POLITECNICO DI TORINO

Master Degree in Computer engineering

Master Degree Thesis

Early Dementia-Disease Detection Prototype System

Supervisor
prof. Elio Piccolo
ing. Marco Bazzani
ing. Enrico Ferrera

Candidate
Davide Visentin

April 2019

Contents

Chapter 1

Introduction

1.1 What is Natural language processing and which are its applications to impaired language

Natural Language Processing (NLP) is a field of artificial intelligence that aims to process in an automatic way the natural language, that is the language talked by human beings. A possible objective is to extract some sort of information from a sequence of utterances. This information can be relatively simple (e.g. the number of words, although also this task can become very complex depending on the language and the definition of word) or very high level (e.g. sentiment classification). Another objective can be to generate utterances in a natural language based on a previously learned language model. This thesis uses NLP techniques only for features extraction.
The advancements in the NLP field have led to its application also in the biomedical sector. Concerning in particular impaired language, two main paths are being followed. One is the study of the disturbed language, both for linguistic research and for diagnostic purposes. The other is the development of tools for linguistic rehabilitation. This thesis falls in the first category, and a review of the state of the art of the application of NLP to the study of impaired language in the particular case of subjects affected by dementia will be given in section 2.1. Concerning the second category, [1] is a meta-analysis (probably a bit outdated, having been published in 2015) of available tools for the rehabilitation of aphasic patients.

1.2 What is Alzheimer's Disease

Alzheimer's Disease (AD) is a chronic neurodegenerative disease. It covers around 80-85% of all cases of dementia. It usually occurs in people older than 65 years, although there are some early-onset forms, mainly due to genetic factors, that begin much before (sometimes already in the 40s).
One of the main symptoms of AD is the loss of memory, that begins with an increased difficulty to consolidate new memories (anterograde amnesia), then proceeds with a progressive retrograde amnesia. The amnesia affects mainly the semantic memory, but in the late stages, also the procedural one is compromised. Another important symptom is the difficulty to plan and to focus on tasks. Both the amnesia and the difficulty to focus and plan affect the language, that acquires some peculiar characteristics that will be described later in this thesis. In late stages, beside a general worsening of cognitive functions, also physiological functions are affected. If there are not co-morbilities, infections, dehydration or malnutrition, that are by far the most common causes of death, the patient dies when the disease compromises the respiratory center in the brainstem. Both the average age at which AD becomes symptomatic and the average life expectancy after the diagnosis strongly depend on the lifestyle of the patient.
The impairment of cognitive and physiological functions is due to an anomalous accumulation of amyloid beta protein in the brain (in the extracellular space) that reduce the ability of the neurons to transmit impulses, and of tau protein ("tau tangles") inside the neurons, that interferes with the transportation of nutrients and other substances inside the affected cells. This causes the death of neurons, whose remains aggregate to amyloid to form plaques. This make the immune system react and the inflammatory process further speed up the death of other neurons. The macroscopic result is a loss of weight and volume of the brain.
The accumulation of amyloid beta is a prodromic sign that can appear several years before the diagnosis. At the beginning, the damages caused by the amyloid are so small that the brain is able to maintain its functions completely. Then, the so-called Mild Cognitive Impairment (MCI) occurs. The deterioration of cognitive functions is usually evident enough to be considered a sign of AD only some year later. Additionally, especially in young patients, the first symptoms are often confused with those caused by other pathologies, both neurological (e.g. frontotemporal dementia) or psychiatric (e.g. depression). Since most therapies (that by now are unable to stop the progression of the disease, but they can alleviate some of the symptoms) give better results when they are started in the early stages of the disease, it's evident how is important to improve the accuracy of the diagnosis.
For a more detailed and extensive description of AD, see for example [2] and [3] or

the website of the Alzheimer's Association[1], where you can also find information about the Italian situation[2].

1.3 Objective

The objective of this thesis is to evaluate the feasibility and eventually build a prototype of an automatic system able to support the physicians in the evaluation of mild cognitive decline from speech characteristics, with particular attention to AD. The main features of the system should be: support for Italian language, complete automation from audio transcription to samples classification, low cost and predisposition to be adapted to other languages or pipeline components.
The objective evolved during the progress of the thesis to cope with various issues that arose and to adapt better to the needs of clinicians of the speech therapy department of the university of Turin with which we collaborated.
Initially, the aim of this thesis was to create a system to help with the diagnosis of Alzheimer based on a Raspberry Pi with an always-on microphone to place at the patient's location. the subject's speech would have been extracted from the audio source and processed to extract useful language features. Then, those features would have been used as inputs for a classifier to extract a diagnostic hypothesis. However, that version of the project was problematic for many reasons.
First of all, it soon appeared to be necessary to recognize speakers, otherwise by indiscriminately analyzing both patient's and caregiver's speech the diagnosis would have been biased toward "healthy". More precisely, it would have been necessary to do speaker segmentation of the audio source. Also assuming that the caregiver and the patient never talk contemporary, this is a very difficult task. The complexity of speaker and speech recognition would have been exacerbated by the fact that these tasks would have been carried on in a noisy environment and with old patients, which voice volume is often quite low.
Another problem is the definition of the time frame to analyze before giving a diagnosis. On one hand, a too short time frame can be little representative of the real condition of the patient (for example if an AD patient has a moment of lucidity or an healthy one has an hesitation). On the other hand, a too large time frame can be not sensitive enough to crisis that sometimes occur in Alzheimer patients. Moreover, it's difficult to establish when to start the time frame.
Due to the aforementioned issues, the project was simplified in order to be concluded in a reasonable amount of time and effort and concentrating more on natural language processing. The new application is intended to be used during a standard

[1]`www.alz.org`

[2]`https://www.alz.org/it/dementia-alzheimers-italy.asp`

examination carried out by a physician in an ambulatory and by using cheap and largely available hardware and software.
Instead of an environmental microphone, a smartphone is used to record the audio. The device is placed on the table between the two interlocutors. The time frame to consider for the analysis corresponds to the duration of the examination (the physician starts the recording at the beginning and stops it at the end). The speaker segmentation problem is still not solved, but due to the specific setting in which the recording is performed, the patient should talk much more than the examiner.
Python is the chosen programming language, because there are a lot of available libraries for machine learning and natural language processing and it makes easy and fast to create a prototype.

1.4 Characteristics of Italian and differences with respect to English

The work of this thesis focus on the Italian language mainly for two reasons. The first is that the long term objective is to build a tool that can be used in the hospitals of Turin and of the rest of Italy, where obviously most of the patients speak only Italian. The second is that most of the researches available in literature work on the English language, but it can be interesting to evaluate how the currently available NLP libraries perform and how the relevant features to discriminate between healthy and AD individuals change when considering another language.
English and Italian are very different languages. Whereas the first is a Germanic language which lexicon has been influenced by Romances languages, especially French, Italian is a pure Romance language, the nearest to Latin. The differences between the two languages refer to all the levels from phonology to semantics and pragmatics, but for the purpose of this thesis, only some of them are relevant.
For what concerns the morphology, it's important to notice that English doesn't inflect lexemes according to gender, whereas Italian does, and the plural form exists only for nouns and is usually easier to form than in Italian. For this reason, often in Italian personal pronouns are omitted, and there is much more freedom in the ordering of the words in the sentence. Moreover, in Italian is very common to implement verbs tenses by applying suffixes or even by changing the stem, while in English many tenses are provided by adding particular words (e.g. will, would, to, etc.).
Syntactically, a relevant difference is that for English, since the meaning of a sentence relies a lot on the order of the words, it's possible to detect questions also when there is no punctuation, while in Italian this is often more difficult. Another important difference, dependent on the morphological ones, is that in Italian there is gender and number agreement between a noun and all the words related to it,

while in English this is not true for adjectives and it's more complicated for pronouns and verbs.
From the semantic point of view, an important difference is the fact that possessive pronouns in English are inflected only based on the gender and number of the subject, while in Italian they are inflected based on the object (e.g. *"Tizio prende il suo ombrello / la sua giacca / le sue scarpe"*, in English "Tizio takes his umbrella / his jacket / his shoes") but there are always different pronouns for singular and plural subjects (e.g. *"Tizio e Caio prendono il loro ombrello / le loro giacche / il proprio ombrello / i propri ombrelli / le proprie giacche"*, in English "Tizio and Caio take their umbrella / their jackets / their umbrella / their umbrellas / their jackets").
Another important difference is the use, in Italian, of the third singular feminine person as a courtesy form to refer to an interlocutor with which the speaker is not familiar (e.g. the physician that is examining the patient). This obviously complicate the anaphora resolution.

Chapter 2

Literature analysis

2.1 Related works

As claimed by [4], Alzheimer's disease (AD) can be predicted effectively by analyzing the changes in the language abilities of the patient. These changes involve both phonetics (e.g. syllables per minute, hesitations), lexicon and semantics (for example wrong words) and pragmatics, while only in late stages of the disease syntax errors are observed. Also [5], a meta-analysis comparing 61 articles concerning language disorders in several forms of dementia and many different languages, comes to similar conclusions for AD.
This fact led to several researches aimed to find metrics to identify AD patients based on the analysis of (more or less) spontaneous speech. Singh ([6][7][8][9]) defined 8 attributes to characterize dysphasic patients: noun, pronoun, adjective and verb rates, Type-Token Ratio, Brunét's Index, Honoré's Statistics and Clause-like Semantic Unit rate (for detail see section 2.2). [10] was able to discriminate AD patients from a control group by using the first seven parameters, while the Clause-like Semantic Unit was found to be not relevant. [11][12] apply these metrics, plus others taken from [13], [14], [15] and [16] (namely: percentage of go-ahead utterances, repeated words, pauses, incomplete words (also called restarts), filler utterances, paraphrasing and syllables per minute (section 2.2)) on three different corpus containing conversations transcribed by hand, finding contrasting results. The differences were explained by comparing with the same metrics the interviewer of the different corpus, finding that in some of them they use a lower lexical richness, probably to adequate to the patients they were talking to. However, it was always possible to classify the speakers using the statistically relevant metrics for each corpus (although with different accuracies and with overfitting for the non-Alzheimer class due to classes unbalance). Other useful attributes were found by [17] (question and confusion rate, no answer count, phonemes per word and word

entropy (section 2.2).
In all the previous cases, the classification was done on manually transcribed conversations. [18] identified 4 acoustics parameters that can discriminate AD patients from other people: articulation rate, speech tempo, hesitation ratio, grammatical errors (only in late stages) (section 2.2). Based on these metrics plus duration and number of silent and filled pauses, [19] built an Automatic Speech Recognition (ASR) software able to classify patients without the need to transcribe by hand the conversations.
Other successful attempts to analyze disfluent speech with ASR were carried out by [20] and [21]. The first is based on a test where the subject has to retell a story, and the score is based on the story elements that are recalled by the patient. So it doesn't use on any of the previously listed features, but it try to align the automatically transcribed text to the reference in order to find which story elements are present (that technique is based on a previous experiment on manually transcribed text, see [22]). The second (that, to be precise, aims to detect progressive aphasia) exploit, additionally to some of the textual and acoustic metrics mentioned before, the familiarity, imageability and age of acquisition of words by consulting special dictionaries.
A very recent paper ([23]) proposed to create a graph representation of the text (manually transcribed in this case) where links represent the co-occurrences of the words, in order to extract a set of topological features (pagerank, betweenness, eccentricity, eigenvector centrality, average degree of the neighbors of a node, average shortest path lenght of a node, degree, assortativity degree, diameter and clustering coefficient (section 2.2)). There features, together with other more traditional metrics, were found to be useful to detect mild cognitive impairment. For the tests, three different corpus were used, included the cookie theft dataset, that is very similar to one of those used in this thesis. In order to cope with the short length of the texts (that make the graph almost linear), edges were added also for all pairs of words that are similar enough.
Another recent paper ([24]) tried to use deep learning for this task, with a top accuracy of 91.1% obtained using a hybrid Convolutional Neural Network - Recurrent Neural Network (CNN - RNN) on manual transcripts annotated with Part-of-Speech tags. However, this result may be optimistic, because the dataset was created by splitting each transcript in utterances and by considering each utterance as a sample. Moreover, the used corpus contains multiples transcripts for many subjects. So, if no particular countermeasures were taken (and the paper doesn't mention them), the training, development and test set were not independent.
Concerning Italian language, [25] is a relevant paper published by a group of the University of Bologna few months before the end of this thesis. In that research, 96 audio files are manually transcribed, then a large set of acoustic and linguistic features is extracted and evaluated to determine the relevance of each of them.

2.2 Analysis of disfluencies

2.2.1 Nouns, pronouns, verbs and adjectives rates

Anomic aphasia is one of the most evident symptoms in AD subjects. So the Noun Rate (2.1), computed as the number of nouns divided by the number of words, tends to be lower than in healthy people. The nouns that the patient is not able to recall are often substituted by pronouns (so the Pronoun Rate, (2.2), increases) and sometimes by circumlocutions containing also adjectives and verbs (so the results of equations (2.3) and (2.4) may be a little higher than for healthy people). In the case of the Italian language, the PR increases also because AD patients tend to explicit them much more often than usual (Italian often doesn't require to explicit personal pronouns, see section 1.4). Other frequent replacements for the forgot nous are the so-called "thing words", that are nouns with a very generic meaning (although they can have also more specific meanings), like "cosa", "coso", "roba", "affare", "tipo", "tizio" and so on. The presence of a lot of these generic nous can affect the significance of the NR, but from the collected bibliography is not clear if it's better to consider them or not (in the second case, they should be removed only from the noun or also from the word count?).

$$NR = \frac{N}{W} \tag{2.1}$$

$$PR = \frac{P}{W} \tag{2.2}$$

$$AR = \frac{A}{W} \tag{2.3}$$

$$VR = \frac{V}{W} \tag{2.4}$$

Additionally from the already mentioned statistics, this thesis will evaluate the opportunity to take into account the moving average version of them. That is, a sliding window will be moved on the text, one word at a time, and each of the aforementioned rates will be calculated for the window at every step. At the end, for each statistic, the average will be calculated. This technique is historically used for other metrics (e.g. subsection 2.2.2) to avoid biases due to the variable length of the analyzed texts, but it's not evident if it can be useful also in this case.

2.2.2 Lexical richness indexes

The reuse of words already retrieved from the semantic memory is one of the peculiar coping mechanisms used by AD patients to reduce the effort needed to continue a conversation. The magnitude of this phenomenon can be measured by three metrics: Type Token Ratio (TTR), Brunét's Index (BI) and Honoré's Statistics (HS). TTR (2.5) is computed as the cardinality of the used vocabulary (the number of different words) over the total number of words. Note that sometimes in literature the TTR is the reciprocal or the percentage of the definition given here. In any case, the main disadvantage of the TTR is that for short utterances it tends to represent an exaggeratedly high lexical richness, and the opposite for very long utterances.

$$TTR = \frac{V}{W} \tag{2.5}$$

An improved version of TTR, independent from the text length, is the Moving Average TTR (MATTR, [26]). Analogously to the moving average version of the parts-of-speech rates, MATTR is based on a sliding window of fixed size that move through the text. For each position, TTR is computed. At the end, the average is computed.
Other two metrics that are independent from text length are Brunét's Index (BI) (2.6) and Honoré's Statistics (HS) (2.7). In particular, Honoré's Statistics takes into consideration also the number of words that appear only once (V_1).

$$BI = W^{V^{0.165}} \tag{2.6}$$

$$BI = \frac{100log(N)}{1 - \frac{V_1}{V}} \tag{2.7}$$

The richer the language, the higher TTR and HS are and the lower BI is.

