



UNIVERSITAT DE
BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

**Study of Different Models for
Sentiment Analysis and Language
Representation**

Autor: Núria López Raich

Directors: Dr. Josep Vives i Santa-Eulàlia

Dr. Jordi Vitrià Marca

Tutor: Oriol Barnaus Garcia

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, June 13, 2022

Contents

Introduction	1
1 How do we Represent Words?	4
1.1 Word Embeddings	4
1.2 One-Hot Encoding	5
1.3 Word2vec	5
1.3.1 The Skip-Gram Model	6
1.3.2 The Continuous Bag of Words Model	7
1.4 The Arithmetic of Word Vectors	9
2 Long Short-Term Memory Network	13
2.1 Recurrent Neural Networks	13
2.2 Activation Functions in a LSTM	14
2.3 LSTM Architecture	15
2.3.1 Input Gate, Forget Gate and Output Gate	16
2.3.2 Candidate Memory Cell	17
2.3.3 Memory Cell	17
2.3.4 Hidden State	18
2.4 Topology of tanh and Sigmoid Layers	18
2.5 Hyperparameters	19
2.5.1 Number of Nodes and Hidden Layers	19
2.5.2 Number of Units in a Dense Layer	20
2.5.3 Dropout	20
2.5.4 Weight Initialization	20
2.5.5 Learning Rate	21
2.5.6 Momentum	21
2.5.7 Number of Epochs	21
2.5.8 Batch Size	21

3	Logistic Regression Model	22
3.1	Generalized Linear Models	22
3.2	Interpreting Parameters in Logistic Regression	24
3.3	Logistic Function	25
3.4	The Multinomial Logistic Regression Model	25
3.5	Fitting Logistic Regression Models	26
3.6	Maximum Entropy Models	26
3.7	Cost Function	27
3.8	Gradient Descent	29
3.9	Assumptions	30
4	Naive Bayes Model	32
4.1	Definition	32
4.2	Parameter Estimation and Event Models	33
4.2.1	Gaussian Naive Bayes	33
4.2.2	Multinomial Naive Bayes	33
4.2.3	Bernoulli Naive Bayes	33
4.3	The Optimality of Naive Bayes	34
4.4	Assumptions	38
4.5	Relationship Between Naive Bayes Classifiers and Logistic Regression	38
5	Results, Analysis and Comparison Between the Models Used	39
5.1	About de Dataset	39
5.2	Advantages and Disadvantages	40
5.3	Results of the Model Evaluation	41
5.4	Conclusions and Future Work	46
A	Implementation	47
	Bibliography	48

Abstract

In this project we study three different models of artificial intelligence to carry out the process of sentiment analysis, which consists of determining the polarity of a text; that is, detecting whether it is positive, negative or neutral.

The first model studied is a neural network, specifically a long short-term memory, which uses deep learning techniques. We delve deeper into the study of its structure and operation, unmasking all the mathematics behind it.

The other two models belong to machine learning: logistic regression and Naive Bayes. We emphasize the study of its parameters and optimization, with the intention to understand the learning process of each one.

Finally, we apply the results and techniques developed to implement a Python program with each model in order to detect the sentiment of thousands of reviews from social media of different bars and restaurants. We dedicate a whole chapter to give the results, the analysis of each one and a comparison between them.

Abstract en Català

En aquest projecte estudiem tres models diferents d'intel·ligència artificial per dur a terme el procés d'anàlisi de sentiments, el qual consisteix en determinar la polaritat d'un text; és a dir, detectar si és positiu, negatiu o neutre.

El primer model estudiat és una xarxa neuronal, concretament una long short-term memory, la qual usa tècniques de deep learning. Aprofunditzem sobretot en l'estudi de la seva estructura i el seu funcionament, desenmascarant totes les matemàtiques que hi ha al seu darrere.

Els altres dos models són de machine learning: la regressió logística i l'Ingenu Bayes. Fem èmfasi en l'estudi dels seus paràmetres i optimització, amb la intenció d'entendre el procés d'aprenentatge de cadascun.

Finalment, apliquem els resultats i tècniques desenvolupades per implementar un programa en Python amb cada model per tal de detectar el sentiment de milers de comentaris en xarxes socials de diferents bars i restaurants. Dedicuem tot un capítol a donar els resultats, l'anàlisi de cadascun i una comparació d'aquests.

Acknowledgments

Especially, I would like to thank my advisors Jordi and Josep, for accepting the work proposal and guiding me. I appreciate all the information sources you have provided to me and all the answers given to my questions.

I'd also like to acknowledge the help of Oriol, this project would not have been possible without his help and his continuous support. Besides my mentor, I would like to thank the rest of the data science team, for welcoming me so well.

Last but not least, I would like to thank my caring, loving and supportive family and friends.

And above all, to Marta, Rafa and Olga.

Abbreviations

CBOW	Continuous Bag of Words
GLM	Generalized Linear Model
GNB	Gaussian Naive Bayes
LR	Logistic Regression
LSTM	Long Short-Term Memory
ML	Machine Learning
MSE	Mean Squared Error
NB	Naive Bayes
NLP	Natural Language Processing
RNN	Recurrent Neural Network

Introduction

Thanks to language, humans are able to turn invisible ideas into visible things. Natural language processing refers to the branch of artificial intelligence concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics with statistical, machine learning and deep learning models. Together, these technologies enable computers to process human language in the form of text “understand” its full meaning, complete with the speaker or writer’s intent and sentiment. Several NLP tasks break down human text in ways that help the computer make sense of what it’s ingesting. One of them, in which we focus on this present work, is sentiment analysis. Sentiment analysis, sometimes also called opinion mining, is a NLP technique used to determine the polarity of a given text; that is, to detect whether data is positive, negative or neutral.

Study of Different Models for Sentiment Analysis and Language Representation, the title of this research work, aims to go beyond the study of three different models to carry out the process of classify texts according to their sentiment and unmask everything that it is relevant about them.

Before explaining and understanding the whole process of how the three models work, it is necessary to explain explain how a machine understands our language, and therefore, how it is able to represent words. In the first introductory chapter, [Chapter 1](#), we present the concept of word embedding, the technique of mapping words to real vectors. Some celebrated results about them are proved. In 2013, a team at Google led by Tomas Mikolov² created word2vec, a word embedding toolkit that can train vector space models faster than the previous approaches. We dedicate the next sections in this chapter to talk about it. Finally, to demonstrate the importance of the benefits it gives, at the end of the chapter a whole section dedicated to the arithmetic of word vectors is exposed, giving a practical example.

²Tomáš Mikolov is a Czech computer scientist working in the field of machine learning.

[Chapter 2](#) presents the first model: long short-term memory. LSTM is an artificial neural network used in the fields of artificial intelligence and deep learning. This type of networks were invented by Hochreiter and Schmidhuber³ in 1997 and set accuracy records in multiple applications domains. First results were already reported in Hochreiter's diploma thesis in 1991 which analyzed and overcame the famous vanishing gradient problem. In machine learning, the vanishing gradient problem is encountered when training artificial neural networks with gradient-based learning methods and backpropagation. In such methods, during each iteration of training each of the neural network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. The aim of the following sections is to discover what is special about these models that make this problem disappear. For that purpose, we will explain its structure, the topology of its layers and the hyperparameters⁴ that must be taken into account for its proper implementation.

Slightly changing the point of view, [Chapter 3](#) delves into the fields of probability and statistics. Logistic regression is a classification model that is very easy to implement and performs very well on linearly separable classes. It is one of the most widely used algorithms for classification in industry too, which makes it attractive to play with. Here we discuss its definition and parameter interpretation, as well as we describe the main assumptions about building a LR model.

To complete the study of the three models, [Chapter 4](#) presents a probabilistic classifier: Naive Bayes, which is based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Apart from explaining the model and the estimation of its parameters, an important result is given: we present a sufficient condition for the optimality of Naive Bayes under the Gaussian distribution. The proof of this result can be found in section 4.3 of this chapter.

The results developed on the previous chapters allow us, in [Chapter 5](#), to analyse how these models behave in practice. We start with an explanation about the dataset we are using to train our models, followed by an argumentation of advantages and disadvantages of each one. Finally, the last section summarize our conclusions, discuss potential improvements to the models and raise future work.

³Hochreiter and Schmidhuber are two computer scientists that have published increasingly sophisticated versions of LSTM

⁴In machine learning, a hyperparameter is a parameter whose value is used to control the learning process.

Lastly, let us remark we have written a Python program of the models, available in the appendix (see [A](#)), that includes an algorithm that reads some commentaries and concludes whether the sentiment of the text is positive, neutral or negative.

The precursor reason that has led me to carry out this work starts as soon as I joined the data science team to perform my internship in the [UVE Solutions](#) company. The first task I was given was to retrieve an LSTM model previously implemented by a teammate. I had to be able to make it work and understand its performance.

The main objective about its implementation is to be able to predict the sentiment of some commentaries from reviews on social media, so we can enrich our database. One can think that we will always have the stars to indicate the polarity of the text, but not all platforms have one, and with the program implemented, we may be able to extract the sentiment of commentaries without needing to have a previous score assigned. This is when I thought of implementing two more different models, to analyse what their differences would be and if it would improve or worsen the predictions quality of the actual model. So one of the goals I set myself is to get a 90% accuracy in both models, building a simpler models to understand, program and train. Being able to count on this tool will be very useful and with so many utilities, from identifying trends of public opinion to evaluate customer attitudes and emotions towards a specific product line or service.

Last but not least, another objective I would like to achieve, is to ensure that the theme is passionate and clear to all of you, readers.

Chapter 1

How do we Represent Words?

Machine and deep learning algorithms can't work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers. In this chapter we introduce the exact mechanism and the math behind word embeddings. The general idea is to find a map from words to vectors such that word similarity and vector-similarity are in correspondence. Whilst vector-similarity can be readily quantified in terms of distances and angles, quantifying word-similarity is a more ambiguous task.

This introductory chapter is intended to cover all the mentioned tasks. [Section 1.1](#) set the definition of word embedding, [Section 1.2](#) explains a first simple approach to represent words and finally [Sections 1.3](#) and [1.4](#) explain in detail the word2vec tool, with an interesting justification about the behaviour doing algebra with vectors.

1.1 Word Embeddings

Natural language is a complex system used to express meanings. In this system, words are the basic unit of the meaning, but words aren't things that computers naturally understand. By encoding them in a numeric form, we can apply mathematical rules and do matrix operations to them, for example.

The technique of mapping words to real vectors is called *word embedding*. It represents words or phrases in vector space with several dimensions. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc.

In recent years, word embedding has gradually become the basic knowledge of natural language processing.

1.2 One-Hot Encoding

A simple way to represent words is through one-hot representations.

Definition 1.1. Let's index the vocabulary V by the set $\{1, \dots, N\}$. A *one-hot representation* of a word is a N -dimensional vector with only one non-zero position corresponding to the index of that word.

That is, to obtain the one-hot vector representation for any word with index i , we create a N -length vector with all zeros and set the element at position i to 1. In this way, each word is represented as a vector of length N , and it can be used directly by neural networks.

Although one-hot word vectors are easy to construct, is not ideal. On the one hand, viewing all the words as discrete units leads us to a sparsity problem. Since there can be a huge corpus of words, representing and storing them as one-hots can be extremely expensive.

