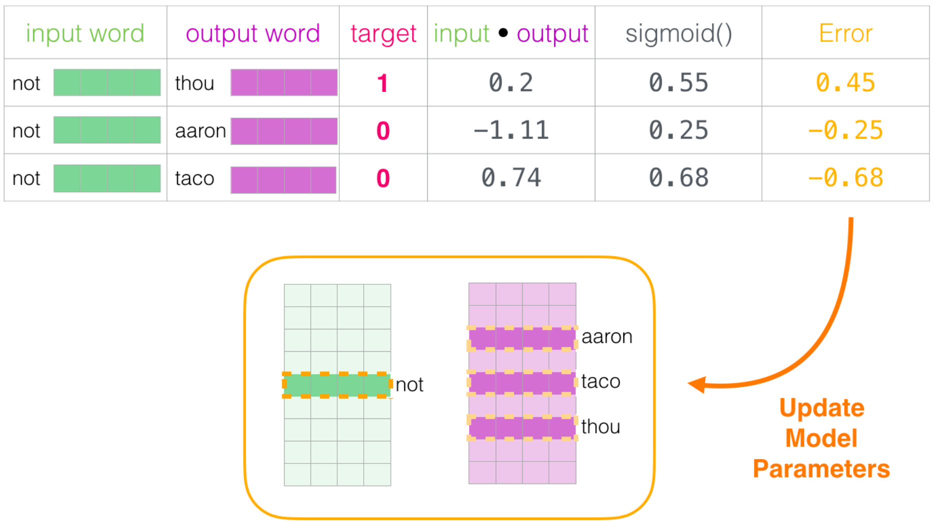
2.3.4 Training process

Before the training process starts, we determine the size of our vocabulary and the size of the embedding (number of dimensions of the vectors).

At the start of the training phase, we have two matrices: an embedding matrix and a context matrix. These two matrices have an embedding for each word in the vocabulary, so the dimensions of the matrices are

$$dimension\ of\ the\ matrix = size\ of\ vocabulary \times size\ of\ embeddings \tag{2.4}$$

The model starts initializing randomly the matrices. Given the input word, it considers the embeddings for the word itself and the context, composed of both positive and negative examples. Subsequently, it calculates the similarity between the input and output embeddings using the dot product. These values are then processed by a sigmoid function which transforms them in the range [0; 1]. The sigmoid value is the model prediction. We can compare this prediction with the real value, calculating the error value. This error value can be used to adjust the embeddings. We repeat this process to input every word in the dataset. Every repetition of the process is said to be an *epoch*. After a certain number of epochs, we expect the model to have the best possible word vectors in the embedding matrix. Figure 2.6 by Alammar summarizes the process [Alammar, 2019].



input word		output word		target	input • output	sigmoid()	Error
not	🟩	thou	🟪	1	0.2	0.55	0.45
not	🟩	aaron	🟪	0	-1.11	0.25	-0.25
not	🟩	taco	🟪	0	0.74	0.68	-0.68

Figure 2.6: Training process of the language model [Alammar, 2019]

2.4 GloVe

The GloVe algorithm was created by Pennington et al. at Stanford in 2014 [Pennington et al., 2014]. GloVe, short for Global Vectors, "is a word vector representation method where training is performed on aggregated global word-word co-occurrence statistics from the corpus" [CR, 2020]. It is a count-based model that tries to show that it is possible to use matrix factorization methods[3] to keep the statistical co-occurrence data while being able to capture the meanings of words. It is, in a certain way, an improvement of the bag of words model with the addition of the words' meaning feature. It was motivated by the idea that context window-based methods suffer from the disadvantage of not learning from the global corpus statistics. GloVe is able to capture the relationships between words by using the ratios of their co-occurrences with other words [Brich, 2018]. In other words, GloVe looks at how often a word *w1* that appears in the context of a word *w2* within all the corpus of texts.

Rather than using a window to define the local context (local context window methods are CBOW and Skip-gram), GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. Pennington et al. state that training multiple instances of a neural network and then combining the results can be less prone to over-fitting and can improve the performance for certain types of neural networks. The resulting embeddings are

[3]A matrix factorization is a way of reducing a matrix into its constituent parts.

constructed by summing the matrices.

2.5 fastText

FastText is a library created by Bojanowski et al. at Facebook in 2017 for efficient learning of word representations and sentence classification [Bojanowski et al., 2017]. FastText is considered an extension of Word2vec which enriches the word embeddings with character-level information. It can use either Skip-gram or CBOW architectures.

Bojanowski's team considers ignoring the morphology of words a limitation, especially for languages with large vocabularies and many rare words. Many word formations follow rules, making it possible to improve vector representations for morphologically rich languages by using character-level information. FastText proposes a new approach with respect to this.

As opposed to Word2vec, which treats each word as an atomic entity, fastText represents each word as the word itself, plus a bag of character n-gram, and two special symbols $<$ and $>$ at the beginning and end of words. N-gram means that the size of the sub-word is of length n characters, counting $<$ and $>$ as a character. For example, the representation of the word *where* with n= 3 is:

<where>, <wh, whe, her, ere, re>

It is important to note that the sequence <her>, corresponding to the word *her* is different from the tri-gram her from the word *where*.

When calculating the word embedding, the representation is given by the sum of each n-gram embedding, where each n-gram representation is calculated as previously seen for the Word2vec model.

This model allows sharing the representations across words, thus allowing to learn representation for rare words that are better in comparison with the other models. In the case of fastText, even if words are rare their character n-grams are still shared with other words, having a significant number of neighbors, while for Word2vec a rare word has only a few neighbors.

Moreover, if a word is not present in the vocabulary, its representation is given by the sum of its n-gram vectors. Likewise, the word *aquarium* can be split into

<aq, aqu, qua, uar, ari, riu, ium, um>

As the model encounters the word *aquarius*, it might not recognize it. Of course, the two words share many n-gram parts, so the model can embed *aquarius* near *aquarium*.

Given its robustness to infrequent missing words, the model is also more robust to the size of the training data. It has been proven that adding more data does not always lead to improved results. The proposed approach provides good word vectors even when using very small training datasets [Bojanowski et al., 2017]. This can be seen in figure 2.7. The performance of the fastText model on only 1% of the Wikipedia corpus has a correlation coefficient of 45, which is better than the performance of CBOW trained on the full dataset (43).

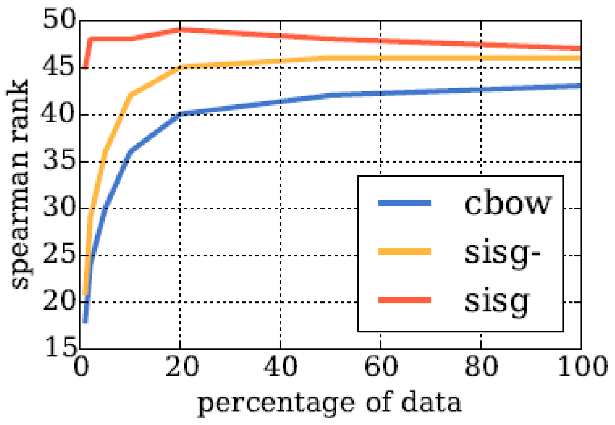


Figure 2.7: Influence of the size of the training data on the performance on a word similarity task (English RW dataset).
Abbreviations are used: sisg : Subword Information Skip-Gram corresponds to the fastText algorithm, sisg- the out of vocabulary words are replaced with zero vectors, cbow : cbow model [Bojanowski et al., 2017, p. 7]

To summarizing, the advantages of fastText over Word2vec are:

- It generates better word embeddings for rare words.
- It can generate embeddings for out-of-vocabulary words
- The algorithm is more robust towards the size of the training data
- It can be faster to learn

These properties are precisely the reason why we chose the fastText model over Word2vec and GloVe.

2.6 Verify the linguistic knowledge

Based on what we explained so far, we can represent each word as a multi-dimensional vector, and be able to extract information from these representations. In doing so, we do not know what each dimension codes for, but we can test practically if the representation works. In fact, it is the only way to test if the model learned word meaning.

Since the quantity of dimensions does not allow representing the vector in a Cartesian plane, to try to visualise the environment, the embeddings have to undergo a dimension reduction, with unavoidable data loss. Figure 2.8 shows an example of vector space obtained by Word2vec. In the figure, we can notice how related words are closer in the space.

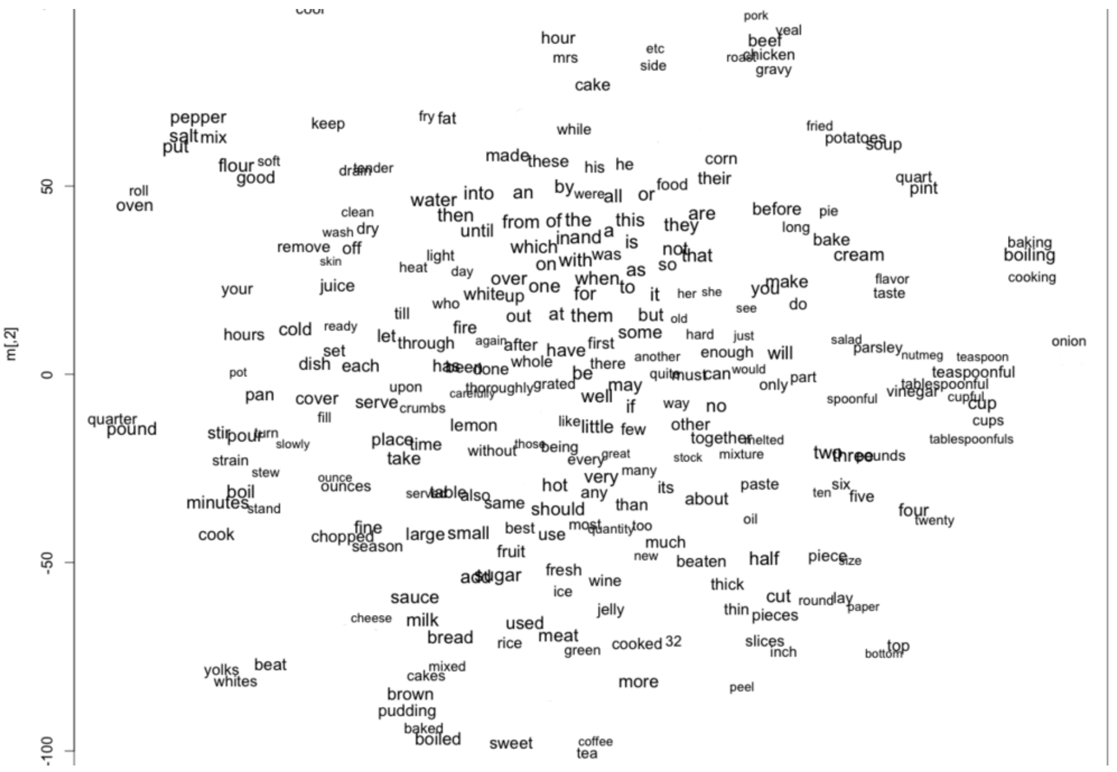


Figure 2.8: Simplification of the vector space. Credits: `https://towardsdatascience.com/mapping-word-embeddings-with-word2vec-99a799dc9695`

We will exploit the graphical example of Alammar [Alammar, 2019]: given a 50-dimensional vector representing the word "king", each value is in a range [-2; 2], where a color from red to blue shows the number in a graphical way (figure 2.9).

Comparing different vectors (figure 2.10), we can notice how words that in-

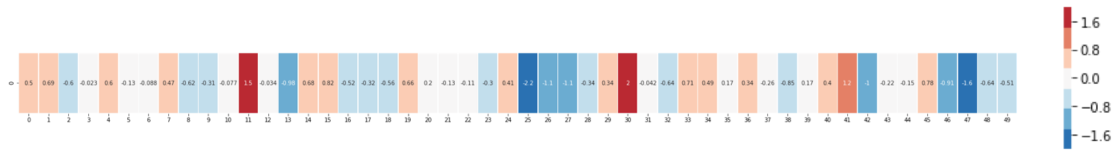


Figure 2.9: Vector of the word "king" in color range [Alammar, 2019]

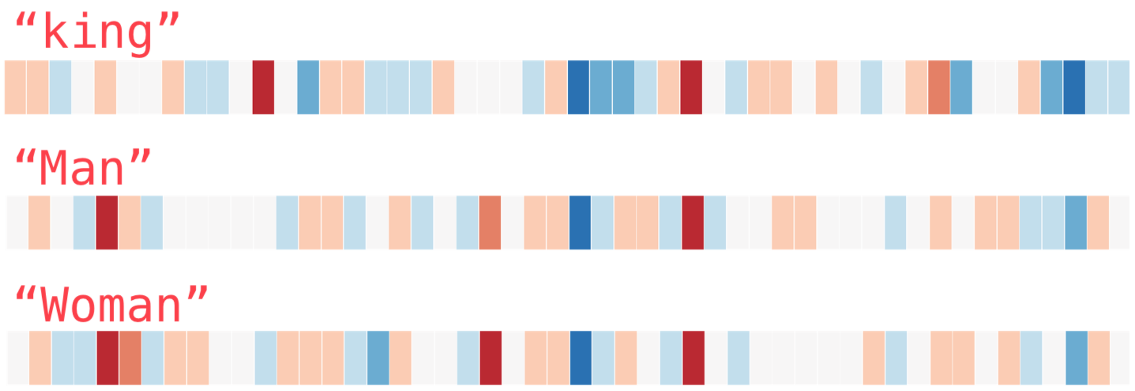


Figure 2.10: Vectors of the words "king", "man" and "woman" in color range [Alammar, 2019]

tuitively are close in meaning correspond to similar vectors. Indeed, the vectors for "man" and "woman" are more alike with respect to the vector of the word "king".