2.2.3 Cohesion

The cohesion is the property of a text that makes all its parts to be correctly linked between each other so that they may combine in a significant way. AD subjects usually have more difficulties to formulate cohesive sentences because they don't find the correct words or they lose the focus, especially when the sentence is long. A metric that has been proposed to quantify the cohesion of a text is the Clause-like Semantic Unit (CSU) rate.
Basically, a CSU is a cohesive utterance. It's computed according to a set of rules described for example in [7], that take into account the usage of conjunctions, the repetition of words, the automatisms and many other elements in order to split a disfluent text into cohesive units. At the end, the CSU rate is computed as the number of CSU per 100 words. The lower, the better.

2.2.4 Prosody, repetitions and restarts

The difficulty to retrieve the correct words and to stay focused affect also the prosody. In particular, AD subjects' speech is usually slower and contain more and longer pauses (in some papers called hesitations) than in the case of healthy people. The speed of the speech is usually calculated in syllables per minute, because syllables can be retrieved from text and have a more constant duration than words. Pauses can be considered by taking into account only their frequency or also their duration, or by simply considering them in the computation of the syllables per minute. Alternatively to syllables per minute it's possible to calculate the articulation rate (i.e. number of phonemes per second ignoring pauses) or the speech tempo (i.e. number of phonemes per second including pauses).
Where a pause occurs and what happens after it is also important. In particular, in literature are usually considered the case where there is a repetition of one or more words around the pause and the one where the pause occurs in the middle of a sentence (sometimes also in the middle of a word), and after the pause the speaker starts a new, different sentence. The last case is called restart.

2.2.5 Pragmatics

An AD subject usually adopt some strategies to try to hide his deficits. An example is the use of go-ahead utterances, that are composed by few words or "noises" (1, 2 or 3 in most cases) just to tell the interlocutor that he is listening and he wants him to continue to talk (e.g: "uh huh", "ok", "I understand" etc.). Some of these utterances can be used also as fillers during hesitations.
Another example is paraphrasing. This can be used both to reduce the effort in producing an answer by reworking the question or to cope with the difficulty to retrieve a word by circumlocuting it.
Other pragmatic features considered in literature are question and confusion rate. The first is calculated as the number of "question words" (e.g. the w-words) divided by the total number of words, whereas the second is, analogously, the rate between the number of words like "maybe", "perhaps", etc. and the total number of words. A defect of the question rate is that some of the question words can be used also outside questions, and that's particularly relevant for Italian language, where for example "what" can be *"che cosa"* but also *"cosa"*, that also means "thing", or *"che"*, that can be an adjective, conjunction or pronoun and is used in a lot of situations. So it's at least necessary to also consider the POS, but especially for *"che"* the tagger can have difficulty to distinguish if it is an interrogative adjective or pronoun or a conjunction.

Chapter 3

Techniques and tools

3.1 Relevant Natural Language processing (NLP) techniques

3.1.1 Basics

From the lexico-syntactic point of view, it's necessary to individuate words and possibly base noun phrases (that are composed by a noun plus eventually words that describe that noun; e.g.: "the world's largest tech fund") and sentences. These techniques are called word, noun and sentence chunking or tokenization. For each of these types of chunking, several diffentent algorithms can be used. The simplest merely split the text based on specific delimiters (spaces for words chunking, punctuation for sentences tokenization). Obviously these symple techniques doesn't allow to implement noun chunking, and give poor results also for the other two kind of tokenizations (consider for example the cases where there are acronyms). More sophisticates methods, instead, after the basic splitting extract a set of features from the text of each aspirant token to check through a classifier if it can be confirmed or not. An example of this technique is the punkt algorithm used for sentence chunking in NLTK ([27], [1]). Always NLTK provides another method for word tokenization, based on regular expressions. Both classification and regular expressions allow to obtain better results and to implement also noun chunking. Yet different methods are proposed by spaCy [28], that for word tokenization apply the basic step and then, for each chunk, checks if it matches an exception rule (e.g. "let's" must be splitted in "let" and "'s", whereas "U.K." must be kept as is) or if an affix can be split off (e.g. punctuation, quotation marks, parentheses, hyphens,

[1]`https://www.nltk.org/_modules/nltk/tokenize/punkt.html`

etc.). [29] provides an interesting and effective way to add punctuation to completely "flat" text, like the one returned by the majority of the speech recognition tools. It can be used as a preprocessing step before performing tokenization.
Other two foundamental techniques are Part-Of-Speech (POS) tagging and lemmatization. The first associate to each word a tag corresponding to its part of speech (noun, adjective, verb, pronoun, etc.). When a word can be more than one POS, the tagger usually take a choice by applying a probabilistic model (e.g.: in english a word following "the" is probably a noun). The second takes words already tagged and search their lemma in a lookup table. Some lemmatizer also returns some morphological attributes such as gender, number, tense, etc.. Often, the verb tenses and the different kinds of pronouns and adverbs are recognized already by the POS tagger. Another important task is syllabification.

3.1.2 Syntax parsing

Syntax parsing for natural language is a challenging task, due to the ambiguity of the language (at all levels: consider for example polysemic words, figurative language, the influence of prosody, etc.). This ambiguity is explained by Chomsky ([30][31]) by assuming that language has two level: a deep structure that is the meaning of a sentence, and a surface structure that is the sequence of words that compose the sentence. A set of transformations can be applied to the surface structure without changing the deep one (e.g. "John eat an apple" and "An apple is eaten by John"). At the same time, the same deep structure can be realized by multiple surface structures. The class of grammars that can generate a natural language is therefore that of transformational grammars. The Chomsky initial theory as been modified multiple times. One of the most important variants is the one described by Chomsky himself in [32] that, among other things, introduces the concepts of empty categories and X-bar theory.
Empty categories are placeholders for referring expressions, pronominals and anaphors. They are used to reconduct some kind of sentence to a standard grammar. A tipical example are the wh-questions, like "where is John gone?". In this example, the empty categories theory says that "where" has moved from the end to the beginning of the sentence, leaving a so called trace in the original position.
The X-bar theory states that all syntagmas can be reduced to analogous parse trees. In particular, each syntagma is an X phrase composed by an X-bar and a specifier (e.g. a determiner). The X-bar can be composed either by another X-bar and an

adjunct, or by an head X and zero or more complements[2]. The X is a placeholder for the part-of-speech of the head, and the position of specifier, adjunct and complements depends on the language, the type of sentence and pragmatic factors (e.g. what is the theme and what is the rheme).
In [33] Chomsky presents the bare phrase structure, that is an attempt to overcome x-bar theory with an even simpler set of rules ("move" and "merge"). However this theory has been criticized to be an unmotivated radical change to previous theories, to it still has not replaced the X-bar theory.
The above theories allow to implement phrase structure grammars, that make possible to parse sentences by using a set of reduction rules.
An alternative way to parse sentences is dependency parsing. In this case, instead of reducing sequences of words to syntagmas, the objective is to find the dependency relationships between words (e.g. in subject-verb-object languages, a noun or a pronoun on the left side of a verb depends on it as a subject).
Different dependency grammars use more or less of the theories seen above. For example, many of them assume that each verb has a valence, and they always use endocentric constructions (a grammatical construction is endocentric if it has the same linguistic function as one of its parts) like the X-bar theory (where a syntagma has the same function of its head). Dependency grammars are also usually aware of empty categories.
An important concept used many dependency grammars is catena ([34] [35]). Assuming that a sentences exists in two dimensions, the x-axis represent the precedence between words, and a continuous combination of elements along this axis is a string, whereas the y-axis represent the dominance relationships beween elements, and a continuous combination along this axis is a catena. Catenaes are used for example to linearize (remove crossing dependency links) parse trees of sentences containing empty categories or to better interpret idiosyncratic language.

3.1.3 Semantic analysis

For what concerns semantics, it's needed to retrieve the semantic relationships between words, such as synonymy, hyponymy (is-a relation, e.g. "cat" is an hyponym of "feline") and hypernymy (the reverse, e.g. "feline" is an hypernym of "cat"), meronymy (part-of or member-of relation, e.g. "finger" is a meronym of "hand") and holonomy (the opposite of meronymy), troponymy (usually defined only for verbs, is the case when an action is "a way to do" another action, e.g. to lisp is a

[2]The difference between adjunct and complement is that, in valency theory, complements are mandatory elements that saturate the syntagma head valence (usually only verbs have complements), whereas the adjuncts are optional.

troponym of to talk) and entailment (only for verbs). These relationships are transitive (e.g. a cat is an hyponym of "feline" and "feline" is an hyponym of "animal", so "cat" is also an hyponym of "animal"), an the meronymy/holonymy, troponymy and entailment relations are inherited by the hyponyms of the linked terms. When two terms share an hypernym, they are called coordinate terms.
A convenient way to represent these relationships is to use a semantic network. In this way is also possible to estimate the semantic similarity between two terms (usually two lemmas) by looking their distance from the nearest common hypernym, if they are coordinate terms, or the distance between the two if one is an hypernym of the other, and eventually by considering also the other relations between them. Note that this estimate can be inaccurate if different parts of the network have different granularities. An example of general purpose semantic network is WordNet [36].
An alternative way to evaluate the semantic similarity between two words is to train a model that associate to each word a vectorial representation, so that the similarity between two vectors is roughtly proportional to the similarity between the corresponding words. This technique is called word embedding, because it maps a corpus, that can be seen as a space with one dimension for each different word, to a space with many less dimensions (e.g. few hundred). The vectors can have different meaning. In the case of Explicit Semantic Analysis (ESA, [37]), for example, each word is associated to a vector of explicit features, like the Wikipedia entries where it appears [38]. It's important that the features are ortogonal. Latent Semantic Analysis (LSA, [39]), instead, can be trained on any corpus, and the vectors are post-processed to reduce the number of dimensions (the input is an N * M matrix, where N is the cardinality of the vocabulary and M is the number of texts in the corpus, and the output is a matrix N * X, where is is a relatively small number) by using the singular-value decomposition. The downside is that at the end the information contained in the vectors is no more explicit. Moreover, it's very expensive to create the vectors from a large corpus. Both ESA and LSA original purpose was to give a way to measure the semantic relatedness between texts (for example, between Wikipedia articles), but have then been found to be quite effective also to measure the semantic similarity between words.
A more recent word embedding technique is word2vec [40], that can create the vectors by using one among two different models, which are both two layer neural networks trained with unsupervised algorithms. One is a Continuous Bag-of-Words Model (CBWO), that takes as input a context (i.e. a set of words that appear before and after the target word in a point of the corpus) and returns a prediction of the missing word. That prediction is compared with the real target and the error is back-propagated to update the weights. The other is a continuous skip-gram model, that takes as input the target word and returns a prediction of its context, updating the weights depending on the error. The vectorial representations of the words are contained in the matrix of the weights of the connections between the

input and the hidden layer. In fact, that matrix has size VxN, where V is the size of the vocabulary (there is one input neuron for each different word) and N is the length of the vectors (that can be changed by changing the size of the hidden layer) (fig. 3.1). CBWO is faster to train, but skip-gram is more accurate for infrequent words.

Word2vec is much more efficient than previous techniques, and gives a better

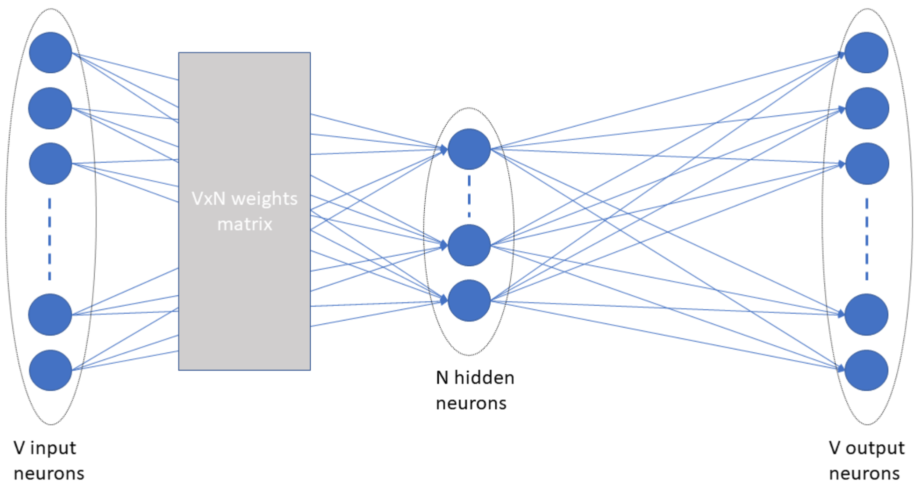

Figure 3.1. The general schema of the neural network used to train the word2vec model. Projection layers are omitted. The matrix depicted contains the words vectors.

representation of the syntactic and semantic relations between words. Additionally to semantic similarity, it allows to perform many other operations by using vector arithmetics. For example, it's possible to answer to question in the form "A is to B as C is to X". Indeed, the vector obtained by doing vector(B) - vector(A) + vector(C) is probably very near to the correct answer (e.g. nearest_neighbor(vector(king) - vector(man) + vector(woman))=vector(queen)). This have interesting consequences. For example, given a single words couple (plural form, singular form), it's possible to possible to obtain the singular form of any plural word. So, with few predefined words couples, it's possible to implement a lemmatizer. In a similar way is possible to get hypernyms, meronyms and so on. Unfortunately, in practice it's difficult to learn vectors that are adequate for many different kinds of word analogies at the same time. [41] analyzes this problem, also by considering alternative ways to find the unknown term of the analogy.

The main competitor to word2vec today is GloVe (Global Vectors, [42]). This

model is trained in two steps. First, a words co-occurrences matrix (where each element (i, j) indicates the number of times the word i occurs in the context (defined as in wordnet) of word j) is constructed from the entire corpus. Then, for each non-zero element of the matrix, one of the two words is considered as been part of the context, and its vector is not modifies, while the vector of the other word is computed so that its dot product with the vector of the context word is equal to the logarithm of the conditional probability that that word occurs given that the context word has occurred. This second step is repeated until convergence or until a maximum number of epochs.
GloVe performs generally better than word2vec on many tasks, but the co-occurrences matrix is very expensive to build, especially from the memory usage point of view.
Both word2vec and Glove can be directly applied on preprocessed text to improve their accuracy. For example, they both can handle n-grams if the words are linked with a symbol (like underscore, e.g. New_York), and they can distinguish polysemic words if the text is modified by appending to each word its POS tag and the optional entity tag (e.g. person, organization, country etc.) ([43]).
FastText [44] is another embedding technique, where the vectorial representation of a word is computed as the sum of the vectors associated to the n-grams composing it. This allows to compute on the fly a representation also for words not present in the training corpus, which is particularly useful for morphologically rich languages.
A different way to quantify the coherence (in particular the internal one) of a text is presented in [45]. Here, to each word, a set of concepts are associated. The internal coherence of a text is computed by choosing one concept for each word so that the sum of the distances in an ontology between the chosen concepts of all the possible pair of words is minimized. The lower that value, the more coherent is the text. Some ontology allow to find all the concepts that are "evoked" by a word, so the process can be fully automatized. The paper also suggest to use the described algorithm only when the ratio between the number of words associated with at least one concept and the total number of words is above some threshold, in order to avoid strange results when many words cannot be mapped to concepts in the ontology.

3.1.4 Discourse analysis

Another important NLP task useful for this thesis in co-reference resolution, which objective is to find sets containing an antecedent and one or more mentions referring to the same entity referred by the antecedent (similarly for cataphora). A simplified variant of co-reference resolution is anaphora resolution, where the constraint that the antecedent and the mention (called anaphor in anaphora resolution) must refer to the same entity is removed (e.g. in the sentence "no man said he was hungry", "man" is the anaphoric antecedent of "he", but there is not co-reference).