Besides that, one-hot word vectors cannot accurately express the similarity between different words, such as the cosine similarity. Seeing its definition:

Definition 1.2. The *cosine similarity* of two vectors $x, y \in \mathbb{R}^d$ is the cosine of the angle θ between them:

$$\cos(\theta) = \frac{x^T y}{\|x\| \|y\|} \in [-1, 1]. \quad (1.1)$$

Since the one-hot vectors of any two different words are necessarily orthogonal, taking the dot product of even two synonyms would yield a similarity score of 0. So one-hot vectors cannot encode similarities among words.

1.3 Word2vec

The word2vec algorithm was proposed to address the above issue. This tool creates a vector representation of words based on the corpus we are using and it manages to capture the semantic representation of words in a vector. It maps each word to a fixed-length vector, and these vectors can better express the similarity and analogy relationship among different words.

Word2vec is not a single algorithm, it consists of models for generating word embeddings. These models are shallow two-layer neural networks having one input layer, one hidden layer and one output layer. Word2vec utilizes two architectures: the continuous bag of words and skip-gram. For semantically meaningful representations, their training relies on conditional probabilities that can be viewed as predicting some words using some of their surrounding words in corpora.

In the following, we proceed to explain them briefly.

1.3.1 The Skip-Gram Model

The skip-gram model aims to predict all the contextual words given only the central word. Here, each word has two d -dimensional-vector representations for calculating conditional probabilities. That is, for any word with index i in the dictionary, we denote by $v_i \in \mathbb{R}^d$ and $u_i \in \mathbb{R}^d$ its two vectors when used as a center word and a context word, respectively. The conditional probability of generating any context word w_o (with index o in the dictionary) given the center word w_c (with index c in the dictionary) can be modeled by:

$$P(w_o|w_c) = \frac{\exp(u_o^T v_c)}{\sum_{i \in \mathcal{V}} \exp(u_i^T v_c)}, \quad (1.2)$$

where the vocabulary index set $\mathcal{V} = \{0, 1, \dots, |\mathcal{V}| - 1\}$.

Consider a text sequence of length T , where the word at time step t is denoted as $w^{(t)}$. Assume that context words are independently generated given any center word. For context window size m , the likelihood function of the skip-gram model is the probability of generating all context words given any center word:

$$\prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w^{(t+j)}|w^{(t)}), \quad (1.3)$$

where any time step that is less than 1 or greater than T can be omitted.

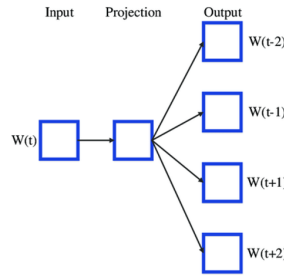


Figure 1.1: The Skip-Gram Model Architecture.

The skip-gram model parameters are the center word vector and context word vector for each word in the vocabulary. In training, we learn the model parameters by maximizing the likelihood function, i.e., maximum likelihood estimation, a method of estimating the parameters of an assumed probability distribution, given some observed data. This is equivalent to minimizing the following loss function:

$$-\sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w^{(t+j)}|w^{(t)}). \quad (1.4)$$

When using stochastic gradient descent¹ to minimize the loss, in each iteration we can randomly sample a shorter subsequence to calculate the (stochastic) gradient for this subsequence to update the model parameters. To calculate this (stochastic) gradient, we need to obtain the gradients of the log conditional probability with respect to the center word vector and the context word vector. In general, according to 1.2, the log conditional probability involving any pair of the center word and the context word w_0 is

$$\log P(w_0|w_c) = u_0^T v_c - \log \left(\sum_{i \in \mathcal{V}} \exp(u_i^T v_c) \right). \quad (1.5)$$

Through differentiation, we can obtain its gradient with respect to the center word vector v_c as

$$\begin{aligned} \frac{\partial \log P(w_0|w_c)}{\partial v_c} &= u_0 - \frac{\sum_{j \in \mathcal{V}} \exp(u_j^T v_c) u_j}{\sum_{i \in \mathcal{V}} \exp(u_i^T v_c)} = u_0 - \sum_{j \in \mathcal{V}} \left(\frac{\exp(u_j^T v_c)}{\sum_{i \in \mathcal{V}} \exp(u_i^T v_c)} \right) u_j \\ &= u_0 - \sum_{j \in \mathcal{V}} P(w_j|w_c) u_j. \end{aligned}$$

Note that this calculation requires the conditional probabilities of all words in the dictionary with w_c as the center word. The gradients for the other word vectors can be obtained in the same way.

After training, for any word with index i in the dictionary, we obtain both word vectors v_i (as the center word) and u_i (as the context word). In natural language processing applications, the center word vectors of the skip-gram model are typically used as the word representations.

1.3.2 The Continuous Bag of Words Model

On the other hand, the continuous bag of words model assumes that a center word is generated based on its surrounding context words in the text sequence. That is, exactly the opposite task of skip-gram model.

For any word with index i in the dictionary, we denote by $v_i \in \mathbb{R}^d$ and $u_i \in \mathbb{R}^d$ its two vectors when used as a context word and a center word, respectively. The conditional probability of generating any center word w_c (with index c in the dictionary) given its surrounding context words $w_{o_1}, \dots, w_{o_{2m}}$ (with index o_1, \dots, o_{2m} in the dictionary) can be modeled by

$$P(w_c|w_{o_1}, \dots, w_{o_{2m}}) = \frac{\exp(\frac{1}{2m} u_c^T (v_{o_1} + \dots + v_{o_{2m}}))}{\sum_{i \in \mathcal{V}} \exp(\frac{1}{2m} u_i^T (v_{o_1} + \dots + v_{o_{2m}}))}, \quad (1.6)$$

¹Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.

where the vocabulary index set $\mathcal{V} = \{0, 1, \dots, |\mathcal{V}| - 1\}$.

Consider a text sequence of length T , where the word at time step t is denoted as $w^{(t)}$. For context window size m , the likelihood function of the CBOW model is the probability of generating all center words given their context words:

$$\prod_{t=1}^T P(w^{(t)} | w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)}), \quad (1.7)$$

where any time step that is less than 1 or greater than T can be omitted.

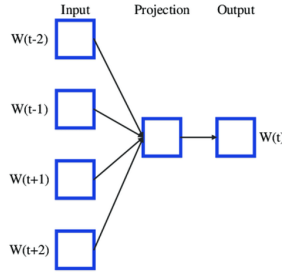


Figure 1.2: The CBOW Model Architecture.

Training continuous bag of words models is almost the same as training skip-gram models. The maximum likelihood estimation of the CBOW model is equivalent to minimizing the following loss function:

$$- \sum_{t=1}^T \log P(w^{(t)} | w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)}). \quad (1.8)$$

Notice that

$$\log P(w_o | \mathcal{W}_o) = u_c^T \bar{v}_o - \log \left(\sum_{i \in \mathcal{V}} \exp(u_i^T \bar{v}_o) \right),$$

where $\mathcal{W}_o = \{w_{o_1}, \dots, w_{o_{2m}}\}$ and $\bar{v}_o = \frac{v_{o_1}, \dots, v_{o_{2m}}}{2m}$.

Through differentiation, we can obtain its gradient with respect to any context word vector v_{o_i} ($i = 1, \dots, 2m$) as

$$\frac{\partial \log P(w_c | \mathcal{W}_o)}{\partial v_{o_i}} = \frac{1}{2m} \left(u_c - \sum_{j \in \mathcal{V}} \frac{\exp(u_j^T \bar{v}_o) u_j}{\sum_{i \in \mathcal{V}} \exp(u_i^T \bar{v}_o)} \right) = \frac{1}{2m} \left(u_c - \sum_{j \in \mathcal{V}} P(w_j | \mathcal{W}_o) u_j \right).$$

1.4 The Arithmetic of Word Vectors

An important aspect of these representations is the ability to solve word analogies of the form “A is to B what C is to X” using simple arithmetic. For example, it allows us to solve the following question: “Man is to king as woman is to _ ?” We’ll see that the following relation of sum of vectors is fulfilled:

$$u_{queen} = u_{king} - u_{man} + u_{woman}.$$

Although introducing strong independence assumptions between the elements of the context, as we already saw, the skip-gram variant is very effective in practice. Below we provide a theoretical justification for the presence of additive compositionality in word vectors learned using this model. In particular, we show that additive compositionality holds in an even stricter sense (small distance rather than small angle) under certain assumptions on the process generating the corpus. As a corollary, we explain the success of vector calculus in solving word analogies.

A natural way of capturing the compositionality of words is to say that the set of context words $C = \{c_1, \dots, c_m\}$ has the same meaning as the single word c if for every other word w , $p(w|c_1, \dots, c_m) = p(w|c)$. Although this is an intuitively satisfying definition, we never expect it to hold exactly; instead, we replace exact equality with the minimization of Kullback-Leibler divergence.

Definition 1.3. For discrete probability distributions P and Q defined on the same probability space \mathcal{X} , the *Kullback–Leibler divergence* (also called *relative entropy*) from P to Q is defined to be

$$D_{KL}(P||Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right). \quad (1.9)$$

So we state that the best candidate for having the same meaning as the set of context words C is the word

$$\operatorname{argmin}_{c \in V} D_{KL}(p(\cdot|C)||p(\cdot|c)). \quad (1.10)$$

Definition 1.4. We refer to any vector that minimizes (1.10) as a *paraphrase* of the set of words C .

There are two natural concerns with (1.10). The first is that, in general, it is not clear how to define $p(\cdot|C)$. The second is that KL-divergence minimization is a hard problem, as it involves optimization over many high dimensional probability distributions. Our main result shows that both of these problems go away for any language model that satisfies the following two assumptions:

1) For every word c , there exists Z_c such that for every word w ,

$$p(w|c) = \frac{1}{Z_c} \exp(u_c^T v_w). \quad (1.11)$$

2) For every set of words $C = \{c_1, \dots, c_m\}$, there exists Z_C such that for every word w ,

$$p(w|C) = \frac{p(w)^{1-m}}{Z_C} \prod_{i=1}^m p(w|c_i). \quad (1.12)$$

Theorem 1.5. *The skip-gram model satisfies the first assumption, and the second one too, when $m \leq \Delta$.*

Proof. Clearly, the skip-gram model satisfies the first assumption by definition. Now, let's prove that it satisfies the second one, with the restriction that $m \leq \Delta$. First, assume that $m = \Delta$. In the skip-gram model target words are conditionally independent given a context word, i.e., $p(c_1, \dots, c_m|w) = \prod_{i=1}^m p(c_i|w)$.

Applying Baye's rule,

$$\begin{aligned} p(w|c_1, \dots, c_m) &= \frac{p(c_1, \dots, c_m|w)p(w)}{p(c_1, \dots, c_m)} = \frac{p(w)}{p(c_1, \dots, c_m)} \prod_{i=1}^m p(c_i|w) \\ &= \frac{p(w)}{p(c_1, \dots, c_m)} \prod_{i=1}^m \frac{p(w|c_i)p(c_i)}{p(w)} = \frac{p(w)^{1-m}}{Z_C} \prod_{i=1}^m p(w|c_i), \end{aligned}$$

where $Z_C = \frac{1}{\prod_{i=1}^m p(c_i)}$.