Another way to show the working of the system is the analogy task. In this assignment, we can add and subtract word embeddings and obtain a reasonable result. The most famous example is shown by the formula:

king - man + woman = queen

The fastText tutorial page [4] offers the example with the words:

berlin - germany + france = paris

Again, another common task is finding the nearest neighbors, which are the closing words in the vector space. For example, the 10 top nearest neighbors of the word "asparagus" are:

beetroot 0.812384

tomato 0.806688

horseradish 0.805928

spinach 0.801483

licorice 0.791697

lingonberries 0.781507

[4]FastText tutorial page: `https://fasttext.cc/docs/en/unsupervised-tutorial.html`

asparagales 0.780756

lingonberry 0.778534

celery 0.774529

beets 0.773984

In our case, to verify the linguistic knowledge acquired by the model, we will test it on a linguistic task, namely the colexification phenomenon. We will explain the issue in the next chapters.

Chapter 3

Setting up the research problem

In this chapter we will set up the research problem (section 3.1), and explain why we decided to address it (section 3.3). We will heavily rely on the study of Yang Xu et al. on colexification, written in the paper "Conceptual relations predict colexification across languages" [Xu et al., 2020a], analyzing how they developed their analysis (section 3.2). Lastly, we will also describe the development plan to follow in the next chapters and phases of work (section 3.4).

3.1 Problem

The phenomenon under investigation is **colexification**:

> A given language is said to *colexify* two functionally distinct senses if, and only if, it can associate them with the same lexical form [François, 2008, p. 170]

It is not difficult to find cases that illustrate the situation, for example, the Italian word "dito" means both finger and toe while the English word "uncle" could mean mother's brother, father's brother or aunt's husband.

It is not clear where colexification originates: is it a case of metaphor, metonymy, hyperonymy, analogical extension or other phenomena? Does it derive from a historical change? Is it influenced by the geographical situation of the speakers? François' idea is that if two words colexify in at least one language, the brain has identified a semantic connection between the meanings [François, 2008, p. 172].

Anyway, finding the exact reason why colexification happens and which are the trigger factors is an unresolved issue whose development goes beyond the scope of this project.

Under the assumption of François, two similar concepts are more expected to be conveyed through a single term with respect to two concepts pertaining to completely different fields. For example, the Italian word "dito" conveys two similar meanings: "finger" and "toe", that is "any of the five separate parts at the end of the foot/hand". At the same time, to avoid incurring in communication's misunderstandings, two concepts that are very related in meaning with a high probability use distinguished lexemes (this is the case, for example, of "brother" and "sister" [Regier et al., 2016]). In other words, semantic closeness between the meanings is a necessary but not sufficient condition to have cases of colexification.

The term avoids distinguishing between polysemy, homonymy, vagueness or other similar concepts keeping the mechanism very simple. Specifically, it is said that colexifications *covers* both polysemy and homonymy [Pericliev, 2015, p. 72]. Pericliev defines homonymy as the result of mere chance coincidence or merging of forms of distinct words, while polysemy as the result of real semantic changes, revealing general cognitive processes or culture-specific semantic associations. To distinguish a colexification as a case of polysemy or homonymy is a complex problem, that should be solved individually in each observed case in a specific language. Pericliev proposes two examples to understand the issue. The Modern English words *arms* "upper limbs" and *arms* "weapons" are homonyms because in Middle English they were distinct words: the word for upper limbs with the form *earmes* (from Old English *earm*) and that for weapons *armes* (from Old French *arme*). Some phonological processes converged them to a single modern form, but whether or not these processes would change the distinct word-forms to an identical form is a fact of purely coincidental nature. In contrast, Modern English *arm*, meaning either "upper limb" or "a support (as on a chair) for the elbow and forearm" is a single polysemous word in which the first meaning developed (via metaphor) into the second. For the purposes of this thesis, we will not treat polysemy, homonymy and colexification as three different phenomena, but rather, we will consider both polysemy and homonymy as colexifications.

Moreover, in the definition of colexification it is not specified whether the two words should be synchronous, that is pertaining to the same chronological version of the language. For example, it may happen that a certain word form has a meaning in one age and acquires a new meaning in the following period. Anyway, since we can consider two diachronic varieties as two full-fledged separate languages, it follows that the terms should belong to the same chronological stage of a language. However, this is a personal choice.

To shed some light on the problem and understand where the focus is, we will consider the work of Xu et al. [Xu et al., 2020a]. The approach of Xu and colleagues is to take into account various languages and try to infer linguistic patterns. Indeed, some colexifications have been proven to be more frequent than others (this is the case of "fire" and "flame"). Their consequent hypothesis is that cross-linguistic variation in colexification frequency is non-arbitrary and reflects a general principle of cognitive economy. Specifically, "the observed gradient of colexification probability across languages tracks the strength of conceptual relatedness in language users' minds" [Xu et al., 2020a]. As consequence, meanings that require less cognitive effort to relate should be more likely to be colexified.

The key factor to stress, following these statements, is that it should be possible to predict if and how much a pair of concepts colexify using only their semantic closeness, keeping in mind that this is not the only factor which triggers colexification.

There are many different ways to operationalize semantic closeness, depending on the chosen model. In this case, the semantic closeness is defined as cosine value between the word embeddings in the vector space. We will study how Xu et al. operationalized the similarity and verified the hypothesis.

3.2 Xu et al.'s Approach

3.2.1 Data used

To verify their hypothesis, Xu et al. collected linguistic data. The material used was obtained from the database "Intercontinental Dictionary Series (IDS)" [Borin et al., 2013], which contains 1310 entries of meaning organized into 22 semantic domains and identified by one or a few English words. English is the language used as a reference for annotating meanings in the database. This constitutes a limit but is a necessary step to make meanings comparable. It follows that a concept, expressed in some language by a single word, could be annotated in the database as a multi-word entry. When using English as a meta language, sometimes there is the need of multiple words because English does not lexicalize a single word expression for that meaning (or colexifies them; e.g., the "uncle" example above). For example, the Italian words "svegliarsi" and "banano" are translated respectively into the English terms "wake up" and "banana tree". They decided to restrict the set of examined colexifications to concepts expressible in single word forms in English, at the level of granularity provided by the IDS database. Moreover, since English is the meta-language, the English entries were not considered.

The original database is composed of 453 975 data points where, as previously written, the number of concepts is 1310. The initial number of languages was 329, while the number of families was 60. Some of the languages have identical 3-letter ISO language codes in the IDS, for this reason, they decided to work with the set of languages that have uniquely identifiable ISO codes according to Glottolog [Hammarstrom et al., 2020], leaving 246 languages of 41 language families, 5 climate categories and 5 geographical regions. Xu's team did not remove diachronic varieties, keeping old languages like Ancient Greek, Latin, Old Prussian [Xu et al., 2020b]. The data draws on a subset of the world's language families and has uneven representations from the language families (e.g., Indo-European is more represented than other language families) [Xu et al., 2020a, p. 10].

For each language, they searched for colexifications where two concepts share the same word form. At this point, they had available a data frame of colexifications distributed through linguistic families, climate categories and geographical regions.

Afterward, they calculated the frequency of colexification for each possible concept pair controlling separately depending on family, climate and geography.

For every group, they performed stratified bootstrapping with replacement [1], repeating the bootstrap 1000 times. After this process, they obtained an aggregated estimate of the colexification frequencies across all language families. [Xu et al., 2020b, p. 5-6].

In addition to the stratified bootstrapping procedure, they also used Monte Carlo simulation [Everett et al., 2015]. Such a procedure allows only a single language to be sampled from each of the 41 language families so that families over-represented in the IDS database (such as Indo-European family) would be treated equally in sample size as those families with fewer languages. As for the stratified bootstrapping, they generated 1000 rounds of Monte Carlo sampled colexification matrices for analyses of all the language families [Xu et al., 2020b, p. 8].

3.2.2 Measures of conceptual association

They considered some variables of conceptual relatedness and used these variables to perform three tasks:

- Show that the variables can distinguish between colexification patterns that are attested and those that are unattested
- Show which variable best predicts which concepts are more frequently colexified
- Show how the degree of associativity recapitulates the gradient of colexification frequency

The considered variables are associativity, similarity, usage frequency, concreteness and emotional valence. Associativity is the expected probability with which a person will associate one word with another word (one or a set of seed words is presented to a participant who responds with the first word that comes to mind, obtained using Human Brain Cloud HBC [2] association game and Uni-

[1]The bootstrap is a method for estimating the distribution of an estimator or test statistic by re-sampling one's data [Horowitz, 2001]. With replacement mechanism, one entry can be selected more than once.

[2]`https://www.humanbraincloud.com/`

versity of South Florida USF [Nelson et al., 1998] word association esperiment. Similarity is given by the standard cosine distance between word embeddings of two concepts (i.e. their distributional similarity), obtained by a word embedding model. Usage frequency is the average frequency obtained by the frequency of words in the Google Book English corpus [Michel et al., 2011]; while concreteness and emotional valence are values obtained by human ratings [Brysbaert et al., 2013]. For the present purposes, we will only pay attention to similarity because it is the one obtained by a computational linguistic model.

To operationalize the concept of similarity, the team opted for scalable vector-based models, in particular, Word2Vec embedding [Mikolov et al., 2013a,Mikolov et al., 2013c] in its pre-trained version [3].

In doing so, for a given pair of concepts, they took the standard cosine distance between the word embeddings of those concepts, obtaining a numerical value.

3.2.3 Tasks implementation

To solve the first task, for any pair of the IDS concepts, the logistic regression model predicts whether it should be colexified or not in any of the languages. In other words, they reduced the problem to binary classification. The data was split into two groups: attested, defined as pairs of colexified concepts attested in at least two languages, and null, defined as pairs of concepts that are attested for one or none of the languages in the set. They applied a 10-fold cross-validation[4], where 10% of the data was the test set, keeping the remaining 90% as training data. Because there are many more pairs in the null group than in the attested one, they balanced the data by sampling from the null group and equating the sample size to that of the attested group. To evaluate the variables, they per-

[3]`https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit`

[4]Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it.

In k-fold cross-validation, the original sample is randomly partitioned into k equal size sub-samples. Of the k sub-samples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k sub-samples used exactly once as the validation data [Vanschoren, 2021].

formed a logistic regression analysis.

The variables that capture conceptual relations (associativity and similarity), performed well above chance (50%). In particular, similarity obtained a value near 80% of predictive accuracy. Figure 3.1 shows how the measures performed.

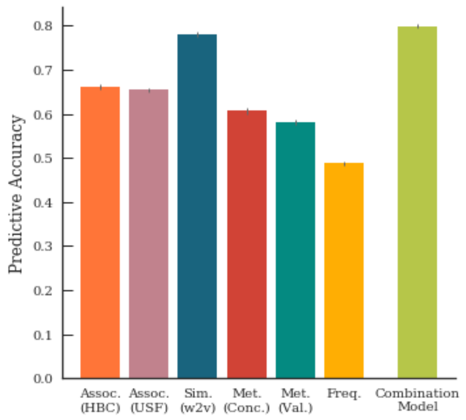


Figure 3.1: Cross-validated predictive accuracies in classifying colexified versus non-colexified sense pairs from individual variables and variables in combination, using logistic regression. Abbreviations are used for the following variables: Assoc : Associativity, Sim : Similarity, Met. (Conc.) : Metaphoricity (Concreteness), Met. (Val.) : Metaphoricity (Valence), Freq. : Frequency [Xu et al., 2020a].

The second set of analyses goes beyond binary predictions. Specifically, they evaluated how well each variable accounts for cross-linguistic variability in how frequently pairs of concepts are colexified.