The basic anaphora resolution algorithm consists in trying to find, for each anaphor, the nearest (within a maximum distance) noun with compatible gender and number. This is already a quite difficult task for Italian, since it's an inflected language and the possessive pronouns are inflected according to the object. More complex approaches assign a score based on the distance from the anaphor and the role of the noun in the sentence (subject, object, etc.) to each possible antecedent, then choose the one with the highest score. In our case, however, it's not necessary to find the most probable antecedent, but it's sufficient to determine if there is at least one compatible with the anaphor.
If for anaphora resolution the most simple approach can be applied, for co-reference resolution this is not true. The difference is that, while in anaphora resolution an antecedent among the candidate ones is always selected, in the case of co-reference it's possible that none of the antecedents that syntactically agree with the anaphor also refer to the same entity. So it's necessary to perform at least semantic checks, and preferably also pragmatic ones (the latter are useful for example to interpret idiomatic expressions).
The usual approach to co-reference resolution is to use a pipeline that first search for all possible mentions, then apply a set of check at all levels (from lexical to discourse ones) to find the true co-references. A famous implementation of this technique is [46]. An alternative, that among other differences uses Word2Vec instead of Wordnet to compute semantic distances, is [47].
An accurate co-reference resolution is a prerequisite for more complex discourse analysis. An example [48], that is a technique to evaluate the local coherence (i.e. coherence inside a discourse segment, as defined in [49]) of discourse. This method is based on the concept that each utterance of a coherent discourse as one backward-looking center, that is the most important (from the point of view of discourse structure) mention to an entity in one of the previous utterances, and a set of forward-looking centers, that may be referred by the backward-looking centers of the subsequent utterances. Note that a center can syntactically be either very simple (e.g. a pronoun or a noun) or very complex (e.g. a noun phrase or even a complex sentence, such as "The man that is talking on phone to the woman that is walking" in the utterance "The man that is talking on phone to the woman that is walking is my friend"), and they can be nested. So, another prerequisite for [48] is to syntactically parse the text.

3.2 Tools

3.2.1 Speech recognition

Since speech recognition is not one of the main concerns of this thesis, the choice of the tool to be used has been restricted to out-of-the-box, easy to use cloud solutions. In particular, the selection has been made among Google Cloud Speech [3], Watson Speech to Text [4], Bing Speech Recognition [5], Amazon Transcribe [6] and Wit.ai [7].
Note: the information below has been last updated in September 2018. Those tools evolve constantly and the comparison may no more represent well the current situation.
The characteristics used to evaluate the different solutions are: support for Italian language, possibility to perform speaker recognition, price and usage limits, possibility to perform NLP and parameters tuning possibilities. The results are shown in Table 3.1
From that table is possible to see that Watson speech to text and Amazon transcribe, although very powerful, cannot be used because they don't support Italian language. Among the others, Wit.ai is interesting because can provide NLP for the transcribed audio, especially complex intent analysis. In the other hand, it's speech recognition capabilities are quite basic and not tunable, because that's not the core business of Wit.ai.
Given the above considerations, the final choice has been made between Google cloud speech and Bing speech to text. The first has a longer free trial period and an higher starting credit. Apart these parameters, it's difficult to tell which of them is cheaper, because Google make the user pay depending on the total length of the submitted audio (each transaction is rounded up to a multiple of 15 seconds), whereas in the case of Bing the payment depend on the number of transactions. Considering the other characteristics, Bing allows to choose between three recognition modes, that are predefined configurations of the recognizer optimized to handle different types of speech. In particular the conversation mode improves the accuracy of the recognition when the submitted audio is a conversation between humans (i.e. with the following characteristics: open dictionary, disfluencies, etc.). An advantages of Google cloud speech, instead, is the possibility to suggest utterances (called "phrases hints" in the documentation) to the recognizer. These must

[3]https://cloud.google.com/speech/

[4]IBM, https://www.ibm.com/watson/services/speech-to-text/

[5]Microsoft Cognitive Services, https://azure.microsoft.com/en-us/services/cognitive-services/speech/

[6]https://aws.amazon.com/it/transcribe/

[7]https://wit.ai/

	Italian support	**Speaker recognition**	**Price and usage limits**	**NLP**	**Parameters tuning possibilities**
Google cloud speech	Yes	Beta for some languages (not Italian) or separate API	1yr free trial, after is free up to 60mins/mo. Starting credit	With separate API	Limited. The most interesting is the possibility to suggest "phrases". Words duration.
Watson speech to text	No	Integrated, only for a subset of the supported languages	Free up to a mins/mo limit, then different tariff based on the usage	With separate API	Large (included noise treshold and custom models). Words duration.
Bing speech to text	Yes	With separate API	1mo free trial, then free up to 5000 transactions per month. Starting credit.	With separate API	Recognition modes: interactive, conversation, dictation. Utterances duration. Beta version with customizable models.
Amazon transcribe	No	Integrated	60mins/mo free for 1yr, 0.0004USD/s if over 60mins or after trial period	With separate API	Custom models, words duration.
Wit.ai	Yes	No	Free and virtually unlimited usage	Some integrated capabilities	None for speech recognition (except language).

Table 3.1. Comparison of different speech recognition solutions as in September 2018. Google cloud speech is the selected one.

be specified in each request where they are needed and they can also be utterances that are not in the dictionary. When the recognizer has a doubt between a suggested utterance and another, if the confidence of the other utterance is below a given threshold, it will choose the suggested one. The recognizer can apply this

policy also to a part of a phrase hint. On the other hand, Bing speech to text provide, as a beta version feature, the possibility to define custom language and acustic models. These allow the user to emulate the Google words hints feature and to implement even more complex features, such as recognition of dialectal speech or particular noises.
An additional characteristic that has been taken into account is the length of the audio that can be submitted in a single transaction, because it has been noted that if an utterance is split in two (or more) requests, the words around the split(s) are often not correctly recognized. So it's important to minimize the intra-utterance splits. Google Cloud Speech allows to submit audio up to 1 minute per request in the case of synchronous or streaming transactions and up to 80 minutes for each asynchronous request (but audio longer than 1 minute must reside in a Google cloud storage bucket). Bing speech to text, instead, permits to send at most 15 seconds per request. Client libraries provided by Microsoft permit to overcome this limit, but it's not clear if they use another type of request or they simply split the audio in 15 seconds long chunks and then concatenate the obtained texts. Anyhow, there is not an official python library.
In conclusion, Google cloud speech as been chosen for this thesis, since it's cheap (and with a long trial period) and easy to use, and it provide a python library, the possibility to submit long audio tracks and to suggest utterances to the recognizer. Additionally, a set of beta features have been recently added (although some of them don't support Italian language) that make the Google solution even more preferable.

3.2.2 Natural language processing

NLTK vs spaCy

NLTK (Natural Language ToolKit) and spaCy are two of the most known and used python NLP libraries. Although they both implement the most of the basic NLP techniques, they are based on different philosophies and each of them provide some technique that is not offered by the other.
NLTK [27] born as a didactic tool providing several alternative algorithms for the main NLP tasks and support material and tools to help the students to evaluate and compare the different solutions and to extend the toolkit with their own algorithms.
SpaCy [28], instead, is intended to be used in production systems. So it provides an unique and highly optimized algorithm (in particular, the POS tagger is very close to state of the art performances) for each NLP technique.
The first important difference betweeen the two tools is that spaCy has a far better support for Italian language, providing models for tokenization, lemmatization,

POS tagging, morphology analysis, dependency parsing, named entities recognition and context vectors (but not word2vec or GloVe, although it's possible to add them). NLTK, instead, only provide language agnostic tokenizers and english support for all the other components. That said, NLTK has the advantage to provide interfaces for ontologies like WordNet and FrameNet (subsection 3.2.2), and in the case of WordNet also Italian is supported (by querying the MultiWordNet ontology).
Other characteristic os spaCy are the fact that the text is analyzed by a pipeline that contains all the necessary steps in the correct order (e.g. the POS tagging must be performed before dependency parsing) and the results are contained in a Doc object, that is a sequence of Token objects, each of them representing a word with the associated lemma, POS tag, morphology, vector, parent in the dependency tree etc.. From a Doc is also possible to obtain a list of sentences or noun chunks, that are represented by Span objects. The pipeline can be modified by adding or removing stages, and it's possible to (re)train one or all the stages. Doc, Span and Token objects can contain attributes defined by the user at runtime, with optional methods (e.g. getter and setter) and default values. In NLTK, instead, all the components are independent and it's up to the user to combine them in a logical way. Moreover, the results are almost always returned as strings.
For all the considerations above, in this thesis spaCy (version 2.0.12) will be used for all the low level analysis (tokenization, lemmatization, POS tagging, and parsing), while NLTK will be used to query Wordnet (i.e. its multilingual version) and Framenet.

WordNet vs FrameNet vs BabelNet

WordNet [36] is a lexical database and semantic network. When a user performs a query for a word, a set of synsets ("synonyms sets") is returned. Each synset represent one of the meaning that can be associated to that word, and allow to retrieve all the synonym lemmas. The query can be restricted to a single POS (e.g. it's possible to obtain the synsets related to "patient" considered as noun).
Each synset is identified by an ID in the form lemma.POS.number, where the lemma is the most significant lemma in the synset and number is used to disambiguate synsets with the same lemma and POS and represent the relative commonness (the lower the number, the more the synset is common with respect to the others with the same lemma and POS). Synsets are linked to each other by means of different types of relation that represent most of the relationships between words described in subsection 3.1.3. Another interesting feature of WordNet is the possibility to obtain information on the valence and types of parameters of verbs (through the so called verb frames). There is also a multilingual version of WordNet, that support also Italian language, although with some limitations.
FrameNet [50] is another lexical database, but it's based on frame semantics. This

means that the meanings of a word are described by the set of frames that are "evoked" by that word.
Each frame represent a concept (e.g. a situation, an action, a relation, an entity, etc.) and contains a set of Frame Elements (FEs) that define that concept (e.g. the frame "Revenge" contains the FEs: Offender, Injury, Injured_Party, Avenger and Punishment). Words that evoke that frame are called Lexical Units (LUs) and are concrete instances of some of the FEs. Frames are linked to each other by means of inheritance, subframe and usage relationships. Moreover, some LUs, frames and FEs are annotated with semantic types (e.g. positive or negative, that can be useful for sentiment analysis, artifact or natural etc.).
Compared to WordNet, FrameNet has a smaller vocabulary but allows to explore different relationships between the words, and gives more syntagmatic information than the very simple verb frames provided by WordNet. [51] is an example on how to exploit both ontologies for text understanding. Unluckily, FrameNet currently doesn't support italian language.
BabelNet [52] is similar to WordNet (it's based on synsets connected to each other), but with some important difference. First of all, it has been created by automatically integrating information coming from many different sources, including WordNet, FrameNet, Wikipedia, ImageNet etc.. This allowed to create very large semantic networks for most of the existing languages (for some of them, only a subset of the sources was available), and make possible to use BabelNet also for translation. Moreover, the usage of wikipedia allowed to insert additional links between synsets that are related to each other in a way that is different from those considered by WordNet, and the integration of ImageNet make possible to associate images to synsets.
The richness of BabelNet in terms of both synsets and relationships can be an advantage or a disadvantage depending on the application (e.g. if you search "cane", that is "dog" in Italian, one of the more relevant results is a Finnish guitarist). For sure the size of the network (about 29GB) make difficult to use it in local and usually require to send the query to the available web service (that imposes a limit of 1000 queries per day). Another disadvantage is that there are not python APIs for now, while WordNet and FrameNet can be queried by using, for example, NLTK, that also offers some additional utilities (e.g. to compute the similarity between words by considering the distance between the corresponding synsets in WordNet).

Other tools

This section describes other tools that are complementary or alternative to the previous ones.
Since one of the features found in literature is the number of syllables per minutes, and since the syllabification task is not performed by spaCy and NLTK, it's necessary to find a tool for this. There is a syllabifier specific for Italian language named

sylli [53], but unfortunately it's not compatible with python 3. The alternatives are in general hyphenetors. Hyphenation points are the places where a string can be divided with a new line, and are a subset of the syllables separation points. That said, by using some shrewdness (in particular by applying the hyphenator to one word at a time, instead than to the entire text), it's possible to approximate very well the results that would have been returned by a syllabifier. The two most famous python modules suited for that task are pyphen (`http://pyphen.org/`) an PyHyphen (`https://pypi.org/project/PyHyphen/`). Both uses the same dictionaries that are used for hyphenation in programs like LibreOffice and Mozilla Firefox. The main difference is that pyphen is a pure python library, while PyHyphen is an interface to a C library.
Other useful python libraries are Gensim ([54]) and scikit-learn ([55]). The first allows to compute the similarity between words or, more in general, texts by training and using a variety of algorithms, included those for word embedding described in subsection 3.1.3. The second implements several algorithms for data preprocessing and machine learning.
Another tool that is worth to mention is Weka ([56]), an open-source machine learning workbench that allows to design, train, test and use machine learning pipelines of any complexity by means of intuitive graphical interfaces. There are also a lot of plugins available, like for example [57], a module for enabling autoML, that is the automatic selection of the machine learning algorithms and tuning of their hyperparameters.

Chapter 4

Methods

4.1 Dataset

Three datasets have been used for this thesis, two for the Italian language and one for the English one.
The first Italian corpus has been provided by the speech therapy department of the University of Turin (UniTo). The collection have been done by recording 15 healthy subjects and 15 with an initial cognitive decline (i.e. they have been admitted at the Alzheimer evaluation unit of the Martini hospital). For each person, two recordings have been performed: one for the description of a complex picture (fig. 4.1) taken from the ELLM test ([58]) and one for a spontaneous speech test.
The second Italian dataset has been provided by the university of Trento (UniTN). It contains recordings from 8 control and 8 pathological subjects. For each of them there is at least a recording of the cookie theft picture (fig. 4.2) description and one for the description of a tipical day of the subject. For some subject there are also other recordings (e.g. what he/she do when taking a shower, or a recall test using the little red riding hood story).
The English dataset is a subset of the Pitt corpus ([59]) stored on TalkBank[1]. In particular, for both the control and the pathological groups, only the cookie theft picture audio recording whose names ends with 0 have been selected. In this way it's assured that there is only one recording per subject, and it is the earliest one (which is important for the pathological group). This is a suboptimal selection, because in the pathological group there are subjects having different types of dementia at different stages, and in the control group the best choice should have been the one including the best audio quality recording for each subject. However, the access to this corpus has been obtained only in late January 2019, and the selection has to

[1]`https://talkbank.org/`

been done manually, so this is the best that has been possible to do. 70 control and 170 pathological recordings have been extracted.

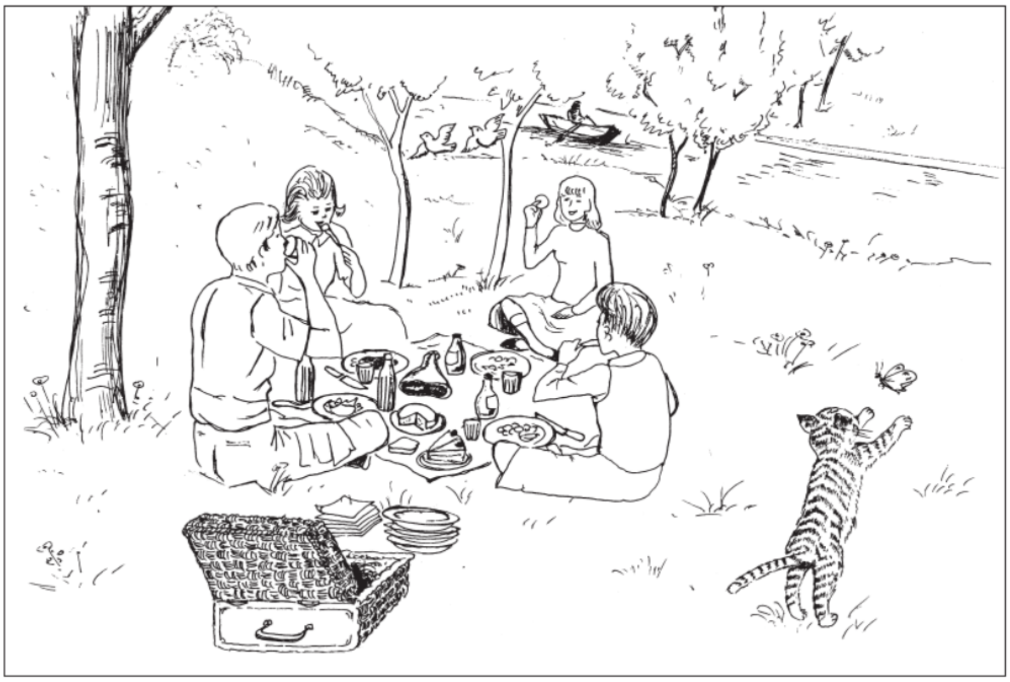

Figure 4.1. Picture to be described during the ELLM test.