This establishes the result when $m = \Delta$. The cases $m < \Delta$ follow by marginalizing out $\Delta - m$ context words in the last equality. \square

Theorem 1.6. *In every word model that satisfies assumptions 1) and 2), for every set of words $C = \{c_1, \dots, c_m\}$, any paraphrase c of C satisfies:*

$$\sum_{w \in V} p(w|c)v_w = \sum_{w \in V} p(w|C)v_w. \quad (1.13)$$

Proof. Note that

$$\begin{aligned} p(w|C) &= \frac{p(w)^{1-m}}{Z_C} \prod_{i=1}^m p(w|c_i) = \frac{p(w)^{1-m}}{Z_C} \exp\left(\sum_{i=1}^m u_{c_i}^T v_w - \sum_{i=1}^m \log Z_{c_i}\right) \\ &= \frac{1}{Z} p(w)^{1-m} \exp(u_C^T v_w), \end{aligned}$$

where $Z = Z_C \prod_{i=1}^m Z_{c_i}$, and $u_C = \sum_{i=1}^m u_{c_i}$.

Minimizing the KL-divergence $D_{KL}(P(\cdot|c_1, \dots, c_m) || p(\cdot|c))$ as a function of c is equivalent to maximizing the negative cross-entropy as a function of u_c , i.e., as maximizing $Q(u_c) = Z \sum_w \frac{\exp(u_c^T v_w)}{p(w)^{m-1}} (u_c^T v_w - \log Z_c)$.

Since Q is concave, the maximizers occur where its gradient vanishes. As

$$\begin{aligned} \nabla_{u_c} Q &= Z \sum_w \frac{\exp(u_c^T v_w)}{p(w)^{m-1}} \left[v_w - \frac{\sum_{l=1}^n \exp(u_c^T v_l) v_l}{\sum_{k=1}^n \exp(u_c^T v_k)} \right] = \frac{\sum_{l=1}^n \exp(u_c^T v_l) v_l}{\sum_{k=1}^n \exp(u_c^T v_k)} \\ &\quad - Z \sum_w \frac{\exp(u_c^T v_w) v_w}{p(w)^{m-1}} = \sum_{w \in V} p(w|c) v_w - \sum_{w \in V} p(w|c_1, \dots, c_m) v_w, \end{aligned}$$

we see that theorem 1.6 follows. \square

Theorem 1.7. *In every word model that satisfies assumptions 1) and 2) and where $p(w) = \frac{1}{|V|}$ for every $w \in V$, the paraphrase of $C = \{c_1, \dots, c_m\}$ is*

$$u_1 + \dots + u_m. \quad (1.14)$$

Proof. Recall that $u_c = \sum_{i=1}^m u_i$. When $p(w) = \frac{1}{|V|}$ for all $w \in V$, the negative cross-entropy simplifies to $Q(u_c) = Z \sum_w \exp(u_c^T v_w) (u_c^T v_w - \log Z_c)$, and its gradient $\nabla_{u_c} Q$ to

$$Z \sum_w \exp(u_c^T v_w) \left[v_w - \frac{\sum_{l=1}^n \exp(u_c^T v_l) v_l}{\sum_{k=1}^n \exp(u_c^T v_k)} \right] = Z \sum_w \exp(u_c^T v_w) v_w - \sum_w \exp(u_c^T v_w) v_w.$$

Thus, $\nabla Q(u_c) = 0$ and since Q is concave, u_c is its unique maximizer. \square

With all the established theory, we are now ready to show that the word vectors encode the semantic relations through linear translations.

Corollary 1.8. Let's rescue the presented case. *Man* and *Woman* share the relationship *Male – Female*. This means that *Man* is the paraphrase of $\{\text{Woman}, X\}$, where X is the abstract set of all the words that encode the relationship *Male – Female*.

Since *King* and *Queen* share the same *Male – Female* relationship, then *King* is the paraphrase of the set of words $\{\text{Queen}, X\}$.

Using Theorem 1.7, we get:

$$\begin{cases} u_{man} = u_{woman} + u_X \\ u_{king} = u_{queen} + u_X \end{cases} \implies u_{queen} = u_{king} - u_{man} + u_{woman}.$$

In the following illustration, we can see more examples about this phenomenon.

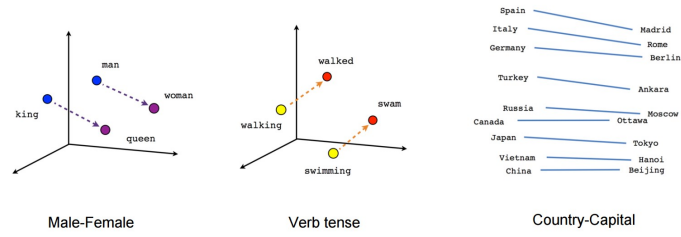


Figure 1.3: Algebra with Word Embeddings.

Chapter 2

Long Short-Term Memory Network

Once we have introduced word embeddings, so we know how to transform words into machine-readable vectors, we can proceed to explain how a RNN works, since it processes a sequence of vectors one by one.

The goal of this chapter is to present the structure of a long short-term memory, a deep learning architecture based on a recurrent neural network. [Section 2.1](#) presents the main idea of recurrent neural networks, [Sections 2.2](#) and [2.3](#) explains the structure and architecture of an LSTM, including an explanation of its activation functions, gates and equations. To go further, in [Section 2.4](#) a theorem is detailed, talking about the topology of the layers when applying the activation functions. Finally, [Section 2.5](#) details the most relevant hyperparameters to have a look at.

2.1 Recurrent Neural Networks

A recurrent neural network is a type of neural network which uses sequential data or time series data. *Sequential data* refers to data whose sequence is important for it to have meaning, such as text. RNNs are good at modelling this type of data due to the ability to retain memory about a sequence. In order to do it, RNNs have a looping mechanism that is used to remember the words that came before each word in the sentence.

Generally, every neural network consists of vertically stacked components that are called *layers*. There are three types:

- An *input layer* that takes as input the raw data and passes them to the rest of the network.

- One or more *hidden layers* that are intermediate layers located between the input and output layer and process the data by applying complex non-linear functions to them. More specifically, the function applies weights to the inputs and directs them through an activation function¹ as the output.
- An *output layer* that takes as input the processed data and produces the final results.

A simple RNN can be illustrated by the following:

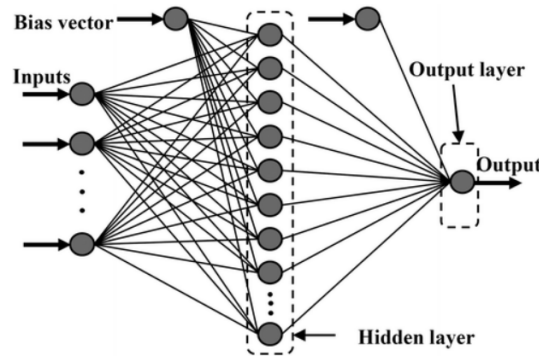


Figure 2.1: A Recurrent Neural Network with One Hidden Layer.

However, RNNs are not good at capturing long-term dependencies, because of gradient vanishing and exploding problems. One approach to tackle such problems is to use different activation functions, composed of gating units. A very successful attempt at this is the long short-term memory, which made it possible to capture distant dependencies between data.

Long short-term memory networks were introduced by Hochreiter & Schmidhuber in 1997. They are a special kind of RNN, able to retain memory over longer periods of time. They have internal mechanisms called gates that can regulate the flow of information, that allows them to decide whether to keep certain information from the previous input or to forget it.

2.2 Activation Functions in a LSTM

Definition 2.1. The mathematical representation of *sigmoid activation function* is:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (2.1)$$

¹Basically, an activation function is just a simple function that helps regulate the values flowing through the network, changing its inputs into outputs with a defined range.

The sigmoid activation function receives input and translates the output values between 0 and 1.

Definition 2.2. The mathematical representation of *tanh activation function* is:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (2.2)$$

On the other hand, the *tanh* activation function squishes values to always be between -1 and 1.

2.3 LSTM Architecture

To better understand diagrams that will come next, first let's give a detailed definition about *point-wise product*, or also called *element-wise product*, a point-wise operation we will use.

Definition 2.3. Given two functions $f, g : X \rightarrow Y$, the *point-wise product*

$$(f \cdot g) : X \rightarrow Y \text{ is defined by } (f \cdot g)(x) = f(x) \cdot g(x).$$

For the element-wise multiplication, we use the dot with the outer circle symbol, \odot , referred to as the *Hadamard product*.

Definition 2.4. For two matrices A and B of the same dimension $m \times n$, the *Hadamard product* $A \odot B$ is a matrix of the same dimension as the operands, with elements given by $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$.

In order to better understand the formulas that follow, mention that we can still perform point-wise operations by invoking the *broadcasting mechanism*. This mechanism first expand one or both arrays by copying elements appropriately so that after this transformation, the two tensors have the same shape. Second, carry out the point-wise operations on the resulting arrays. In this way, we can add matrices with vectors, even if they have different dimensions.

Also comment that the sigma function is applied to each element of the matrix, since it takes real values. To leave already the explanation of the symbols in the diagrams: each *line* carries one vector, from the output of one node to the input of one of more nodes. *Circles* mean point-wise operations and *boxes* are neural network layers. A *line merge* represents a concatenation of vectors, while a *line fork* symbolizes a copy of vectors.

2.3.1 Input Gate, Forget Gate and Output Gate

The data feeding into the LSTM gates are the input at the current time step and the hidden state of the previous time step. They are processed by three fully-connected layers with a sigmoid activation function to compute the values of the input, forget and output gates. As a result, values of the three gates are in the range of $(0, 1)$.

Mathematically, suppose that there are h hidden units, the batch size is n , and the number of inputs is d . Thus, the input is $X_t \in \mathbb{R}^{n \times d}$ and the hidden state of the previous time step is $H_{t-1} \in \mathbb{R}^{n \times h}$. Correspondingly, the gates at time step t are defined as follows: the input gate is $I_t \in \mathbb{R}^{n \times h}$, the forget gate is $F_t \in \mathbb{R}^{n \times h}$ and the output gate is $O_t \in \mathbb{R}^{n \times h}$.

They are calculated as follows:

$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i), \quad (2.3)$$

$$F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f), \quad (2.4)$$

$$O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o), \quad (2.5)$$

where $W_{xi}, W_{xf}, W_{xo} \in \mathbb{R}^{d \times h}$ and $W_{hi}, W_{hf}, W_{ho} \in \mathbb{R}^{h \times h}$ are weight parameters and $b_i, b_f, b_o \in \mathbb{R}^{1 \times h}$ are bias parameters.

Remark 2.5. It is worth mentioning that we have to take into account that the above equations are for only a one-time step. This means that these equations will have to be recomputed for the next time step. Thus, if we have a sequence of 5 time steps, then the above equations will be computed 5 times, one for each time step, respectively.

Remark 2.6. Also, add up that the weight matrices $W_{xi}, W_{xf}, W_{xo}, W_{hi}, W_{hf}, W_{ho}$ and biases b_i, b_f, b_o are not time-dependent. This means that these weight matrices don't change from one time step to another. In other words, to calculate the outputs of different time steps same weight matrices are used.

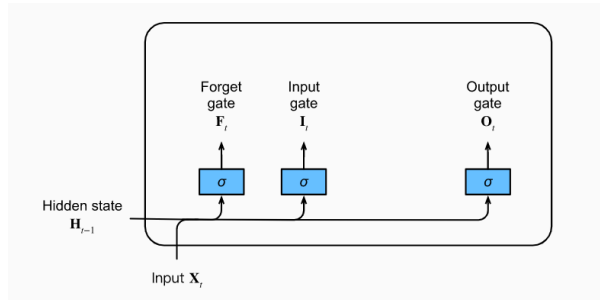


Figure 2.2: Computing the Input, Forget and Output Gates in a LSTM Model.