To evaluate the results, they correlated the variables against colexification frequencies, i.e. the number of times a pair of words is a colexification in any of the languages; showing how among all, associativity best correlates with the cross-linguistic frequency of colexification. Similarity is the second-best variable, with a Spearman's p-value of 0.22[5]. Figure 3.2 shows the correlations between the measures and the colexification frequencies in the family-controlled set.

Finally, they examined the relationship between the orders of association of two concepts and the cross-linguistic frequency of their colexification. Until now, the team studied a direct association of two concepts, but it is conceivable that

[5]Citing Wikipedia: "In statistics, Spearman's rank correlation coefficient or Spearman's p, is a non-parametric measure of rank correlation (statistical dependence between the rankings of two variables). It assesses how well the relationship between two variables can be described using a monotonic function."

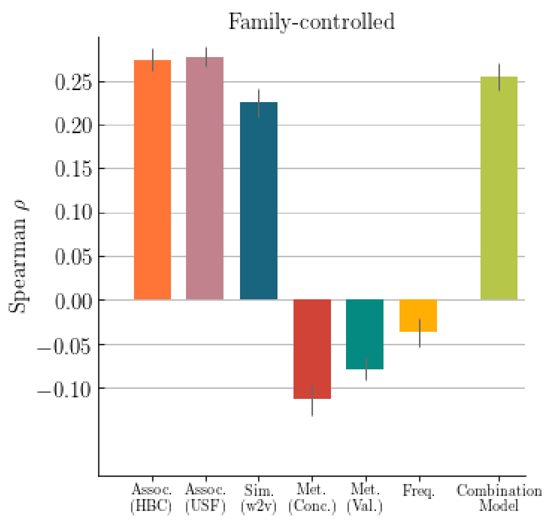


Figure 3.2: Results of variable correlations with colexification frequencies across languages. Abbreviations are used for the following variables: Assoc : Associativity, Sim : Similarity, Met. (Conc.) : Metaphoricity (Concreteness), Met. (Val.) : Metaphoricity (Valence), Freq. : Frequency [Xu et al., 2020a].

colexification may also happen between indirect association. The described example is a possible colexification between "bird" and "sap" where the connecting term is "tree". Under this assumption, pairs that are more remotely associated should be less likely to enter the lexicon, compared to those that are more directly associated. In this case, given that the only variable in use is associativity, we will not delve into the development. In a nutshell, the frequency of colexification shows a monotonic decrease as the order of association increases.

3.3 Results and discussion

In all three tasks, the results lead support to the initial hypothesis of a positive relationship between semantic closeness and the emergence of colexification.

Through the whole paper, Xu et al. established the importance of conceptual relations in colexification processes. The three phases of work suggest that conceptual relations are central to predicting whether a pair of concepts will be colexified in a language and how frequently they will tend to colexify. The outcome is a reflection of cognitive economy, verifying the initial statement of the key constraint of conceptual relatedness in colexification processes.

The work of Xu's team was successful: they were able to test the hypothesis and verify the theory. There are different ways to operationalize semantic closeness, among which. Xu et al. look at associativity and word-embedding based similarity. However, their choices instantiate just one of many ways of codifying associativity and similarity. It is an open question whether other ways of operationalizing distributional similarity will fare better or worse. In replicating their work, we will try other ways to operationalize semantic relatedness. Our focus is on Natural Language Processing models, for this reason, we will only concentrate on the variable of distributional similarity using another model (not Word2vec but fastText) and seeing how the results change. Additionally, besides the pre-trained version of the model, we will train our own models, changing the parameter of window size whose effect is on the dimension of the context window.

In conclusion, it is necessary to highlight that the main goal of the project is not to replicate or outperform the studied work, even if we do perform a detailed replication of an aspect of their analysis. The real purpose is to analyze the colexification phenomenon through a computational model of Natural Language Processing, carrying out all the phases of work (from the data collection to the discussion of the results), taking care of an aspect that in the studied paper was not touch: model training and hyper-parameters selection, with a particular emphasis on the dimension of the context window.

3.4 Development plan

The following section describes the phases of work, foreshadowing the steps that will be carried out in chapters 4 and 5.

3.4.1 Data used

To start to replicate Xu's team work, we need to collect linguistic data. The chosen database is CLICS3 [Rzymski et al., 2020], which aggregates multiple data sets, including the IDS database used by Xu's team. In this way we are able to follow their reasonings, expanding the amount of data used. To give the idea, the number of considered linguistic varieties improved from 246 to 2990. The CLICS database structure will contain linguistic forms, expressed concepts, linguistic varieties, linguistic families, but no colexifications. To obtain a set of colexifications it will be necessary to pre-process the available data.

We will dive into the data understanding part, laying the foundations for the data cleaning step. After the data cleaning phase, we will prepare the linguistic concepts to obtain a colexifications data frame.

Following the logic argued by Xu et al., we will perform a bootstrap, reducing the number of bootstrapped samples from 1000 to 100. After that, we will calculate the probability of a concept pair to be a colexification, in a similar fashion as Xu and colleagues did to compute their frequencies.

3.4.2 Measure of conceptual association

We will use a model of distributional similarity as well. The idea is that it will be possible to predict that concepts that are used in more similar linguistic contexts should be more frequently colexified across languages, that is, colexified with a higher probability. The measure of conceptual association that we will take into consideration is similarity, obtained by the fastText model (contrary to Word2vec model used by Xu's team). For the first phase of analyses, the model in use will be the pre-trained one [Grave et al., 2018].

Changing the model differentiates the two works in the biggest way. In fact, if we had decided to continue working with Word2vec, the innovation points would have been too limited. FastText is undoubtedly a less studied and known model but has few positive aspects that separate it from other models. The two main being its speed in training (hence the name) and its ability to compute word representations for words that did not appear in the training data [Bojanowski et al., 2017]. Using the fastText model allowed me to study a less used model, able to grasp sub-word information. By the way, the fact that it is not famous as the Word2vec model had some side effects. For example, the availability of documentation is shallow with respect to Word2vec, resulting in more challenging research for the model structure and understanding of the mechanism. The phase of troubleshooting was naturally more demanding, having at disposal only a few examples of use. For the same reason -the lack of examples-, the number of decisions to take was greater than the one we would have taken when using a very popular model. On the other hand, the speed of training got us designing a phase of model training from the beginning, making the whole project more attractive and stimulating. In addition, using a less known model made me develop several job-oriented skills such as problem-solving, decision making and autonomy.

3.4.3 Predictive steps

We will propose few questions we want data to answer, namely:

- How well can the cosine similarity values help us in predicting whether two meanings are highly colexified across different languages?
- How well can the cosine similarity values help us in predicting whether two meanings are a colexification at all?

As Xu et al. did, we will set up the problem in a binary manner, being able to use the logistic regression model. The algorithms that can solve a binary classification problem are several, to name a few Naive Bayes, Decision Tree, Support Vector Machine. Anyway, the Logistic Regression model is the baseline regarding binary classification and is the one that combines high-performance results with a small number of features. Indeed, in our case, we will only use the cosine similarity values to obtain a predictive result.

The three main metrics used to evaluate a classification model are accuracy, precision, and recall. Accuracy is defined as the percentage of correct predictions for the test data.

$$Accuracy = \frac{correct\ predictions}{all\ predictions} \tag{3.1}$$

Precision is defined as the fraction of relevant examples (true positives) among all of the examples which were predicted to belong in a certain class.

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{3.2}$$

Recall is defined as the fraction of examples that were predicted to belong to a class with respect to all of the examples that truly belong in the class.

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{3.3}$$

Accuracy is particularly informative when the classes are evenly distributed. If the classes are unbalanced, accuracy could lead us to think we have a magnificent model when maybe that is not the case. For instance, if there is only one case of colexification but 999 negative cases, then the model could generalize that all the values are negatives. This would lead to an accuracy of 99,9%. Great, isn't it? The downside is that the model has not learnt anything about the problem we were interested in. We could say that the performance is splendid, but what if that single missed prediction is the one we care most? Precision and recall, on the other hand, both verify the model has actually learned something, checking both for overlooking and misclassification [Jordan, 2017]. In the example, both precision and recall values would have been 0. In other words, precision and recall are more complete measures, but one does not include the other. Which one to choose? F1 score comes in support of seeking a balance between precision and recall.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.4}$$

When we will evaluate the results obtained from the logistic regression, we will precisely use the F1 score value.

3.4.4 Training the models

Corpus

We will likewise perform a phase of model training. This stage will begin with the download of the training corpus (selected as English Wikipedia), followed by its cleaning according to the kind of data expected by the model. The Wikimedia Foundation provides free and large text data that are expected to be reliable and in (hopefully) correct English, giving its reliance on crowd control. However, the corpus needs to be in plain text format, which means deprived of images, labels, notes, comments, or other metadata.

Architecture

When training the fastText model, we can choose between two approaches: Skip-gram or CBOW. The topic has already been studied in section 2.3, but will be briefly reviewed here for convenience.

Both are architectures to learn the underlying word representations for each word by using neural networks. In the CBOW model, the training objective is to combine the representations of surrounding words to predict the word in the middle. Similarly, in the Skip-gram model, the distributed representation of the input word is used to predict the context [Kulshrestha, 2019, Mikolov et al., 2013b]. In both cases, the models involve a specific context (i.e. the surrounding words) whose dimension change dependently on the value of the hyper-parameter, considering different number of neighbouring words. To illustrate the difference, consider the following example from the fastText webpage:

Poets have been mysteriously silent on the subject of cheese

where the target word is *silent*. A Skip-gram model tries to predict the target using a random close-by word, like *subject* or *mysteriously*. The CBOW model takes all the words in a surrounding window, like *been, mysteriously, on, the*, and uses the sum of their vectors to predict the target [Facebook Inc., 2020b]. The model architectures of these two methods are shown in figure 3.3.

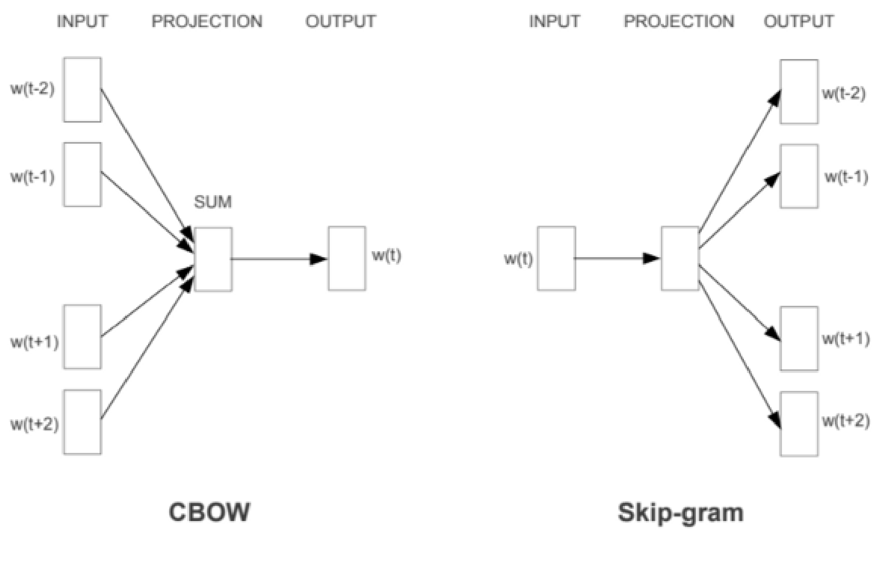


Figure 3.3: Graphical representation of the CBOW model and Skip-gram model [Mikolov et al., 2013b]

In practice, Skip-gram gives better word representations when the monolingual data is small, while CBOW is faster and more suitable for larger datasets [Mikolov et al., 2013a]. The Skip-gram and CBOW models can be trained on a large corpus in a short time and tend to learn very similar representations for languages [Mikolov et al., 2013b]. To decide which model to use, we will try both and choose the one with the higher performances.

Window size hyper-parameter

In the model training, we will probe the parameter of window size, keeping all the other hyper-parameters with their default values. The window size parameter represents the dimension of the context window in the training phase. Our goal is to see if the dimension has an effect on the logistic regression performance and in that case, which is the best value to set.

Previous studies have already detected that the window size is a crucial factor that directly affects the word vector representations. This is because distributional semantic models represent words through real-valued vectors of fixed dimensions, based on the distributional properties of these words observed in large corpora [Lison and Kutuzov, 2017]. Larger windows are known to induce embeddings that are more topical or associative, improving their performance on analogy test sets, while smaller windows induce more functional and synonymic models, leading to better performance on similarity test sets [Goldberg, 2017].

Jurafsky and Martin [Jurafsky and Martin, 2000] state that shorter context windows tend to lead to representations that are a bit more syntactic, since the information is coming from immediately nearby words. When the vectors are computed from short context windows, the most similar words to a target word tend to be semantically similar words with the same parts of speech. When vectors are computed from long context windows, the highest cosine words to a target word tend to be words that are topically related but not similar.