4.2 Speech recognition

As written in subsection 3.2.1, Google Cloud Speech has been chosen as speech recognition tool.
There are three ways to use it: in synchronous mode, it's possible to transcribe at most 1 minute of audio (that can be sent from the local machine or stored on Google Cloud Storage) per request and the thread is blocked until it receives the response; in asynchronous mode, it's possible to transcribe audio tracks that are long up to 180 minute each (but those longer than 1 minute must be stored in Google Cloud Storage), and the request method immediately returns an handle that can be used to retrieve the result when it's ready; in streaming mode, the duration of a single audio track is again up to 1 minute (it can only be sent from local machine, at a rate as similar as possible to real time), and it's possible to receive so-called interim results as soon as the recognizer is able to update the transcription relying on the new data obtained from the client.

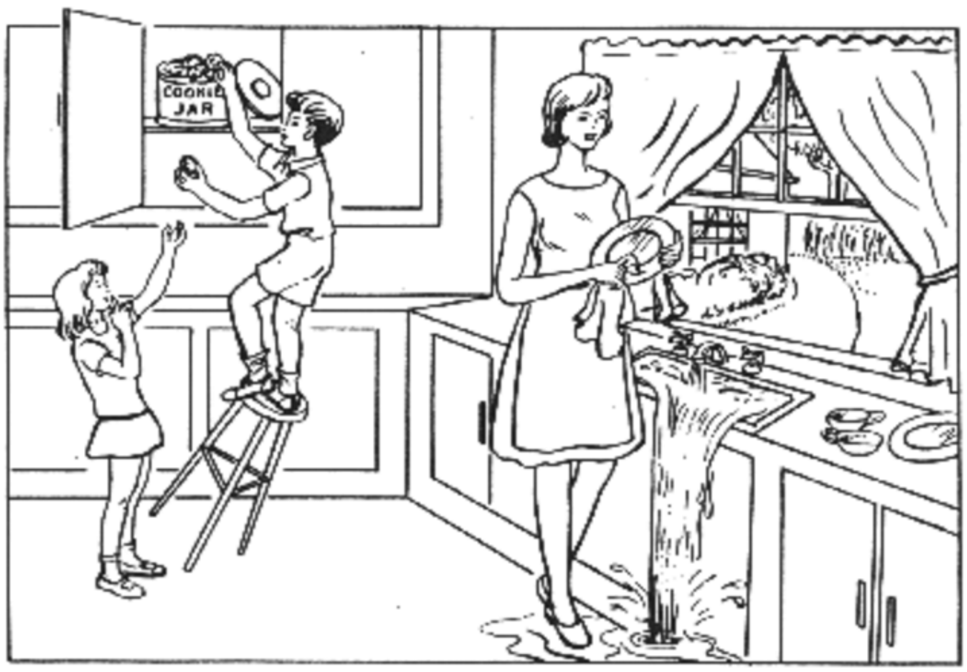

Figure 4.2. Cookie theft picture.

All the three methods described above support the same set of configuration option (only streaming recognition can have additional options) that includes the audio encoding[2] and sample rate (that can be omitted if declared in the file header), the language, hints to improve the recognition of specific strings, the maximum number of alternative transcripts that can be returned, the indication of the duration of each word of the first alternative, the activation of punctuation recovery (beta version), etc..
Since the dataset is composed by a set of audio files with a length in general greater than 1 minute, the more appropriate mode is the asynchronous one. The main alternative would have been the synchronous mode, that has the advantage to not require to upload the file to Google Cloud Storage, but make necessary to split the audio track in segments shorter than 1 minute. This can worsen the recognition for two reasons. First of all, the break can happen in the middle of a word (but this can be avoided in most cases by choosing a point where the volume is below a given threshold). Furthermore, the recognizer try to guess each word by looking also at its context, so it's better to avoid to split the audio in random positions.
Rows from 190 to 210 of the code in appendix A show how the Google Cloud

[2]For details, see `https://cloud.google.com/speech-to-text/docs/encoding`

Speech service is consumed to transcribe an audio file already loaded on Google Cloud Storage and how the text is extracted from the response, concatenated and entered in the spaCy pipeline. That piece of code also load the spaCy models for the Italian language and a word2vec trained model, and add a special case to the spaCy tokenizer to correctly recognize the word "po'".

4.2.1 Recognition accuracy

Speech recognition accuracy as been computed in terms of Word Error Rate (WER, (4.1)), that is the minimum sum of the number of words to be inserted, deleted and modified to change a string into one that is assumed as reference, divided by the number of words in the reference.

$$WER = min(\frac{I + D + M}{N}) \tag{4.1}$$

For this purpose, 9 automatic transcripts per class and the corresponding manual transcripts from the University of Turin dataset have been used. WER has been computed by using the python jiwer package[3], that first remove punctuation and text between square brackets and other delimiters, then replace each distinct word with a number and finally uses an approximated algorithm to compute the Levenshtein distance between the two transformed strings (WER is in fact a Levenshtein distance that uses words instead of characters). Additionally, the correlation between WER and the average word confidence returned by Google Cloud Speech has been computed. This has been done to check if that confidence is a reliable measure of the goodness of a transcript.
Table 4.1 reports the correlation for all the 18 samples and for those of each class. Figure 4.3, instead, plot the distribution of the 18 samples in the WER/confidence space.

Correlation	0.7
Correlation for healthy class	0.8
Correlation for AD class	0.6

Table 4.1. Correlation between WER and confidence.

[3]`https://github.com/jitsi/asr-wer/`

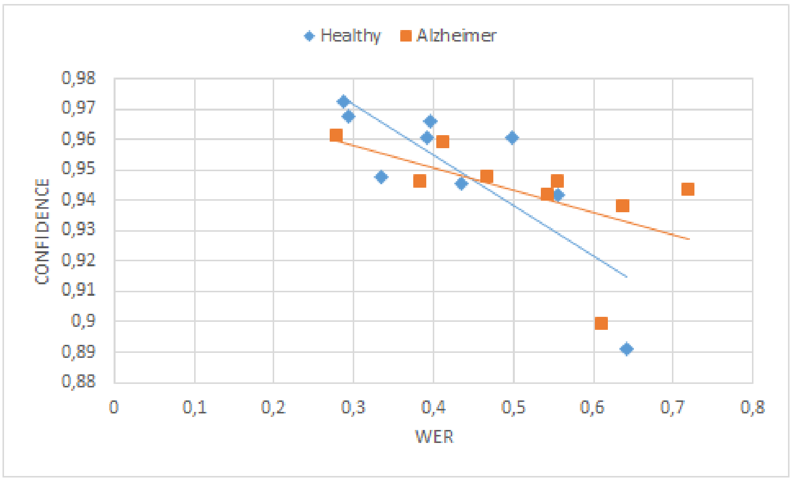

Figure 4.3. WER vs confidence scatter plot, with tendency lines.

4.3 The problem of restoring the punctuation

4.3.1 Some approaches available in literature

Punctuation is very important for the correctness of the results returned by NLP techniques. In fact, both POS tagging and syntax parsing exploit punctuation, and most of the advanced analysis methods are based on these two techniques.
Google Cloud Speech can try, if opportunely configured, to restore punctuation. Unfortunately, that's a beta feature only available for en-US language (although for some strange reason, sometimes in transcripts there is anyway some full stop).
Also spaCy try to restore punctuation, by adding appropriate transitions to its dependency parsing algorithm ([60][61]). This approach give in general poor results when dealing with completely unpunctuated text, at least for the Italian language, and especially when there are also disfluencies.
Another possible approach, described for example in [62] and [29], is to perform sequence to sequence transformation from an unpunctuated text to a punctuated one. In particular, the second paper proposes an encoder-decoder architecture where the encoder is a bi-directional Recurrent Neural Network (RNN) and an attention mechanism is used to make the decoder focus on the relevant context.
The problem of RNNs is that they have substantially a sequential structure, thus they are inefficient when processing long sequences. A more modern approach is to use Convolutional Neural Networks (CNN). The first CNN that was able to become the new state of the art in a sequence to sequence task (i.e. machine translation) is described in [63]. The CNN used for NLP tasks are usually 1-dimensional. A 2D CNN is however used in [64] with success. In that case, the network is very similar

to those used for image recognition, and each "pixel" of the "image" is the embedding of a couple of words, one from the input and one from the output. A mask is applied so that the CNN doesn't try to operate on the "pixels" corresponding to output words that have still to be generated.

4.3.2 Proposed approach

A novel approach proposed in this thesis consists in using the CNN described by [64] to perform joint POS tagging and punctuation restoration. In this way it's possible that the accuracy in both tasks increase with respect to the case where POS tagging is performed before punctuation restoration (so the tagger cannot exploit the information related to the punctuation) or vice versa (with analogous problems).
There are three operations to perform in order to adapt the open-source implementation of the aforementioned CNN available on github[4] to the new task:

- Define a good embedding for POS tags and punctuation insertion operations;
- Collect a big enough dataset and preprocess it;
- Tune the hyperparameters of the network;

The first two points are quite easy. For the embedding, the one already available should be fine. Each token is simply replaced by its position in a vocabulary built at the beginning of the training phase. Concerning the dataset, it must contain text annotated with POS tags. So it's sufficient to take a large treebank (i.e. a corpus annotated with dependency parsing information) and remove the superfluous annotations. A possible suitable treebank is the Paisà corpus ([65]). Once this is done, it's possible to move or swap some of the words, together with their POS tags, in order to simulate disfluent language, to replace punctuation with appropriate "add comma/full stop/question mark/ecc." tags and to split or merge sentences to form what will be called "spans" from now on. Finally, it's necessary to split the data in two separated corpora, aligned at the span level. Now it's sufficient to run the preprocess.py script with suitable parameters to generate all the needed train, validation and test sets in the right format.
The tuning of the hyperparameters must be done before training. Since both the output vocabulary and the alignment between the input and output sequences are simpler than in the case of machine translation (e.g. for the alignment, each pos

[4]`https://github.com/elbayadm/attn2d`

tag is associated to exactly one word, and the alignment is monotonic), the size of the network can maybe be reduced a little.
Once the hyperparameters have been defined and the network configuration has been saved in yaml format in the config folder, it's possible to train and test the model with slightly modified versions of the train.py and generate.py scripts.
The smallest pre-tuned model available on github requires a GPU with at least 16GB of memory to be trained. Considering that the target vocabulary is very small and the task described here is simpler that machine translation (for example, the alignment is monotonic and the "translation" is only 1-to-1 or 1-to-many), it's maybe possible to use a smaller model, but still it would probably not fit in a normal personal computer GPU memory.

4.4 Data augmentation and features extraction

The available dataset is too small to train a classifier. To augment it, it has been chosen to use a sliding window over the text to obtain a large number of samples (i.e. one for each position of the window). The window move by one word at a time, and has a random length (between 100 and 200 words) that vary for each execution of the script but is always the same for the duration of the sliding window loop. A different length for each subject has been chosen in order to simulate what happens in reality and possibly reduce the significance for classification of features that are affected by text length. To avoid to increase the error rates of the POS tagger and the parser, the sliding window is applied after them.
Rows 219 - 272 of the code in appendix A show how the duration and the average confidence of each window are computed, the initialization of the lists that will hold the results and the beginning of the loop that will contain all the features extraction operations. In the following, this cycle will be called "window loop". This method has the advantage to be very easy and to return samples corresponding to parts of what the subject has really said, so at least the basic features should be quite consistent. On the other hand, it make difficult or even impossible to compute those metrics that are related to the structure of the discourse. Another disadvantage is the fact that the samples related to the same speaker are not independent from each other, so having, for example, one thousand samples from 30 subjects is not the same than having them from one thousand speakers. Although, it's always better than having only 30 samples.

4.4.1 Prosodic features

The only prosodic feature extracted is syllables per minute. Pauses are included in the computation, since Google Cloud Speech doesn't allow to isolate them (the

duration of each word include the adjacent pauses, see section 4.2).
Rows from 326 to 333 in appendix A use the pyphen library to syllabify the words of the window (one at a time to make the results of pyphen, that is an hyphenator, more consistent with those of a true syllabifier) and exploit the window duration computed outside the window loop to compute the syllables per minute.

4.4.2 Parts-of-speech and lexical features

The POS rates for adjectives, nouns, pronouns and verbs are computed both in the traditional way and by computing the moving average over a window of 30 words. Additionally, a noun rate that ignore very common nouns is computed, in both ways.
The function `pos_rate`, declared at row 53 of appendix A, takes as input a sequence of spaCy tokens (a doc or a span) and returns the POS rates for it. In the window loop, from line 273 to line 298, that function is called to extract the features described above.
Concerning lexical richness, the only features computed are TTR and MATTR, that are extracted by the code at rows from 300 to 317 in appendix A. A set is exploited to compute the number of distinct words.

4.4.3 Semantic and pragmatic features

Extracting significant semantic and pragmatic features in the case of this thesis is in general very difficult. This is due both to the fact that most of them rely on accurate syntax parsing, but this is not possible in this case (see section 4.3 for details). So, the metrics computed in this thesis are only repetitions rate, rate of pronominal anaphora that have an antecedent in the previous N words and two indexes for internal coherence derived from [45].
Concerning repetitions, the function called `repetitions_rate` declared at row 74 af the code in appendix A computes the rate of repetitions of `rep_len` words separed by `gap_len` words (e.g.: "quel giorno ehm quel giorno" is a repetition of the two words "quel" and "giorno" separed by the filler word "ehm"). That function is called at lines from 319 to 324 to calculate the repetitions of one or two words separated by zero, one or two other words.
The rate of correct anaphora is computed by means of an adapted and simplified pronominal anaphora resolution function, called `pronominal_anaphora_resolution_rate` and declared at row 85 of appendix A. Since there are no clues about verbal turns and speakers' characteristics, it's not possible to make considerations about the correct use of first and second person pronoun, so only third person ones are considered.
Lines from 94 to 100 try to find clitic pronouns postfixed to verbs, by exploiting a

suitable dictionary containing these pronouns, declared at row 21. These pronouns are correctly recognized by spaCy only when they are not suffixed to a verb, so this dictionary is necessary in order to know which suffixed are pronouns. This method is not perfect because it only considers the end of the word, but sometimes there are more than one third person clitic pronouns suffixed to a verb, and usually in these cases there are also other intricacies (e.g. "prendiglielo", can mean "take it to/from him/her/them", and there is an "e" between "gli" and "lo"). Anyhow, it works well in most cases.
Another interesting piece of code is the one from line 90 to line 93. This if statement is responsible to find pronouns (except suffixed clitics). In doing this, it tries also to find qualificative and numeral pronouns. Unfortunately, this kind of pronoun is not recognized by spaCy, and actually it's often not even detected by this piece of code.
When a pronoun is detected, lines from 101 to 111 try to find an antecedent for the pronoun. This is done in a very simplistic way, by looking if there is a noun with compatible gender and number (considering also the cases in which the pronoun is masculine and the antecedent is neutral and when the pronoun has an indefinite number)in the text preceding the pronoun.
The previous steps are repeated for each word. Then, the rate between the number of pronouns with an antecedent and the total number of pronouns is returned. If there are not pronouns, 1 is returned. This is more consistent than returning 0 or a negative number.
Lastly, the internal coherence is evaluated by using two different metrics named "ontoscore" to stress their similitude with the technique defined in [45]. The first is named `w2v_ontoscore` (declared at row 116 of appendix A) because it uses word2vec to compute the similarity between words, whereas the second one is named `wordnet_ontoscore` (declared at row 131). The main difference between the two is that in the first the similarity is calculated between the words as they appear in the text and without taking into account that they may be polysemic, whereas the one based on wordnet makes the evaluation basing on the synsets associated (by using the code at lines 212 - 217) with the lemmas of the words.
Both functions return, additionally to the result, the rate of "covered words", that are the words present in the ontology and not in the stop words list of spaCy. `w2v_ontoscore` actually return two coherence measures, one based on the cosine similarity between the normalized vectors corresponding to the words and the other based on the euclidean distance between the original vectors.