2.3.2 Candidate Memory Cell

Next we introduce the candidate memory cell $\tilde{C}_t \in \mathbb{R}^{nh}$. Its computation is similar to that of the three gates described above, but using a \tanh function as the activation function. This leads to the following equation at time step t :

$$\hat{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c), \quad (2.6)$$

where $W_{xc} \in \mathbb{R}^{dxh}$ and $W_{hc} \in \mathbb{R}^{hxh}$ are weight parameters and $b_c \in \mathbb{R}^{1xh}$ is a bias parameter.

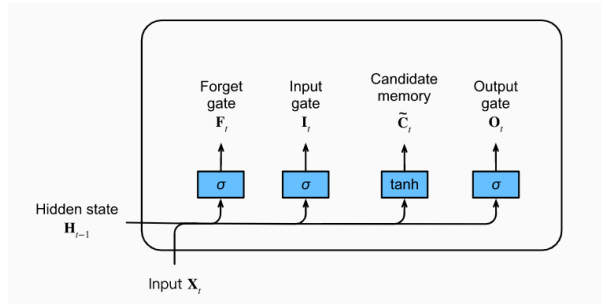


Figure 2.3: Computing the Candidate Memory Cell in a LSTM Model.

2.3.3 Memory Cell

It's now time to update the old cell state C_{t-1} , into the new cell state C_t . The previous steps already decided what to do, we just need to actually do it. To govern input and forgetting we have two dedicated gates for such purposes: the input gate I_t governs how much we take new data into account via \tilde{C}_t and the forget gate F_t addresses how much of the old memory cell content $C_{t-1} \in \mathbb{R}^{nh}$ we retain. We arrive at the following update equation:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t. \quad (2.7)$$

If the forget gate is always approximately 1 and the input gate is always approximately 0, the past memory cells C_{t-1} will be saved over time and passed to the current time step. This design is introduced to alleviate the vanishing gradient problem and to better capture long range dependencies within sequences.

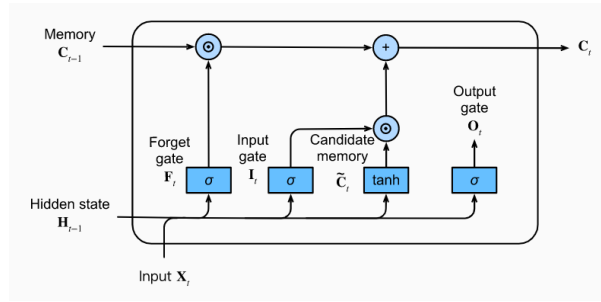


Figure 2.4: Computing the Memory Cell in a LSTM Model.

2.3.4 Hidden State

Finally, we need to decide what we're going to output, so we need to define how to compute the hidden state $H_t \in \mathbb{R}^{n \times h}$. We put the cell state C_t through \tanh and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to, that is:

$$H_t = O_t \odot \tanh(C_t). \quad (2.8)$$

Whenever the output gate approximates 1 we effectively pass all memory information through to the predictor, whereas for the output gate close to 0 we retain all the information only within the memory cell and perform no further processing.

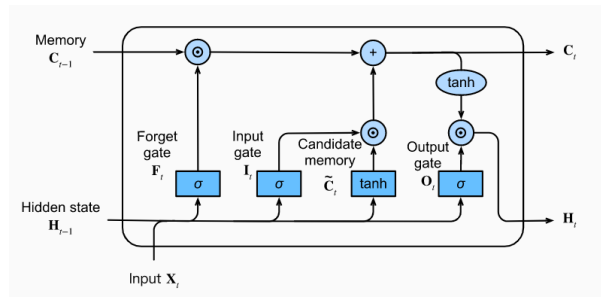


Figure 2.5: Computing the Hidden State in a LSTM Model.

2.4 Topology of tanh and Sigmoid Layers

Each layer stretches and squishes space, but it never cuts, breaks or folds it. Intuitively, we can see that it preserves topological properties. For example, a set will be connected afterwards if it was before (and vice versa).

Transformations like this, which don't affect topology, are called *homeomorphisms*. Formally, they are bijections that are continuous functions both ways.

Theorem 2.7. Layers with N inputs and N outputs are homeomorphisms, if the weight matrix W is non-singular. (Though one needs to be careful about domain and range.)

Proof. Let's consider this step by step:

- First, let's assume W has a non-zero determinant. Then it is a bijective linear function with a linear inverse. Linear functions are continuous, so multiplying by W is a homeomorphism.
- Second, remember that translations are homeomorphisms.
- Last, *tanh* and sigmoid are continuous functions with continuous inverses. They are bijections if we are careful about the domain and range we consider. Applying them point-wise is a homeomorphism.

Thus, if W has a non-zero determinant, our layer is a homeomorphism. \square

Remark 2.8. This result continues to hold if we compose arbitrarily many of these layers together.

2.5 Hyperparameters

When designing a model, the most important step among all is perhaps the training of such a model so that it is capable of making robust predictions in any new testing data. It is thus pertinent to choose a model's hyperparameters (parameters whose values are used to control the learning process) in such a way that training is effective in terms of both time and fit.

It is important to have in mind that tuning the most relevant hyperparameters for LSTM makes a difference in order to get the optimum results. To achieve it, in this section we mention the most relevant.

2.5.1 Number of Nodes and Hidden Layers

There is no final number on how many nodes (hidden neurons) or hidden layers one should use, so depending on the individual problem a trial and error approach will give the best results.

As a general rule, one hidden layer will work with most simple problems and two layers with reasonably complex ones. Also, while many nodes within a layer can increase accuracy, fewer number of nodes may cause underfitting².

2.5.2 Number of Units in a Dense Layer

A dense layer is the most frequently used, which is basically a layer where each neuron receives input from all neurons in the previous layer — thus, “densely connected”. Dense layers improve overall accuracy and 5–10 units or nodes per layer is a good base. So the output shape of the final dense layer will be affected by the number of neuron or units specified.

2.5.3 Dropout

Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network.

Every LSTM layer should be accompanied by a dropout layer. Such a layer helps avoid overfitting³ in training by bypassing randomly selected neurons, thereby reducing the sensitivity to specific weights of the individual neurons. While dropout layers can be used with input layers, they shouldn’t be used with output layers as that may mess up the output from the model and the calculation of error. While adding more complexity may risk overfitting (by increasing nodes in dense layers or adding more number of dense layers and have poor validation accuracy), this can be addressed by adding dropout.

2.5.4 Weight Initialization

Ideally, it is better to employ different weight initialization schemes according to what activation function is used. However, more commonly a uniform distribution is used while choose initial weight values. It is not possible to set all weights to 0.0 as the asymmetry in the error gradient is brought out by the optimization algorithm; to begin searching effectively. Different set of weights results in different starting points of the optimization process, potentially leading to different final sets with different performance characteristics. Weights should finally

²Underfitting happens when a machine learning model is not complex enough to accurately capture relationships between a dataset’s features and a target variable.

³Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit to additional data or predict future observations reliably.

be initialized randomly to small numbers (an expectation of the stochastic optimization algorithm, otherwise known as stochastic gradient descent) to harness randomness in the search process.

2.5.5 Learning Rate

This hyperparameter defines how quickly the network updates its parameters. Setting a higher learning rate accelerates the learning but the model may not converge or even diverge. Conversely, a lower rate will slow down the learning drastically as steps towards the minimum of loss function will be tiny, but will allow the model to converge smoothly.

2.5.6 Momentum

The momentum hyperparameter has been researched into to integrate with RNN and LSTM. Momentum is a unique hyperparameter which allows the accumulation of the gradients of the past steps to determine the direction to go with, instead of using the gradient of only the current step to guide the search.

2.5.7 Number of Epochs

This hyperparameter sets how many complete iterations of the dataset is to be run. While theoretically this number can be set to an integer value between one and infinity, this should be increased until the validation accuracy starts to decrease even though training accuracy increases (and hence risking overfitting).

2.5.8 Batch Size

This hyperparameter defines the number of samples to work on before the internal parameters of the model are updated. Large sizes make large gradient steps compared to smaller ones for the same number of samples “seen”.

Chapter 3

Logistic Regression Model

Now we slightly change the point of view and focus on a more statistical and probabilistic environment for the following two chapters. In this one we study logistic regression closely. As preparation for this chapter, we first introduce in [Section 3.1](#) generalized linear models and its components, to understand better the model we are dealing with. In [Sections 3.2](#) and [3.3](#) we present its definition and parameter interpretation, as well as we present the logistic function. [Section 3.4](#) generalize the model to multiple predictors, some of which may be qualitative, and the one that we use in our program to not only classify negative and positive, but neutral cases too. Later, [Sections 3.5](#) and [3.6](#) try to reveal the logic of fitting logistic regression models. Finally, [Sections 3.7](#) and [3.8](#) talk about the cost function and how we can utilize gradient descent to compute the minimum cost. The last section, [Section 3.9](#), describes the main assumptions about building a LR model.

3.1 Generalized Linear Models

Generalized linear models have three components: first, the *random component*, to identify the response variable Y and its probability distribution. Second, the *linear predictor*, to specify the explanatory variables through a prediction equation that has linear form. And last, the *link function*, that specifies a function of $E(Y)$ that the GLM relates to the linear predictor. Let's see a better definition of each one.

Definition 3.1. The *random component* of a GLM identifies the response variable Y and assumes a probability distribution for it. Standard GLMs treat the n observations on Y as independent. We denote those observations by (y_1, \dots, y_n) .

The assumed distribution will depend on the data we are working with. When the observations are binary, we assume a binomial distribution for Y . For example,

to model success or failure, each y_i might be the number of successes out of a certain fixed number of trials.

On the other hand, when each observation is a count, we then assume a distribution for Y that is defined on all the nonnegative integers, usually the Poisson or the negative binomial.

Otherwise, if each observation is continuous, such as a subject's weight in a dietary study, we might assume a normal or a gamma distribution for Y .

Definition 3.2. The *linear predictor* of a GLM specifies the explanatory variables. The name reflects that the variables enter linearly as predictors on the right-hand side of the model equation, in the form:

$$\alpha + \beta_1 x_1 + \dots + \beta_p x_p. \quad (3.1)$$

Definition 3.3. The expected value $\mu = E(Y)$ of the probability distribution of Y has a value that varies according to values of the explanatory variables. The *link function*, specifies a function g that relates μ to the linear predictor as

$$g(\mu) = \alpha + \beta_1 x_1 + \dots + \beta_p x_p. \quad (3.2)$$

The link function g connects the random component with the linear predictor function of the explanatory variables. Link function literally "links" the linear predictor and the parameter for probability distribution.

Definition 3.4. The *general equation* for GLM is:

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \dots + \beta_k X_k. \quad (3.3)$$

Remark 3.5. The β 's in a GLM are coefficients or weights assigned to the predictor variables, i.e., the X 's on the right hand side of the prediction equation. The first β , β_0 , is a constant. That is, it is the same for every observation regardless of any values on any of the X 's.

Instead, the other β 's are all associated with a variable. Because the variable is multiplied by the β , the β is a "weight" that determines how much the X contributes to prediction.

We can note an interesting property about these coefficients and a very simple proof of its interpretation.

Theorem 3.6. A β gives the predicted change in Y for a one unit change in the X , keeping everything else constant.