Lapesa et al. focused on the paradigmatic vs. syntagmatic relations that hold between words. Paradigmatic relations hold between words that occur in similar contexts but that do not co-occur between them (for example, synonyms *frigid - cold* and antonyms *cold - hot*. Syntagmatic relations hold between words that co-occur and therefore exhibit a similar distribution across contexts. They tested the model with a window size of dimensions 1, 2, 4, 8, 16. Their results show that a very small context window (i.e., one word) is sufficient for all paradigmatic relations [Lapesa et al., 2014].

Another similar experiment was conducted by Lison and Kutuzov who discovered that for a lexical similarity task narrow window size performs best, while for analogy task larger context windows are usually beneficial (but not always!) [Lison and Kutuzov, 2017]. They do not comment exactly which is the number to choose, but they prove to have tested the model on window sizes 1, 2, 5, and 10.

Considering these studies, we can predict that the most accurate window size to choose for our purpose will be narrow, but we cannot predict exactly which size. This is because we assume colexification depends on similarity between words. A model which best capture word similarity values, best predict cases of colexifications. Looking at the previous studies, we are aiming to the best paradigmatic and lexical similarity results.

Chapter 4

Implementation and evaluation of the project

Now that the development plan has been set up, the following chapter will explain the implementation of the project, with its evaluation. In this chapter we will only use the pre-trained version of the fastText model, leaving the customized versions of the model for the next chapter.

The chapter will begin with a data management phase. Section 4.1 will explain how to obtain the data, section 4.2 will look deeper into the data, trying to understand its structure and meaning, while section 4.3 will clean the data, keeping only relevant information.

Section 4.4 will explain the data management steps that include the creation of the colexification data frame, the bootstrap process, and the calculus of probabilities associated with each meaning pair.[1]

Section 4.5 will describe how we obtained the cosine similarity values for the colexified meaning pairs using the pre-trained fastText model.

Section 4.6 will analyze the obtained probabilities and cosine similarity values and study their distributions.

Section 4.7 will use the Logistic Regression model to answer two questions:

- *How well can the cosine similarity values help us in predicting whether two meanings are highly colexified across different languages?*

[1]The data cleaning, bootstrapping and probabilities computations were coded with Lucía Pitarch Ballesteros.

- *How well can the cosine similarity values help us in predicting whether two words are a colexification at all?*

Finally, section 4.8 will discuss the results obtained from the Logistic Regression with this first pre-trained version of the model.

4.1 Data Collection

The data used is obtained from the online Database of Cross-Linguistic Colexifications, first published in 2014 and available for use in its third edition (CLICS3) [Rzymski et al., 2020]. The CLICS project aims to provide free and transparent linguistic data, allowing to investigate colexifications through global and areal semantic networks. It is therefore the best source to collect the data for our purpose.

The CLICS project aggregates various databases, allowing us to dispose of a huge amount of data. In doing so, the CLICS team uniformed the entries establishing a standard way to record a word's meaning.

As we will see in the next section, we do not have a ready-to-use list or table of colexifications, which can be built by processing the data (see section 4.4.1).

The pre-processed CLICS data is readily available from a SQL database, from the CLICS3 page on Github[2]. Our starting point is given by its conversion into a more convenient CSV format.
The code performing this operation is in the file *00_DataCollection.ipynb*.

4.2 Data Understanding

The raw file that contains the CLICS data is composed of 1 390 594 rows and 16 columns. Its structure is well depicted in table 4.1.

As anticipated, the data frame collects data from different databases. This is why we have dataset IDs as well as columns that may sound redundant but which are necessary to try to standardize the information (for example, *Form* vs

[2]https://github.com/clics/clics3

Table 4.1: Explanation original raw data frame

Column	Description	# values bc	# NaN	# values ac
dataset_ID	name of the dataset	30	0	30
Form_ID	unique identifier of the form	1 390 584	0	1 202 817
Form	the concept, expressed in its language	913 591	87	802 148
clics_form	the form, written in a normalized manner	706 169	441	623 902
gloss_in_source	the concept, expressed in English, as written in the original database	7260	64 676	6108
Concepticon_ID	unique identifier of the concepticon	2919	0	2396
Concepticon _Gloss	the concept, expressed in English, in a normalized way	2919	0	2396
Ontological _Category	general category the concept refers to	6	0	5
Semantic_Field	the subject the concept refers to	24	108	24
variety	language variety	3050	0	2990
Glottocode	unique identifier of the linguistic variety [Hammarstrom et al., 2020]	2279	19627	2238
ISO639P3code	linguistic code	1845	206 325	1807
Macroarea	macroarea of the variety	6	87 845	6
Family	linguistic family of the variety	201	22 090	201
Latitude	latitude value	1850	226 992	1822
Longitude	longitude value	1835	226 992	1803

# values bc : number of unique values before cleaning; # values ac : number of unique values after cleaning

clics_form). Specifically, the column *Form* lists the word as it's written or pronounced in the original language. It's the case of the word "face" present in Italian both with *faccia* and as its phonetic transcription *fatt͡ʃa*, even though the correct way to write the word is "faccia". As consequence, the column *clics_form* transforms the above entries in *faccia* and *fattsa*. As we see, the symbol t͡ʃ has undergone an orthographic change. The phenomenon happens other times, for example in the Hruso Aka Jamiri language, ʃə and sə were both transcribed with the clics form *s@*. The two words, having two different meanings, will be later considered a colexification, even though we cannot know if this is a false positive case given by the unification. We take into account this issue but we continue with the work using the *clics_form* column as reference for the colexification detection [3]. This is a common trade-off when doing large scale computational work: the amount of data is bigger but the data is more noisy being collected by various sources. The grouping of different single databases also explains why we do not have all the entries for all the rows in the table (among others, *Latitude* and *Longitude* show a big amount of missing entries).

The number of unique entries between *variety* and *Glottocode* columns is different because of the way the two values are conceived. The variety is an easily readable way to define a language (e.g. Italian), while the glottocode is a unique code identifier that refers to that language using the Glottolog standard [Hammarstrom et al., 2020] (e.g. ital1282).

Since there is no perfect definition of what a language is, we are using the state of the art (Glottolog) as our best approximation. This means that sometimes the variety column assigns different names to the same glottocode, given the fact that there can be multiple names or super/sub classifications associated with the same language. For instance, "Dammai Dibin" and "Dammai Rurang" both are associated with miji1239. This is likely because they are local variants that Glottolog does not consider two distinct languages. In other cases, it is because languages were written down in different ways across data sets, keeping in mind that the CLICS data is an aggregation of multiple heterogeneous data sets. For example "Latin" and "latin-std" both refer to lati1261. To summarize, we should use the

[3]Thanks to Lucía Pitarch Ballesteros for the issue detection.

Glottocode column as the main reference for a language and use the *variety* column only if we need to filter or search for human-readable strings.

Some columns contain null values, as shown in table 4.1. We can notice that between the most important columns (which are *clics_form*, *Concepticon_Gloss*, *variety*, *Glottocode*, *Family*), *Family* and *Glottocode* contains a big amount of null values (respectively 22 090 and 19 627).

Detaching from the table and looking deeper into the data, we notice that the data frame does not contain duplicated rows.

Regarding the distribution of glottocodes per family, most families only have few members, with a mean of 11 and a standard deviation of 42. This means that only a few families attest a number of languages way bigger than the majority of families. There are more than one family with only a single member, while the largest family (Austronesian) has 395. The first percentile is 1, the second is 2, and the third is 4. Figure 4.1 shows the distribution of languages per family.

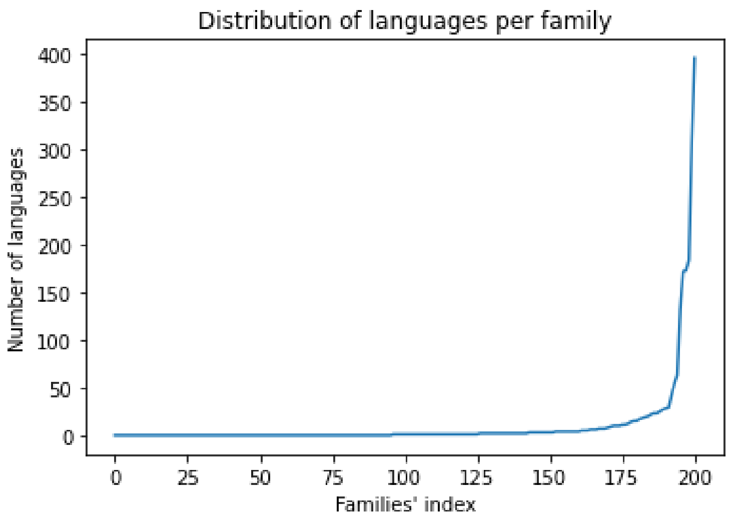


Figure 4.1: Distribution of languages per family.

The distribution of concepts per family, instead, is the following: the smallest family (Pahoturi) has 4 concepts while the largest (Indo-European) has 181 678, with a mean of 6808. The standard deviation is 24 540, being the first percentile 201, the second 895 and the third 1776. This means that a huge amount of families attest only a few concepts, while the biggest amount of concepts is attested in the most studied linguistic families. This distribution is depicted in figure 4.2.

We noticed some entries from different diachronic varieties, for example, "An-

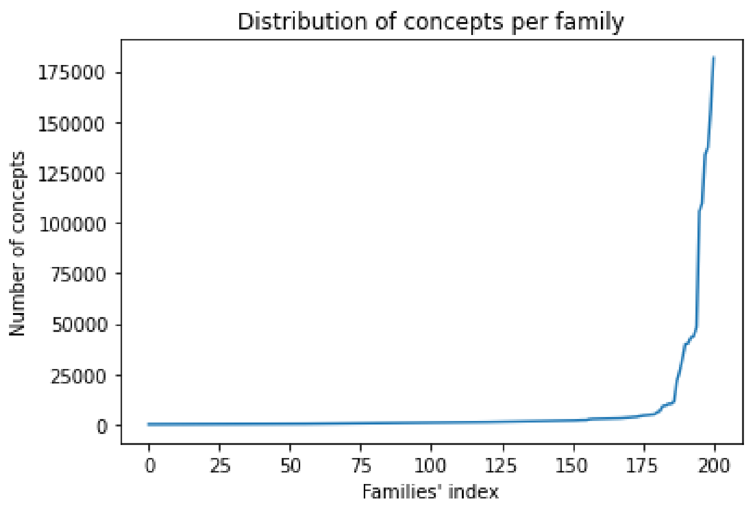


Figure 4.2: Distribution of concepts per family.

cient Greek", "Old Prussian", "Middle Welsh", "Modern Armenian", plus the classical language Latin. As we will see in the next section, these varieties will be discarded. The numbers of rows that contain one of the diachronic keywords in the variety column are shown in table 4.2.

Table 4.2: Count of diachronic varieties

Key word	**Number**
Ancient	4164
Old	17566
Classical	663
Middle	6096
Modern	4256
Proto	2757
Gotic	1198
Latin	3621

908 Concepticon glosses contain multiword forms, parentheses or the disjunction "or". This is the case of "HAIR (HEAD)", "HOW MANY" and "BELOW OR UNDER". As we will see in the next section, these forms could create problems in the following phases of work and need to be treated accordingly.
The code inspecting the data is in the file *01_DataUnderstanding.ipynb*.
Table 4.3 shows as an example an entry of the data frame.

Table 4.3: *Example row of the CLICS data frame*

dataset_ID	ids
Form_ID	170-1-100-1
Form	mondo
clics_form	mondo
gloss_in_source	world
Concepticon_ID	965
Concepticon_Gloss	WORLD
Ontological_Category	Person/Thing
Semantic_Field	The physical world
variety	Italian
Glottocode	ital1282
ISO639P3code	ita
Macroarea	Eurasia
Family	Indo-European
Latitude	43.0464
Longitude	12.6489

4.3 Data Cleaning

As seen in the previous section, the data frame contains words from different diachronic varieties. For consistency with the colexification definition (see section 3.1), we decided to discard entries of diachronic varieties, leaving only modern languages. The entries whose variety contains the words *Old, Middle, Classical, Ancient, Proto, Gothic*, as like Latin, will not be considered.

As anticipated, English is the meta-language used to express the concepts in a uniform way, but it is also the language used to train the model and to base our proxies of semantic similarity on. If we took into account English entries, we would create a bias in the model, using the training language as a metric to evaluate testing data. Therefore, we decided to not consider its rows as well, deleting 9564 entries.

Accordingly, the first phase of the cleaning code consists of tokenizing the variety column to better identify the languages that contain the aforementioned unwanted characteristics. The rows with these keywords have been flagged in a new column and consequently deleted. The deleted rows are 40 417.

