Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Deep Learning Text Classification for Medical Diagnosis

Relatore:
Chiar.mo Prof.
Danilo Montesi

Correlatore:
PhD.
Stefano Giovanni Rizzo

Presentata da:
Alberto Drusiani

II Sessione
2017/2018

Al nonno Nino, compagno di giochi e maestro silente

Sommario

La codifica ICD è lo standard internazionale per la classificazione di malattie e disturbi correlati, stilata dall'Organizzazione Mondiale della Sanità, e introdotta per semplificare lo scambio di dati medici, velocizzare le analisi statistiche e rendere efficienti i rimborsi assicurativi. La classifficazione manuale dei codici ICD-9-CM richiede tuttora uno sforzo umano che implica uno spreco di risorse non indifferente e per questo nel corso degli anni sono stati presentati diversi metodi per automatizzare il processo. In questa tesi viene proposto un approccio per la classificazione automatica di diagnosi mediche in codici ICD-9-CM utilizzando le Reti Neurali Ricorrenti, in particolare il modulo LSTM, e sfruttando il word embedding. I risultati sono stati soddisfacenti in quanto siamo riusciti ad ottenere un accuratezza migliore rispetto alle Support Vector Machines, il metodo tradizionale più utilizzato. Oltretutto abbiamo mostrato l'efficacia dei modelli di embedding di dominio specifico rispetto a quelli generali.

Abstract

The ICD coding is the international standard for the classification of diseases and related disorders, drawn up by the World Health Organization. It was introduced to simplify the exchange of medical data, to speed up statistical analyzes and to make insurance reimbursements efficient. The manual classification of ICD-9-CM codes still requires a human effort that implies a considerable waste of resources and for this reason several methods have been presented over the years to automate the process. In this thesis an approach is proposed for the automatic classification of medical diagnoses in ICD-9-CM codes using the Recurrent Neural Networks, in particular the LSTM module, and exploiting the word embedding. The results were satisfactory as we were able to obtain better accuracy than Support Vector Machines, the most used traditional method. Furthermore, we have shown the effectiveness of specific domain embedding models compared to general ones.

Contents

Chapter 1

Introduction

The International Classification of Diseases (ICD) was created by WHO for the purpose of assigning universally recognized and unique codes to diseases, symptoms and signs. It is used for insurance reimbursements, sharing of statistical data and several versions have been released over the years. The ninth version (ICD-9-CM) was presented in 1978 and consists of more than 16 thousand codes, while the tenth version (ICD-10-CM) in more than 68 thousand. The classification is made by qualified personnel starting from a textual diagnosis written by a doctor, which may contain typing errors and abbreviations, especially in departments where they are diagnosed in a short time, such as emergency rooms. Considering the fact that a diagnosis can contain more than one pathology or symptom, each of these can be labeled with an arbitrary number of codes, which makes the classification even more complex. Given the amount of existing codes, manual classification work requires a lot of effort from qualified personnel, in terms of time and expense, and using an automatic classification system would make this process faster and cheaper, limiting human effort and waste of resources. However, automatic classification must be able to handle factors that can be easily resolved by a human, such as typing errors and word semantics.

There are several studies that investigate different approaches to the problem of ICD classification, including the classification of the text through machine learning. To take advantage of machine learning in classification problems, a supervised learning approach is used, which consists in training an automatic classifier starting from already labeled

examples, and therefore in our case, of medical diagnosis with the corresponding associated codes.
In this thesis an automatic ICD-9-CM classifier was developed using the deep learning, the state of the art of machine learning that exploits neural networks, and more particularly the LSTM model, a type of recurrent neural network widely used for its effectiveness. We were given a dataset of about 15,000 Italian diagnoses labeled by the Forlì emergency room and used for the training of the classifier. We have chosen to study and test the ICD-9-CM classification as it is officially used in Italy at the time of writing this paper.
During the classification it is necessary to code the text in the most appropriate way and for this purpose we have used word embedding, considered the state of the art in the representation of the text. Word embedding consists in representing the text with dense vectors of real numbers, through which it is possible to capture semantic relations between words and allows a substantial reduction of the encoding dimensions compared to traditional methods. Word embedding models are trained on large general text corpora, such as Wikipedia, or on specific domain corpora. We have tested the ICD-9-CM classification using general and specific word embedding models provided by a relative thesis, in order to show a possible increase in performance due to the different approaches.

Chapter 2

Related Works

The goal of automatically assigning an ICD code to a textual diagnosis is difficult given the size of all the possible codes.
The classic approach has taken SVM for a long time as a state of the art but in recent years the efficiency of neural networks has been confirmed also in NLP problems.
The most recent approaches use the Convolutional Neural Networks (CNN) and the Long Short Term Memory (LSTM) model, which proved to be particularly effective for this type of classification.
Unfortunately, the most recent studies use datasets labeled with ICD-10-CM codes, as this is the most recent encoding. In this paper we will propose an approach that uses neural networks to classify ICD-9-CM codes, as in Italy it is the official codification currently used.

2.1 ICD-9-CM Coding

The International Classification of Diseases, Ninth Revision, Clinical Modification (ICD-9-CM) is used in assigning codes to clinical diagnosis.
It has been drawn up by the World Health Association to facilitate international comparability of diagnosis and speed up the reimbursement process of medical insurance.
It contains more than 16,000 different codes and each code consists of a maximum of 5 characters and a minimum of 3, where the first character is alphanumeric and the remain-

ing ones are numeric.
The structure of a code is as follows:

$$YXX.XX$$

The structure is hierarchical: the three-character codes are the root codes and consequently two levels of nesting are possible that indicate an expansion of the parent code. For example, the code 410 corresponds to the diagnosis *acute myocardial infarction* and its child code 410.2 corresponds to *acute myocardial infarction of the inferolateral wall*, which is precisely an expansion of the first.

2.2 Machine learning ICD classification

Several studies have been made to try to solve the problem of automatic classification of ICD-9-CM diagnoses using machine learning techniques, each with different methods and characteristics.
Larkey et al. 1995 [1] implemented three multilabel classifiers for medical discharge summaries labeled with ICD9 codes, using KNN, relevance feedback and Bayesian independence. They noted that a combination of different classifiers produced better results than any single type of classifier. Lita et al. 2008 [2] focused on correct classification of high-frequency diagnosis codes with a large dataset of about 90000 diagnosis and about 2700 distinct ICD9 codes associated, using Support Vector Machine (SVM) and Bayesian Ridge Regression with promising results. Zhang et al. 2008 [3] presented a hierarchical classifier that exploited the taxonomy of the ICD-9-CM coding, using the Support Vector Machine method. The corpus they used is the official dataset of the 2007 Computational Medicine Challenge. This challenge corpus consists of 1,954 radiology reports from the Cincinnati Children's Hospital Medical Center, and contains only 45 distinct ICD-9-CM codes. This hierarchical approach has also been studied and evaluated by Perotte et al. 2013 [4], using the MIMIC public dataset consisting of about 21,000 documents. They confirmed that the hierarchical SVM classifier performed better than non-hierarchical, with an F-score of 39.5% and 27.6%, respectively, with an improved recall at the expense of precision. Several studies have shown that one of the main obstacles of the ICD classi-

fication is the imbalance of the classes in the datasets, which can significantly worsen the performances of the classifiers. For example, Kavuluru et al. 2015 [5] found that 874 of 1,231 ICD-9-CM codes in UKLarge dataset have less than 350 supporting data, whereas only 92 codes have more than 1,430 supporting data. The former group has macro F1 value of 51.3%, but the latter group only has 16.1%. To resolve data imbalance problem, they used optimal training set (OTS) selection approach to sample negative instance subset that provides best performance on validation set. However, OTS did not work on UKLarge dataset because several codes have so few training examples that even carefully selecting negative instances could not help. More modern approaches have been used for the ICD-10-CM classification, for example from Lin at al. 2017 [6]. They compared the performance of traditional pipelines (NLP plus supervised machine learning models) with that of word embedding combined with a Convolutional Neural Network (CNN). They used these methods to identify the chapter-level ICD-10-CM diagnosis codes in a set of discharge notes, using a dataset of about 104000 documents, and obtaining very promising results. However, it must be considered that the classification at chapter level reduces the problem to 21 classes and significantly increases performance. The case study of Baumel et al. 2017 [7] is similar to that described in this paper. They compared 4 models for multi-label classification, proposing a model that exploited a type of recurrent neural network (GRU) similar to LSTM and using the MIMIC dataset described above.

Chapter 3

Text Classification

Text classification is a text mining related task. It consists in assigning one or more classes to a text document, in order to classify it according to certain rules.
Formally we want to find a function that maps a set of properties of a document, let's call them features (x), in a set of classes, let's call them labels (y).
Mathematically:

$$(x_1, x_2, x_3, ..., x_n) \rightarrow (y_1, y_2, y_3, ..., y_m)$$

For example, we might want to assign the corresponding ICD-9-CM code to the textual diagnosis *myocardial infarction*, that would be 410. It is therefore necessary to extract some features from the diagnosis text in order to use them for classification.
There are two main approaches to extract information from text automatically:

- **Linguistic**: text is considered as a sequence of sentences and the grammatical structure must be maintained and considered. To use this approach, it is necessary to have complex structures such as trees, built by experts, that indicate the relationship and hierarchy among all the words in the vocabulary of a language.
- **Machine Learning**: the set of words belonging to a document is used as a set of features, without maintaining either the syntactic structure or the order. By using this approach are therefore considered features, for example, the only frequency or presence of a word in a document.

Machine learning approach uses large amounts of textual data (called corpus) and tries

to infer the right class for each document.
In general, machine learning techniques can be divided into two categories:

- **Supervised**: to reach the goal it is necessary to have a labeled dataset, where for each input there is an associated label that indicates the correct output (ground truth). This type of approach is used for automatic classification and will be shown in more detail in this chapter.

- **Unsupervised**: the input data does not have an associated output, so there is no ground truth. Techniques belonging to this category can not be used directly for classification due to the lack of ground truth, which is necessary for classification. Unsupervised learning is used for problems such as clustering and anomaly detection.

In this chapter I will analyze the approach that uses the supervised machine learning technique, focusing also on the representation of text and in particular on the meaning of *word embedding*.
The following section briefly shows why the automatic ID9-CM classification is considered a complicated type of classification.

3.1 Text classification in ICD-9-CM

As mentioned in the second chapter, ICD-9-CM classification contains more than 16000 codes, and this fact makes the ICD-9-CM classification a multi-class problem. Furthermore, if we consider that hospital diagnoses can contain multiple pathologies and symptoms, the classification also becomes multi-label. Given the multilabel nature and the large amount of classes, the ICD-9-CM automatic classification is considered a complex problem in terms of accuracy.
For example, the diagnosis "*obstructive sleep apnea syndrome*" may have associated the following codes[1]:

- **327.23**: obstructive sleep apnea (adult) (pediatric).

- **470**: deviated nasal septum.

[1]The diagnosis and associated codes were provided by the Forlì hospital.