Proof. Assume GLM equation of the form of equation 3.3 and concentrate on the i th X . We can write this equation as:

$$\hat{Y}_0 = \dots + \beta_i X_i. \quad (3.4)$$

Now change the value of X_i from X_i to $X_i + 1$. The predicted value is now:

$$\hat{Y}_1 = \dots + \beta_i X_i + 1. \quad (3.5)$$

Subtracting equation 3.4 from 3.5 gives:

$$\hat{Y}_1 - \hat{Y}_0 = \beta_i X_i + 1 - \beta_i X_i = \beta_i. \quad (3.6)$$

□

So a β gives the predicted change in Y for a one unit increase in X .

Remark 3.7. Note carefully that the actual magnitude of a β 's is a function of the units of measurement of its X . Suppose X was measured in milligrams. The β would give the predicted change in \hat{Y} for a one milligram increase in X . If we changed the scale of X from milligrams to micrograms, then the β in the new equation would give the change in \hat{Y} from a one microgram change in X . One can therefore arbitrarily make a \hat{Y} larger or smaller by simply changing the scale of its variable.

This scale property of β leads to one of the most important cautions in interpreting the results from a GLM: never compare the β 's across variables to determine the importance of the variables in prediction.

3.2 Interpreting Parameters in Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression. Very simplistically explained, logistic regression works as follows.

Definition 3.8. For a binary response variable Y and an explanatory variable X , let $\pi(x) = P(Y = 1|X = x) = 1 - P(Y = 0|X = x)$. The *logistic regression model* is

$$\pi(x) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}. \quad (3.7)$$

Definition 3.9. Equivalently, the log odds, called the *logit*, has the linear relationship

$$\text{logit}[\pi(x)] = \log \frac{\pi(x)}{1 - \pi(x)} = \beta_0 + \beta_1 x. \quad (3.8)$$

Notice that this equates the logit link function to the linear predictor.

3.3 Logistic Function

As we design the logistic regression algorithm, one of the things we might naturally want is for the hypothesis $h_\theta(x)$ to output values $\in \{0, 1\}$. To that end, we choose our hypotheses $h_\theta(x)$ as

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where g is a sigmoid function which we already defined in 2.1.

Since $\theta^T x$ could be < 0 and > 1 , which is not very natural for a binary classification problem with labels $\in \{0, 1\}$, logistic regression passes it through the sigmoid function $g(z)$ which forces the input from $\in (-\infty, \infty)$ to an output $\in (0, 1)$.

The logistic sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

3.4 The Multinomial Logistic Regression Model

In the previous lines we focused on the use of the logistic regression model when the outcome variable is dichotomous or binary. This model can be easily modified to handle the case where the outcome variable is nominal with more than two levels.

Definition 3.10. Like ordinary regression, *multinomial logistic regression* extends to models with multiple explanatory variables. For instance, the model for $\pi(x) = P(Y = 1)$ at values $x = (x_1, \dots, x_p)$ of p predictors is

$$\text{logit}[\pi(x)] = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p. \quad (3.9)$$

Remark 3.11. The alternative formula, directly specifying $\pi(x)$, is

$$\pi(x) = \frac{e^{\alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p}}{1 + e^{\alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p}}. \quad (3.10)$$

The parameter β_i refers to the effect of x_i on the log odds that $Y = 1$, controlling the other x_j . For instance, e^{β_i} is the multiplicative effect on the odds of a 1-unit increase in x_i , at fixed levels of other x_j .

3.5 Fitting Logistic Regression Models

The coefficients β of the logistic regression algorithm are unknown, and must be estimated based on the available training data. This is done using maximum-likelihood estimation. In a general sense, the method of maximum likelihood yields values for the unknown parameters that maximize the probability of obtaining the observed set of data. In order to apply this method we must first construct a function, called the *likelihood function*. This function expresses the probability of the observed data as a function of the unknown parameters.

Definition 3.12. Assuming that the m training examples are generated independently, we can write down the *likelihood* of the parameters as

$$\mathcal{L}(\theta) = \prod_{i=1}^m \left((h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}} \right).$$

Since it is easier to maximize the log likelihood rather than the likelihood, and we will need this formula in a further section, we have the following:

$$\log \mathcal{L}(\theta) = \sum_{i=1}^m \left(y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right).$$

Now, per maximum likelihood estimation, we need to find the values that maximize this function. Thus, the resulting estimators are those that agree most closely with the observed data. To find this values, we need to run the gradient ascent. The update rule is given by

$$\theta \leftarrow \theta + \alpha \frac{\partial}{\partial \theta} \log \mathcal{L}(\theta).$$

3.6 Maximum Entropy Models

In information theory¹, the entropy of a random variable is the average level of "information", "surprise" or "uncertainty" inherent to the variable's possible outcomes. Mathematically:

Definition 3.13. Given a discrete random variable X , with possible outcomes x_1, \dots, x_n which occur with probability $P(x_1), \dots, P(x_n)$, the *entropy* of X is formally defined as:

$$H(X) = - \sum_{i=1}^n P(x_i) \log P(x_i). \quad (3.11)$$

¹Information theory is the scientific study of the quantification, storage, and communication of digital information.

Note: The choice of base for log, the logarithm, varies for different applications.

Classification in machine learning often employs a standard loss function, called cross entropy loss, that minimizes the average cross entropy between ground truth and predicted distributions.

Definition 3.14. The *cross-entropy* of the distribution q relative to a distribution p over a given set is defined as follows:

$$H(p, q) = -E_p[\log(q)], \quad (3.12)$$

where $E_p[\cdot]$ is the expected value operator with respect to the distribution p .

Remark 3.15. The definition may be formulated using the Kullback–Leibler divergence, $D_{KL}(p\|q)$, which we already defined in 1.3:

$$H(p, q) = H(p) + D_{KL}(p \parallel q).$$

Theorem 3.16. *Maximizing the likelihood with respect to the parameters θ is the same as minimizing the cross-entropy.*

Proof. Given N conditionally independent samples in the training set, then the likelihood of the parameters θ of the model $q_\theta(X = x)$ on the training set is:

$$\mathcal{L}(\theta) = \prod_i q_\theta(X = i)^{Np(X=i)},$$

so the log-likelihood, divided by N is

$$\frac{1}{N} \log(\mathcal{L}(\theta)) = \frac{1}{N} \log \prod_i q_\theta(X = i)^{Np(X=i)} = \sum_i p(X = i) \log q_\theta(X = i) = -H(p, q),$$

so that maximizing the likelihood with respect to the parameters θ is the same as minimizing the cross-entropy. \square

3.7 Cost Function

The cost function summarizes how well the model is behaving. In other words, it tells us how close the model's predictions are to the actual outputs for a given set of parameters. The cost function has its own curve and its own gradients. The slope of this curve tells us how to update our parameters to make the model more accurate.

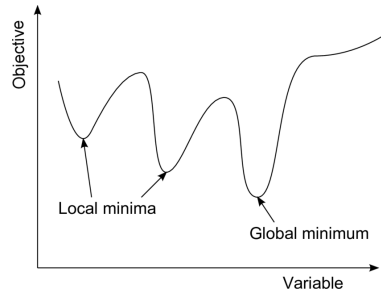


Figure 3.1: Global and local minima

In linear regression, we use MSE as the cost function, but in logistic regression, using the mean of the squared differences between actual and predicted outcomes as the cost function might give a wavy, non-convex solution; containing many local optima, as we can see in Figure 3.1.

In this case, finding an optimal solution with the gradient descent method is not possible. Instead, we use a logarithmic function to represent the cost of logistic regression. It is guaranteed to be convex for all input values, containing only one minimum, allowing us to run the gradient descent algorithm.

Precisely, the cost function used in logistic regression is *Log Loss*. Let's see its formal definition.

Definition 3.17. The *Log Loss*, which is the negative average of the log of corrected predicted probabilities for each instance, is

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right), \quad (3.13)$$

where the logs are natural logarithms.

Computing the gradient, which we will need in the next section, requires the partial derivative of the loss function with respect to each parameter. For this reason we expose the following theorem.

Theorem 3.18. (Partial Derivative of the Cost Function for Logistic Regression)

The partial derivative of the logistic regression cost function with respect to θ is

$$\frac{\partial J(\theta)}{\partial \theta_j} = \nabla_{\theta_j} J(\theta) = \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}). \quad (3.14)$$

Proof. Remember our original cost function is of the form:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right).$$

Now, $\log h_\theta(x^{(i)}) = \log \frac{1}{1+e^{-\theta x^{(i)}}} = -\log(1 + e^{-\theta x^{(i)}})$, and

$$\begin{aligned} \log(1 - h_\theta(x^{(i)})) &= \log\left(1 - \frac{1}{1 + e^{-\theta x^{(i)}}}\right) = \log(e^{-\theta x^{(i)}}) - \log(1 + e^{-\theta x^{(i)}}) \\ &= -\theta x^{(i)} - \log(1 + e^{-\theta x^{(i)}}). \end{aligned}$$

Plugging in the two simplified expressions above in our original cost function, we obtain:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left(-y^{(i)}(\log(1 + e^{-\theta x^{(i)}})) + (1 - y^{(i)})(-\theta x^{(i)} - \log(1 + e^{-\theta x^{(i)}})) \right),$$

which can be simplified to:

$$\begin{aligned} J(\theta) &= -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)}\theta x^{(i)} - \theta x^{(i)} - \log(1 + e^{-\theta x^{(i)}}) \right) \\ &= -\frac{1}{m} \sum_{i=1}^m \left(y^{(i)}\theta x^{(i)} - \log(1 + e^{\theta x^{(i)}}) \right), \end{aligned}$$

where the second equality follows from:

$$-\theta x^{(i)} - \log(1 + e^{-\theta x^{(i)}}) = -[\log(e^{\theta x^{(i)}}) + \log(1 + e^{-\theta x^{(i)}})] = -\log(1 + e^{\theta x^{(i)}}).$$

Now, it only remains to compute the partial derivative of the last equation with respect to θ_j using the following:

$$\frac{\partial}{\partial \theta_j} y^{(i)}\theta x^{(i)} = y^{(i)}x_j^{(i)} \frac{\partial}{\partial \theta_j} \log(1 + e^{\theta x^{(i)}}) = \frac{x_j^{(i)} e^{\theta x^{(i)}}}{1 + e^{\theta x^{(i)}}} = x_j^{(i)} h_\theta(x^{(i)}).$$

Finally, plugging in the two components above in the expression for $\frac{\partial J(\theta)}{\partial \theta_j}$, we obtain the end result:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^m \left((h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \right).$$

□

3.8 Gradient Descent

We already have all the tools to investigate how we can utilize the gradient descent algorithm to calculate the optimal parameters.

As we already state, gradient descent is an iterative optimization algorithm, which finds the minimum of a differentiable function. In this process, we try different values and update them to reach the optimal ones, minimizing the output.

In the following discussion, we will apply this method to the cost function of logistic regression in order to find an optimal solution minimizing the cost over model parameters:

$$\min_{\theta} J(\theta).$$

Let's assume we have a total of n features. In this case, we have n parameters for the θ vector. To minimize our cost function, we need to run the gradient descent on each parameter θ_j :

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

Furthermore, we need to update each parameter simultaneously for each iteration. In other words, we need to loop through the parameters $\theta_0, \theta_1, \dots, \theta_n$ in vector $\theta = [\theta_0, \theta_1, \dots, \theta_n]$.