The *Concepticon_Gloss* column contains all the glosses we will use in the fastText model. This column contains multiword entries, e. g. "ALLOW OR PERMIT", "BE ABLE", "ANIMAL (CLASSIFIER)". To understand whether to consider or not these entries, we first checked how the fastText model worked. The pre-trained model used a single-word segmented corpus [Grave et al., 2018]. This could could lead us to think to not query it with multi-word glosses. On the contrary, the model do provide a specific function to obtain a single vector from a line of text: *get_sentence_vector()* [Mohr, 2021a], [Facebook Research, 2019]. As consequence, we can consider multi-word entries without incurring into errors. Furthermore, if we call the single word function and the multi-word function on the same word (*get_word_vector()* compared to *get_sentence_vector()*), the cosine distance between the vectors obtained is 1.0, showing how the multi-word mechanism works exactly as the single-word one.

Another issue is that some parenthetical specifications, e.g., "PLAY (VERB)", or entries containing *OR*, are not concept-related per-se. They account for more than one concept or specify it too much, removing part of the meaning from the concept. It is unclear how to interpret a vector derived from these surface forms. Multi-word expressions, instead, provide the meaning of a single concept and could be considered, in a way, a *lack* in the English language (given the fact that one word is sufficient for the expression in other languages).

We expect the model to return random or unreasonable vectors for the first two cases, and to work better for the latter. For this reason, the entries containing OR or parenthesis have been deleted, removing 147 351 rows. This operation has been done flagging each row that contained "OR " or "(" and deleting them.

As we saw in the previous section, there are no null values in the *Concepticon_Gloss* column, so we do not need to take counter measures.

Lastly, it is recommended to convert all uppercase into lowercase as a good practice to normalize the text [Bhattacharjee, 2018], although the model works in both cases. This is why we transformed every concepticon gloss in lowercase,

saving the modified data frame in a CSV file.
The code of this phase is in the file *02_DataCleaning.ipynb*.

After the data cleaning, the number of linguistic families we are working on is 201, while the number of linguistic varieties is 2990 (check table 4.1 to see all the values).

4.4 Data Preparation

4.4.1 Create colexification data frame

Now that we have a cleaned version of the data, we want to permutate all the concepticons, obtaining a colexification data frame. To do so, we first select only the relevant columns (*clics_form, Concepticon_ID, Glottocode, Concepticon_Gloss, Family, variety*), removing the duplicated rows and those that contained null values. The number of rows in this data frame is 1 148 664.

This intermediate data frame was joined with itself through an *inner_join* operation on the columns *clics_form, Glottocode, Family, variety*. In this way we obtain a data frame where, in a linguistic variety, a certain clics form has two concepticon glosses, resulting in a colexification entry. The cases where the concepticons were duplicated or permutated have been removed. The number of rows of this new data frame is 124 187 for a total of 56 415 unique colexification word pairs. For the record, the number of families we are working with has lowered to 181 with a distribution of colexifications per family that reflects the previous ones (see figure 4.3).
The code performing this operation is in the file *03_CreateColex.ipynb*.
Table 4.4 shows a line of the data frame as an example.

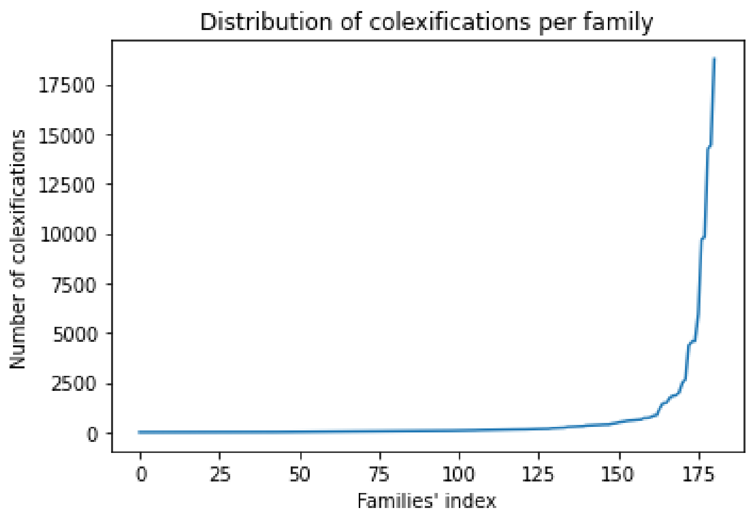


Figure 4.3: Distribution of colexifications per family.

Table 4.4: Example row of the colexification data frame

clics_form	uomo
Concepticon_ID.x	683
Glottocode	ital1282
Concepticon_Gloss.x	person
Family	Indo-European
variety	Italian
Concepticon_ID.y	1554
Concepticon_Gloss.y	man

4.4.2 Bootstrap

Citing J. L. Horowitz, "the bootstrap is a method for estimating the distribution of an estimator or test statistic by resampling one's data" [Horowitz, 2001, p. 3161]. It consists of sampling the data to create a bootstrapped data distribution to apply all the computations on. The method is proven to reduce biases and errors, approximating the original distribution.

In our case, we wanted to select a specific number of varieties for every language family, creating a new colexifications data frame to base our calculus on. The number we chose is the counting of varieties of a specific family, allowing a replacement mechanism: each variety can be selected more than once.

To do the bootstrapping, we created a data frame with only *Family* and *Glot-*

tocode columns, removing the duplicated rows. For every family, we chose *n* random varieties, where *n* is the number of varieties available in that family, in a replacement manner, creating a list of desired varieties. So for example, for the family Mayan (which has two varieties whose glottocodes are: kekc1242 and tzot1259), we will select two varieties, and the varieties selected can be repeated, meaning that we could pick tzot1259 twice. Secondly, for every variety in the list, we took all its colexifications and put them into another data frame.

Since we select all the colexifications for all the varieties considered, one can ask if this is truly useful. Despite the fact that the amount of data considered is completely comparable with the original one, after the bootstrap the data we are going to work on has undergone a shuffling. If we repeat the process another time, it is expected that the size will be almost unchanged, even though the considered observation may vary, consequently changing the calculated values (we will examine this part in the next section). By implication, we are faking to have more data than the available one, making the calculated values more reliable on an average scale.

We decided to perform the bootstrap 100 times, obtaining 100 sample data frames.

4.4.3 Calculating Probabilities

Every bootstrap returns a data frame of all the colexifications for the randomly chosen varieties. From this data frame, we grouped depending on the linguistic family and we counted the number of varieties that have the same specific colexification. This number has been then divided by the total number of varieties for that family, meaning, the maximum possible number of varieties in which the colexification could appear. This gave us the probability of finding a specific colexification within a family. The process balanced the data relative to the language family: a family that has 1000 members contributes the same amount of (weighted) information than one that only has 1 member.

Following the case of Mayan already used in the previous paragraph, when the bootstrap takes the colexifications of both varieties, the pair *(air, wind)* colexifies in both, resulting in 1 as a probability value. The pair *(alone, only)*, instead, only colexifies in one of the varieties, thus the value is 0.5.

Subsequently, we grouped only depending on the colexification, summing up all the probabilities and dividing by the total number of families, obtaining the probability of finding a specific colexification among all the families. In this case, if the value is 1 it means that all the families show that colexification, the lower the value, the less attested that colexification is.

That is to say, with this process, we calculate a value that represents how probable it is that a specific pair of words is attested as colexification in any of the language families taken into account.

To summarize, for every bootstrapped data frame, we choose the varieties, we take every colexification of those varieties, we calculate the probabilities and save all the obtained probabilities in a list. Given that we repeat the steps for every bootstrapped data frame, this procedure allows us to have 100 computations of the probabilities (one per table).

Each cycle consisting of the bootstrap and the calculus of probabilities, lasted approximately 1 minute and 40 seconds, with a total time of 2 hours and 48 minutes.

The code performing both the bootstrap and the calculus is in the file *04_BootstrapProbabilities.ip*

4.5 Getting cosine similarity values

For this first phase of work, we used the English pre-trained version of the fastText model [Grave et al., 2018]. The model has been trained on Common Crawl and Wikipedia using the hyper-parameters shown in table 4.5.

Table 4.5: Main hyper-parameters of the pre-trained fastText model.

Dimension	300
Epoch	1
Learning rate	0.05
Loss function	skipgram negative sampling
min Count	5
Model	CBOW
window size	5

To compute the cosine similarity values, we load the colexifications data frame and, row by row, obtain the vector of the two concept words using the model function (*get_sentence_vector*); if one of the vectors does not exist, the default value for the vector assigned is 0 and the word is listed in a file. The default value 0 is a meaningless number used only to avoid code-breaking while keeping track of the not computed vectors would help us in a debugging phase. We have to keep in mind that the fastText model can manage out of vocabulary words, returning a vector even for a word that was not in the corpus. This means that, ideally, the described possibility never happens, but anyway, the aforementioned code constitutes a sound mechanism to avoid any kind of error. When querying the model with the vector, the only columns we care about are *Concepticon_Gloss_x* and *Concepticon_Gloss_y* because they are the ones that convey the meaning in the English language.

At this point, we calculate the cosine similarity using the two vectors and applying the dot product definition: let a and b two vectors, θ their angle

$$cos(\theta) = \frac{a \cdot b}{||a|| \cdot ||b||} \tag{4.1}$$

Since our definition of cosine similarity depends on the dot product, the range the cosine values can address is [-1; 1], given -1 as negative correlation (completely opposite vectors), 1 as positive correlation (identical vectors), and 0 as lack of correlation [FEL, 2020].

For example, in the first row of the colexifications data frame, we take the two concepts (gold, bamboo), obtain their vectors and calculate the cosine similarity as the cosine between the vectors' angle (value 0.25).

The code loading the model and calculating the cosine similarity values is in the file *05_gettingCosines.ipynb*.

4.6 Analyses

4.6.1 Study of the probabilities

To analyze the distribution of probabilities we grouped together all the results obtained by the bootstraps, calculating the average probability for every colexification.

Overall, we consider 56 415 colexifications on the total 100 bootstrapped data frames, with a mean of 40 980 colexifications per bootstrap.

The values of the probabilities among the colexifications are extremely low for most of the pairs: the mean value is 1.5×10^{-3} with a standard deviation of 5.4×10^{-3}. The minimum value is 2.1×10^{-5} for the pair *(dust, mosquito)* while the maximum is 0.19 for the pair *(moon, month)*.

As we can see from figure 4.4, the values show a logarithmic distribution. The majority of colexification has a low probability value, being only the 2.3% of them to have a value higher than 0.01. This is expected. Colexification is an infrequent phenomenon. Consequently, frequency rates computed across languages are very low. Notwithstanding, we can still see that some colexifications are much more frequent than others, relatively speaking.

Figure 4.5 shows a histogram with the distribution of probabilities. Again, we see a drastic split in the values: 99% of colexifications are between 0 and 0.0195, while the remaining 1% covers the whole range of probabilities.

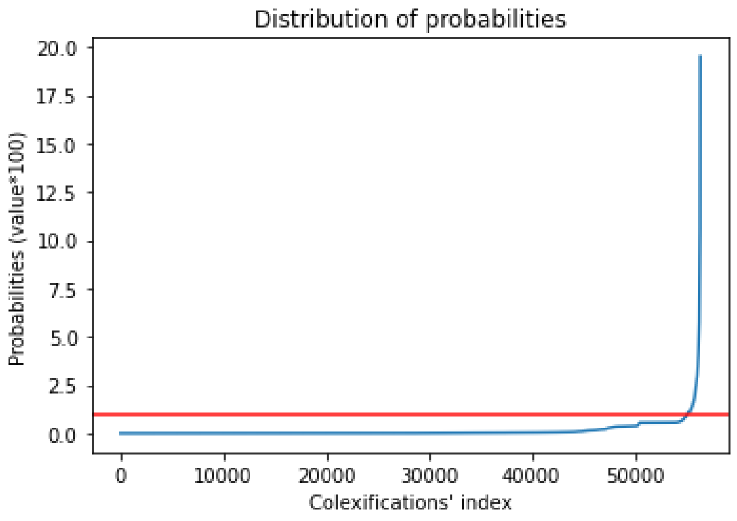


Figure 4.4: Distribution of probabilities. The red line is set at 1.

4.6.2 Study of the cosine similarity values

We then wanted to take into account the cosine similarity values obtained from the pre-trained fastText model (see section 4.5).