- **478.0**: hypertrophy of nasal turbinates.
- **474.00**: chronic tonsillitis.
- **528.9**: other and unspecified diseases of the oral soft tissues.
- **529.8**: other specified conditions of the tongue

In this case the first code is the one that best describes the textual diagnosis, but all the others may be symptoms that the doctor has associated with the main diagnosis.
It is evident that automating classification starting from textual diagnoses only as a ground truth is complex and requires a large amount of data.

3.2 Representation of text

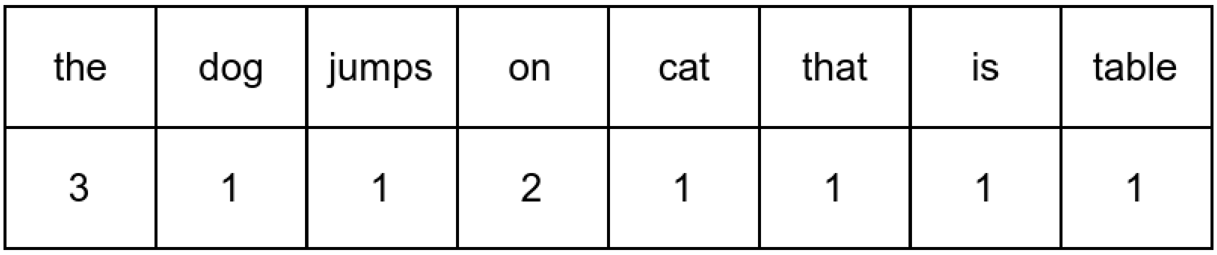
Machine learning algorithms are not able to use text as it is and for this reason it is necessary to represent it numerically. The most natural way to manage text collections are the vectors and, as shown below, the most used encodings exploit this data structure. There are several ways to encode text so that it can be manipulated by a computer and in this section the two most relevant to understand this elaborate will be shown.

3.2.1 Traditional representation

The *bag-of-words* model consists in counting the occurrences of the words contained in a document in order to create a dictionary that indicates the frequency of each word. In this way any information about the structure of the document is lost and only the multiplicity of each word is maintained, which is used as a training feature for a machine learning model. In spite of this de-structuring, the *bag-of-words* model has been found to be sufficient in some contexts, but takes too much account of the most frequent words. For example the following sentence:

the dog jumps on the cat that is on the table

would be encoded as shown in Figure 3.1.



the	dog	jumps	on	cat	that	is	table
3	1	1	2	1	1	1	1

Figure 3.1: *Bag-of-words* encoding.

It is evident that the less relevant words (such as articles and propositions) are the ones with the greatest occurrences while making little information useful for understanding the sentence.
To avoid this problem it is possible to use the *tf-idf* approach, abbreviation for "term frequency-inverse document frequency". This method makes it possible to calculate the relevance of each word to a set of documents and is often used as a weighting factor by classification algorithms.
A further defect of this type of encoding is the dimension of the vectors: each document is encoded with a dimension vector dependent on the total number of words present in the corpus, which generates a sparse matrix of enormous size and not very efficient to manipulate. For this reasons these methods are considered **sparse** representations, where sparse means that the most frequent value in the data structure is zero. Let's see a slightly more complex example.
Consider having a corpus containing the following documents (sentences, in this case):

- **Document 1**: *Bob is a friend of Alice*
- **Document 2**: *Alice is the sister of Carla*
- **Document 3**: *Carla is eight years old*

The bag of words representation would be like the one shown in Figure 3.2.
It is evident that the concept of sparse is emphasized exponentially with large datasets.

	Bob	is	a	friend	of	Alice	the	sister	Carla	eight	years	old
1	1	1	1	1	1	1	0	0	0	0	0	0
2	0	1	0	0	1	1	1	1	1	0	0	0
3	0	1	0	0	0	0	0	0	1	1	1	1

Figure 3.2: *Bag-of-words* representation of three documents: the first row contains the set of words in the documents, the following are associated with the corresponding document.

.

3.2.2 Distributional representation

While the *bag-of-words* model is one of the simplest, *word embedding* is considered the state of the art in terms of text representation.
Word embedding, known also as *distributional representation of words*, allows to capture the semantics of words starting from a corpus of textual data, and to construct a multi-dimensional vector of real numbers for each word. The vectors are constructed in such a way that every semantically similar word has a similar vector. Conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension. Every single word is encoded in a vector with a fixed size and it can be assumed that every dimension is associated with some semantic concept, as shown in Figure 3.3. Word embedding is based on the simple idea that the semantics of a word depend on the context, i.e. the words that precede it and that follow it [9]. The context is therefore a window of neighboring words and this window has a size that is a configurable parameter of the model. Modern approaches use neural networks to create the **embedding layer**: this layer is obtained by giving the starting text (corpus) encoded with a sparse representation as input to a neural network that encodes them to a dense representation. Details about the general functioning of neural networks will be discussed in the next chapter. It is worth to underlining that the term word embedding does not refer to a precise implementation but to the whole set of

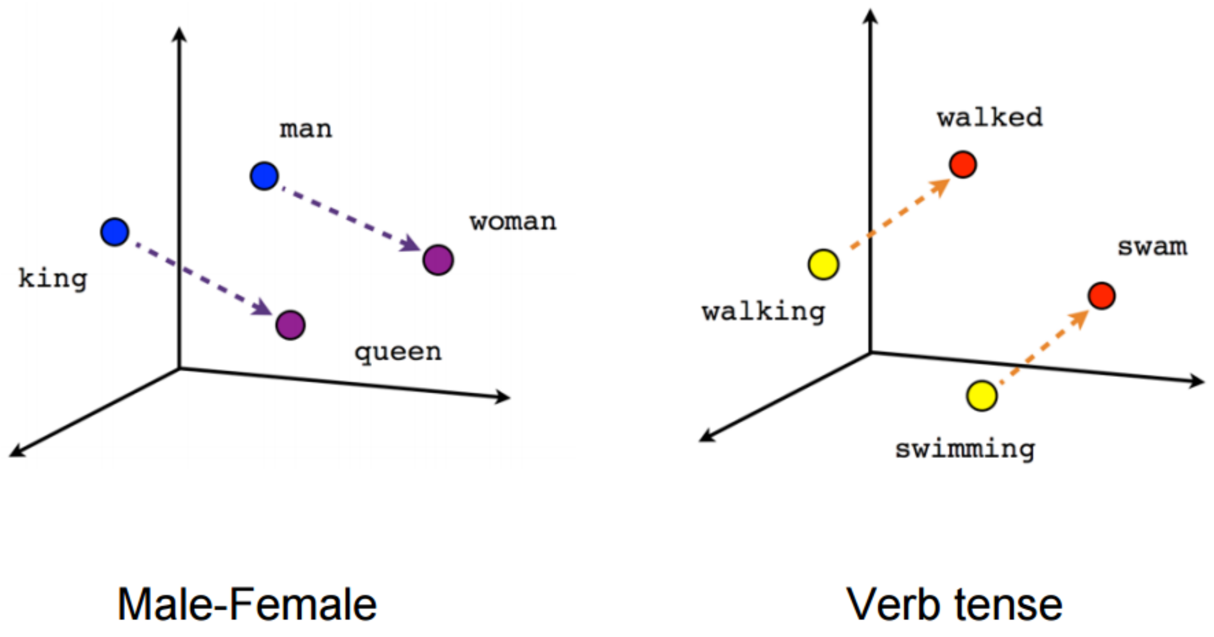


Figure 3.3: Relationships between words are captured with similar distances between the arguments of the relationship [8].

techniques used to reach the goal. One of the most famous technique today is Word2Vec and it was developed by Tomas Mikolov, et al. at Google in 2013 [10].
Word embedding can therefore be used as a representation of the input to a textual classifier.
There are two main ways to take advantage of word embedding:

- **Pre-trained models**: as the title suggests, it consists in using models that are already trained, usually on large public dataset like Wikipedia or Google News. These models are available on the web and do not require a word embedding training.

- **Train new model**: it consists in training a new model using a custom text corpus as input. This method can be used to create specific domain embedding models.

In this paper both pre-trained models available on the web and specific domain models developed in a related thesis will be used.

3.3 Supervised classification

Supervised training, as mentioned above, requires a labeled dataset from which to determine a function that classifies a document based on certain features. Dataset is usually divided into training set and test set: the first is used to infer the function and the second to determine the accuracy.

Classification can be defined as the general task of predicting a categorical variable's value to assign to a set of features given as input.
This task can be divided into two other sub-categories:

- **Binary classification**: there are two possible classes in output.
- **Multi-class classification**: the possible classes in output are minimum three.
- **Multi-label classification**: a generalization of the multi-class where the output can be composed of more than one class at the same time.

In the following subsections these types of classification will be shown in more detail.

3.3.1 Binary classification

The classic example of binary classification concerns the automatic distinction between spam and non-spam emails. For simplicity it is possible to consider spam as a 0 label and no spam as 1. It is therefore necessary to find a function (i.e. learning a model) that maps, for example, the body text of a mail in 0 or 1.
Formally a function f from n real values x_i to one categorical value y (0 or 1 in this case):

$$y = f(x_1, ..., x_n)$$

When f is a linear function on the input variables x_i, it can be used the **logistic regression** classification method, that will be shown in the next steps.
A linear function is a function in the form:

$$y = f(x_1, ..., x_n) = w_1 x_1 + ... + w_n x_n + b$$

where x_i are the inputs, the coefficients w_i are *weights* and b is the *bias*.
Conceptually this function generalizes the function of the line $f(x) = mx + q$ for n inputs.
It is evident that all x_i are known a priori so the unknowns remain the *weights* and the *bias*.
Using logistic regression method, the **hypothesis function** is given by the following formula:

$$h_{logistic}(x_1, ..., x_n) = sig(f(x_1, ..., x_n))$$

where f is linear and $sig(t)$ is a special case of the *logistic function*.
The **sigmoid** function has a domain of all real numbers with return value in the $[0, 1]$ interval, as shown in Figure 3.4. It is described as:

$$sig(t) = \frac{1}{1 + e^{-t}}$$

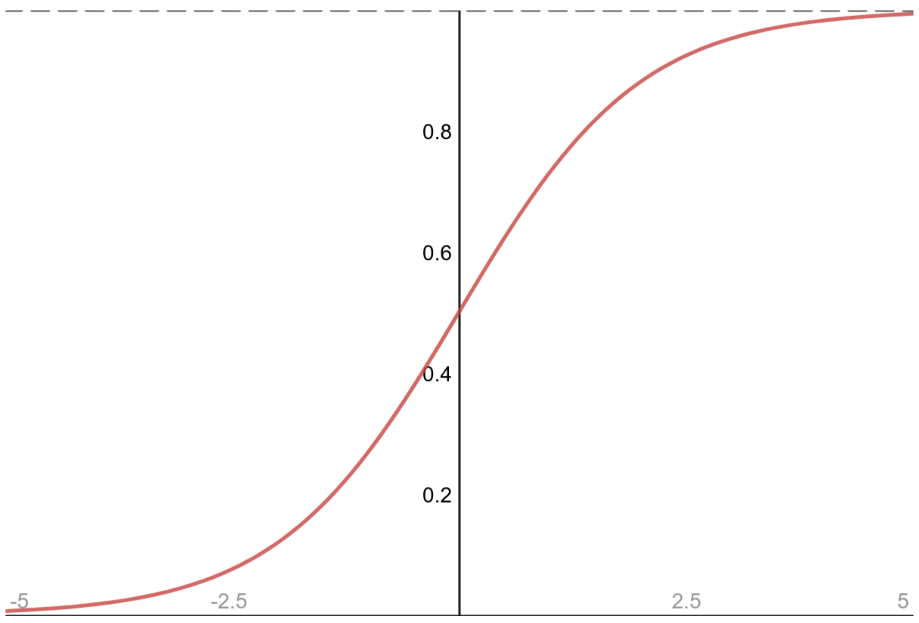


Figure 3.4: The sigmoid function behaves like a straight line for values close to 0 and converges on 0 and 1 for large values (in absolute terms).