To complete the algorithm, we need the value of $\frac{\partial}{\partial \theta_j} J(\theta)$, which we already have computed using theorem 3.18:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m \left((h_{\theta}(x^{(i)}) - y^{(i)}) x_j \right).$$

Plugging this into the gradient descent function leads to the update rule:

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m \left(h_{\theta}(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

By iterating over the training samples until convergence, we reach the optimal parameters θ leading to minimum cost.

3.9 Assumptions

We can't build a logistic regression model without understanding the basic assumptions of it. In the following lines, we are going to explain these assumptions in depth and discuss briefly the techniques to check these assumptions for the given data.

- The most critical assumption of logistic regression is that there should be little or no *multicollinearity* in the provided dataset. This condition occurs when the features or independent variables of the dataset are highly correlated to each other in a manner, that they do not contribute unique or independent information in the regression model.

If a model has correlated variables, it becomes hard to determine which variable contributes to estimating the target variable. If the level of correlation is high between variables, it leads to problems while fitting and interpreting the model.

The most popular approach to detect multicollinearity is by using the correlation matrix, which measures the correlation degree between the independent variables in a given dataset.

- Logistic regression is very sensitive to *outliers*. It assumes that there are no extreme outliers or influential observations in the given dataset. We can get unexpected outcomes due to the presence of just one outlier in our data.

The most common way to test for extreme outliers and influential observations in a dataset is to calculate Cook's distance for each observation.

Definition 3.19. *Cook's distance*, often denoted as D_i , is used in regression analysis to identify influential data points that may negatively affect your regression model. The formula for Cook's distance is:

$$D_i = \frac{r_i^2}{p * MSE} * \frac{h_{ii}}{1 - h_{ii}}$$

where r_i is the i^{th} residual, p is the number of coefficients in the regression model, MSE is the mean squared error and h_{ii} is the i_{th} leverage value.

- The last assumption we comment is that it requires quite large sample sizes. The size of the dataset should be large enough to make suitable conclusions from the logistic regression model.

Chapter 4

Naive Bayes Model

In this chapter we are going to present the theory of probabilistic classifiers, to study deeply Naive Bayes classifier, a simple one belonging to this family.

In machine learning, a probabilistic classifier is a classifier that can predict, given an observation of an input, a probability distribution over a set of classes, rather than only outputting the most likely class that the observation should belong to.

[Section 4.1](#) covers the main idea of the model and its definition. [Section 4.2](#) talks about parameter estimation and finally [Section 4.3](#) provides a theorem of sufficient and necessary conditions for the optimality of Naive Bayes. [Section 4.4](#) comment the main assumptions we make when implementing a Naive Bayes model. To conclude this chapter, and with it, the two statistical models studied, the last section, [Section 4.5](#), argue some relationship between LR and NB.

4.1 Definition

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. That is, changing the value of one feature, does not directly influence or change the value of any of the other features used in the algorithm.

Bayes' theorem states the following relationship, given class variable y and dependent feature vector (x_1, \dots, x_n) :

$$p(y|x_1, \dots, x_n) = \frac{p(y)p(x_1, \dots, x_n|y)}{p(x_1, \dots, x_n)}. \quad (4.1)$$

4.2 Parameter Estimation and Event Models

A class's prior may be calculated by assuming equiprobable classes or by calculating an estimate for the class probability from the training set. To estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric models for the features from the training set. The assumptions on distributions of features are called the *event model* of the Naive Bayes classifier. These assumptions lead to distinct models. In the following lines we present 3 examples.

4.2.1 Gaussian Naive Bayes

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. Suppose the training data contains a continuous attribute, x . The data is first segmented by the class, and then the mean and variance of x is computed in each class. Let μ_k be the mean of the values in x associated with class C_k , and let σ_k^2 be the Bessel corrected variance of the values in x associated with class C_k . Suppose one has collected some observation value v . Then:

$$p(x = v|C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}. \quad (4.2)$$

4.2.2 Multinomial Naive Bayes

With a multinomial event model, samples (feature vectors) represent the frequencies with which certain events have been generated by a multinomial (p_1, \dots, p_n) , where p_i is the probability that event i occurs. The likelihood of observing a vector $x = (x_1, \dots, x_n)$ is given by

$$p(x|C_k) = \frac{(\sum_{i=1}^n x_i)!}{\prod_{i=1}^n x_i!} \prod_{i=1}^n p_{k_i}^{x_i}. \quad (4.3)$$

4.2.3 Bernoulli Naive Bayes

In the multivariate bernoulli event model, features are independent binary variables describing inputs. If x_i is a boolean expressing the occurrence or absence of the i 'th term from the vocabulary, then the likelihood of a document given a class C_k is given by

$$p(x|C_k) = \prod_{i=1}^n p_{k_i}^{x_i} (1 - p_{k_i})^{1-x_i}. \quad (4.4)$$

4.3 The Optimality of Naive Bayes

We just saw that the different Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $p(x_i|y)$. In this section we present a sufficient condition for the optimality of Naive Bayes under the gaussian distribution, and show theoretically when Naive Bayes works well.

Typically, an example E is represented by a tuple of attribute values (x_1, x_2, \dots, x_n) , where x_i is the value of attribute X_i . Let C represent the classification variable, and let c be the value of C . The probability of an example $E = (x_1, x_2, \dots, x_n)$ being class c is

$$p(c|E) = \frac{p(E|c)p(c)}{p(E)}, \quad (4.5)$$

according to Bayes Rule.

Definition 4.1. A *Bayesian classifier* is a function defined as follows:

$$f_b(E) = \frac{p(C = +|E)}{p(C = -|E)}. \quad (4.6)$$

Assume that all attributes are independent given the value of the class variable, that is:

$$p(E|c) = p(x_1, x_2, \dots, x_n|c) = \prod_{i=1}^n p(x_i|c), \quad (4.7)$$

the resulting classifier is then:

$$f_{nb}(E) = \frac{p(C = +|E)}{p(C = -|E)} \prod_{i=1}^n \frac{p(x_i|C = +)}{p(x_i|C = -)}. \quad (4.8)$$

The function $f_{nb}(E)$ is called a *Naive Bayesian classifier*.

Definition 4.2. S_{++}^n is the space of *symmetric positive definite $n \times n$ matrices*, defined as

$$S_{++}^n = \{A \in \mathbb{R}^{n \times n} : A = A^T \text{ and } x^T A x > 0 \text{ for all } x \in \mathbb{R}^n \text{ such that } x \neq 0\}. \quad (4.9)$$

Definition 4.3. A vector-valued random variable $X = [X_1, \dots, X_n]^T$ is said to have a *multivariate normal (or Gaussian) distribution* with mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in S_{++}^n$ if its probability density function is given by

$$p(x; \mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}. \quad (4.10)$$

We write this as $X \sim \mathcal{N}(\mu, \Sigma)$.

Let us restrict our discussion to two attributes X_1 and X_2 , and assume that the class density is a multivariate gaussian in both the positive and negative classes. That is,

$$p(x_1, x_2, +) = \frac{1}{\sqrt{2\pi|\Sigma|_+}} e^{-\frac{1}{2}(x-\mu^+)^T \Sigma_+^{-1}(x-\mu^+)}, \quad (4.11)$$

$$p(x_1, x_2, -) = \frac{1}{\sqrt{2\pi|\Sigma|_-}} e^{-\frac{1}{2}(x-\mu^-)^T \Sigma_-^{-1}(x-\mu^-)}, \quad (4.12)$$

where $x = (x_1, x_2)$, Σ_+ and Σ_- are the covariance matrices in the positive and negative classes respectively, $|\Sigma|_+$ and $|\Sigma|_-$ are the determinants of Σ_+ and Σ_- , Σ_+^{-1} and Σ_-^{-1} are the inverses of Σ_+ and Σ_- , $\mu^+ = (\mu_1^+, \mu_2^+)$ and $\mu^- = (\mu_1^-, \mu_2^-)$, μ_i^+ and μ_i^- are the means of attribute X_i in the positive and negative classes respectively, and $(x - \mu^+)^T$ and $(x - \mu^-)^T$ are the transposes of $(x - \mu^+)$ and $(x - \mu^-)$.

We assume that two classes have a common covariance matrix $\Sigma_+ = \Sigma_- = \Sigma$, and X_1 and X_2 have the same variance σ in both classes.

Now we are able to follow our discussion with all the expressions defined.

When applying a logarithm to the bayesian classifier, defined in 4.1, we obtain the classifier f_b below:

$$f_b(x_1, x_2) = \log \frac{p(x_1, x_2, +)}{p(x_1, x_2, -)} = -\frac{1}{\sigma^2}(\mu^+ + \mu^-)\Sigma^{-1}(\mu^+ - \mu^-) + x^T \Sigma^{-1}(\mu^+ - \mu^-). \quad (4.13)$$

Then, because of the conditional independence assumption, we have the correspondent Naive Bayesian classifier f_{nb} :

$$f_{nb}(x_1, x_2) = \frac{1}{\sigma^2}(\mu_1^+ - \mu_1^-)x_1 + \frac{1}{\sigma^2}(\mu_2^+ - \mu_2^-)x_2. \quad (4.14)$$

Assume that $\Sigma = \begin{pmatrix} \sigma & \sigma_{12} \\ \sigma_{21} & \sigma \end{pmatrix}$.

X_1 and X_2 are independent if $\sigma_{12} = 0$. If $\sigma \neq \sigma_{12}$, we have $\Sigma^{-1} = \begin{pmatrix} \frac{-\sigma}{\sigma_{12}^2 - \sigma^2} & \frac{\sigma_{12}}{\sigma_{12}^2 - \sigma^2} \\ \frac{\sigma_{12}}{\sigma_{12}^2 - \sigma^2} & \frac{-\sigma}{\sigma_{12}^2 - \sigma^2} \end{pmatrix}$.

Note that, an example E is classified into the positive class by f_b , if and only if, $f_b \geq 0$. Similarly with f_{nb} . Thus, when f_b or f_{nb} is divided by a non-zero positive constant, the resulting classifier is the same as f_b or f_{nb} . Then,

$$f_{nb}(x_1, x_2) = (\sigma_1^+ - \sigma_1^-)x_1 + (\sigma_2^+ - \sigma_2^-)x_2, \quad (4.15)$$

and

$$f_b(x_1, x_2) = \frac{1}{\sigma_{12}^2 - \sigma^2} (\sigma_{12}(\mu_2^+ - \mu_2^-) - \sigma(\mu_1^+ - \mu_1^-))x_1 + \frac{1}{\sigma_{12}^2 - \sigma^2} (\sigma_{12}(\mu_1^+ - \mu_1^-) - \sigma(\mu_2^+ - \mu_2^-))x_2 + a, \quad (4.16)$$

where

$$a = -\frac{1}{\sigma_{12}^2 - \sigma^2} (\sigma^2(\mu^+ + \mu^-)\Sigma^1(\mu^+ \mu^-),$$

a constant independent of x .