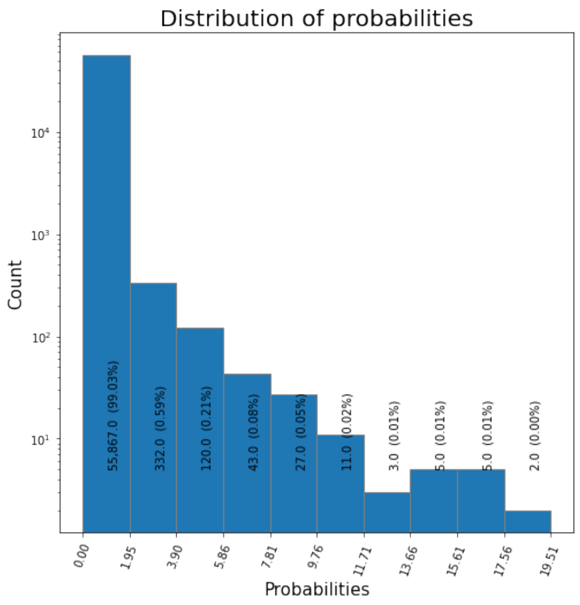


Figure 4.5: Histogram with the distribution of probabilities.

As mentioned, the cosine similarity range is [-1; 1], being -1 the relation between perfect opposite vectors, 0 a relation between vectors with no correlation, and 1 two complete identical vectors. Specifically, in our case, the values of the cosine similarity values range between -0.098 and 0.997, with very low negatively correlated vectors.

The mean value in the data is 0.21 with a standard deviation of 0.14. The most negatively correlated colexification is *(launder, deaf)* while the most positively correlated pair is *(three days before now, four days before now)*.

Figure 4.6 represents the sigmoidal distribution of cosine similarity values, showing that, considering the whole range that in our case is [-0.1; 1], the majority of colexifications has a value below the median.

4.6.3 Study of the correlation between probability and cosine similarity

The plot 4.7 instead, combine both the probability with the cosine similarity values, showing how the higher probability values have a cosine similarity that ranges between 0.2 and 0.9, while the higher cosine values have very low probabilities. As such, tables 4.6 and 4.7 shows the top five probabilities and cosine similarity value pairs. To better understand why the higher cosine values are

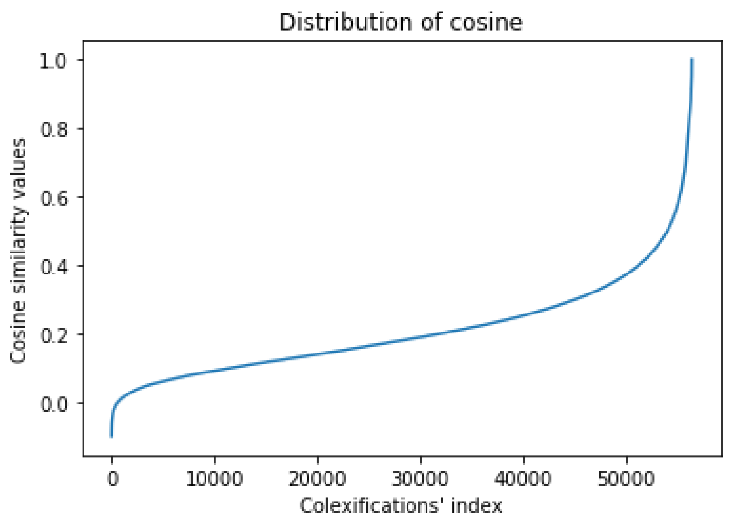


Figure 4.6: Distribution of cosine similarity values.

actually not frequently colexified, we have to remember the theory lying under the colexification itself. If it is true that two similar meanings are easily conveyed with a single word form, it is also true that meanings that are so similar that could cause misunderstandings, are hardly a case of colexification (section 3.1). This is exactly the case of pair of words with very high cosine similarity values.

Table 4.6: Top 5 probabilities values pairs

Pairs	**Probabilities**	**Cosine similarity**
moon, month	0.195	0.22
tree, wood	0.182	0.40
flesh, meat	0.171	0.50
grandson, granddaughter	0.169	0.86
dish, plate	0.164	0.49

Table 4.7: Top 5 cosine similarity values pairs

Pairs	**Probabilities**	**Cosine similarity**
three days before now, four days before now	5.1×10^{-5}	0.997
mother's sister, father's sister	0.05	0.989
mother's brother, father's brother	0.06	0.989
twenty four, twenty five	1.4×10^{-4}	0.986
north, south	0.02	0.978

At a glance, we could say there is no correlation between the two variables. In the following phases of work, we will use the cosine similarity values to predict the probability ones (see section 4.7). We are going to use a single feature, derived from something completely different, to predict a complex phenomenon. If the two features do not even correlate, the difficulty of the task improves. Since the variables do not show a Gaussian-like distribution, we cannot use covariance or Pearson's correlation to verify if there is correlation. Still we can calculate Spearman's correlation, whose value is 0.138, in a range [-1; 1], confirming the previous idea that between the two variables there is no strong correlation. By comparison, Xu's result of Spearman's correlation was 0.22, which was considered as a high value compared by the correlation obtained with other features (see 3.2.2).

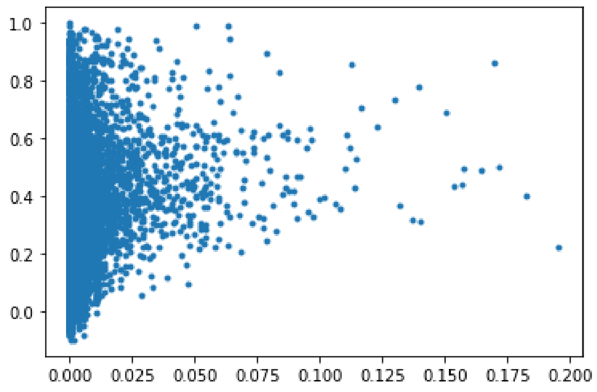


Figure 4.7: Scatter plot of cosine similarity with probabilities values.

These analyses can be found in the file *06_AnalysisBootCosine.ipynb*

4.7 Logistic Regression

Now that we have calculated the probabilities and the cosine similarity values for every colexification, we will try to answer two questions:

- How well can the cosine similarity values help us in predicting whether two words are colexified with high probability across different languages?
- How well can the cosine similarity values help us in predicting whether two words are a colexification at all?

To answer the questions, we will apply a Logistic Regression model to the data.

4.7.1 First version

In this first version of the Logistic Regression, we will try to predict the most probable colexifications.

To begin with, we load the first bootstrapped table and define a label for every row: 1 for the highly probable colexifications, 0 for the rest. But how much is "highly probable"? If we look back at the section 4.6.1, we see that only roughly 2% of the colexifications have a probability value higher than 0.01. For this reason, the top 2% has been labeled with value 1, defining this amount as highly probable.

At this point, we load the Logistic Regression model from scikit learn [Pedregosa et al., 2011], setting the test size at 0.2. The data the model relies on are only the cosine similarity values and the boolean labels. After the training, the accuracy we obtain in the test data is 0.98, which could be considered a great result! However, if we look at the classification report, shown in table 4.8, we see that the model is having a great response just because it is assigning 0 to the majority of pairs. This usually happens in an unbalanced situation, where a model generalizes the problem, putting the most attested label to almost all the entries.

As consequence, the obtained result is meaningless, and we can avoid repeating the process for the other bootstrapped tables, opting for another approach that will be laid out in the following section.

Table 4.8: Classification report for first version of Logistic Regression.

Label	Precision	Recall	F1-score	Support
0	0.98	1.00	0.99	8567
1	0.00	0.00	0.00	183

4.7.2 Second version

In the previous section, the problem we encountered dealt with an unbalance in the data. To solve the situation, once we set the top 2% of highly probable colexifications, we need to take roughly the same amount of entries for the not highly probable ones.

This time, the obtained accuracy is 0.78 (which is still a good result!), with a classification report as the one of table 4.9.

Table 4.9: Classification report for second version of Logistic Regression.

Label	Precision	Recall	F1-score	Support
0	0.79	0.78	0.78	179
1	0.77	0.78	0.77	171

We can consider the procedure as sound, and run the code on large scale, meaning on every bootstrapped sample setting 10 different random states [4].

Of course, the quantity of entries we use in the model is quite small, because of the balancing operation. A way to consider a bigger quantity of colexifications is defined in the following version.

4.7.3 Third version

As seen previously, the amount of colexification used by the logistic regression model is quite small. A solution to this could be changing the percentage of highly probable colexifications, but this operation could be misleading because in that way we will not have true *high* probable entries. Another resolution could be to change the whole question the model is trying to answer. As such, instead of finding the most probable colexifications using the cosine similarity, we want to use the logistic regression model to identify whether a pair of words is a colexification at all.

To do this, we set all the colexifications' labels as 1, meaning that the word pairs are real colexification; and we create random word pairs to be set as 0. The latter will be created from the existing list of colexifications, randomly picking pair of words and checking whether they are attested as colexifications. In the case the pair is not attested, it cannot be considered a colexification and labeled as 0.

Obviously, we need to keep the data balanced, to avoid incurring in the error of the first version. For this reason, we will create as much unattested colexification as attested ones.

[4]Setting a random state allows us to obtain always the same results when the number does not change.

The accuracy we obtain for this task is 0.64, with a classification report detailed in table 4.10.

Table 4.10: Classification report for third version of Logistic Regression.

Label	Precision	Recall	F1-score	Support
0	0.62	0.70	0.66	8734
1	0.66	0.57	0.61	8678

Again, we consider the values as valid, and we perform the operation on every bootstrapped table, setting 10 different random states as we did for the previous version.

The code that implements this section is in file *07_LogisticRegression_3ver.ipynb* and *08_LogReg_100pretr.ipynb*.

4.8 Discussion of the results

In this phase of work, we decided to consider only the F1 score value, since this number easily sums up both precision and recall (see section 3.4.3).

The results obtained from the 100 bootstrappings are almost unchanged through the tables. For the second version, which balanced the data with an equal number of 1 and 0 labels, the standard deviation is 0.02, with a mean value of 0.80. On the other hand, in the third version, where we created unattested colexifications, the standard deviation is 0.006 and the mean is 0.63. In both cases, the value is far from the random baseline, set at 0.5 given the two possibilities in the labels. As previously described (section 4.7.1), the first version was not performed on all the bootstrapped tables to avoid incurring in a bias in the model.

The mechanism of bootstrapping allowed us to see how much the values could change considering different re-samplings of data. We noticed a nearly null variation, meaning that the model is robust to changes in the input.

We can then consider the results a success and a confirmation of the initial assumption: the distributional similarity between two words helps in predicting whether a pair can (with a high probability) be a colexification. Moreover, we noticed that the logistic regression model was able to grasp some aspects of the data that were invisible from our initial correlation tests. In section 4.6.3 we stated

that we would hardly have success in the prediction given the lack of correlation between the variables, indeed, we were wrong.
Table 4.11 summarizes the results.

Table 4.11: Result of the Logistic Regression on the two versions, averaged between 100 stratified boot-straps.

Task	Average F1-score	Standard Deviation
high probability	0.80	0.02
is colexification	0.63	0.006

Chapter 5

Training models

We saw in the previous chapter how we collected and manipulated the data to apply the pre-trained fastText and Logistic Regression models.

In this chapter, we will introduce some modifications to the process explained above. We will prepare a training corpus and train our own fastText models, changing the value of window size.

The chapter will start with the collection and management of the training data: section 5.1 will explain how we obtained the data to train the model, how we cleaned it obtaining various files, and how we joined those files. Section 5.2 will take care of the models' training to end with section 5.3 that will discuss the obtained results.

5.1 Training corpus

In this phase of work, we want to train our own fastText models. To do so, the starting point is to collect a training corpus.

5.1.1 Downloading the corpus

We decided to use the English Wikipedia as a training corpus, given its dimension, linguistic reliability, and availability to the public. The first step is to download a Wikimedia dump [1].

The code that performs the download operation was adapted from the fast-

[1]Wikimedia downloads: `https://dumps.wikimedia.org/`

Text's GitHub repository [2]. The bash file performs the first cleaning phase of the Wikimedia dump, removing tables, converting links and performing a brief normalization on the text.

The downloaded file is called *enwiki-latest-pages-articles.xml.bz2*, and has a size of 17 GB. The Wikimedia organization periodically updates its dump files, allowing people to get the most up-to-date corpus, but unfortunately, they do not label the files with a version number. To specify which dump we worked on, we can only say the download was performed on 19 April 2021 [3].

5.1.2 Cleaning the corpus

From the first code, we obtained a zipped version of the English Wikipedia dump. To unzip the file, we used the WikiExtractor module [Attardi, 2015] which creates various text files of fixed size (in our case, we opted for 1 GB). Each extracted file has the following format: contains various articles in an XML structure bounded by a *doc* tag containing id, URL and title of the article. The XML element contains the article in plain text, maintaining upper and lower cases, having the first line as the title of the article [Attardi, 2021].

Afterward, every file has been cleaned removing the XML *doc* tags, putting everything lowercase, and removing URLs, multiple spaces and punctuation. The file that performs this operation is *cleaning_wiki.py*. Lowering the words, removing consecutive spaces and punctuation were not mandatory but recommended steps [Bhattacharjee, 2018, p. 62].