The hypothesis function estimates the probability of a certain class given certain inputs. Since the probability of an event is always between 0 and 1, it is reasonable to consider these two numbers as the output classes, as mentioned above. Since the binary classification can have only two possible output classes, if the hypothesis function returns a number greater than or equal to 0.5 then the output is rounded to 1, otherwise to 0. As shown above, the guarantee that the output of $h_{logisitc}$ is between 0 and 1 is given by the application of the sigmoid to f. A schematic diagram for binary classification is shown in Figure 3.5

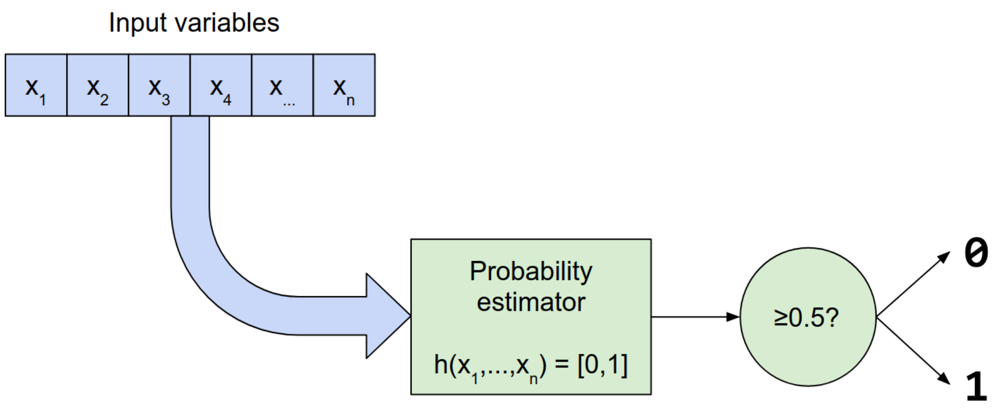


Figure 3.5: Binary classification diagram: features are given as input to a probability estimator that compute an output between 0 and 1.

Loss function

A **loss function** measures how bad a classifier is, comparing the real output value (ground truth) and the predicted one. As mentioned before, the $h_{logistic}$ can be considered as a probability estimator for the output class.
Given a training input x there are two possible cases:

- **True y = 0**: the $h_{logistic}$ should returns 0 and every value above it is considered an error. For this reason $h_{logistic}$ output can be used as the loss.

- **True y = 1**: the $h_{logistic}$ should returns 1. In this case a reasonable loss can be $1 - h_{logisitic}$.

In both cases the loss value depends on the hypothesis function which in turn depends on *weights* and *bias*. It has already been shown that the $h_{logistic}(x_1, ..., x_n) = sig(w_1x_1 + ... + w_nx_n + b)$ and each w_i measure how important the variable x_i is in predicting the positive class (for example "1"). This is given by the fact that if a *weight* is high, the associated variable will strongly contribute to the sum, which is proportional to the probability score. If otherwise a *weight* is close to 0, the associated variable will not be significant to the sum.

Optimization

The minimization of the loss function requires the tuning of the *weights* and the *bias*. For this purpose it is used an optimization algorithm and the most popular one in machine learning is called **gradient descent**.
Gradient descent algorithm is described in the following way:

1. Compute the gradient of the loss function. The gradient is described as a generalization of derivative for multiple variables.
2. If the gradient is positive it means that the loss is increasing, otherwise it is decreasing. Because the goal is to find the minimum, it is necessary to decrease the weight, if the gradient is positive, and increase the weight, if the gradient is negative.
3. Update the hypothesis with the new weights.
4. Compute again the loss.

These steps are repeated until the loss is smaller than a given tolerance.

Fitting the model

Given this learning model, it is necessary to fit it with training inputs. As mentioned at the beginning of this section, the starting dataset must be divided into a **training set** and **test set**.
The former is used for training and consequently for the minimization of the loss function.

The latter is used for estimating accuracy, once the model has completed training. Usually the two sub-sets are obtained following the Pareto's principle [11]:

- 80% training set.
- 20% testing set.

In fact, the Pareto's principle states that, for many events, roughly 80% of the effects come from 20% of the causes, and therefore a test set of this size is sufficient for a good approximation of accuracy.

3.3.2 Cross validation

K-fold cross validation technique consists in rotating over training and test split, so that each sample is used both for training and testing. The goal of cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or selection bias [12].
One rotation is called "fold" and the total number of folds is based on the fraction size of the test set.

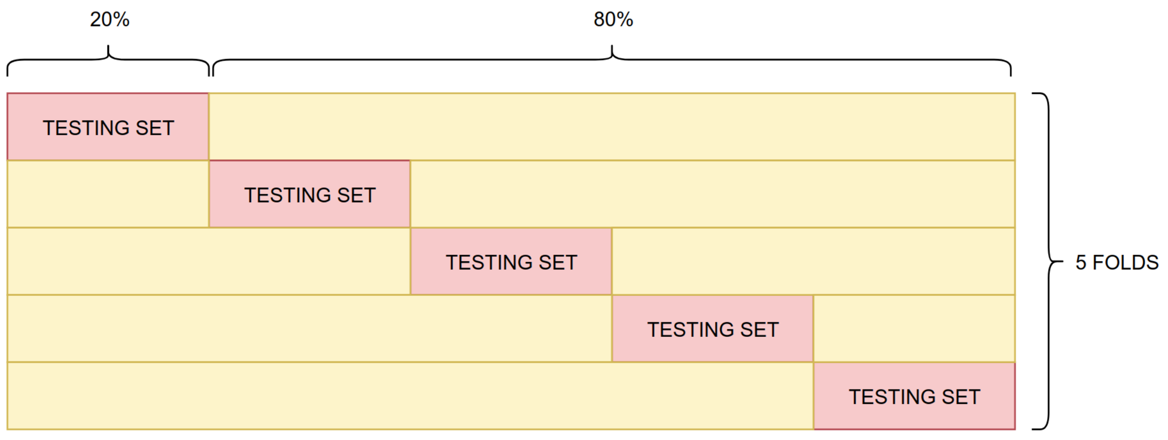


Figure 3.6: 5-fold cross validation: each fold is used both for training and for the test. In this case 5 iterations are necessary.

By using the Pareto's principle the testing set would be of size 20%. Then the number of folds would be 5 as shown in Figure 3.6.
The final accuracy by using the cross validation method is the mean of every fold's accuracy.
The procedure can be summarized in the following steps:

1. Shuffle the initial dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
   (a) Take the group as test dataset.
   (b) Take the remaining groups as a training dataset.
   (c) Fit the model on the training set and evaluate it on the test set.
   (d) Retain the evaluation score and discard the model.
4. Summarize the skill of the model by using the sample of model evaluation scores.

In summary, cross validation allows to obtain more precise estimates regarding the performance of the model [13].

3.3.3 Overfitting

Overfitting is the phenomenon that exists when a model is very accurate in classifying the data of the training set but not in the test set. For example, after the training of a model it can happen that the value of the loss function is very close to zero for the training data but considerably higher for test values. This means that the model is extremely good on training data but it is not able to generalize on new data, as shown in Figure 3.7 [14].

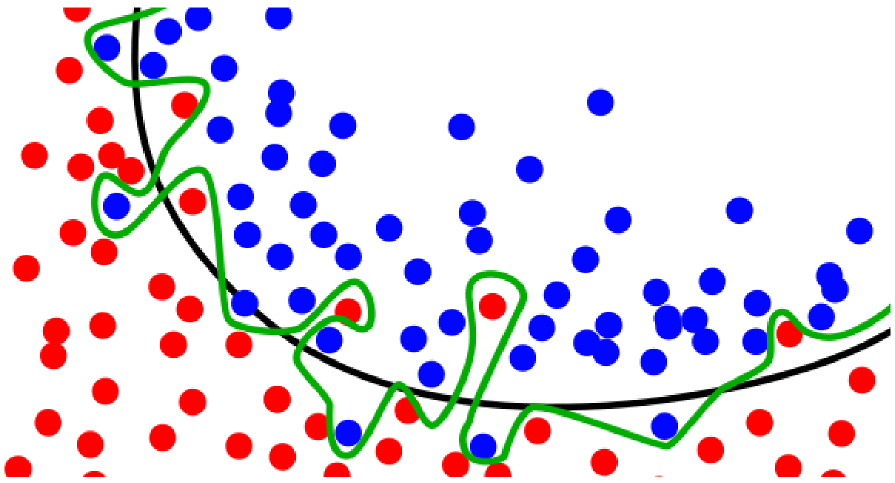


Figure 3.7: The green line represents the overfitted model, while the black line represents the normalized model [15].

Intuitively it means that the model has found a naive way to fit the training set data instead of learning a pattern that can be used on any data.
The use of **regularization techniques** allows avoiding overfitting. These techniques consists in a modification of the loss function adding an additional value that increases as the value of *weights* increase. In this way *weights* are prevented from being too large.

3.3.4 Multi-class and multi-label classification

As I mentioned at the beginning of this section, **multi-class classification** allows multiple output classes, only one of which is output for each input data. An example of multi-class classification concerning flowers, extracted from the famous *Fisher's Iris* dataset [16], is shown in Table 3.1.

Sepal length	Sepal width	Petal length	Petal width	Class
6,3	2,3	4,4	1,3	Versicolor
6,2	3,4	5,4	2,3	Virginica
5,2	3,4	1,4	0,2	Setosa
6,9	3,1	5,4	2,1	?