4.4.4 Co-occurrences graph features

These features are those listed in [23]. The co-occurrences graph is built (after having removed the stop words and normalized to lowercase the others) by using a

function provided by textacy ([66]), that is a library that exploit spaCy to provide higher level functions and utilities. The graph is then enriched by connecting each pair of words that have a similarity score (i.e. cosine similarity) higher than 0.5 in the word2vec model already mentioned. This is done, as suggested in [23], because otherwise a co-occurrences graph built on a short text is often almost linear (it's not very probable that a word appear in multiple contexts), and that would make the topological features extracted from it not significant. All these operations are performed at rows from 374 to 385 of the code in appendix A.
The code from line 386 to line 401 computes the set of topological features listed in [23] and add the obtained values to the lists defined outside the sliding window loop. Notice that some function returns a list of values, one for each node. In these cases, the average is calculated.

4.5 Data preprocessing

4.5.1 Outliers

Outliers are samples that are significantly different from the others belonging to the same class. Their influence on the prediction accuracy of the classifier depends on their number, the model used and what they represent. Sometimes, outliers are even more useful than the other points. The "difference" named before can be evaluated in different ways. In this thesis the average distance from the k nearest neighbors is used.
The first step is to plot the samples sorted by the 4 nearest neighbors distance versus the distance. Hopefully, the plotted points will trace a sort of curve with one or more knees. A knee separates the points that are inside a denser cluster from those that are more widespread. So it's possible to visually determine the percentage of outliers.
In the case the percentage of outliers or their value may significantly affect the model performances, the second step is to run the DBSCAN algorithm([67]) by using as epsilon value the distance corresponding to the knee identified before. The outliers will be included in a fake cluster (with id = -1 in most implementations). Otherwise, it's sufficient that the features standardization is performed in a way that is not too sensitive to outliers (see subsection 4.5.2).

4.5.2 Features selection and transformation

Both features selection and features transformation can be used to reduce the dimensionality of the data. This is useful to speed up model training and reduce the

risk of overfitting (on the other side, too few features reduce the accuracy). Concerning the last point, [68] find some rule of thumb to estimate a good number of features for some model, depending on the number of instances and the correlation between features. At the same time, that paper states that those rules are valid only in some case, and also find that sometimes the optimal number of features doesn't increase monotonically with the number of samples.
Features selection reduce the dimensionality of the data by removing the features that are expected to be less relevant for the classification task. An exhaustive search of the optimal subset is an NP-complete problem with complexity proportional to the cardinality of the power set of the original features set. So it's usually necessary to use some heuristics, that can be based on the intrinsic characteristics of the considered subset (filter methods) or use a model for the evaluation (wrapper methods). Filter methods are faster, but wrapper ones may give better results.
Features transformation is based on a different approach. In this case, the original features are transformed and combined so that data is mapped on a lower dimensional space. An example of this approach is the Principal Components Analysis (PDA), which linearly combines the original features so to project the data in a space that maximize its variance.

Correlated features

In general, a good features set should include those that are strongly correlated with the class but little correlated between each other. The first part of the previous statement is always true, whereas the second one is not. In fact, two features that are strongly correlated between each other should be both kept if the correlation add some information useful for the classification. Instead, if the correlation between the features doesn't add any information (e.g. there is a mathematical relationship between them), one of them should be discarded.
The computation of the correlation with a categorical class is in general problematic, because the Pearson correlation value changes depending on the numeric value assigned to each class. Luckily, when there are two classes only the sign changes, so the absolute value of the correlation can still be useful. However, it's often better to use Mutual Information (MI) instead of correlation. MI basically indicates how much the knowledge of a variable increases the knowledge of another one, and it's expressed as in equation (4.2).

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{4.2}$$

Features standardization

Features standardization (sometimes called normalization) transforms the original features values by scaling and translating them so to obtain comparable values for all the features. This step is crucial when dealing with classifiers based on distances, like the k-NN one, but it's useful also for other models. For example, a standardized data space tends to speed up the convergence of neural networks. However, also this operation is not always useful, because it's possible that the scale or offset difference between features it's itself an information useful for the classifier.
The normalization should always be performed first on the training set, then on the test set by using the parameters fitted during training. This is because the test set should represent the data that will be used for prediction once the model is deployed, and that data is not known neither at training nor at testing time, so the most consistent option is to standardize both the testing and any future data by using the parameters used for the training set.
There are different ways to standardize features. Typical approaches consist in subtracting the mean and divide by the standard deviation, the difference between the third and the first quartiles or the difference between the maximum and the minimum values. Using standard deviation or quartiles for the division tends to make the normalization less sensitive to outliers.

4.6 Hyperparameters tuning and model training and evaluation

Hyperparameters tuning, training and testing are performed with a nested cross validation (CV). The inner CV is used to select the features and optimize the hyperparameters of the classifier so to maximize the accuracy on a validation set, whereas the outer one test the optimized model (retrained on train + validation sets) on an independent test set. The partitioning of the data in training, validation and test sets is performed so that the instances belonging to the same patient are all in the same set. This assures the independence between the three sets.

4.6.1 First approach: using the UniTo spontaneous speech corpus

The first attempt to train a model was made by using only the spontaneous speech part of the UniTo corpus and augmenting it as described in section 4.4. The image description part was not used because the transcripts in this case are much shorter, thus the data augmentation procedure would have produced a negligible

number of instances compared to the other part of the corpus. Additionally, those few instances would have probably been quite different from those obtained from spontaneous speeches, thus probably worsening the accuracy of the model.
Additionally to features and class, each instance is associated to a "sample" field that contains an anonymized identifier of the subject. Before any other operation, the instances are shuffled and features, class and sample fields are stored in three different arrays.
The nested CV is performed by using as a model a pipeline composed by three steps: standardization, features selection and classifier. In particular, the standardization is performed by zeroing the average and scaling based on the interquartile range, the feature selection is done by taking the N (where N is an hyperparameter to be tuned) features with greater mutual information with the class, and the classifier is a multilayer perceptron. The inner CV performs a random search for the optimal hyperparameters over an N-dimensional distribution that vary depending on the considered classifier. Anyway, one of the hyperparameters is always the number of features to select. The optimization is performed for each inner fold, then the best model among all folds is retrained on the training + validation sets and is passed to the outer CV, where it's evaluated on a test set.
Unfortunately, the model trained this way is very unstable, i.e. the optimal hyperparameters and the accuracy vary a lot depending on how the dataset is split. This is due to the fact that the instances belong to only 22 people, so they don't represent the entire population, and the samples in the training set may be very different from those present in the validation or test sets. Use also the other Italian corpora would not be useful, because they are all very small, and while the number of samples increases, also the size of the population increases.

4.6.2 Second approach: transfer learning from a model trained on the English corpus

To solve the problem of the too small size of the Italian corpora, the idea was to try training the model on a large English corpus and then use part of the Italian ones just to tune it for the real target language. This method is called transfer learning (because the knowledge encoded in a model trained for a task it transfered to another task) and today is usually used in deep learning, but can actually be exploited in any case where there are few data for the target task but many for a similar one. The efficacy depends on how much they are similar.
The first step was to try to tune, train and test a model only on the Pitt corpus, to see if the results are consistent with those found in literature. A nested CV on the same pipeline used previously already gives an accuracy above 80&, as can be seen in section 5.2. In that section it can be noticed that the score is further improved by substituting the neural network with an ensemble. Furthermore, the variance is

smaller than in the first approach.
Once assessed that the work described in this thesis gives good results on a large enough corpus, the next step was to implement a naive form of transfer learning. First of all, subset of features that have an high mutual information with the target variable in both the English and the Italian datasets was manually selected. Then, a pipeline (without the features selection stage) was tuned and trained on the English corpus. Finally, the pipeline was retrained in "warm start" mode, that is, by using the neural network weights fitted previously as starting point. This very simplistic implementation gives results comparable with those obtained with the first approach (see section 5.2). However, it also suggests that transfer learning may make sense for this kind of tasks, and a more sophisticate version of it might actually improve accuracy.

4.6.3 Evaluation metrics

The metrics described in this section are defined in term of number of True Positive (TP, i.e. positive instances that are predicted as positive), True Negative (TN, i.e. negative instances that are predicted as negative), False Positive (FP, i.e. negative instances that are predicted as positive) and False Negative (FN, i.e. positive instances that are predicted as negative). These numbers can be effectively represented in a so-called confusion matrix (fig. 4.4). Scikit learn default score

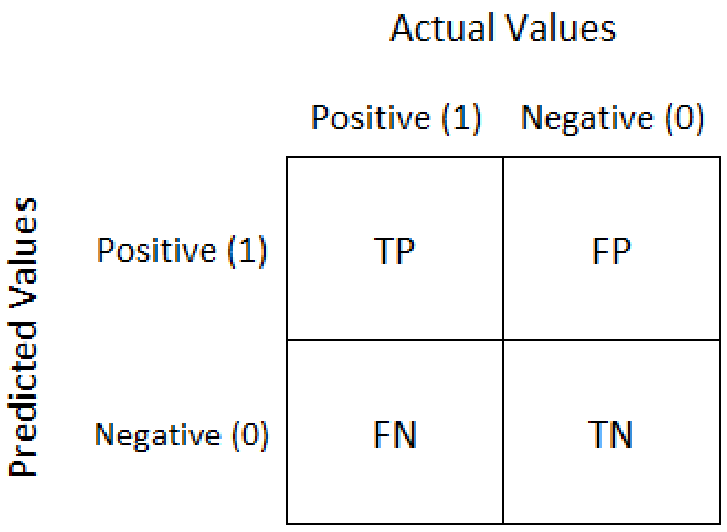

Figure 4.4. Confusion matrix for binary classification.

metric for classification is accuracy, that is the number of correctly predicted instances divided by the total number of instances (4.3). The advantage of accuracy is that it describes the overall performances of the model with a single number. The disadvantage is that it doesn't make evident an eventual unbalancing among the classes.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

More adequate metrics to use in case of a biased classifier are precision (4.4), recall (4.5) and F1 score (4.6). For the positive class, precision is the fraction of instances predicted as positive being actually positive. Recall is the fraction of actually positive instances being predicted as positive. The F1 score is the harmonic mean of precision and recall. Analogously for the negative class. These metrics are computed for each class, so a bias of the model can easily be detected.

$$precision_T = \frac{TP}{TP + FP},\ precision_N = \frac{TN}{TN + FN} \tag{4.4}$$

$$recall_P = \frac{TP}{TP + FN},\ recall_N = \frac{TN}{TN + FP} \tag{4.5}$$

$$F1_P = 2 \cdot \frac{precision_P \cdot recall_P}{precision_P + recall_P},\ F1_N = 2 \cdot \frac{precision_N \cdot recall_N}{precision_N + recall_N} \tag{4.6}$$

Chapter 5

Results

5.1 First approach

The model trained only on UniTo samples (spontaneous speech and ELLM picture description for 12 physiological and 12 pathological subjects) has a very variable accuracy. fig. 5.1 plots the accuracy for 50 runs of a 5x5 nested cross-validation (so there are 50x5 accuracies). Data has been shuffled before each run. The histogram shows accuracies varying from about 0.55 to around 0.75, with two peaks distant 0.15 from each other. The mean is 0.65, with a standard deviation of 0.061.

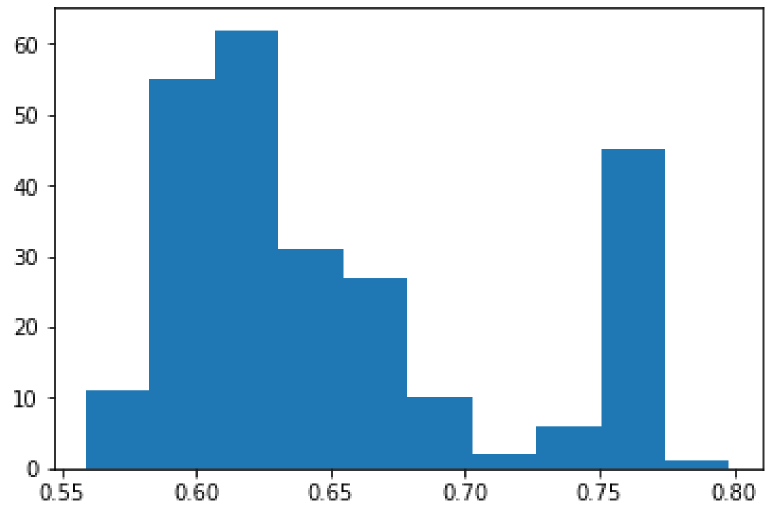

Figure 5.1. First approach accuracy for 50 nested cross validations.

5.2 Second approach

The first step of the second approach was to tune, train and test a neural network only on the selected subset of the Pitt corpus (cookie theft picture description for 70 physiological and 170 pathological subjects). The tuning was performed only for the number of features, the number of neurons of each hidden layer (a multilayer perceptron with two hidden layer was used), the initial value of the learning rate and alpha, a regularization parameter. fig. 5.2 show the distribution of the accuracies obtained with 200 runs of a 5x5 nested CV. The mean is 0.84, with a standard deviation of 0.016. This means that the variance is one order of magnitude smaller than in the previous approach (about 0.0003 versus 0.004).

After having checked that the system works well with the Pitt corpus, the second

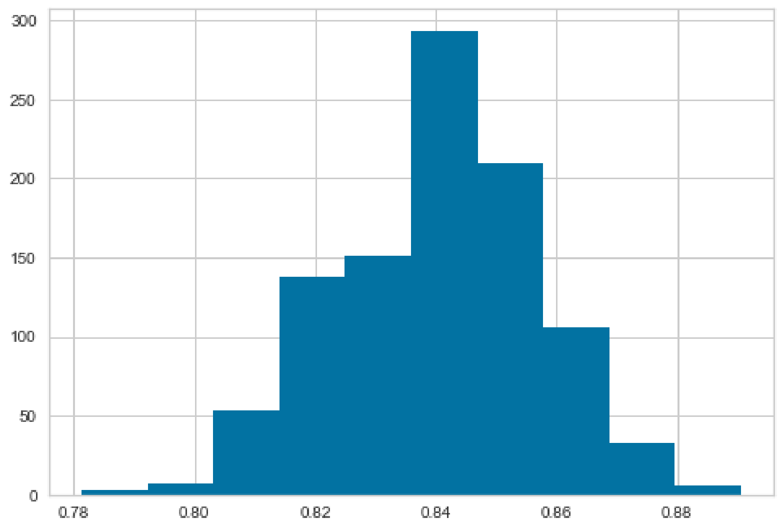

Figure 5.2. Accuracy distribution for multilayer perceptron on the Pitt corpus.

step was to train the model on the entire English corpus, then further train it on part of the UniTo dataset and test it on the remaining part. In this case, 4 features have been manually selected among those that have an high mutual information score with respect to the class in both the English (fig. 5.3) and the Italian (fig. 5.4) datasets. The chosen features are Pronoun Rate (PR), Moving Average Type-Token Ratio (MATTR), coverage (i.e. the percentage of words that can be found in the used word2vec model) and syllables per minute. Although MATTR has an opposite behavior in the two datasets (In the English one it's greater for physiological individuals than for pathological ones, as shown in fig. 5.5, whereas in the Italian one is the opposite, as shown in fig. 5.6), including it actually improves significantly both accuracy mean and standard deviation.