An option for the cleaning could have been to remove also stop words, but looking at the colexifications' list, we noticed that some concepts appeared in the stop words list. Therefore, removing stop words from the corpus would mistakenly compromise the training. As regards numbers, no actions were taken, given that numbers would probably not influence the training [4].

[2]FastText's Github repository: `https://github.com/facebookresearch/fastText/blob/master/get-wikimedia.sh`

[3]Visit `https://dumps.wikimedia.org/enwiki/latest/` for a complete list of the most updated English dumps

[4]For some suggestions in the text pre-processing check: `https://stackoverflow.com/questions/62244474/text-preprocessing-for-text-classification-using-fasttext`

The cleaned files have been joined together in a single corpus file of size 13 GB through a concatenation.

5.2 Training the models

5.2.1 Choosing the architecture

Now that the corpus is ready, we need to understand which model to use for training: CBOW or Skip-gram. We decided to be pragmatic training two versions of the model with the following command:

fasttext MODEL -input corpus_wiki -output result/standard_model

where *MODEL* is either CBOW or Skip-gram, *corpus_wiki* is the training corpus and *standard_model* is the name we assign to the output model. The hyper-parameters chosen for the training are the default ones, shown in table 5.1 [Facebook Inc., 2020a].

Table 5.1: Default main hyper-parameters of the fastText model.

Dimension	100
Epoch	5
Learning rate	0.1
Loss function	softmax
min Count	1
window size	5

The models we obtain have a size of 2.5 GB and took 31.5 and 44.5 hours for training, respectively for the CBOW and the Skip-gram models.

To choose the best model, we used them both to get the cosine similarity values. Subsequently, we used the cosine similarity values in the Logistic Regression model. The average F1 scores of the two models are in table 5.2. As we can see, the best results were obtained using the Skip-gram model.

Table 5.2: Results of the trained fastText models.

Model	Task	Average F1-score
CBOW	high probability	0.723
Skip-gram	high probability	0.747
CBOW	is colexification	0.576
Skip-gram	is colexification	0.604

Comparing the tables 4.5 and 5.1, we see that the hyper-parameters used by the models are different. Moreover, knowing that even the training corpus is different, it is clear that we cannot compare the results obtained by the pre-trained and trained models. As a matter of fact, the results obtained by the pre-trained model are higher with respect to the ones we got with our model. It is clear that the bigger dimension of the training corpus, combined with a different setting in the hyper-parameters, has improved the model performances. Nevertheless, we decided to use the default values of the hyper-parameters for the following phases of work. As consequence, the results obtained from the trained model will be treated as our baseline for future analysis. Lastly, since the Skip-gram model performed better than the CBOW, we will use the former.

5.2.2 Changing window size

The next steps consist of training other models changing the parameter of window size and keeping the other hyper-parameter as default. Due to the time consumption for the training phase, we opted for training the models on a small section of the corpus. If it is true that the bigger the training corpus is, the higher the performances are, it is also expected that the results we obtain when changing the window size are completely comparable through the models once fixed a predefined training corpus common for every model. Furthermore, it has been proven that fastText model is robust to small training corpus (see section 2.5). This choice speeds up the process, reduces the memory usage and increases the reproducibility of the work. The size chosen for the training corpus is 1 GB, obtained from the first part of the whole corpus we managed previously (section 5.1).

We trained the models with different window sizes: 1, 2, 3, 5, 7, 10, 20, 50. When choosing the values, we thought it was interesting to see how the values changed with very low window sizes (1, 2, 3), then we opted for the default one (5), we moved on to bigger ones but keeping the values close to the default (7, 10) and finally, we went to very wide ones (20, 50). The resulting models have a size of approximately 1 GB. On every trained model, we ran the code explained in the previous chapter obtaining the results listed in table 5.3. The code performing the operations is in the file *10_LogReg_models.ipynb*.

Table 5.3: Results of the fastText models trained with different window sizes.

WS	Hours training	avg F1 high prob	avg F1 is colex
1	2	0.751	0.587
2	2.5	0.754	0.594
3	3	0.753	0.598
5	4.5	0.745	0.598
7	5	0.741	0.594
10	6	0.738	0.594
20	10.5	0.726	0.593
50	16	0.707	0.594

5.3 Analysis of the results

The first aspect to analyze is the chosen architecture: Skip-gram. The Skip-gram model took more time to train in comparison with the CBOW model. The reason lies behind the number of operations Skip-gram has to run: for every word in the context window, the model tries to predict the target word, check its performance and correct it backward. On the contrary, with the CBOW approach, all the vectors of the words in the context window are averaged together in a single vector used as input data to obtain the target word. As result, if the window size is 3 the Skip-gram model has to repeat the operation 3 times, while the CBOW model only 1, and if we change the window size to 7 the number of operations executed by Skip-gram become 7, while remains stationary at 1 for CBOW. Luckily, the extra time used by Skip-gram helps in obtaining higher performances [Mohr, 2021b].

This leads us to the second thing to notice: the amount of time necessary for the training. The wider the window size, the higher the number of hours necessary for the training. It is a logical consequence of the explained approach.

We then move to the analysis of the results. Through the 100 bootstrapped tables, as previously noticed (section 4.8), the values keep a low standard deviation, being stuck to a constant number across the iterations. Figures 5.1 and 5.2 depict how the values are spread.

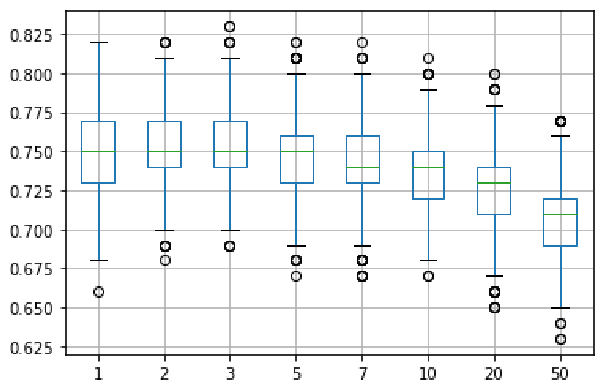


Figure 5.1: Box plot of the results for the balanced version.

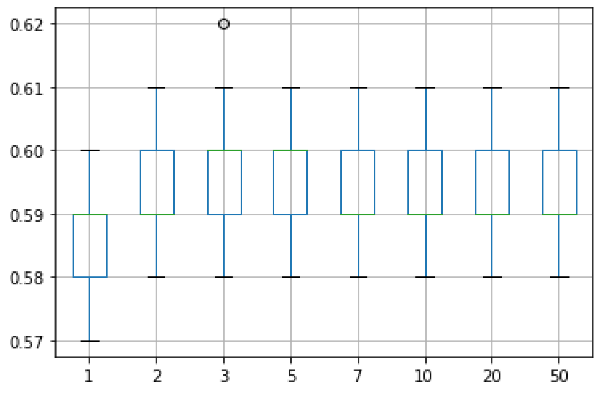


Figure 5.2: Box plot of the results for the unattested version.

As regards the query on whether the cosine similarity can help us in predicting if two words are colexified with high probability across different languages, the results are moderately higher with respect to whether the cosine similarity values can help us in predicting if two words are a colexification at all. This means that the first task was harder than the second, probably given the fact that in the first step we split the pairs of words depending on their value. In doing so, the hypothesis is that we separate concepts that are highly similar between them to concepts which happen to be a colexification for reasons other than meaning similarity (e.g. culture or linguistic development). On the other hand, in the second task, the set of positive examples is composed of pairs that may be both highly correlated as well as just above an ideal threshold of correlation, making the positive group impure.

In both cases, the results we obtained are above the random value (0.5 having only two possibilities), allowing us to declare that the jobs were successful. The values do not radically change through the columns, but we can observe how a narrow window size has better effects on the logistic regression results. Specifically, for the task of labeling the high probable pairs, the best window size is 2, with a result that is quite different respect to the wider window size (0.754 vs 0.707). As a general rule it looks like the wider the window size, the worst the performance on this task. For the second task, which consist in labeling if a pair could be a colexification at all, the window sizes 3 and 5 obtain the best results. Through all the column, we do not see much difference in the results (0.598 for the best, 0.587 for the worst). Interestingly, in this case the worst performance was obtained by the most narrow window size (1). Doing a trade-off, we can say that a window size of 3 gets the best results in both tasks.

Finally, comparing the best results of the models trained on 1 GB of text with respect to the results of the model trained on 13 GB (0.753 and 0.598 for the small vs 0.747 and 0.604 for the big), it isn't worth the trouble. The enormous amount of text taken into account has almost no improvement on the performances. It should be recalled that the Skip-gram model has proven to works well with a small amount of the training data [Kulshrestha, 2019]. Presumably, if we used the CBOW model, the performances using only 1 GB of text would have been much worst. In addition, this is one of the key features of fastText itself: even if we have a small training corpus, the word embeddings do not improve proportionally to the dimension.

Chapter 6

Conclusions

In the thesis we investigated the linguistic knowledge acquired by word embedding models.

As we discussed, finding ways to represent words in computational ways is both a necessary and not straightforward task to do. There is the need to allow algorithms to include semantic sensibility in the computation process to perform a multiplicity of tasks, from sentiment analysis to semantic search. But, there is no obvious best way to represent linguistic information. It is therefore important to investigate the linguistic knowledge different computational methods acquire, and how different parameters affect this outcome.

After an overview of the first methods to represent words and the study of the three most widely and known word embeddings models (Word2vec, GloVe and fastText), we decided to use the latter. It distinguishes from the others for its capacity to build good word vectors for out-of-vocabulary or rare words, its speed in training and its robustness towards the (small) size of the training data.

Given the impossibility to show the vectors without some kind of loss, and realizing that we do not know the meaning of each vector dimension, we need to test practically if the word embeddings have some sort of linguistic knowledge. Accordingly, to inspect the knowledge gained by the model, we decided to test it on a linguistic problem. We studied the past research of Xu et al. on colexification [Xu et al., 2020a], and, based on that, we supposed word embeddings would have been sensitive to that linguistic phenomenon. In this thesis, we used much more and more diverse data than the study of Xu's team (for example, the number of languages has increased from 246 to 2990). Through the whole process, we dealt with the dataset collection, its understanding and its cleaning. The used model was fastText, in its pre-trained version plus few different trained versions depending on the window size. We looked at the influence that different window sizes had on the effect of the predictions of a logistic regression model and we could verify which kind of linguistic knowledge the model acquired.

To describe with more care the steps taken, we studied the phenomenon of colexification, described as cases where, in a given language, two functionally distinct senses can be associated with the same lexical form [François, 2008, p. 170]. Its origin, despite the fact of not being clear, is thought to rely on a semantic connection between the two meanings [François, 2008, p. 172]. Given that word embeddings allow us to calculate semantic closeness as the value of cosine between the angles of their vector representations, obtaining the cosine similarity values is precisely our first assignment. To solve it, we used the pre-trained version of the fastText model.

We wanted the data to answer two questions:

- How well can the cosine similarity values help us in predicting whether two meanings are highly colexified across different languages?
- How well can the cosine similarity values help us in predicting whether two words are a colexification at all?

To be able to solve these questions, we used a logistic regression model. The model was able to identify correctly the majority of queried word pairs as high probable colexifications, or as colexifications at all. The obtained results are satisfying: 0.80 on the first task and 0.63 on the second, with a baseline set at 0.50, given the binary choice.

Subsequently, we also performed a phase of model training, changing the value of the window size. This is considered to be one of the most important hyper-parameters of the model, given the model's architecture itself. The tested sizes were 1, 2, 3, 5, 7, 10, 20, and 50, to be understood as symmetric number of words before and after the target word.

Overall, for our purpose, we can say dimension 3 is the window size dimension that leads us to the best performances. This result aligns with the previous studies in the field, stating that a narrow window size is the best for paradigmatic and lexical semantic tasks. Surprisingly, depending on the task, we had little or almost no variation in the results. For the task of labeling the high probable pairs, through the window sizes, the values changed in a range from 0.754 to 0.707. For the second task, consisting in labeling if a pair could be a colexification at all, apart from the window size 1 whose value is 0.587, the remaining values have a variation of only 0.004.

These results show that the second task is a lot less sensitive to the window size dimension, and more broadly, show that it is not always true that the window size is an important parameter, contrary to claims of other studies [Sarkar and Howard, 2019].

We can conclude the project was successful, showing that the word embedding model, specifically fastText, is able to acquire semantic knowledge of the embedded words. Furthermore, we discovered that, depending on the queried task, there is window size invariance.

This work suggests that there is a difference between highly frequent (the top 2%) vs. the other colexifications. The results we obtained show that the former are sensitive to the window size, while the latter are not. At present, it is unclear why that is. A future development could involve an in-depth study of the difference between highly and lower frequent colexifications.