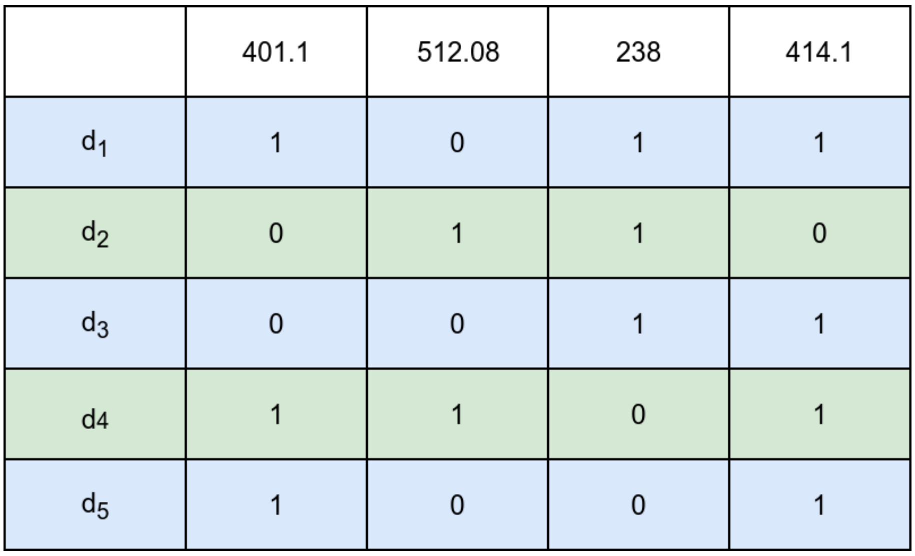
Table 3.1: Fragment extracted by the multi-class Iris dataset.

In this case the inputs are the measurements of the petals and the sepals, the output is the class (ground truth) and the unknown is the class of Iris in the last row.
As already shown, a binary classifier can be considered as a probability estimator between only two classes (0 and 1), but this concept can be generalized for an arbitrary number of possible classes. There are two ways to adapt binary classifiers to a K-classes classification problem [17]:

- **One vs One (OVO)**: this strategy consists in training a single classifier per class where the samples of the corresponding class are considered as positive and all the others as negatives. This approach needs to train K classifiers.
- **One vs Rest (OVR)**: each possible pair of classes is taken in consideration in order to train a different classifier. This approach needs to train K*(K-1)/2 classifiers.

Multi-class classification can still have only one output for every set of features and this does not solve classification problems that intrinsically require more output, like ICD classification.
For this type of problems it is therefore necessary to use the **multi-label classification** that generalizes the multi-class classification allowing more than one classes for each instance.
Intuitively a multi-label classifier maps inputs x inputs into binary vectors y, assigning 0 or 1 for each label if it is associated with that input or not. In Figure 3.8 it is shown an example of multi-label classification with five samples and four classes: when a class is associated with a certain sample, the corresponding row-column intersection is marked with 1, 0 otherwise. There are several ways to generalize multi-class classification to multi-label, below are shown the most used [18]:

- **Binary Relevance**: as many classifiers are trained as the number of classes where each single binary classifier is associated with a class. Then, given an unseen sample, the final classifier will return as positive only the positive classes returned by the corresponding binary classifiers.
- **Label Powerset**: the multi-label problem is reduced to a multi-class using only one multi-class classifier that is trained on all the unique combinations present in the training set.



	401.1	512.08	238	414.1
d_1	1	0	1	1
d_2	0	1	1	0
d_3	0	0	1	1
d4	1	1	0	1
d_5	1	0	0	1

Figure 3.8: Multi-label example diagram. Every d_i diagnosis is associated with with more than one ICD-9-CM code at the same time.

- **Classifier Chain**: the first classifier is trained on the input data and then each next classifier is trained on the input space and all the previous classifier in the chain.

It is possible to implement a multi-label classifier also by using a neural network with an output neuron for each class, and this will be shown in this paper in the following chapters.

Chapter 4

Neural networks

Artificial neural networks (ANN) are brain-inspired systems, that is, they try to replicate the mechanisms of human reasoning [19]. The neural network itself isn't an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Neural networks are not specifically programmed about the problem to be solved, but they tend to learn using a large number of labeled examples (supervised learning).

They are widely used to date in the fields of computer vision, sentiment analysis and speech recognition. Intuitively, an ANN consists of two or more layers, each of which consists of an arbitrary number of **neurons**. The neurons that make up a layer are "combined" and connected with the neurons of the next layer through **edges**. The analogy with the human brain is evident, where the neurons are connected by the synapses. The more examples and new inputs the program sees, the more accurate the results become because the program learns with experience; this is especially true when the program receives feedback on whether its predictions or outputs are correct.

This concept can best be understood with an example. Imagine the "simple" problem of trying to determine whether or not an image contains a cat. While this is rather easy for a human to figure out, it is much more difficult to train a computer to identify a cat in an image using classical methods. Considering the diverse possibilities of how a cat may look in a picture, writing code to account for every scenario is almost impossible. But using machine learning, and more specifically neural networks, the program can exploit a generalized approach to understanding the content in an image. Using several layers of functions to decompose the image into data points and information that a computer can use, the neural network can start to identify trends that exist across the many examples that it processes and classify images by their similarities [20].

4.1 Architecture

An artificial neuron is a mathematical function that maps n inputs $x_1, ..., x_n$ in an output y. Each inputs x_i is multiplied by a weight w_i and then all weighted inputs are summed together. This computation, as mentioned in the previous chapter, can be written as:

$$w_1x_1 + ... + w_nx_n + b$$

or compacted in:

$$l = b + \sum_{i=0}^{n} w_i x_i$$

where l is the **logit**. Then it is applied an **activation function** that takes the logit as input and returns an output:

$$y = f(l)$$

This computation can be visualized in Figure 4.1.

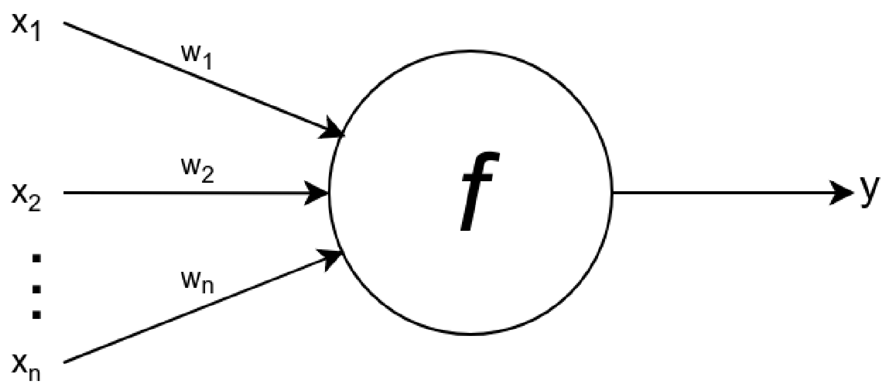


Figure 4.1: Neuron takes n weighted inputs and returns an output.

Different activation functions are used, including the sigmoid that I showed above.

Therefore, using the sigmoid it is clear that the output of each neuron must be a real number between 0 and 1.
The neurons are connected in such a way as to form a network. Artificial neural networks are composed of **layers**, each of which is formed by an arbitrary number of neurons. It is possible to line up many layers and connect them so that every neuron in a layer is connected to all the neurons in the next layer through edges. Each edge has an associated weight which, as shown above, contributes to the function calculated by each neuron. The first layer is the **input layer**, the last is the **output layer** and each layer between is a **hidden layer**.
Neural networks can be used to solve classification problems.
As already mentioned, to do supervised learning it is necessary a labeled dataset, where features are inputs and labels are outputs. Intuitively, the features are given as input to the first layer of the neural network and the output is expected in the last layer. In classification problems, the number of possible outputs is the number of classes, so it is reasonable to think of having in the output layer a number of neurons corresponding to the number of classes of the problem. Using the same reasoning, the input layer must be made up of a number of neurons equal to the number of features contained in the dataset. Imagine you have to classify mail as spam or not spam: in this case a binary classifier is needed and then it could consist in a neural network whose output layer contains two neurons, one associated with class "not spam" and one associated with class "spam". The input layer could reasonably have as many neurons as the words contained in the mail set. The hidden layer(s) may instead contain an arbitrary number of neurons. A diagram of this configuration is shown in Figure 4.2. As mentioned above, if it is used the sigmoid as an activation function, then every neuron outputs a number between 0 and 1;it is therefore considered a "turned on" neuron if its output is close to 1 and "off" if it is close to 0. This means that in the output layer the class predicted by the network will be the one associated with the highest value neuron. Taking the example of the binary mail classifier, an email will be considered spam if the output neuron associated with the "spam" class is the one with the highest value.
The main computation is made by hidden layers.
Taking, for example, face images as input, it can be assumed that each hidden layer de-

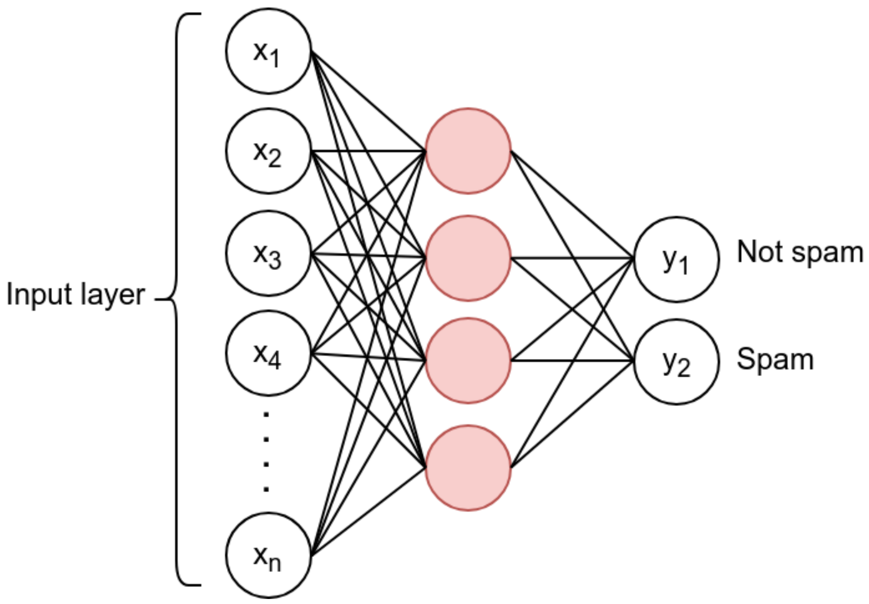


Figure 4.2: The input layer contains as many neurons as the words contained in the mail set. The hidden layer with four neurons is marked in red.

composes the image into a pattern that is gradually more complex. For example, the first hidden layer could detect edges, the second could detect face parts (mouth, eyes, nose) and so no. This is not necessarily true but makes the behavior of the neural network comprehensible.

As shown above, once given an input, the variables of a neural network are the weights associated with the edges; this means that it is necessary to find the value of the weights in order to have a good classification. So, first of all, the weights are randomized due to the lack of information necessary to set them correctly. Then the first input is inserted into the first layer of the neural network. For simplicity it is considered the input codified as *bag-of-words*. The values of the neurons are calculated from the function seen above, where the inputs are the results of all the neurons of the previous layer, up to the final layer. At this point the neurons of the output layer will contain values in the interval $[0, 1]$ and the class predicted by the network for that input will be the one associated with the neuron with the highest value.