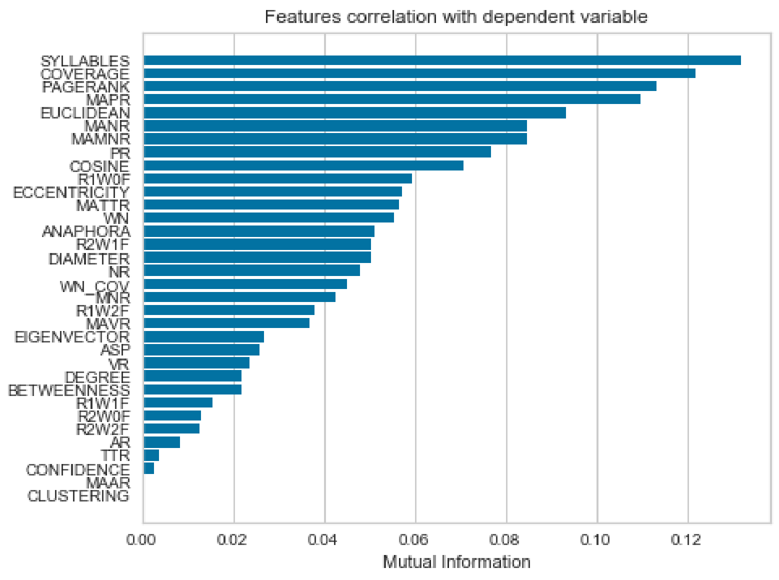

Figure 5.3. Mutual information between features and target variable for the Pitt corpus.

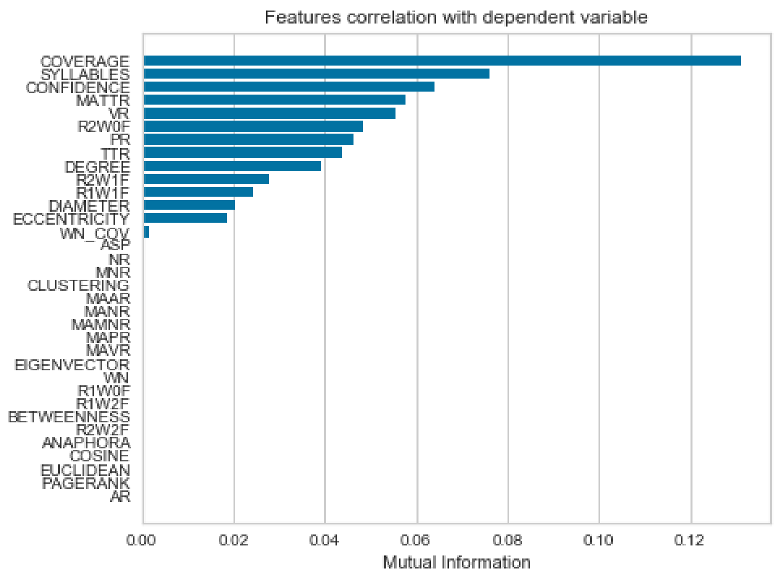

Figure 5.4. Mutual information between features and target variable for the UniTo corpus.

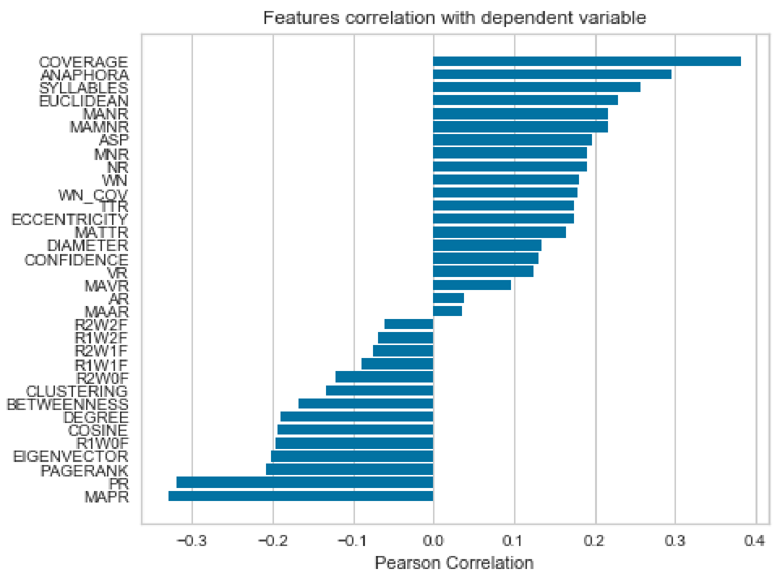

Figure 5.5. Correlation between features and class for the Pitt corpus. Positive values are greater in healthy instances than in AD ones and viceversa.

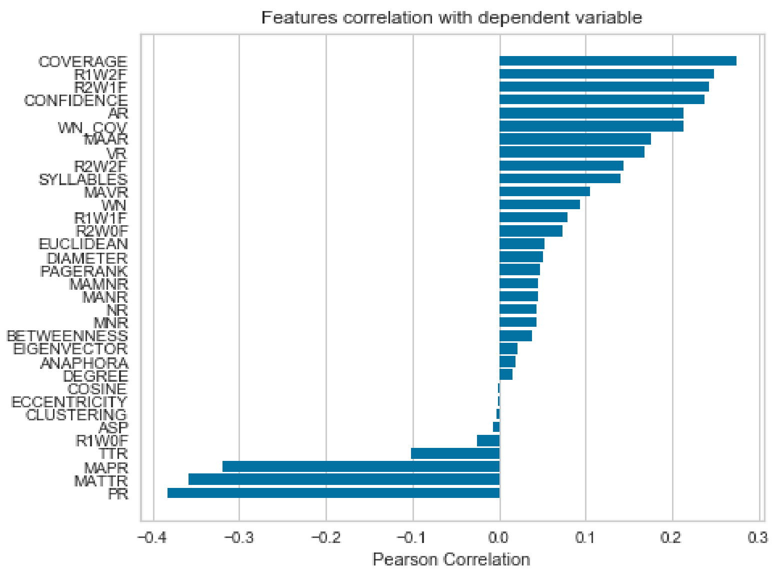

Figure 5.6. Correlation between features and class for the UniTo corpus. Positive values are greater in healthy instances than in AD ones and viceversa.

Before training the model on the entire Pitt corpus, 100 runs of a 5x5 nested CV were performed, with a pipeline without the features selection step, in order to find the optimal hyperparameters. The accuracy (fig. 5.7) has a mean of 0.75 and standard deviation of 0.018, whereas figs. 5.8 to 5.11 show the distribution of the optimal number of neurons for each hidden layer, the optimal alpha in the 0 - 8 and 0 - 1 ranges and the optimal initial value for the learning rate. It's interesting to notice that the value of the last one seems to be not relevant. This is confirmed by figs. 5.12 to 5.14, that show a roughly uniform distribution of the initial learning rate (y axis) in correspondence of the optimal values of the other two hyperparameters and of the different values for accuracy.

At this point, a multilayer perceptron with alpha equal to 0.5 and 90 neurons per hidden layer is trained on the entire Pitt dataset. Then, the same network is further trained 1000 times on 15 random samples of the Unito corpus and tested on the other 9. As shown in fig. 5.15, both mean (0.57) and standard deviation (0.086) are worse than in the old approach.

5.3 Discussion

The implemented pipeline (speech recognition, features extraction and classification) has performances comparable to the state of the art on the Pitt corpus. This demonstrates the goodness of the approach. Moreover, there is room for improvements in several areas, as listed in section 6.1.
However, on the UniTo corpus the performances worsen a lot, both for what concerns the accuracy mean value and the accuracy standard deviation. This is probably due to the small size of the dataset, that cause the model to fit different subset of the complete population depending on which data falls in the training set and which in the test set. Other possible causes of the poor results may be the fact that the subjects in the UniTo dataset have been chosen so to be very close to the boundary between healthy and AD, and that the presence of both image description and spontaneous speech recordings in the corpus make it more heterogeneous than the Pitt one, so it's more difficult to separate the two classes.
Transfer learning, that was introduced to try to solve the problems with the Italian corpus, seems to be not useful. However, the implementation used in this thesis was very naive. In particular, the fact that the pre-trained model is fitted on the second dataset without early stopping strategies may tend to cancel the contribute given by the first training step. However, it's difficult to perform early stopping with a dataset so small, because it requires to keep apart a small (but not too much) dataset that will not be included in neither the training nor the test set.

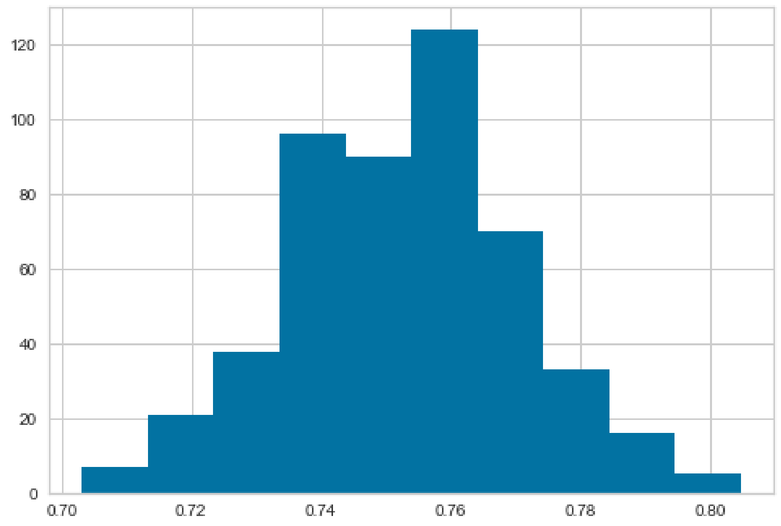

Figure 5.7. Accuracy distribution.

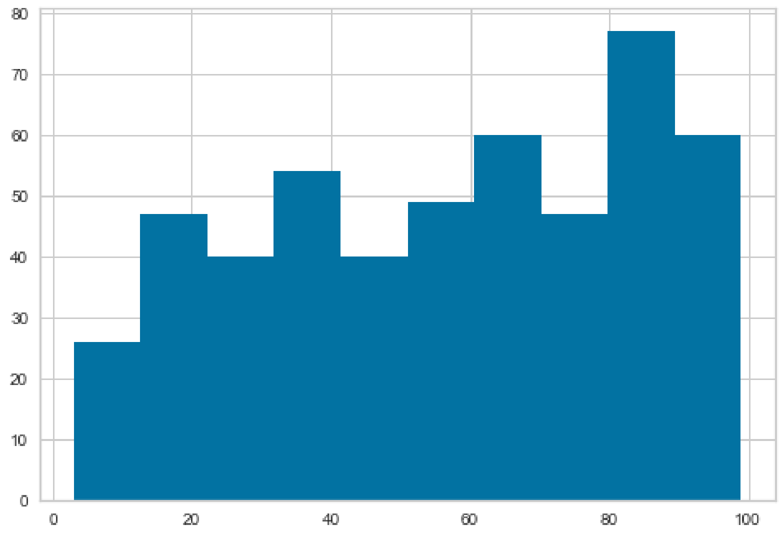

Figure 5.8. Distribution of the optimal number of neurons for each of the two hidden layers.

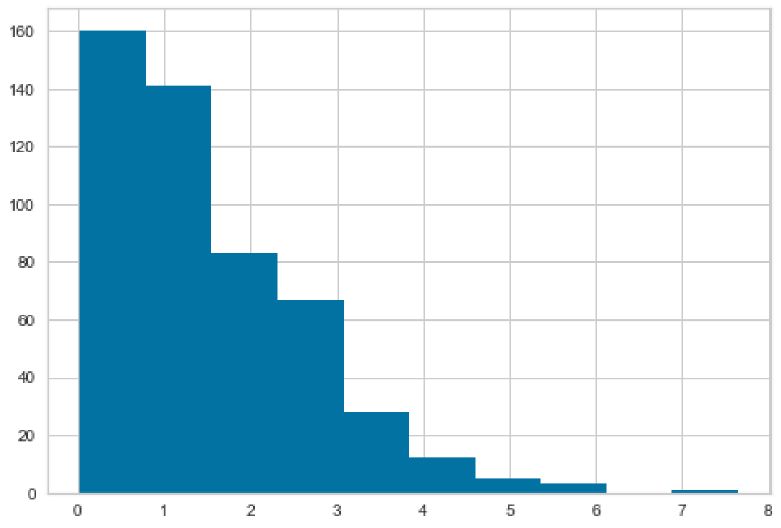

Figure 5.9. Distribution of the optimal alpha regularization term.

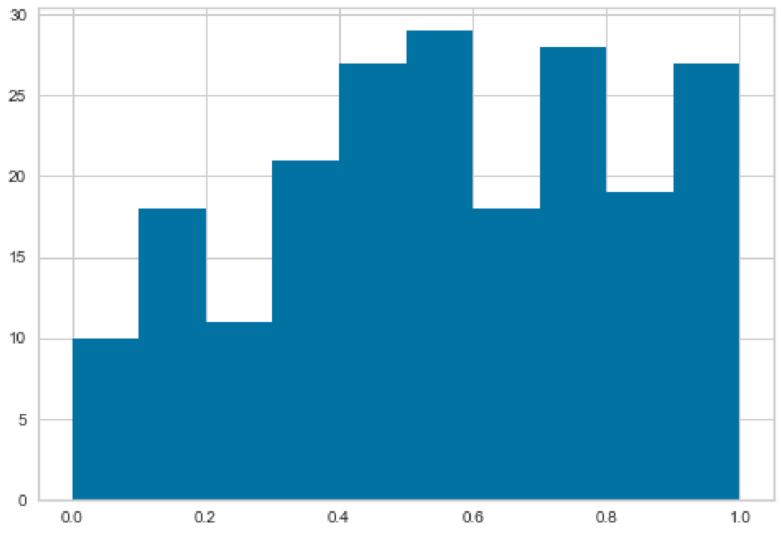

Figure 5.10. Distribution of the optimal alpha in the 0 - 1 range.

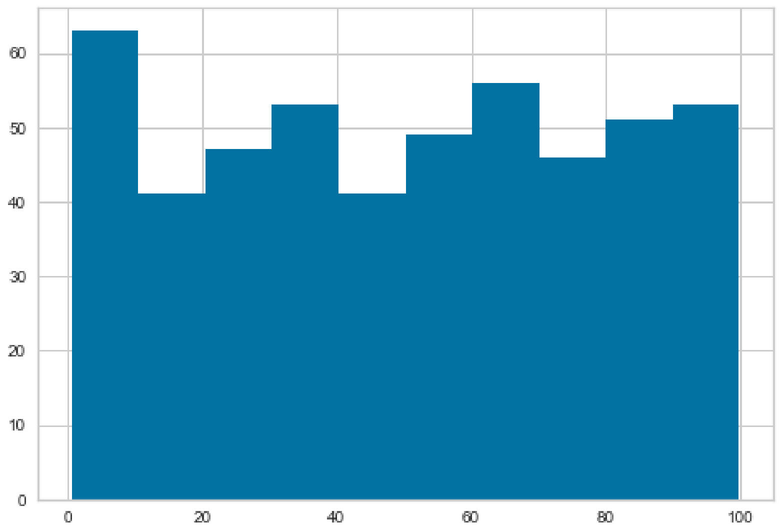

Figure 5.11. Distribution of the optimal initial learning rate.