For any x_1 and x_2 , Naive Bayes has the same classification as that of the underlying classifier if

$$f_b(x_1, x_2)f_{nb}(x_1, x_2) \geq 0. \quad (4.17)$$

That is,

$$\begin{aligned} & \frac{1}{\sigma_{12}^2 - \sigma^2} ((\sigma_{12}(\mu_1^+ - \mu_1^-)(\mu_2^+ - \mu_2^-) - \sigma(\mu_1^+ - \mu_1^-)^2)x_1^2 \\ & + (\sigma_{12}(\mu_1^+ - \mu_1^-)(\mu_2^+ - \mu_2^-) - \sigma(\mu_2^+ - \mu_2^-)^2)x_2^2 \\ & + (2\sigma_{12}(\mu_1^+ - \mu_1^-)(\mu_2^+ - \mu_2^-) - \sigma((\mu_1^+ - \mu_1^-)^2 \\ & + (\mu_2^+ - \mu_2^-)^2))x_1x_2) + a(\mu_1^+ - \mu_1^-)x_1 + a(\mu_2^+ - \mu_2^-)x_2 \geq 0. \end{aligned} \quad (4.18)$$

This last equation represents a sufficient and necessary condition for $f_{nb}(x_1, x_2) = f_b(x_1, x_2)$. Let's try to simplify this.

Let $\mu_1^+ - \mu_1^- = \mu_2^+ - \mu_2^-$. Equation is simplified as

$$w_1(x_1 + x_2)^2 + w_2(x_1 + x_2) \geq 0, \quad (4.19)$$

where $w_1 = \frac{(\mu_1^+ - \mu_1^-)^2}{\sigma_{12} + \sigma}$, and $w_2 = a(\mu_1^+ - \mu_1^-)$.

Let $x = x_1 + x_2$ and $y = w_1(x_1 + x_2)^2 + w_2(x_1 + x_2)$.

Figure (4.1) shows the area in which Naive Bayes has the same classification with the target classifier. So, this shows that under certain condition, Naive Bayes is optimal.

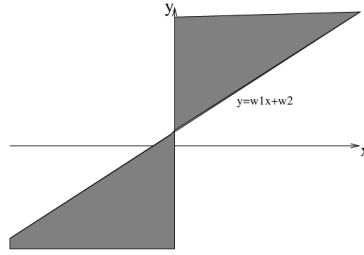


Figure 4.1: Naive Bayes has the same classification with that of the target classifier in the shaded area

The following theorem presents a sufficient condition for that Naive Bayes works exactly as the target classifier.

Theorem 4.4. $f_b = f_{nb}$, if one of the following two conditions is true:

1. $\mu_1^+ = -\mu_2^-, \mu_1^- = -\mu_2^+$, and $\sigma_{12} + \sigma > 0$.
2. $\mu_1^+ = \mu_2^-, \mu_2^+ = \mu_1^-$, and $\sigma_{12} - \sigma > 0$.

Proof.

1. If $\mu_1^+ = -\mu_2^-, \mu_1^- = -\mu_2^+$, then $(\mu_1^+ - \mu_1^-) = (\mu_2^+ - \mu_2^-)$. It is straightforward to verify that $-\frac{1}{\sigma^2}(\mu^+ + \mu^-)\Sigma^{-1}(\mu^+ - \mu^-) = 0$. That is, for the constant a in the equation 4.16, we have $a = 0$. Since $\sigma_{12} + \sigma > 0$, equation 4.19 is always true for any x_1 and x_2 . Therefore, $f_b = f_{nb}$.
2. If $\mu_1^+ = \mu_2^-, \mu_2^+ = \mu_1^-$, then $(\mu_1^+ - \mu_1^-) = -(\mu_2^+ - \mu_2^-)$, and $a = 0$. Thus, equation 4.18 is simplified as $\frac{(\mu_1^+ - \mu_1^-)^2}{\sigma_{12}}(x_1 + x_2)^2 \geq 0$. Since $\sigma_{12} - \sigma > 0$, equation 4.19 is always true for any x_1 and x_2 . Therefore, $f_b = f_{nb}$.

□

This theorem represents an explicit condition that Naive Bayes is optimal. It shows that Naive Bayes is still optimal under certain condition, even though the conditional independence assumption is violated. In other words, the conditional independence assumption is not the necessary condition for the optimality of Naive Bayes. This provides evidence that the dependence distribution may play the crucial role in classification.

4.4 Assumptions

As we already commented at the beginning of the chapter, we make two important naive assumptions:

- All features are independent from each other.

In our case, that means each word is independent from the others, so given a sentence, there is no relation between any two words.

- Every feature contributes equally to the output.

Again, for what concerns us, that means each word contributes equally to the decision of the model, regardless of its relative position in the sentence.

4.5 Relationship Between Naive Bayes Classifiers and Logistic Regression

The page limit disallows presenting complete proofs of the following results. See [NJ01] for a complete study, where Ng and Jordan (2001) present some theoretical and empirical comparisons between logistic regression and the Naive Bayes classifier.

- When the GNB modeling assumptions do not hold, logistic regression and GNB typically learn different classifier functions. In this case, the asymptotic (as the number of training examples approach infinity) classification accuracy for logistic regression is often better than the asymptotic accuracy of GNB. Although logistic regression is consistent with the Naive Bayes assumption that the input features X_i are conditionally independent given Y , it is not rigidly tied to this assumption as is Naive Bayes. Given data that disobeys this assumption, the conditional likelihood maximization algorithm for logistic regression will adjust its parameters to maximize the fit to (the conditional likelihood of) the data, even if the resulting parameters are inconsistent with the Naive Bayes parameter estimates.
- GNB and logistic regression converge toward their asymptotic accuracies at different rates. GNB parameter estimates converge toward their asymptotic values in order $\log(n)$ examples, where n is the dimension of X , the feature vector (x_1, \dots, x_n) . In contrast, logistic regression parameter estimates converge more slowly, requiring order n examples. In the paper, the authors also show that in several data sets logistic regression outperforms GNB when many training examples are available, but GNB outperforms logistic regression when training data is scarce.

Chapter 5

Results, Analysis and Comparison Between the Models Used

So far this research developed all the necessary tools required to study all the math behind the three different chosen models to classify our data. To accomplish this goal, some background in probabilistic theory has been developed, including all the necessary results to be able to understand LR and NB properly, along with the study of recurrent neural networks, required to provide a satisfactory description of the LSTM and its workflow.

Once all the theory is established, we can analyse how models behave in practice. In this chapter, we review the results of the models we built and some analysis and comparison between them. [Section 5.1](#) explains the dataset we are using to train our models, [Section 5.2](#) covers an argumentation of advantages and disadvantages of each model and [Section 5.3](#) talks about the execution time, results and metrics of the resultant programs. Finally, [Section 5.4](#) summarize our conclusions, discuss potential improvements to the models and raise future work.

5.1 About de Dataset

Data comes from internet websites, public and legal sources, given by the company UVE Solutions. The format of the input is an Excel file, and it contains 894.690 entries corresponding to reviews on social media of bars and restaurants, written in Spanish. Each row contains the text commentary (*Comment* column) and the stars received (*Score* column). Score column is the one we use to train our model in order to make predictions on the sentiment, since it is the one that will help us to label the *Sentiment* column. We will assign a 0 to scores with value 1 and 2 (negative class), a 1 to scores with value 3 (neutral class) and a 2 to scores with value 4 or 5 (positive class).

In Figures 5.1 and 5.2 we can see a histogram of the stars and sentiment class distribution, respectively:



Figure 5.1: Stars Distribution.

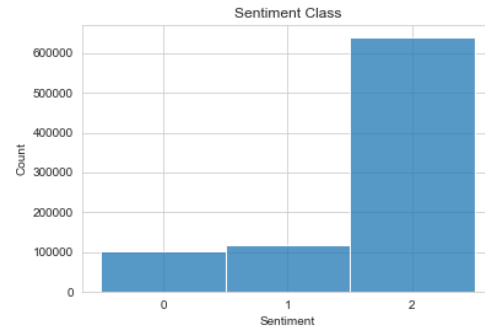


Figure 5.2: Sentiment Class Distribution.

We can observe that we have a lot of reviews with positive ratings, while we have a much smaller sample of negative and neutral commentaries. This makes us think that our models will surely have a good behavior with the positive class, and therefore a higher accuracy, than the rest of the classes.

5.2 Advantages and Disadvantages

When it comes to the execution time, Naive Bayes is the fastest one, since it only takes 4.22 minutes to execute the whole program. Instead, logistic regression and LSTM require 18.59 minutes and 5.47 minutes, respectively. So we can already state that the main advantage of the Naive Bayes model is its simplicity and fast computation time. As we studied in [Section 4.3](#), this is mainly due to its strong assumption that all events are independent of each other. Although this assumption can be incorrect in real life, we can see the model still performs very well in various scenarios.

Regarding the logistic regression model, it attempts to predict precise probabilistic outcomes based on independent features. On high dimensional datasets, like the one we are dealing with, this may lead to the model being overfit on the training set, which means overstating the accuracy of predictions on the training set and thus the model may not be able to predict accurate results on the test set. This usually happens in the case when the model is trained on little training data with lots of features. So on high dimensional datasets, regularization techniques should be considered to avoid overfitting. *Regularization* works by adding a penalty or regularization term to the loss function. It is a form of regression that

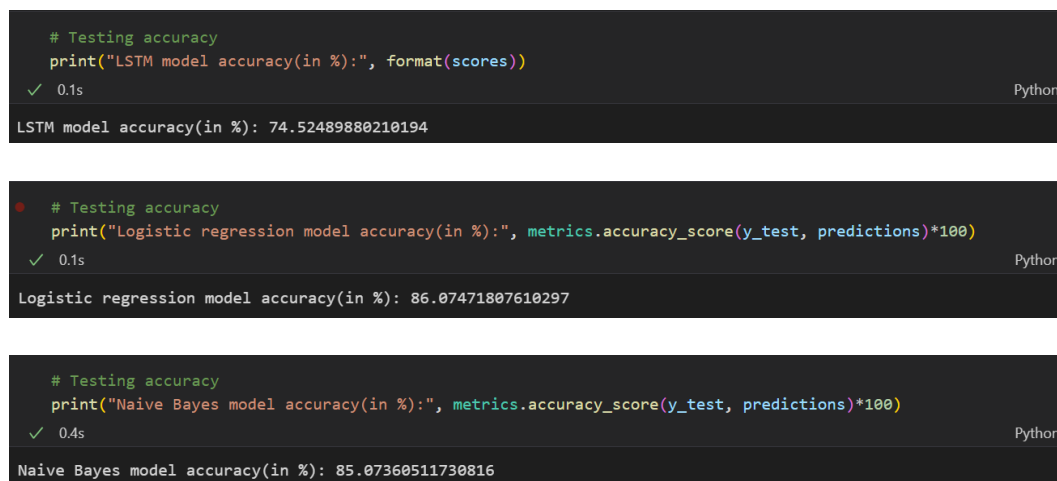
shrinks the coefficient estimates towards zero, so it forces us not to learn a more complex or flexible model.

On the subject of LSTMs, they require a lot of resources and time to get trained and become ready for the desired application. In technical terms, they need high memory-bandwidth because of linear layers present in each cell which the system usually fails to provide for. Often, they are prone to overfitting and it is difficult to apply the dropout algorithm (see [Subsection 2.5.3](#)) to curb this issue.

5.3 Results of the Model Evaluation

Extract metrics to evaluate and analyse the behavior of the models we build is essential, since that lets you see if there is something weird going on with them, how to fix it, and how to ultimately improve their performance.