Once the output is obtained, it is necessary to compare it with the real value obtained from the starting dataset: the two values (the predicted and the real) are given as input

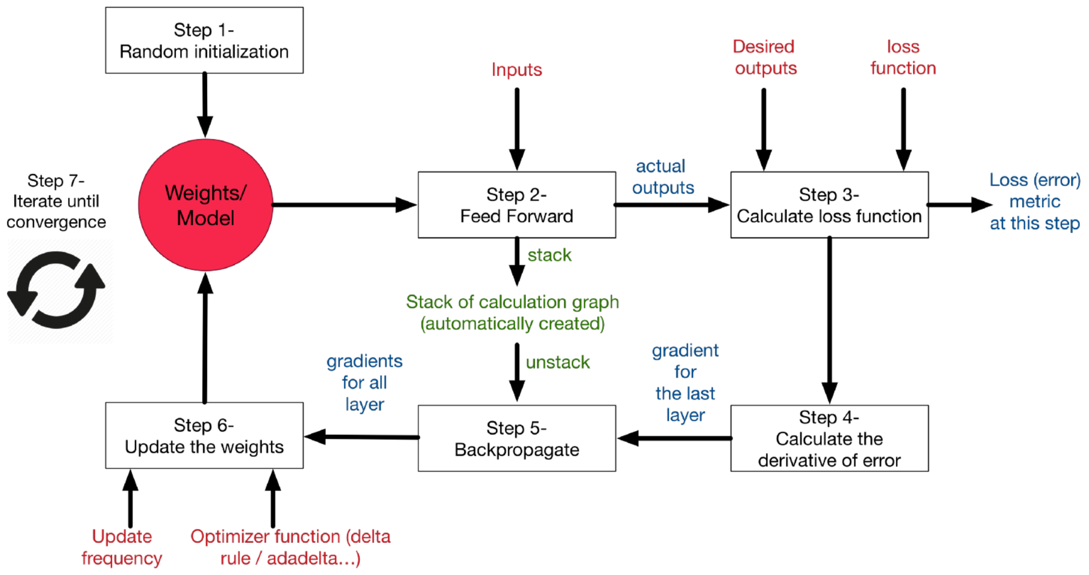


Figure 4.3: Diagram of neural network training [21]. Every complete loop is considered ad an iteration.

to a loss function which, as seen in the previous chapter, calculates the error committed by the classifier. Now we can use an optimization algorithm (like gradient descent) that calculates the derivative of the last layer and decides whether weights should be increased or decreased.

In order to update all the weights of the network it is used a procedure called **backpropagation**: we start to compute the derivative for the last layer, and then move backwards, back-propagating the error to the previous layer. This is the core of the efficiency of neural networks as it allows the updating of the weights in feasible time even for neural networks with hundreds of thousands of edges.

Passing an input forward and then updating the back weights consists of an **iteration**. After the first iteration the procedure is repeated for each data in the training set. Once each input of the training set has been passed into the neural network, an **epoch** is concluded.

The training of a neural network usually consists of several epochs in order to update the weights several times and improve the accuracy of the classification.

4.2 Layers

Artificial neural networks can be implemented in different ways, changing the number of layers (**depth**), the number of neurons for each layer (**width**) and the type of each layer. Depending on the nature of the classification, there are layers that are more efficient than others in certain problems.
In this section I will illustrate the three types of layers that were used to create the classifier shown in the next chapter. In the list below I will briefly describe the simplest layers and then focus in the next sub-section on the most interesting and complex recursive layer.

- **Dense Layer**: the standard layer where each neuron is connected with each neuron of the previous layer. For this reason it is also called *fully connected* layer, as shown in Figure 4.2. In this type of layer all neurons contribute in the output function in the way mentioned at the beginning of this chapter.

- **Embedding Layer**: as shown in the chapter on text classification, there are several ways to encode input data, some of which are more effective than others. As already mentioned, word embedding is able to capture semantic information and can be combined with a neural network using an embedding layer. This type of layer is used as an input layer in combination with the **embedding matrix** obtained from the word embedding model and the training dataset. Intuitively this layer will contain a number of neurons equal to the words of the dataset contained in the trained word embedding model. Implementation details will be shown in the next chapter.

4.2.1 LSTM

Long Short Term Memory networks are a special kind of **RNN (Recurrent Neural Network)**.
Intuitively, a RNN is a neural network capable of "remember" information. Imagine you have to write a diagnosis: your brain would allow you to write a meaningful sentence, so that each word in some way depends on the context given by the previous ones. In fact, the human brain is able to reason using the memory of past events, so that we can

understand and interact with what surrounds us. Unfortunately, the "classical" neural networks, as shown in this chapter, do not have any mechanism to store information that can be used in the future because they are based only on the current input.
Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist, a shown in Figure 4.4.

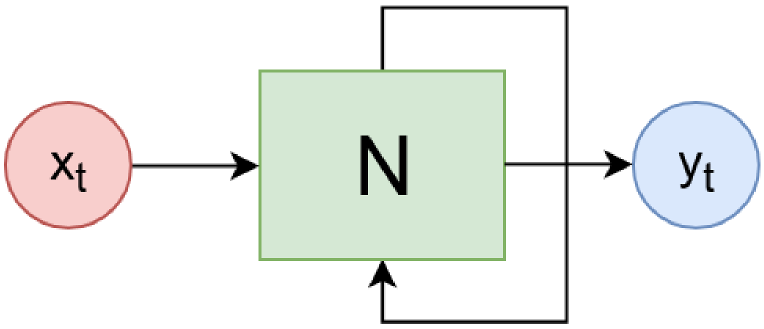


Figure 4.4: A piece of a RNN. The loop allows information to be passed from one step to another.

Therefore, a recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
In this way, during the training, a RNN will be able to use information collected during the training of the previous diagnoses. It is therefore crucial to understand how far the classical RNNs can memorize. Theoretically, they should be able to manage "long-term dependencies". Unfortunately, it has been observed that classical RNNs are not able to use information stored long before and for this reason they can not capture links between distant elements [22].
LSTMs avoid this problem because it is capable of learning long-term dependencies.
A LSTM neural network is a RNN: in fact it inherits the loop concept from recursive networks.
The difference lies in the architecture of each internal module that in the Figure 4.4 is called *N*.
It consists in four layers:

- **Cell**: remembers values over arbitrary time intervals.

- **Forget gate**: "decides" which information (belonging to previous inputs) must be thrown away or kept from the cell.
- **Input gate**: "decides" which new information is going to be stored in the cell state.
- **Output gate**: "decides" which part of the cell state should be output.

These units are interconnected as shown in Figure 4.5.

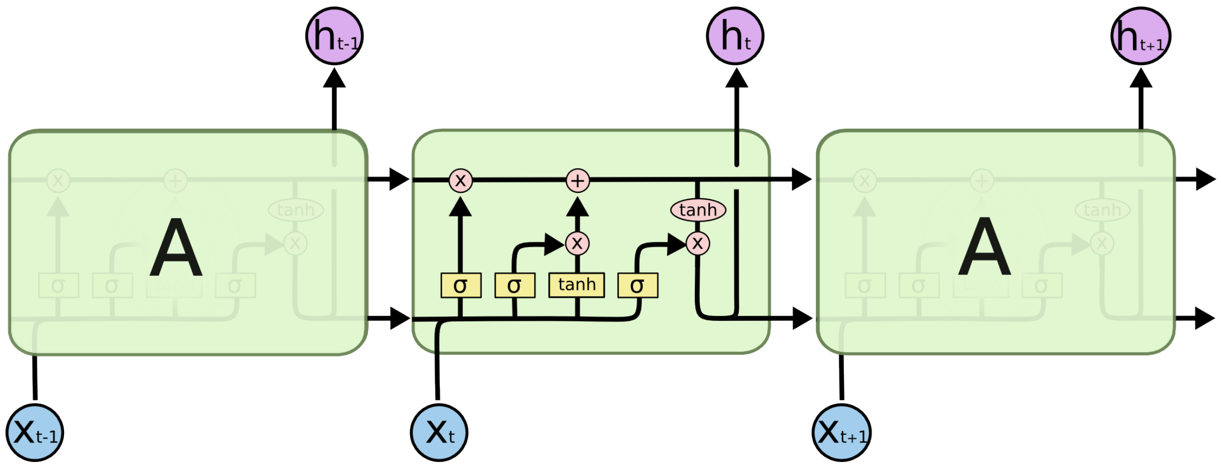


Figure 4.5: LSTM network with view of an internal module [23]. Every σ is a sigmoid gate and *tanh* is the abbreviation for *hyperbolic tangent*.

Let's call the inputs x_t and the outputs h_t. This model passes through the following steps:

1. x_t and h_{t-1} are combined and pass through the forget layer (sigmoid). In this way it forgets unnecessary information for each number in the previous cell state.
2. x_t and h_{t-1} are combined and pass first through the input gate (sigmoid) that computes which values to update. Then they pass trough a *tanh*, that create new candidate values for the cell state.
3. The results in the previous step are combined on order to update the cell state with new useful information.
4. The cell state pass through the output layer (sigmoid), then trough a *tanh* and these results are combined in order to output a filtered version of the cell state values.

Chapter 5

Deep learning for ICD-9-CM classification

This chapter shows an approach to the ICD-9-CM classification that exploits neural networks and word embedding, applied to textual diagnoses written in Italian.

5.1 Dataset and preprocessing

The dataset we used is composed of medical diagnoses extracted from hospital discharge cards provided by the emergency room from G.B.Morgagni-L.Pierantoni hospital, Forlì. The diagnoses are written in Italian and labeled by qualified personnel with the corresponding ICD-9-CM codes. To perform the preprocessing I used *scikit-learn* [24], an open source python library suitable for these purposes. The text has been preprocessed in order to minimize any noise and keep only the significant parts, following the steps below:

- Removal of any empty diagnoses.
- Removal of punctuation and numbers and transformation of all characters into lower cases.
- Tokenization.
- Removal of Italian stop words.

- Removal of insignificant characters (+,-,=,!..).
- Replacement of abbreviations with the corresponding whole word (i.e. "rx" replaced with "right").
- Each diagnosis sequence has been truncated to a maximum of 30 words or, if contained less, by 0-padding up to 30.

5.2 Feature extraction and label representation

To represent the labels associated with each diagnosis, we have created a sparse binary matrix of dimension $[diagnosis, classes]$, where each element is 1 if the diagnosis corresponding to the row has associated the label corresponding to the column, 0 otherwise.
Once we obtained the list of diagnoses in the form of tokens and the label matrix, we divided them randomly into training set and test set, respectively 80% and 20% of the original dataset. Unfortunately, the existence of classes present only once in the whole dataset did not allow us to use the cross validation technique. For implementation reasons it is in fact necessary for a class to appear at least twice.
We subsequently created an **embedding matrix** using a *word2vec* model trained in a related thesis. I have been provided with different *word2vec* models, each trained on a different corpus. In this way we were able to evaluate any performance boost due to the specific word embedding domain for this particular problem.
Test results with the different word embedding models will be shown in the next chapter.
Each element of the embedding matrix contains the dense vector representation of a word present in the diagnostic dataset, provided by the word embedding model.