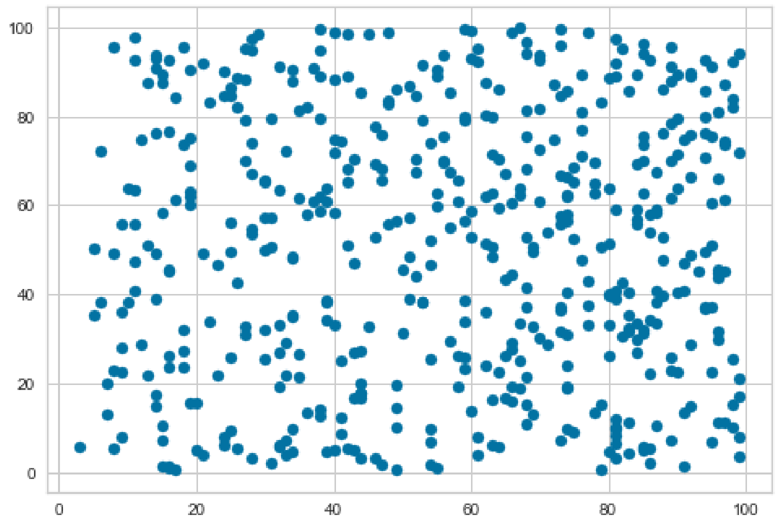

Figure 5.12. Number of neurons vs initial learning rate.

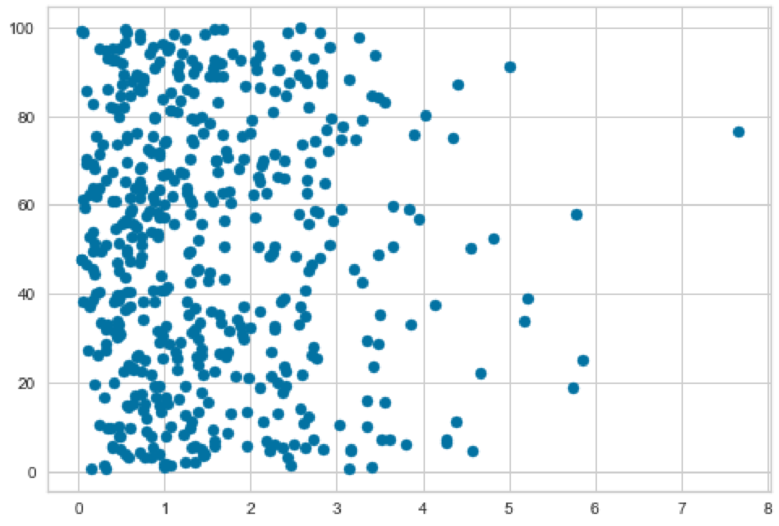

Figure 5.13. Alpha vs initial lerning rate.

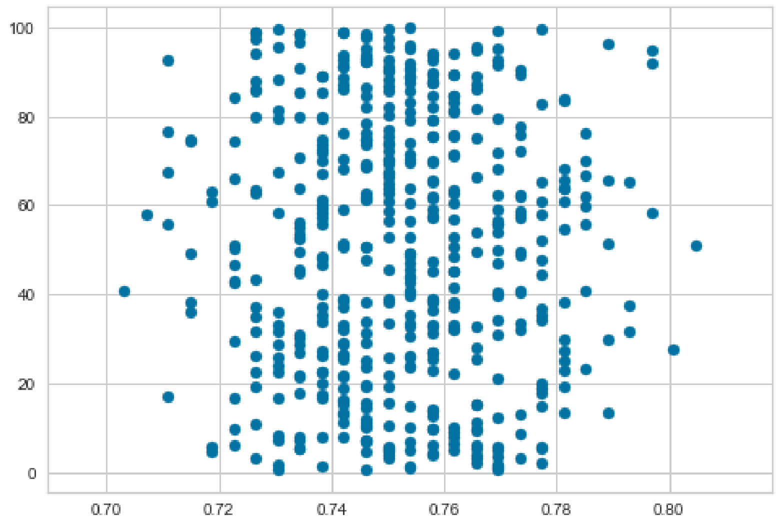

Figure 5.14. Accuracy vs initial learning rate.

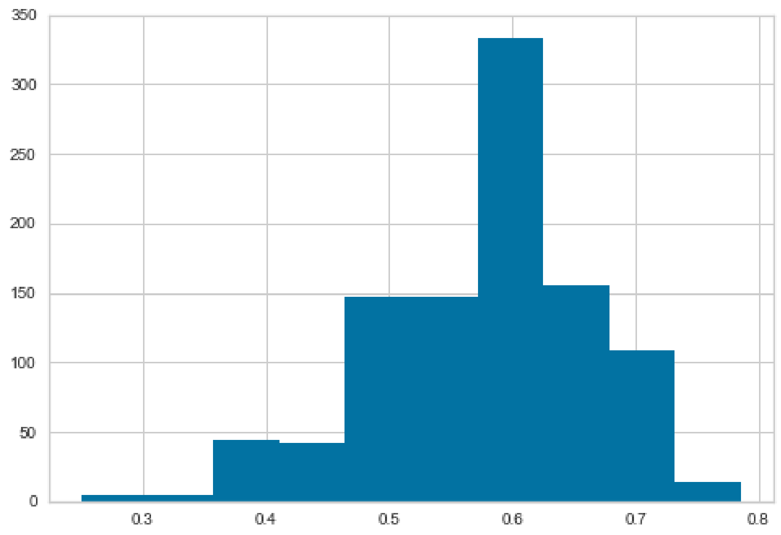

Figure 5.15. Accuracy on the UniTo dataset after transfer learning.

Chapter 6

Conclusions

The aim of this thesis can be summarized in the following question: is it possible to build an automatic, low cost and flexible system able to distinguish Italian speakers suffering of mild cognitive decline from those who are not? To answer this question, a set of 34 features has been extracted, covering prosodic, lexical, morphosyntactic and semantic aspects of the language. The text on which those features have been computed has been obtained by automatically transcribe audio tracks using a general purpose speech recognizer (i.e. Google Cloud Speech). Then a neural network has been tuned and trained with nested cross validation. The hyperparameters tuning involve also the selection of an optimal number of features, ranked basing on the mutual information with respect to the target variable.
The question stated above has not received a definitive answer, probably due to a combination of scarcity of data (12 pathological and 12 physiological individuals) and low signal to noise ration of the Italian dataset. However, the fact that the system performs very well (i.e. accuracy of about 0.84) on a large (170 pathological and 70 physiological subjects) English corpus and that it gives anyhow an accuracy around 0.65 for the Italian corpus (but with an high variance) is promising.
It's planned to continue the collaboration with the speech therapy department of UniTo, which will try to enlarge the dataset. On the technical side, the project can be improved in different directions, as described in the following section.

6.1 Future works

This thesis can be the starting point of more complex works. Improvements are possible in many area, including speech and speaker recognition, punctuation restoration, features extraction and classification. Many of these, however, depend on the availability of a large enough dataset.
Concerning speech recognition, it would be important to tune a recognizer with

specific models. Some cloud services already provide this option, and the other will probably follow this trend in the near future. Speakers segmentation would also give a significant improvement, and it's another feature that is being adopted by most services. This however require a training phase of the recognizer for each different pair of speakers.
Concerning punctuation restoration, section 4.3 provides some literature reference and proposes a novel approach for joint punctuation insertion and POS tagging with quite detailed instructions to train the model. A future work may implement that and other approaches and choose the best one. In any case, this task can probably be solved effectively only with deep learning techniques, and an high performance computation platform is needed for most of them. Once punctuation is available, syntax parsing should perform better, thus allowing to extract complex features that depend upon it.
Classification can be probably improved by considering different features selection algorithms (and maybe combining them as in [69] and [70]) and classifiers, and by exploiting better ensemble methods. To improve the performances on small datasets, an approach suggested by Dr. Rossella Muò (UniTo) can be tried. The idea is to start training the model on more compromised subjects for the AD group, which are easier to find and record, then progressively refine it with less compromised individuals.

6.2 Acknowledgments

The work of this thesis has been possible thanks to the datasets provided by Rossella Barile, Francesca Fusero, Dr. Anna Accornero and Dr. Rossella Muò of the speech department of the university of Turin (UniTo corpus), Dr. Brian MacWhinney, Dr. Davida Fromm and the authors of [59] (DementiaBank), Dr. Maddalena Costanzo and Dr. Gabriele Miceli (Trento corpus). Drs. Accornero and Muò also provided useful suggestions and guidance along the way.
Special thanks to Dr. Ilaria Bosi (LINKS foundation) and Pietro Vigorelli (Gruppo Anchise association) for helpfulness.

Bibliography

[1] C. Zheng, L. Lynch, and N. Taylor, "Effect of computer therapy in aphasia: a systematic review," *Aphasiology*, vol. 30, no. 2-3, pp. 211–244, 2016.

[2] A. Alzheimer's, "2015 alzheimer's disease facts and figures.," *Alzheimer's & dementia: the journal of the Alzheimer's Association*, vol. 11, no. 3, p. 332, 2015.

[3] R. Mayeux, "Genetic epidemiology of alzheimer disease," *Alzheimer Disease & Associated Disorders*, vol. 20, no. Supplement 2, 2006.

[4] G. Szatloczki, I. Hoffmann, V. Vincze, J. Kalman, and M. Pakaski, "Speaking in alzheimer's disease, is that an early sign? importance of changes in language abilities in alzheimer's disease," *Frontiers in aging neuroscience*, vol. 7, p. 195, 2015.

[5] V. Boschi, E. Catricala, M. Consonni, C. Chesi, A. Moro, and S. F. Cappa, "Connected speech in neurodegenerative language disorders: a review," *Frontiers in psychology*, vol. 8, p. 269, 2017.

[6] S. Singh, "Linguistic computing in speech and language disorders," in *Proceedings of the 5th International Conference on Speech Science and Technology*, pp. 5–8, 1994.

[7] S. Singh, *Computational analysis of conversational speech of dysphasic patients.* PhD thesis, University of the West of England at Bristol, 1996.

[8] S. Singh, "Computational linguistics for analysing conversation in speech and language disorders," in *Proc. 3rd International Conference on Statistical Analysis of Textual Data, Rome*, pp. 11–13, 1995.

[9] S. Singh and T. Bookless, "Analysing spontaneous speech in dysphasic adults," *International Journal of Applied Linguistics*, vol. 7, no. 2, pp. 165–181, 1997.

[10] R. S. Bucks, S. Singh, J. M. Cuerden, and G. K. Wilcock, "Analysis of spontaneous, conversational speech in dementia of alzheimer type: Evaluation of an objective technique for analysing lexical performance," *Aphasiology*, vol. 14, no. 1, pp. 71–91, 2000.

[11] C. Guinn, B. Singer, and A. Habash, "A comparison of syntax, semantics, and pragmatics in spoken language among residents with alzheimer's disease in managed-care facilities," in *Computational Intelligence in Healthcare and e-health (CICARE), 2014 IEEE Symposium on*, pp. 98–103, IEEE, 2014.

[12] C. I. Guinn and A. Habash, “Language analysis of speakers with dementia of the alzheimer’s type.,” in *AAAI Fall Symposium: Artificial Intelligence for Gerontechnology*, pp. 8–13, Menlo Park, CA, 2012.

[13] J. S. Yaruss, “Real-time analysis of speech fluency: Procedures and reliability training,” *American Journal of Speech-Language Pathology*, vol. 7, no. 2, pp. 25–37, 1998.

[14] H. Bortfeld, S. D. Leon, J. E. Bloom, M. F. Schober, and S. E. Brennan, “Disfluency rates in conversation: Effects of age, relationship, topic, role, and gender,” *Language and speech*, vol. 44, no. 2, pp. 123–147, 2001.

[15] B. H. Davis and M. Maclagan, “Examining pauses in alzheimer’s discourse,” *American Journal of Alzheimer’s Disease & Other Dementias®*, vol. 24, no. 2, pp. 141–154, 2009.

[16] C. Pope and B. H. Davis, “Finding a balance: The carolinas conversation collection,” *Corpus Linguistics and Linguistic Theory*, vol. 7, no. 1, pp. 143–161, 2011.

[17] A. Khodabakhsh, S. Kusxuoglu, and C. Demiroglu, “Natural language features for detection of alzheimer’s disease in conversational speech,” in *Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on*, pp. 581–584, IEEE, 2014.

[18] I. Hoffmann, D. Nemeth, C. D. Dye, M. Pákáski, T. Irinyi, and J. Kálmán, “Temporal parameters of spontaneous speech in alzheimer’s disease,” *International journal of speech-language pathology*, vol. 12, no. 1, pp. 29–34, 2010.

[19] L. Tóth, G. Gosztolya, V. Vincze, I. Hoffmann, G. Szatlóczki, E. Biró, F. Zsura, M. Pákáski, and J. Kálmán, “Automatic detection of mild cognitive impairment from spontaneous speech using asr,” in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[20] M. Lehr, E. Prud’hommeaux, I. Shafran, and B. Roark, “Fully automated neuropsychological assessment for detecting mild cognitive impairment,” in *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[21] K. Fraser, F. Rudzicz, N. Graham, and E. Rochon, “Automatic speech recognition in the diagnosis of primary progressive aphasia,” in *Proceedings of the fourth workshop on speech and language processing for assistive technologies*, pp. 47–54, 2013.

[22] E. T. Prud’hommeaux and B. Roark, “Alignment of spoken narratives for automated neuropsychological assessment,” in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pp. 484–489, IEEE, 2011.

[23] L. B. d. Santos, E. A. Corrêa Jr, O. N. Oliveira Jr, D. R. Amancio, L. L. Mansur, and S. M. Aluísio, “Enriching complex networks with word embeddings for detecting mild cognitive impairment from speech transcripts,” *arXiv preprint arXiv:1704.08088*, 2017.

[24] S. Karlekar, T. Niu, and M. Bansal, "Detecting linguistic characteristics of alzheimer's dementia by interpreting neural models," *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018.

[25] D. Beltrami, G. Gagliardi, R. Rossini Favretti, E. Ghidoni, F. Tamburini, and L. Calza, "Speech analysis by natural language processing techniques: a possible tool for very early detection of cognitive decline?," *Frontiers in Aging Neuroscience*, vol. 10, p. 369, 2018.

[26] M. A. Covington and J. D. McFall, "Cutting the gordian knot: The moving-average type–token ratio (mattr)," *Journal of quantitative linguistics*, vol. 17, no. 2, pp. 94–100, 2010.

[27] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pp. 63–70, Association for Computational Linguistics, 2002.

[28] M. Honnibal and I. Montani, "spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing," *To appear.*

[29] O. Tilk and T. Alumäe, "Bidirectional recurrent neural network with attention mechanism for punctuation restoration.," in *Interspeech*, pp. 3047–3051, 2016.

[30] N. Chomsky, "Syntactic structures," 1957.

[31] N. Chomsky, "Aspects of the theory of syntax," tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE RESEARCH LAB OF ELECTRONICS, 1964.

[32] N. Chomsky, *Lectures on government and binding: The Pisa lectures.* No. 9, Walter de Gruyter, 1993.

[33] N. Chomsky, "A minimalist program for linguistic theory. the view from building, 20, 1-52," 1993.

[34] W. O'grady, "The syntax of idioms," *Natural Language & Linguistic Theory*, vol. 16, no. 2, pp. 279–312, 1998.

[35] T. Osborne, M. Putnam, and T. Groß, "Catenae: Introducing a novel unit of syntactic analysis," *Syntax*, vol. 15, no. 4, pp. 354–396, 2012.

[36] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[37] E. Gabrilovich and S. Markovitch, "Feature generation for text categorization using world knowledge," in *IJCAI*, vol. 5, pp. 1048–1053, 2005.

[38] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis.," in *IJcAI*, vol. 7, pp. 1606–1611, 2007.

[39] S. T. Dumais, "Latent semantic analysis," *Annual review of information science and technology*, vol. 38, no. 1, pp. 188–230, 2004.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[41] T. Linzen, "Issues in evaluating semantic spaces using word analogies," *arXiv preprint arXiv:1606.07736*, 2016.