A second point to inspect in future could be a systematic evaluation of linguistic tasks in which window size does or does not matter. What do tasks in each group have in common? The goal, again, would be to learn about the linguistic knowledge acquired by word embeddings, as a function of window size but on a more general level.

Another idea could be to use Language Models (e.g. BERT) for the analysis, looking at how (and if) the results change. This would shed a light on the differences between the models.

Finally, there could be used not only English word embeddings but other languages to check the robustness of the results reported here.

Acknowledgments

I thank Thomas Brochhagen for his great work as traineeship tutor and for his indispensable supervisor role in the writing of this thesis.

References

[Alammar, 2019] Alammar, J. (2019). The Illustrated Word2vec. `http://jalammar.github.io/illustrated-word2vec/`, Accessed: 23/06/2021.

[Attardi, 2015] Attardi, G. (2015). Wikiextractor. `https://github.com/attardi/wikiextractor`.

[Attardi, 2021] Attardi, G. (2021). File format. `https://github.com/attardi/wikiextractor/wiki/File-Format`, Accessed: 08/06/2021.

[Bhattacharjee, 2018] Bhattacharjee, J. (2018). *FastText Quick Start Guide: Get Started with Facebook's Library for Text Representation and Classification*. Packt Publishing.

[Biberauer et al., 2010] Biberauer, T., Holmberg, A., Roberts, I., and Sheehan, M. (2010). *Parametric Variation: Null Subjects in Minimalist Theory*. Cambridge University Press.

[Bloomfield, 1922] Bloomfield, L. (1922). *The American Journal of Philology*, 43(4):370–373.

[Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

[Borin et al., 2013] Borin, L., Comrie, B., and Saxena, A. (2013). *The Intercontinental Dictionary Series - a rich and principled database for language comparison*, pages 285–302. De Gruyter Mouton.

[Brich, 2018] Brich, T. (2018). Semantic sentence similarity for intent recognition task.

[Brownlee, 2017] Brownlee, J. (2017). *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery.

[Brysbaert et al., 2013] Brysbaert, M., Warriner, A., and Kuperman, V. (2013). Concreteness ratings for 40 thousand generally known english word lemmas. *Behavior research methods*, 46.

[Chomsky, 1957] Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co., The Hague.

[Chomsky, 1965] Chomsky, N. (1965). *Aspects of the Theory of Syntax*. The MIT Press, Cambridge.

[CR, 2020] CR, A. (2020). Word Embeddings in NLP — Word2Vec — GloVe — fastText. `https://medium.com/analytics-vidhya/word-embeddings-in-nlp-word2vec-glove-fasttext-24d4d4286a73`, Accessed: 29/06/2021.

[Everett et al., 2015] Everett, C., Blasi, D., and Roberts, S. (2015). Climate, vocal folds, and tonal languages: Connecting the physiological and geographic dots. *Proceedings of the National Academy of Sciences of the United States of America*, 112.

[Facebook Inc., 2020a] Facebook Inc. (2020a). List of options. `https://fasttext.cc/docs/en/options.html`, Accessed: 10/06/2021.

[Facebook Inc., 2020b] Facebook Inc. (2020b). Word representations. `https://fasttext.cc/docs/en/unsupervised-tutorial.html`, Accessed: 16/06/2021.

[Facebook Research, 2019] Facebook Research (2019). Fasttext readme. `https://github.com/facebookresearch/fastText/blob/master/python/README.md#model-object`, Accessed: 11/05/2021.

[FEL, 2020] FEL, T. (2020). Interpreting negative cosine similarity. Cross Validated. URL:https://stats.stackexchange.com/q/446554 (version: 2020-04-02).

[Firth, 1957] Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. 1952-59:1–32.

[François, 2008] François, A. (2008). *Semantic maps and the typology of colexification: Intertwining polysemous networks across languages*, pages 163–215.

[Geeraerts, 1997] Geeraerts, D. (1997). *Diachronic Prototype Semantics: A Contribution to Historical Lexicology*. Oxford University Press.

[Goldberg, 2017] Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*, volume 37 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool, San Rafael, CA.

[Grave et al., 2018] Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[Hammarstrom et al., 2020] Hammarstrom, H., Forkel, R., Haspelmath, M., and Bank, S. (2020). *Glottolog 4.3*. Jena.

[Harris, 1954] Harris, Z. (1954). Distributional structure. *Word*, 10(2-3):146–162.

[Hockett and Hockett, 1960] Hockett, C. F. and Hockett, C. D. (1960). The origin of speech. *Scientific American*, 203(3):88–97.

[Horowitz, 2001] Horowitz, J. L. (2001). Chapter 52 - the bootstrap. volume 5 of *Handbook of Econometrics*, pages 3159–3228. Elsevier.

[Jackendoff and Peruzzi, 1998] Jackendoff, R. and Peruzzi, A. (1998). *Linguaggio e natura umana*. Collezione di testi e di studi. Il Mulino.

[Jones, 1972] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21.

[Jordan, 2017] Jordan, J. (2017). Evaluating a machine learning model. `https://www.jeremyjordan.me/evaluating-a-machine-learning-model/`, Accessed: 16/06/2021.

[Jurafsky and Martin, 2000] Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st edition.

[Kulshrestha, 2019] Kulshrestha, R. (2019). NLP 101: Word2Vec - Skip-gram and CBOW. *Toward Data Science*.

[Lapesa et al., 2014] Lapesa, G., Evert, S., and Schulte Im Walde, S. (2014). Contrasting syntagmatic and paradigmatic relations: Insights from distributional semantic models. pages 160–170.

[Lison and Kutuzov, 2017] Lison, P. and Kutuzov, A. (2017). Redefining context windows for word embedding models: An experimental study. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 284–288, Gothenburg, Sweden. Association for Computational Linguistics.

[Michel et al., 2011] Michel, J.-B., Shen, Y., Aiden, A., Veres, A., Gray, M., Pickett, J., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J., Pinker, S., Nowak, M., and Aiden, E. (2011). Quantitative analysis of culture using millions of digitized books. *Science (New York, N.Y.)*, 331:176–82.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In Bengio, Y. and LeCun, Y., editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.

[Mikolov et al., 2013b] Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *ArXiv*, abs/1309.4168.

[Mikolov et al., 2013c] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26.

[Mohr, 2021a] Mohr, G. (2021a). How does pre-trained fasttext handle multi-word queries? `https://stackoverflow.com/a/66251182/14984896`, Accessed: 11/05/2021.

[Mohr, 2021b] Mohr, G. (2021b). Why does skipgram model take more time than cbow. `https://stackoverflow.com/a/48349115/14984896`, Accessed: 11/06/2021.

[Nelson et al., 1998] Nelson, D. L., McEvoy, L., C., and Schreiber, T. A. (1998). The University of South Florida word association, rhyme, and word fragment norms.

[Norris, 2018] Norris, S. (2018). An Idiot's Guide to Word2vec Natural Language Processing. `https://opendatascience.com/an-idiots-guide-to-word2vec-natural-language-processing/`, Accessed: 24/06/2021.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Pericliev, 2015] Pericliev, V. (2015). On colexification among basic vocabulary. *Journal of Universal Language*, 16:63–93.

[Rajasekharan, 2020] Rajasekharan, A. (2020). What are the main differences between the word embeddings of ELMo, BERT, Word2vec, and GloVe? `https://qr.ae/pGN2Fp`, Accessed: 29/06/2021.

[Ramos, 2003] Ramos, J. (2003). Using tf-idf to determine word relevance in document queries.

[Regier et al., 2016] Regier, T., Carstensen, A., and Kemp, C. (2016). Languages support efficient communication about the environment: Words for snow revisited. *PLOS ONE*, 11(4):1–17.

[Rosch, 1978] Rosch, E. (1978). Principles of categorization. In Rosch, E. and Lloyd, B. B., editors, *Cognition and Categorization*, pages 27–48. Erlbaum, Hillsdale, NJ.

[Rzymski et al., 2020] Rzymski, Christoph, and Tresoldi, T. e. a. (2020). The database of cross-linguistic colexifications, reproducible analysis of cross-linguistic polysemies. *Scientific Data*, 7.

[Sarkar and Howard, 2019] Sarkar, A. and Howard, M. (2019). Scale-dependent relationships in natural language. *CoRR*, abs/1912.07506.

[Storks et al., 2020] Storks, S., Gao, Q., and Chai, J. Y. (2020). Recent advances in natural language inference: A survey of benchmarks, resources, and approaches.

[The Association for Computational Linguistics, 2005] The Association for Computational Linguistics (2005). What is computational linguistics? `https://www.aclweb.org/archive/misc/what.html`, Accessed: 14/12/2020.

[Turing, 1950] Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, LIX(236):433–460.

[Vanschoren, 2021] Vanschoren, J. (2021). 10-fold crossvalidation. `https://www.openml.org/a/estimation-procedures/7`, Accessed: 17/06/2021.

[Xu et al., 2020a] Xu, Y., Duong, K., Malt, B. C., Jiang, S., and Srinivasan, M. (2020a). Conceptual relations predict colexification across languages. *Cognition*, 201:104280.

[Xu et al., 2020b] Xu, Y., Duong, K., Malt, B. C., Jiang, S., and Srinivasan, M. (2020b). Conceptual relations predict colexification across languages (supplementary material). *Cognition*, 201:104280.

[Zipf, 1949] Zipf, G. K. (1949). *Human Behavior and the Principle of Least Effort*.

Appendix A

Differences between Xu et al. and this project

Table A.1: Differences between Xu et al.'s project and present project

Factor	**Xu's project**	**Present project**
Data base used	IDS	CLICS
Number of languages	246	2990
Use diachronical varieties	Yes	No
Type of concepts	Single-word	Multi-word*
Bootstrapping	Yes	Yes
Monte Carlo	Yes	No
Number of bootstraps	1000	100
NLP model	Word2Vec	FastText
Metric to evaluate Logistic Regression	Accuracy	F1 score

* We removed multi-word expressions containing "or" or parenthesis

Appendix B

List of files available on GitHub

The following is a list of the files available in the GitHub repository of the project, available at: `https://github.com/sarabert96/Colexification` [1]

Main folder

- 00_DataCollection: writes a CSV with the data from CLICS3 (out df_all_raw.csv)
- 01_DataUnderstanding: inspect the CSV file
- 02_DataCleaning: remove diachronic varieties, multiword concepticons and lowercase every concepticon (out df_all_ok.csv)
- 03_CreateColex: creates a dataframe of colexifications (out df_colexifications.csv)
- 04_BootstrapProbabilities does a bootstrapping (100 cycles) and for every cycle it calculates the probability of finding a specific colexification in any of the families (out statistics_time.txt, listBoot.txt)
- 05_gettingCosines: given the colexifications dataframe (out of 03), it returns another dataframe with the calculation of cosine similarity between a pair of concepts (a colexification). Uses the pre-trained version of FastText. (out df_colex_cosines.csv)
- 06_AnalysisBootCosine: analyze the distribution of probabilities calculated from the bootstrapped code, and the cosine values for every pair of that bootstrapped cycle

[1] Last update 29/06/2021

- 07_LogisticRegression_3ver: 3 versions of application of Logistic Regression on the output of only one cycle of the bootstrap
- 07b_gettingCosines_support: the file uses the FastText model to calculate the cosine similarity values. Serves as a support for the file 07
- 08_LogReg_100pretr: apply the Logistic Regression model on every output of the 100 cycles. Uses the pre-trained version of FastText. (out df_logreg_bal.csv, df_logreg_un.csv)
- 09_AnalysisLRpT : analyze the results of the Logistic Regression of file 08
- 10_LogReg_models: apply the Logistic Regression model on every output of the 100 cycles with 8 different models.
- 11_ModelResults : analyze the results of the Logistic Regression applied using the 8 different models
- HP_FT: inspect the hyper-parameters of the pre-trained version of FastText

HPC folder

1. get_wikimedia.sh: download Wikipedia dump (out bz2 file)
2. cleaning_attardi.sh: cleans the Wikipedia dump using WikiExtractor function
3. cleaning.sh: for every output of the previous file, calls cleaning_wiki.py function
4. cleaning_wiki.py: cleans the Wikipedia dump obtained by the WikiExtractor function, putting everything lowercase, removing links and punctuation
5. joining.sh: for every cleaned file in the folder, it appends it to the general corpus file (out corpus_wiki)
6. training_ft.sh: code to train the FastText model using the outputted corpus
7. trainingFT_models.sh: code to train the FastText model with different window sizes (NB uses a small version of the wikipedia corpus)

8. log_reg.sh: calls log_reg_model.py

9. log_reg_model.py: for every model trained, applies the Logistic Regression model as seen in the notebook file 08 (out df_logreg_bal.csv, df_logreg_un.csv)