5.3 Neural network model

For the classification task we have exploited a neural network, implemented using *keras* [25], an open source python library. Keras makes it easy to create a deep learning model by choosing the hyper-parameters, the input and output dimensions of the layers and the type of layer, using a few methods. The final configuration of the neural network was chosen after numerous trials with different combinations and different types of layers, including Convolutional Neural Networks (CNN). It has been noted in other studies that the latter, although created for the recognition of images, are quite efficient even in the context of medical text classification [6], but in our case they have not proved satisfactory. We have also tried to exploit LSTM and CNN together, without achieving good results. The best configuration among those tested is composed of the following three layers, as shown in Figure 5.1:

1. **Embedding layer**: this layer acts as an input layer and does nothing more than point to the embedding matrix created earlier to obtain the vector representations of each word contained in the input diagnosis.
   The vector values contained in the table are potentially trainable, but we decided not to train them because we used a model of word embedding specifically trained for this domain. As mentioned above, we have chosen 30 as the input dimension. The output size of this layer depends on the size of embedding, in our case 200 or 300 depending on the model.

2. **LSTM layer**: the LSTM layer is the core of the neural network. After several tests with different dimensions (20, 50, 100, 200, 300, 500), the best output size was found to be 300. We added a dropout coefficient of 0.5 as the network tended to overfit very quickly. In this way, half of the output neurons of this layer are "switched off" at each iteration, in order to avoid overfitting phenomena. The dropout coefficient has been tested for various values, starting from 0.1 up to 0.8.
   The activation function used is the hyperbolic tangent, as shown in the previous chapter in the LSTM section.

3. **Dense layer**: the output layer is fully connected and has a size equal to the number of classes in the dataset. We have chosen the sigmoid as an activation function, as it allows a multi-label classification where the choice of each class is independent from the others. In fact, the codes associated with a diagnosis are considered all with the same weight and without an order of relevance.

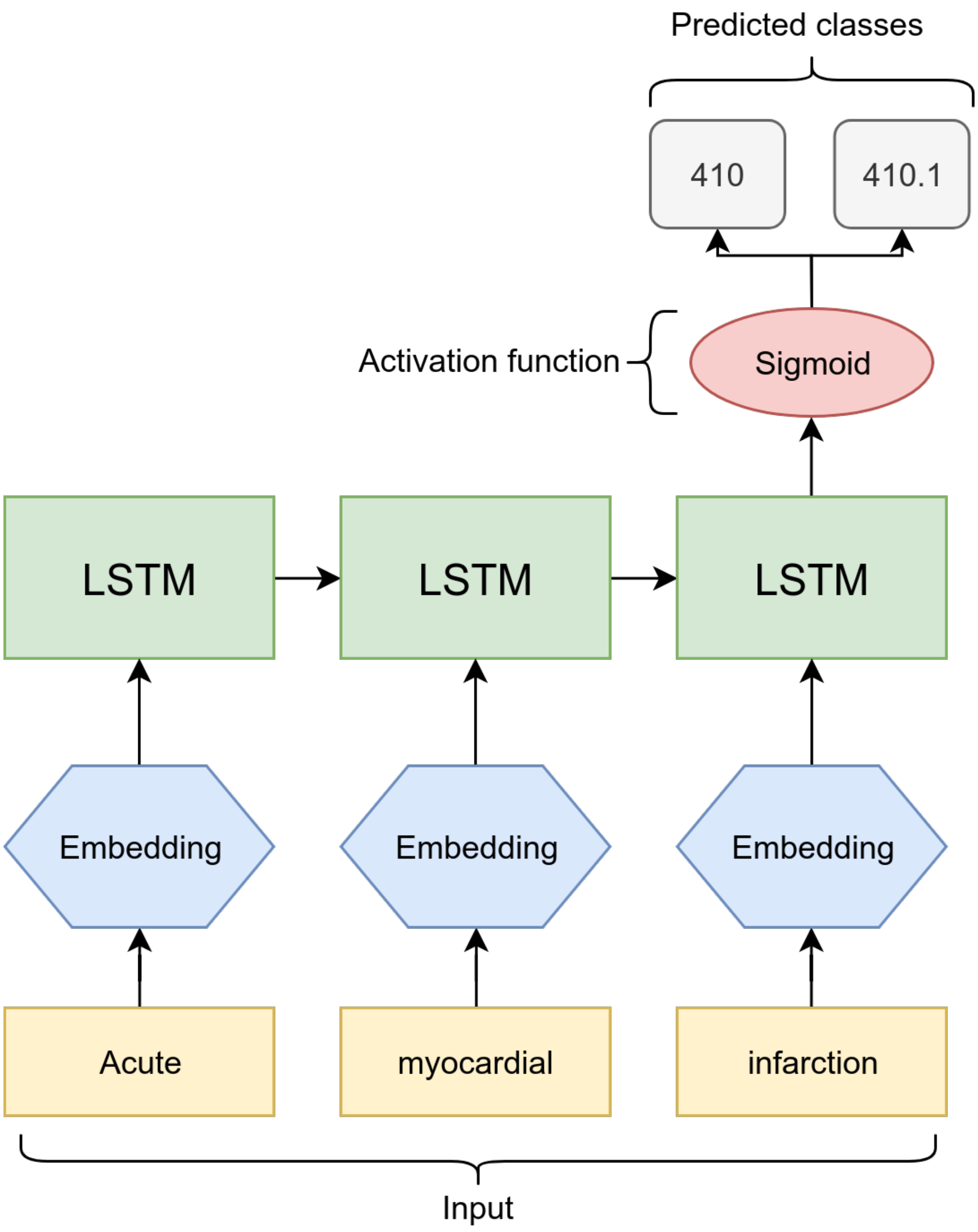


Figure 5.1: A simple example of LSTM architecture combined with word embedding and multi-label output. It must be specified that there is a sigmoid gate for each output class.

5.4 Optimization

The search for the best loss function was one of the most difficult obstacles. Initially we did not take into account the imbalance of the dataset and consequently the classifier learned to always predict the most frequent class in order to minimize the error. Later we tested different loss functions that considered the imbalance of the dataset and we used the one that allowed better accuracy.
The **loss** function we used is taken from *tensorflow* [26], a python library for machine learning used for research and production at Google. This specific function allows to manage the imbalance of the classes by modifying the weights of each prediction based on the frequency in the dataset of the aforementioned class. This function computes the sigmoid cross entropy and, by tuning a parameter, allows one to trade off recall and precision by up- or down-weighting the cost of a positive error relative to a negative error.
For brevity, let x = predicted label, z = true label, q = weight parameter.
Formally the loss is:

$$\begin{aligned}
& qz(-log(sigmoid(x)) + (1-z))(-log(1-sigmoid(x))) \\
&= qz(-log\left(\frac{1}{1+e^{-x}}\right)) + (1-z)(-log\left(\frac{e^{-x}}{1+e^{-x}}\right)) \\
&= qz(log(1+e^{-x})) + (1-z)(-log(e^{-x}) + log(1+e^{-x})) \\
&= qz(log(1+e^{-x})) + (1-z)(x + log(1+e^{-x}) \\
&= (1-z)x + (qz + 1 - z)(log(1+e^{-x})) \\
&= (1-z)x + (1+(q-1)z)(log(1+e^{-x}))
\end{aligned}$$

The trend of the loss value of the network using one of the embedding models is shown in Figure 5.2. The tuning of the hyper-parameters was done quite exhaustively, testing several tens of combinations to achieve better accuracy than the baseline with SVM. Several optimization algorithms have been tested and we have chosen to use **Adam** [27], with a **learning rate** of 0.0001. Learning rate is a hyper-parameter that controls how much we are adjusting the weights of our network with respect the loss gradient. The default Adam

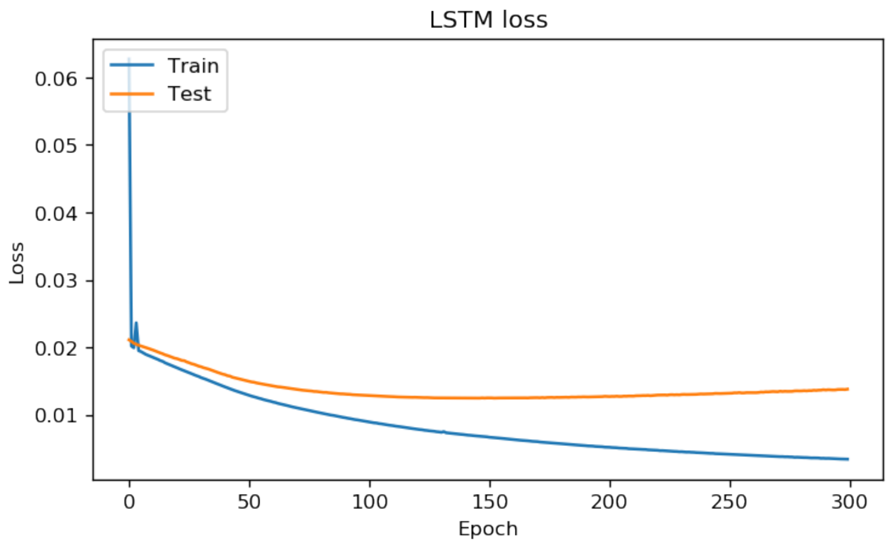


Figure 5.2: The chart shows the progress of the training and test loss using LSTM and JoinData.

model provided by Keras uses a learning rate of 0.001 that was too fast for our purposes, reaching very quickly the nearest local minima and consequently oscillating considerably. We chose 300 training periods after testing different sizes (10, 50, 100, 200, 300, 500, 1000) and noting that with less epochs we did not reach convergence and with more epochs we risked overfitting.

We chose a batches size of 32, because by increasing it, the accuracy was reduced, and decreasing it resulted only in an increase in training time. We set the network in such a way that it stopped the training in case the loss did not decrease from one epoch to the other, still allowing a patience of 200 epochs in order to overcome any stalls. No increase in performance was noted by adding depth to the network, for example with multiple LSTM layers. The values returned in output from the network (therefore between 0 and 1) have been approximate to the nearest integer: in this way the values lower than 0.5 are interpreted as 0 and the greater ones as 1. We noticed that, by adding a considerable patience, the loss value increased slightly, still obtaining a better F1 score.

Chapter 6

Results

In this chapter the results obtained with the classifier illustrated in the previous chapter are shown as opposed to a traditional machine learning method. As already mentioned, we used word embedding for text classification, testing different models provided by a relative thesis.

Dataset details

As mentioned in the previous chapter, we used a dataset composed of 14305 diagnosis. Despite preprocessing, many typos remain that can not be deleted or corrected. After preprocessing, there are 11,076 different words left and each diagnosis contains from 1 to 172 words, with an average of 13.7 words each. Each diagnosis has 1 to 15 associated ICD-9-CM codes, with an average of 3.4 codes and a total of 3248 different classes. The distribution of the classes is extremely unbalanced, as certain codes appear much more frequently than others. This fact is reasonable as there are more frequent conditions than others in real cases, as shown in Figure 6.1. The most frequent code is 401.1 which corresponds to *benign essential hypertension* and consists of 3.6% of all the total labels in the dataset. On the other hand, there are 1067 codes that appear once on all labels and each accounts for 0.002%, for example the code associated with *atresia and stenosis of large intestine, rectum, and anal canal.* The Figure 6.2 shows the 10 most frequent classes.