The accuracy that we have obtained for each one can be seen in Figure 5.3.



```
# Testing accuracy
print("LSTM model accuracy(in %):", format(scores))
✓ 0.1s Python
LSTM model accuracy(in %): 74.52489880210194
```

```
# Testing accuracy
print("Logistic regression model accuracy(in %):", metrics.accuracy_score(y_test, predictions)*100)
✓ 0.1s Python
Logistic regression model accuracy(in %): 86.07471807610297
```

```
# Testing accuracy
print("Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, predictions)*100)
✓ 0.4s Python
Naive Bayes model accuracy(in %): 85.07360511730816
```

Figure 5.3: Testing Accuracy of the Three Models.

To go further, let's define some metrics we should take into account in order to evaluate the predictions of our models.

Definition 5.1. *Precision* metric is the ratio of True Positive and the sum of True Positive and False Positive.

Definition 5.2. *Recall* metric is the ratio of True Positive and the sum of True Positive and False Negative.

We can see a visual representation of these two metrics in Figure 5.4.

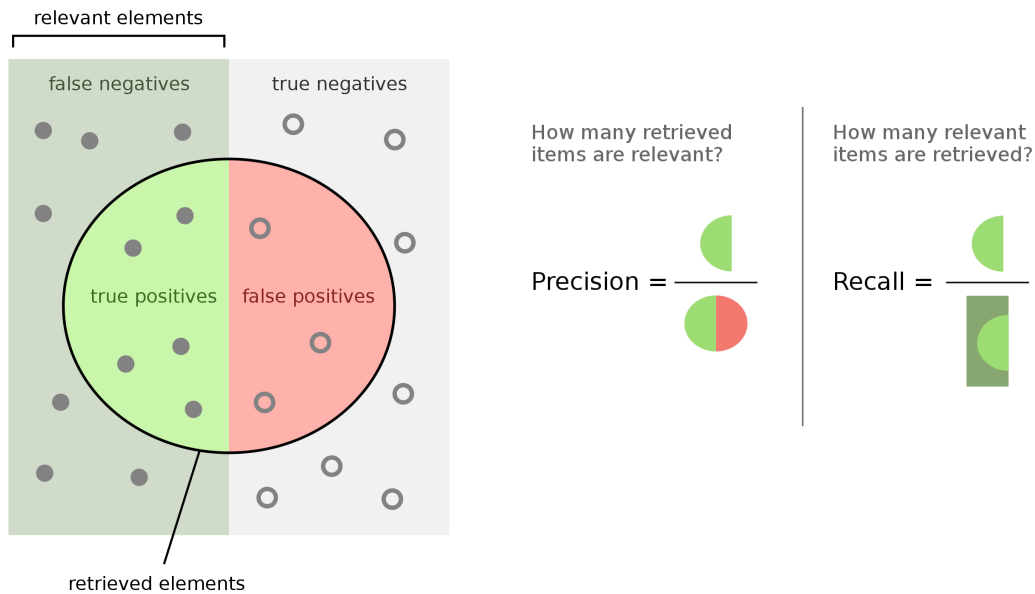


Figure 5.4: Precision and Recall Computation.

Definition 5.3. The *F1 score* metric gives the combined result of Precision and Recall. Precisely, it is a harmonic mean of Precision and Recall:

$$F1score = \frac{2}{\left(\frac{1}{Recall}\right) + \left(\frac{1}{Precision}\right)}. \quad (5.1)$$

Definition 5.4. *Support* is the number of actual occurrences of the class in the dataset. It doesn't vary between models, it just diagnoses the performance evaluation process.

In the following figures, 5.5, 5.6 and 5.7, we can see a classification report of its predictions.

Note: For the LSTM, due to memory problems, since we are not able to allocate the gibibytes we are required, we have only been able to test 5000 entries, so it gives us very low values of precision and recall. Remark these are not entirely true, since we are comparing different proportions, but we still show them to get an idea of the distribution of the predictions with a small sample.


```
# Metrics
print(classification_report(predictions, y_test))
✓ 0.2s
```

	precision	recall	f1-score	support
0	0.76	0.79	0.77	20786
1	0.41	0.56	0.48	18380
2	0.96	0.91	0.93	142918
accuracy			0.86	182084
macro avg	0.71	0.75	0.73	182084
weighted avg	0.88	0.86	0.87	182084

Figure 5.5: Classification Report for Logistic Regression.

```
# Metrics
print(classification_report(predictions, y_test))
✓ 0.3s
```

	precision	recall	f1-score	support
0	0.74	0.75	0.75	21763
1	0.42	0.53	0.47	19905
2	0.95	0.91	0.93	140984
accuracy			0.85	182652
macro avg	0.70	0.73	0.71	182652
weighted avg	0.87	0.85	0.86	182652

Figure 5.6: Classification Report for Naive Bayes.

```
# Metrics
print(classification_report(pred, y_true))
✓ 0.1s
```

	precision	recall	f1-score	support
0	0.21	0.13	0.16	976
1	0.10	0.13	0.12	515
2	0.70	0.75	0.73	3509
accuracy			0.57	5000
macro avg	0.34	0.34	0.33	5000
weighted avg	0.55	0.57	0.55	5000

Figure 5.7: Classification Report for LSTM.

Definition 5.5. A *confusion matrix* is a $N \times N$ matrix, where N is the number of target classes, used for evaluating the performance of a classification model. The matrix compares the actual target values with those predicted by the machine learning model.

We can see its structure in Figure 5.8.

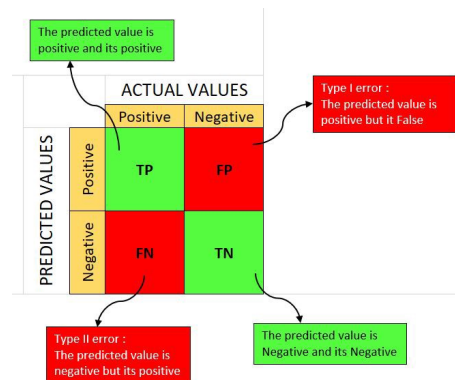


Figure 5.8: Binary Classification Problem (2x2 Matrix).

We have plotted the confusion matrices of the three models in Figures 5.9, 5.10 and 5.11:

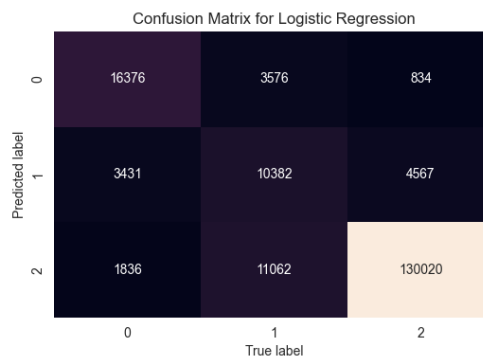


Figure 5.9: Confusion Matrix for Logistic Regression.

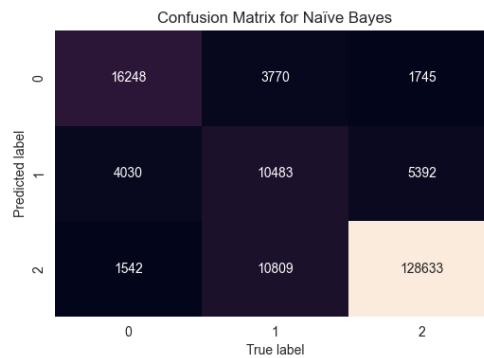


Figure 5.10: Confusion Matrix for Naive Bayes.

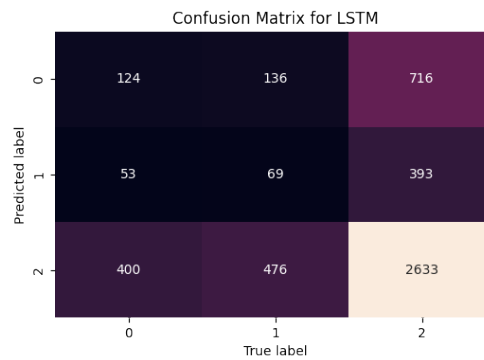


Figure 5.11: Confusion Matrix for LSTM.

In general, after many tests with different comments, we can observe that long comments are very well predicted, whereas the models have a hard time understanding a certain sarcasm, idioms or short phrases. To see a few examples of how the different models evaluate a text, we have invented the following six comments:

- 1) Buen sitio para tapear con buenos precios, buenas tostas y buen vino y una camarera muy atenta y simpática que te recomienda sitios a los que ir a tapear y a cenar.
- 2) No se puede fumar, fatal.
- 3) Cerveza con sabor a alcantarilla.
- 4) Regular.
- 5) Ni fu ni fa.
- 6) Por la boca vive el pez.

Comment	LR	LSTM	NB
1	Positive	Positive	Positive
2	Negative	Negative	Negative
3	Positive	Negative	Negative
4	Negative	Neutral	Negative
5	Neutral	Positive	Neutral
6	Negative	Positive	Positive

Table 5.1: Sentiment Predictions of Invented Commentaries.

In the above table we summarize the predicted values by each model. We can see that they do not differ much, but some perform these cases better than others.

5.4 Conclusions and Future Work

Although 90 % of accuracy has not been reached, which was one of the objectives, both LR and NB have achieved an accuracy of more than 85 %, which is pretty decent.

Concerning to improve the accuracy of these two models, we could be interested in which text features are the most helpful predictors for the classification task. As for the LSTM, it should be said that no iterations or optimizations have been made, hence I think that its accuracy is not reaching its maximum. It remains pending as future work to improve this model, and with it, this metric. It will be useful to have in mind the adjustment of hyperparameters we commented in [Section 2.5](#).

A task already proposed to work later within the company is to create a model that is able to predict not only the sentiment of an entire sentence, but of a single word. It would be very helpful and interesting, because in this way, we could obtain a user rating segmented by products or dishes, for example.

I also think it would be interesting not only to classify a review as negative, neutral or positive, but to increase polarity with other feelings, such as anger or joy.

Appendix A

Implementation

You can find the Python implementation of the models by scanning the following QR code, which is linked to a GitHub repository:



or also by clicking or copying this link in your browser:

<https://github.com/nurialopezraich/Sentiment-Analysis>.

Note: You will only find the implementation of logistic regression and Naive Bayes, the two models that are original work, since all the implementation of the LSTM belongs to UVE Solutions.

Bibliography

- [Agr03] Alan Agresti. *Categorical data analysis*. John Wiley & Sons, 2003.
- [CC18] Anthony L Caterini and Dong Eui Chang. *Deep neural networks in a mathematical framework*. Springer, 2018.
- [GAM17] Alex Gittens, Dimitris Achlioptas, and Michael W Mahoney. “Skip-Gram- Zipf+ Uniform= Vector Additivity”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 69–76.
- [Gol17] Yoav Goldberg. *Neural network methods for natural language processing*. Vol. 10. 1. Morgan & Claypool Publishers, 2017, pp. 1–309.
- [Has+09] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.
- [HLS13] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. Vol. 398. John Wiley & Sons, 2013.
- [Jam20] Alex Pappachen James. *Deep Learning Classifiers with Memristive Networks*. Vol. 14. Springer, 2020.
- [MN19] Peter McCullagh and John A Nelder. *Generalized linear models*. Routledge, 2019.
- [NJ01] Andrew Ng and Michael Jordan. “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes”. In: *Advances in neural information processing systems* 14 (2001).
- [Zha+21] Aston Zhang et al. “Dive into deep learning”. In: *arXiv preprint arXiv:2106.11342* (2021).
- [Zha04] Harry Zhang. “The optimality of naive Bayes”. In: *Aa* 1.2 (2004), p. 3.