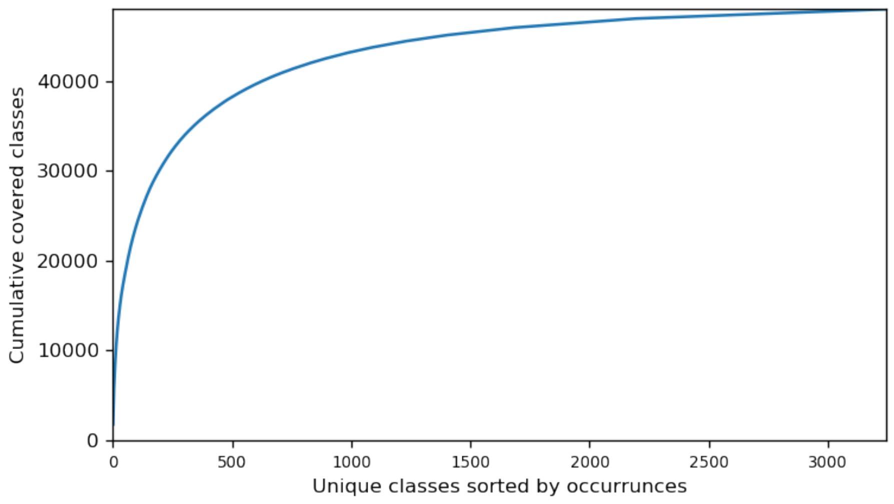


Figure 6.1: The curve shows the percentage of coverage of the total labels with respect to the number of classes in increasing order of occurrences.

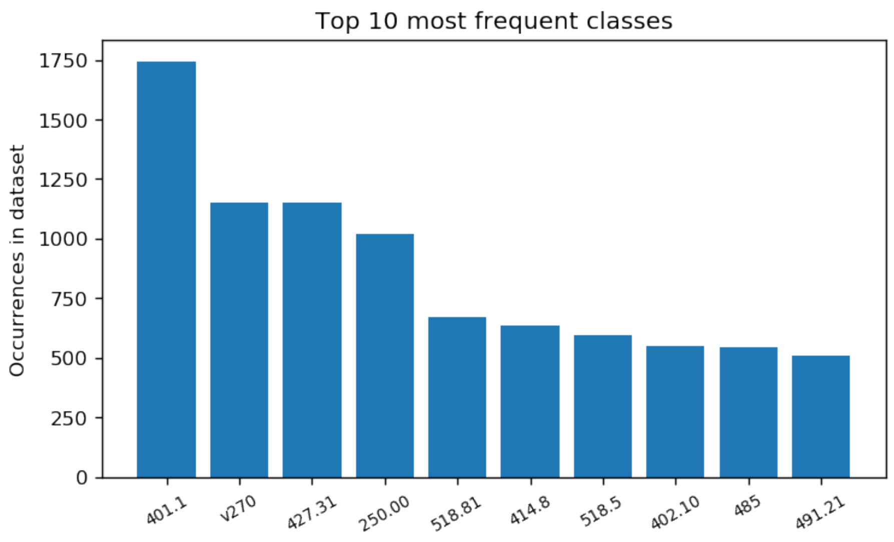


Figure 6.2: The diagram shows the 10 most frequent classes in our dataset. These classes make up 17% of the total labels in the dataset.

Baseline

As a baseline we used a classifier that uses **Support Vector Machine (SVM)** combined with word embedding, trained and tested on the same datasets used for the neural network. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. Given the multi-label nature of the ICD-9-CM classification, we used the One vs Rest approach, thus building 3284 binary SVM classifiers.

Word embedding models

As already mentioned, several models of word embedding have been provided, trained on different domains:

- **General purpose**
  - **General Wikipedia**: all pages of Italian Wikipedia.
  - **TED**: a corpus created starting from the translations of conferences held worldwide.
- **Domain specific**
  - **ICD-9-CM manual**: the descriptions of the diagnoses contained in the official manual.
  - **SDO**: a set of about 700.000 emergency room discharge records provided by Forlì Emergency Room Hospital.
  - **Medical Wikipedia**: all the Wikipedia pages in Italian concerning medical topics.
  - **JoinData**: all domain specific corpus mentioned above trained together.

In this way it was possible to evaluate a possible increase in the performance of the classifier due to the different domains and embedding models, as shown in the next sections. All embedding models and datasets are publicly available [1].

[1]https://drive.google.com/drive/folders/1_7xUVJQbHyDR5cP4dz1uQG9LGDunLYds

Evaluation metrics

To evaluate the accuracy of the classifier we used the **F1-score**, which is defined as follows:

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

where **precision** is the fraction of relevant instances among the retrieved instances, while **recall** is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. We measured three types of F1-score:

- **F1-micro**: calculate metrics globally by counting the total true positives, false negatives and false positives.
- **F1-macro**: calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- **F1-weighted**: calculate metrics for each label, and find their average weighted by support (the number of true instances for each label).

6.1 Evaluation

We performed the tests on the baseline always using the same architecture and changing the embedding models, so as to understand which model would increase the performances more. Due to time constraints, we only tested three embedding templates with the SVM baseline.
The scores of SVM for the tested embedding models ar shown in Table 6.1.

Model	F1-micro	F1-macro	F1-weighted
SDO	0.37936	**0.08545**	0.36578
Medical Wikipedia	0.23026	0.06832	0.27729
JoinData	**0.38625**	0.08495	**0.36919**

Table 6.1: Scores with SVM using different embedding models.

Model	F1-micro	F1-macro	F1-weighted
General Wikipedia	0.36012	0.06345	0.31400
TED corpus	0.31676	0.05080	0.27442
ICD-9-CM manual	0.26799	0.03937	0.21829
SDO	0.42574	**0.08832**	0.38910
Medical Wikipedia	0.38766	0.07462	0.34543
JoinData	**0.42796**	0.08665	**0.39355**

Table 6.2: Scores with LSTM using different embedding models.

The highest score was obtained using the JoinData embedding model, which, as mentioned above, was trained by joining all the medical domain corpora. Even training using the SDO dataset has achieved high scores, probably due to the fact that it also includes the diagnoses used for training the classifier, allowing the word embedding to capture semantic information even from typos.
The score of the F1-macro is considerably low compared to the other two, reflecting the fact that it does not take into account the imbalance of the dataset and confirms that the management of the imbalance is necessary to achieve an acceptable performance.
We used the same method for the neural network too, testing all the embedding models and maintaining the same hyper parameters. The scores of the LSTM neural network for each embedding model are shown in Table 6.2. Also our model confirms the effectiveness of word embedding with specific domain, reaching the best score with word embedding trained on JoinData. The LSTM model is more accurate than the baseline, with an increase in performance of 10.7% for F1-micro, 2% for F1-macro and 6.6% for F1-weighted. The differences between the SVM model and the LSTM neural network are shown in Figure 6.3, Figure 6.4 and Figure 6.5, one for each evaluation metric. As already shown in the table, we have tested the LSTM network with all the embedding models at our disposal, concluding that the best performances are achieved using the medical domain models. Results are summarized in Figure 6.6

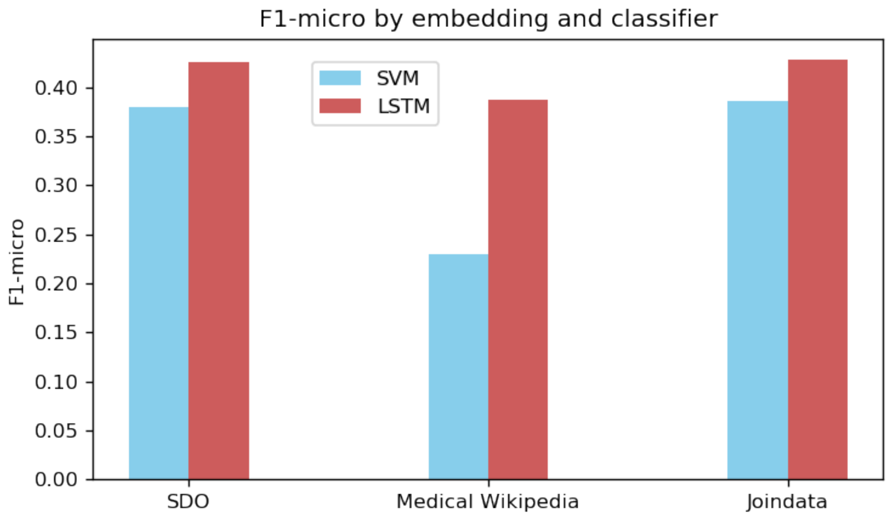


Figure 6.3: F1-micro differences between the two classifiers, grouped by word embedding model.

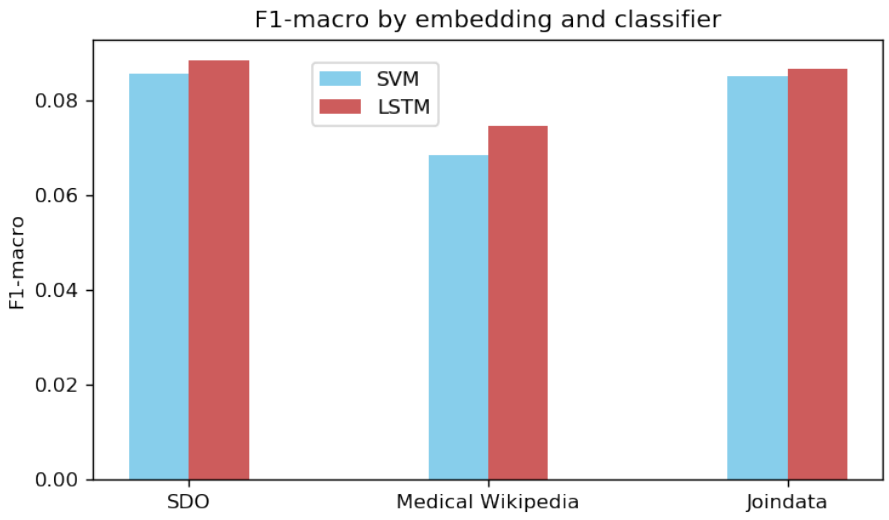


Figure 6.4: F1-macro differences between the two classifiers, grouped by word embedding model.

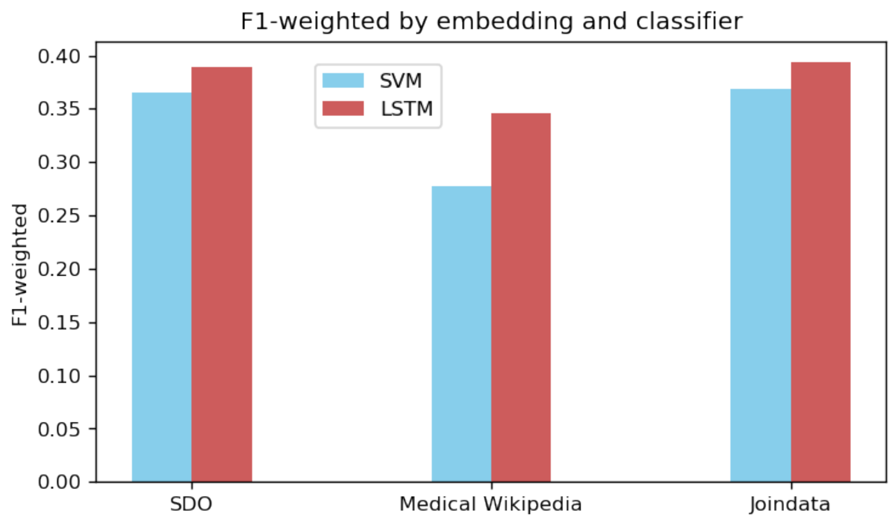


Figure 6.5: F1-weighted differences between the two classifiers, grouped by word embedding model.

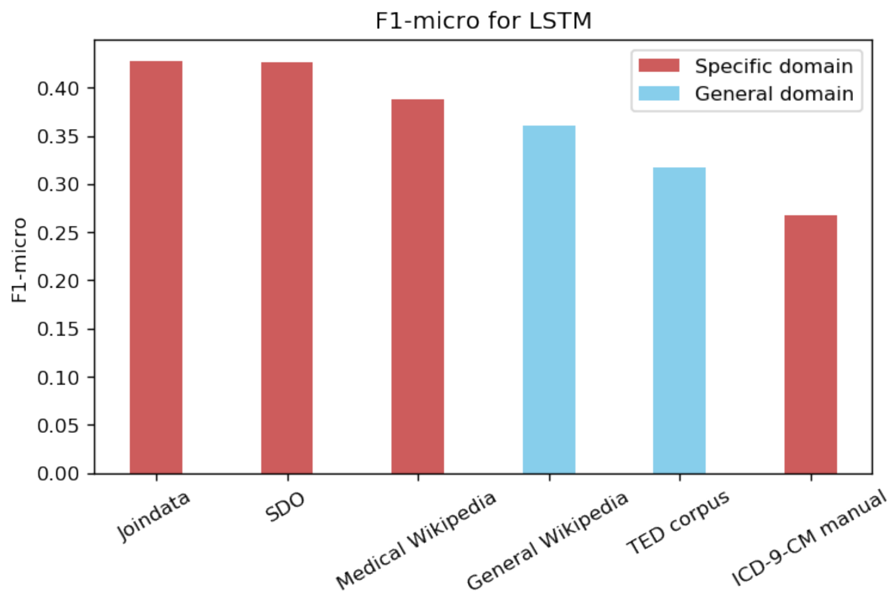


Figure 6.6: LSTM for F1-micro tested on each embedding model.

It is noted, however, that the model trained in the ICD-9-CM manual does not achieve satisfactory results, given that it contains very technical and medium-short textual descriptions. It should also be considered that the medical Wikipedia model achieved an accuracy improvement of 7.6% compared to general Wikipedia model, confirming that, although the former is a subset of the latter, the specificity of the domain influences the final performance.
We also observed an advantage in terms of prediction time of the neural network with respect to SVM, a fundamental parameter in case we want to use the classifier for real-time services. For example, the prediction on the same computer of the test set output, represented by a vector of 2861 elements, took about 1.5 seconds for the LSTM classifier and 392 seconds for SVM, showing a SVM execution time 261% higher than the neural network.

Chapter 7

Conclusion

The ICD classification based on textual data requires a great human effort and a waste of resources, and studies have been carried out for years on the automation of the classification. In this thesis, a classification model based on LSTM neural networks was presented and trained on a small diagnosis dataset labeled with one or more codes, provided by the Forlì emergency room. Our main goal was to overcome the accuracy of a traditional classifier using a deep learning approach combined with word embedding. Our LSTM classifier achieved a 10.7% improvement in F1 compared to SVM, despite the small size of the dataset and the excellent efficiency of the Support Vector Machine. We also compared different models of word embedding provided by a related work, trained both on corpora of general domain and medical domain, showing how the latter contribute significantly to an increase in performance. We observed that the embedding model trained on medical Wikipedia obtained better performances than the one trained on full Wikipedia, showing that the size of the embedding corpus is not necessarily the best choice in cases of classification with specific domains. We have noticed that even by changing the embedding models, the best accuracy was always achieved by the neural network. Moreover, we also noticed a drastic reduction in the prediction time of our model with respect to the baseline, achieving an improvement of 261%, given that, to make the classification with SVM multi-label, we need as many classifiers as the classes in the dataset, inexorably slowing down the process. Another advantage of our model is the possibility of being able to learn incrementally without having to re-train on the whole dataset, as opposed to a

model based on SVM. In this way it is possible to continue to give new inputs to the neural network maintaining the calculated weights up to that moment.
Despite these promising results, the accuracy value does not allow to use this model as a hard classifier, but rather as a means of classification aid, for example by building an ordered list of probable codes to choose from. This soft classification model will be tested in some Italian hospitals, using the codes chosen as new inputs for the neural network, allowing a continuous training and a probable consequent increase in performance.

Bibliography

[1] Leah S Larkey and W Bruce Croft. Automatic assignment of icd9 codes to discharge summaries. Technical report, Technical report, University of Massachusetts at Amherst, Amherst, MA, 1995.

[2] Lucian Vlad Lita, Shipeng Yu, Stefan Niculescu, and Jinbo Bi. Large scale diagnostic code classification for medical patient records. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II*, 2008.

[3] Yitao Zhang. A hierarchical approach to encoding medical concepts for clinical notes. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Student Research Workshop*, pages 67–72. Association for Computational Linguistics, 2008.

[4] Adler Perotte, Rimma Pivovarov, Karthik Natarajan, Nicole Weiskopf, Frank Wood, and Noémie Elhadad. Diagnosis code assignment: models and evaluation metrics. *Journal of the American Medical Informatics Association*, 21(2):231–237, 2013.

[5] Ramakanth Kavuluru, Anthony Rios, and Yuan Lu. An empirical evaluation of supervised learning approaches in assigning diagnosis codes to electronic medical records. *Artificial intelligence in medicine*, 65(2):155–166, 2015.

[6] Chin Lin, Chia-Jung Hsu, Yu-Sheng Lou, Shih-Jen Yeh, Chia-Cheng Lee, Sui-Lung Su, and Hsiang-Cheng Chen. Artificial intelligence learning semantics via external resources for classifying diagnosis codes in discharge notes. *Journal of medical Internet research*, 19(11), 2017.

[7] Tal Baumel, Jumana Nassour-Kassis, Raphael Cohen, Michael Elhadad, and Noemie Elhadad. Multi-label classification of patient notes a case study on icd code assignment. *arXiv preprint arXiv:1709.09587*, 2017.

[8] The skip-gram model. Available at `https://www.tensorflow.org/tutorials/representation/word2vec`.

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

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[11] Jürgen Backhaus. The pareto principle. *Analyse & Kritik*, 2(2):146–171, 1980.

[12] Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(Jul):2079–2107, 2010.

[13] Giovanni Seni and John F Elder. Ensemble methods in data mining: improving accuracy through combining predictions. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 2(1):1–126, 2010.

[14] Chabacano. Overfitting diagram, 2008. [Online; accessed 24-February-2008].

[15] Overfitting diagram. Available at `https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Overfitting.svg/1920px-Overfitting.svg`.

[16] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[17] Christopher M Bishop. Pattern recognition and machine learning (information science and statistics) springer-verlag new york. *Inc. Secaucus, NJ, USA*, 2006.

[18] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[19] Marcel van Gerven and Sander Bohte. *Artificial neural networks as models of neural information processing.* Frontiers Media SA, 2018.

[20] Build with ai | deepai. Available at `https://deepai.org/machine-learning-glossary-and-terms/neural-network`.

[21] Neural network training diagram. Available at `https://cdn-images-1.medium.com/max/1600/1*mi-10dMgdMLQbIHkrG6-jQ.png`.

[22] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[23] lstm diagram. Available at `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png`.

[24] Scikit-learn, a set of python modules for machine learning and data mining. Available at `http://scikit-learn.org`.

[25] Keras: Deep learning for humans. Available at `https://github.com/keras-team/keras`.

[26] Tensorflow, an open source machine learning framework for everyone. Available at `https://www.tensorflow.org/`.

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Appendix A

Code

Classifier training and all tests were performed with an Intel 6700HQ 2.60GHz quad-core CPU and 16GB of DDR4 RAM. As a development environment we used Jupyter Notebook with Python 3.6.7, installed on Ubuntu 18.04 operating system
The following libraries for machine learning were used for the implementation of the classifier: Keras 2.2.2, TensorFlow 1.10.1, scikit-learn 0.19.1.
In this appendix only the most relevant code has been reported.

A.1 Loss

```
def weighted_binary_crossentropy(y_true, y_pred):
    epsilon = tfb._to_tensor(tfb.epsilon(), y_pred.dtype.base_dtype)
    y_pred = tf.clip_by_value(y_pred, epsilon, 1 - epsilon)
    y_pred = tf.log(y_pred / (1 - y_pred))
    loss = tf.nn.weighted_cross_entropy_with_logits(
        targets=y_true,
        logits=y_pred,
        pos_weight=4)
    return tf.reduce_mean(loss, axis=-1)
```

A.2 Model

```
def build_model():
  earlyStopping = EarlyStopping(monitor='val_loss', patience=200)
  model = Sequential()
  model.add(Embedding(vocabulary_size, 200, input_length=30,
    weights=[embedding_matrix],
    trainable=False))
  model.add(LSTM(300, dropout=0.5))
  model.add(Dense(3248, activation='sigmoid'))
  model.compile(loss=weighted_binary_crossentropy,
    optimizer=adam, +
    metrics=['accuracy'])
```

List of Figures

List of Tables

Ringraziamenti

Ringrazio il mio relatore Danilo Montesi per avermi dato la possibilità di lavorare ad un progetto importante e stimolante come questo.

Ringrazio Stefano per la sua immensa disponibilità e pazienza che mi hanno permesso di imparare e mettermi in gioco.

Ringrazio i miei nonni, colonne portanti di una famiglia che si vuole bene.

Ringrazio i miei genitori che mi hanno sempre supportato con entusiasmo in ogni mia scelta.

Ringrazio le mie sorelle per l'affetto e la fiducia dimostrati ogni giorno. So che posso sempre contare su di voi.

Ringrazio i miei zii, i cugini e tutta la famiglia allargata.

Ringrazio Eleonora, compagna di vita e migliore amica. Grazie per essermi stata a fianco in ogni momento ed avermi concesso uno spazio nella tua vita.

Ringrazio gli amici del liceo per continuare a portare avanti un'amicizia importante e non scontata.

Ringrazio Osso, Moros e Cala per le infinite risate e le sessioni di studio migliori di sempre.

Infine ringrazio tutti gli amici e le persone con cui ho condiviso anche un piccolo momento felice.