[42] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[43] A. Trask, P. Michalak, and J. Liu, "sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings," *arXiv preprint arXiv:1511.06388*, 2015.

[44] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[45] I. Gurevych, R. Malaka, R. Porzel, and H.-P. Zorn, "Semantic coherence scoring using an ontology," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 9–16, Association for Computational Linguistics, 2003.

[46] H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky, "Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task," in *Proceedings of the fifteenth conference on computational natural language learning: Shared task*, pp. 28–34, Association for Computational Linguistics, 2011.

[47] S. Agrawal, A. Joshi, J. C. Ross, P. Bhattacharyya, and H. M. Wabgaonkar, "Are word embedding and dialogue act class-based features useful for coreference resolution in dialogue," in *Proceedings of PACLING*, 2017.

[48] B. J. Grosz, S. Weinstein, and A. K. Joshi, "Centering: A framework for modeling the local coherence of discourse," *Computational linguistics*, vol. 21, no. 2, pp. 203–225, 1995.

[49] B. J. Grosz and C. L. Sidner, "Attention, intentions, and the structure of discourse," *Computational linguistics*, vol. 12, no. 3, pp. 175–204, 1986.

[50] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pp. 86–90, Association for Computational Linguistics, 1998.

[51] C. F. Baker and C. Fellbaum, "Wordnet and framenet as complementary resources for annotation," in *Proceedings of the third linguistic annotation workshop*, pp. 125–129, Association for Computational Linguistics, 2009.

[52] R. Navigli and S. P. Ponzetto, "Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network," *Artificial Intelligence*, vol. 193, pp. 217–250, 2012.

[53] L. Iacoponi and R. Savy, "Sylli: Automatic phonological syllabification for

italian," in *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[54] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Citeseer, 2010.

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

[56] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pp. 357–361, IEEE, 1994.

[57] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855, ACM, 2013.

[58] G. Allibrio, A. Gori, G. Signorini, and C. Luzzatti, "Un esame del linguaggio per la diagnosi dei deficit afasici al letto del malato," *Giornale di Psicologia*, vol. 3, no. 1, pp. 7–21, 2009.

[59] J. T. Becker, F. Boiler, O. L. Lopez, J. Saxton, and K. L. McGonigle, "The natural history of alzheimer's disease: description of study cohort and accuracy of diagnosis," *Archives of Neurology*, vol. 51, no. 6, pp. 585–594, 1994.

[60] D. Zhang, S. Wu, N. Yang, and M. Li, "Punctuation prediction with transition-based parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, pp. 752–760, 2013.

[61] A. Björkelund, A. Faleńska, W. Seeker, and J. Kuhn, "How to train dependency parsers with inexact search for joint sentence boundary detection and parsing of entire documents," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, pp. 1924–1934, 2016.

[62] O. Tilk and T. Alumäe, "Lstm for punctuation restoration in speech transcripts," in *Sixteenth annual conference of the international speech communication association*, 2015.

[63] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *arXiv preprint arXiv:1705.03122*, 2017.

[64] M. Elbayad, L. Besacier, and J. Verbeek, "Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction," *arXiv preprint arXiv:1808.03867*, 2018.

[65] V. Lyding, E. Stemle, C. Borghetti, M. Brunello, S. Castagnoli, F. Dellorletta, H. Dittmann, A. Lenci, and V. Pirrelli, "The paisÀ corpus of italian web

texts," *Proceedings of the 9th Web as Corpus Workshop (WaC-9)*, 2014.

[66] B. DeWilde, "textacy documentation," 2017.

[67] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.

[68] J. Hua, Z. Xiong, J. Lowey, E. Suh, and E. R. Dougherty, "Optimal number of features as a function of sample size for various classification rules," *Bioinformatics*, vol. 21, no. 8, pp. 1509–1515, 2004.

[69] Ö. Uncu and I. Türkşen, "A novel feature selection approach: combining feature wrappers and filters," *Information Sciences*, vol. 177, no. 2, pp. 449–466, 2007.

[70] T. Sivapriya, A. Kamal, and P. Thangaiah, "Ensemble merit merge feature selection for enhanced multinomial classification in alzheimer's dementia," *Computational and mathematical methods in medicine*, vol. 2015, 2015.

Appendix A

Code

```python
from google.cloud import speech_v1p1beta1 as speech
from google.cloud.speech_v1p1beta1 import types, enums
import pickle
import spacy
from spacy.attrs import ORTH, LEMMA
import pyphen
import matplotlib.pyplot as plt
import numpy
from openpyxl import Workbook, load_workbook
from nltk.corpus import wordnet as wn
from gensim.models import Word2Vec
from itertools import islice
import textacy
from pprint import pprint
import networkx as nx
from random import randint

class_names=['healty', 'alzheimer']
sample_names=['sano', 'alz']

clitic_pronouns={
   'gli': {'Gender': 'Masc', 'Number': 'Sing'},
   'li': {'Gender': 'Masc', 'Number': 'Plur'},
   'le': {'Gender': 'Fem', 'Number': 'Any'},
   'lo': {'Gender': 'Masc', 'Number': 'Sing'},
   'la': {'Gender': 'Fem', 'Number': 'Sing'}
}

pos_mapping={
   'ADJ': wn.ADJ,
    'NOUN': wn.NOUN,
    'VERB': wn.VERB
}

general_words=['cosa', 'coso', 'cose', 'cosi', 'affare', 'affari', 'tizio
    ↪ ', 'tizia', 'tizi', 'tizie']

class Treenode(object):
    def __init__(self, data=None, parent=None):
        self.data=data
        self.parent=parent
        self.children=[]

def rolling_window(seq, window_size):
    win = [] # First window
    for i in range(window_size):
        win.append(seq[i])
    yield win
    for e in seq: # Subsequent windows
        win[:-1] = win[1:]
        win[-1] = e
        yield win

def pos_rates(doc):
    words_count=0
    adj_count=0
    noun_count=0
    modified_noun_count=0
    pron_count=0
    verb_count=0
    for token in doc:
        words_count+=1
        if token.pos_=='ADJ':
            adj_count+=1
        if token.pos_=='NOUN' or token.pos_=='PROPN':
            noun_count+=1
            if token.text not in general_words:
                modified_noun_count+=1
        if token.pos_=='PRON':
            pron_count+=1
        if token.pos_=='VERB':
            verb_count+=1
```

```python
    return adj_count/words_count, noun_count/words_count, modified_noun_count/words_count, pron_count/words_count, verb_count/words_count

def repetitions_rate(doc, rep_len, gap_len):
    rep_count=0
    for i in range(len(doc)-2*rep_len-gap_len):
        repeated=True
        for j in range(i, i+rep_len):
            if doc[j].text!=doc[j+rep_len+gap_len].text:
                repeated=False
        if repeated==True:
            rep_count+=1
    return rep_count/(len(doc)-2*rep_len-gap_len)

def pronominal_anaphora_resolution_rate(seq):
    pron_count=0
    anaphora_count=0
    for i in range(len(seq)-1):
        is_pronoun=False
        if seq[i].pos_=='PRON' or (seq[i].pos_=='ADJ' and seq[i-1].pos_=='DET' and seq[i+1].pos_!='NOUN'):
            is_pronoun=True
            pron_gender=seq[i].tag_[seq[i].tag_.find('Gender=')+7:seq[i].tag_.find('Gender=')+10]
            pron_number=seq[i].tag_[seq[i].tag_.find('Number=')+7:seq[i].tag_.find('Number=')+11]
        if seq[i].pos_=='VERB':
            for clitic, morph in clitic_pronouns.items():
                if seq[i].text.endswith(clitic):
                    is_pronoun=True
                    pron_gender=morph['Gender']
                    pron_number=morph['Number']
                    break
        if is_pronoun:
            pron_count+=1
            for token in seq[i-1::-1]:
                if token.pos_=='NOUN':
                    ant_gender=token.tag_[token.tag_.find('Gender=')+7:token.tag_.find('Gender=')+10]
                    ant_number=token.tag_[token.tag_.find('Number=')+7:token.tag_.find('Number=')+11]
                    if (pron_gender in ['Mas', 'Fem'] and pron_gender==
                        ↪ ant_gender) or (pron_gender!='Fem' and ant_gender not
                        ↪  in ['Mas', 'Fem']):
                        if (pron_number in ['Sing', 'Plur'] and pron_number==
                            ↪ ant_number) or (pron_number not in ['Sing', 'Plur'
                            ↪ ]):
                            #print(seq[i].text, seq[i].pos_, pron_gender,
                                ↪ pron_number, 'refers to', token.text, token.pos_
                                ↪ , ant_gender, ant_number)
                            anaphora_count+=1
                            break
    if pron_count==0:
        return 1
    return anaphora_count/pron_count

def w2v_ontoscore(seq, w2v_model):
    covered_words=[]
    cosine_tot=0
    euclidean_tot=0
    for w in seq:
        if not w.is_stop and w.text in w2v_model.wv.vocab:
            covered_words.append((w.text, w2v_model.wv[w.text]))
            #print(w.text, numpy.linalg.norm(numpy.array(w2v_model.wv[w.text
                ↪ ])))
    cw_number=len(covered_words)
    for i, w1 in enumerate(covered_words, 1):
        for w2 in covered_words[:i-1]:
            cosine_tot+=w2v_model.wv.similarity(w1[0], w2[0])
            euclidean_tot+=numpy.linalg.norm(numpy.array(w1[1]-w2[1]))
    return cosine_tot*2/(cw_number*cw_number-cw_number), euclidean_tot*2/(
        ↪ cw_number*cw_number-cw_number), cw_number/len(seq)

def wordnet_ontoscore(seq, synsets, start_id, old_covered_words,
    ↪ old_score, cache):
    tree=Treenode(0)
    level_nodes=[]
    covered_words=[]
    stack=[]
    cur_similarity_sum=0
    max_similarity_sum=0
    i=start_id
    for w in seq:
        if len(synsets[i])>0:
            covered_words.append(synsets[i])
```

```
      i+=1
   if covered_words==old_covered_words:
      #print('equal to previous')
      return old_score, len(covered_words)/len(seq), covered_words
   cw_number=len(covered_words)
   if(cw_number>0):
      for s in covered_words[0]:
         n=Treenode(s, tree)
         tree.children.append(n)
         level_nodes.append(n)
      for w in covered_words:
         new_level_nodes=[]
         for n in level_nodes:
            for s in w:
               c=Treenode(s, n)
               n.children.append(c)
               new_level_nodes.append(c)
         level_nodes=new_level_nodes
         #print(len(level_nodes))
      for n1 in level_nodes:
         path=[n1]
         while(n1.parent is not tree):
            n2=n1.parent
            for n in path:
               if (n, n2) not in cache:
                  ps=n.data.path_similarity(n2.data)
                  cache[(n, n2)]=ps
               else:
                  ps=cache[(n, n2)]
               if ps is not None:
                  cur_similarity_sum+=ps
               if (n2, n) not in cache:
                  ps=n2.data.path_similarity(n.data)
                  cache[(n2, n)]=ps
               else:
                  ps=cache[(n2, n)]
               if ps is not None:
                  cur_similarity_sum+=ps
            path.append(n2)
            n1=n2
         if(cur_similarity_sum>max_similarity_sum):
            max_similarity_sum=cur_similarity_sum
      score=max_similarity_sum
      if cw_number>1:
            score/=cw_number*cw_number-cw_number
        return score, cw_number/len(seq), covered_words
    return 0, 0, []

client=speech.SpeechClient();
print("window size: ", window_size)
audio = types.RecognitionAudio(uri='gs://alzthesis/alz1.flac')
config = types.RecognitionConfig(encoding=enums.RecognitionConfig.AudioEncoding.FLAC, language_code='it-IT', enable_word_time_offsets=True, enable_word_confidence=True)
operation = client.long_running_recognize(config, audio)

print('Waiting for operation to complete. Loading some modules.')
model=Word2Vec.load('wiki_iter=5_algorithm=skipgram_window=10_size=300_neg-samples=10.m')
nlp=spacy.load('it_core_news_sm')
nlp.tokenizer.add_special_case("po'", [{ORTH: "po'", LEMMA: 'poco'}])
response = operation.result()

with open('texts/alz1.txt', 'wb') as f:
    pickle.dump(response, f)

string=''
for result in response.results:
print(format(result.alternatives[0].transcript))
string+=result.alternatives[0].transcript+' '

doc=nlp(string)

synsets_list=[]
for token in doc:
    if token.pos_ in pos_mapping.keys() and not token.is_stop:
        synsets_list.append(wn.synsets(token.lemma_, pos=pos_mapping[token.pos_], lang='ita')[:3])
    else:
        synsets_list.append([])

window_size=randint(100, 200)
words_objects=[]
for result in response.results:
    for w in result.alternatives[0].words:
        words_objects.append(w)
windows_times=[]
windows_average_confidences_list=[]
```

```
for words_window in islice(rolling_window(words_objects, window_size),
    ↪ window_size, None):
  windows_times.append(words_window[-1].end_time.seconds-words_window
      ↪ [0].start_time.seconds)
  tot=0
  for w in words_window:
    tot+=w.confidence
  windows_average_confidences_list.append(tot/len(words_window))
average_window_time=sum(windows_times)/len(windows_times)

adj_rate_list=[]
noun_rate_list=[]
modified_noun_rate_list=[]
pron_rate_list=[]
verb_rate_list=[]
maar_list=[]
manr_list=[]
mamnr_list=[]
mapr_list=[]
mavr_list=[]
ttr_list=[]
mattr_list=[]
r1w0f_list=[]
r1w1f_list=[]
r1w2f_list=[]
r2w0f_list=[]
r2w1f_list=[]
r2w2f_list=[]
syllables_per_min_list=[]
anaphora_rate_list=[]
w2v_cosine_ontoscore_list=[]
w2v_euclidean_ontoscore_list=[]
w2v_coverage_list=[]
wordnet_ontoscore_list=[]
wordnet_coverage_list=[]
pagerank_list=[]
betweenness_list=[]
eccentricity_list=[]
eigenvector_list=[]
neighbor_degree_list=[]
average_shortest_path_list=[]
degree_list=[]
degree_assortativity_list=[]
diameter_list=[]
```

```
average_clustering_list=[]
win_number=0
for span in islice(rolling_window(doc, window_size), window_size, None):
    win_number+=1
    print('Processing window '+str(win_number)+' of '+str(len(doc)+1-window_size))
    adj_rate, noun_rate, modified_noun_rate, pron_rate, verb_rate=pos_rates(span)

    adj_rate_list.append(adj_rate)
    noun_rate_list.append(noun_rate)
    modified_noun_rate_list.append(modified_noun_rate)
    pron_rate_list.append(pron_rate)
    verb_rate_list.append(verb_rate)

    adj_rate_list1=[]
    noun_rate_list1=[]
    modified_noun_rate_list1=[]
    pron_rate_list1=[]
    verb_rate_list1=[]
    for win in rolling_window(span, 30):
        adj_rate, noun_rate, modified_noun_rate, pron_rate, verb_rate=pos_rates(win)
        adj_rate_list1.append(adj_rate)
        noun_rate_list1.append(noun_rate)
        modified_noun_rate_list1.append(modified_noun_rate)
        pron_rate_list1.append(pron_rate)
        verb_rate_list1.append(verb_rate)

    maar_list.append(sum(adj_rate_list1)/float(len(adj_rate_list1)))
    manr_list.append(sum(noun_rate_list1)/float(len(noun_rate_list1)))
    mamnr_list.append(sum(modified_noun_rate_list1)/float(len(modified_noun_rate_list1)))
    mapr_list.append(sum(pron_rate_list1)/float(len(pron_rate_list1)))
    mavr_list.append(sum(verb_rate_list1)/float(len(verb_rate_list1)))

    words_count=0
    distinct_words=set()
    for token in span:
        words_count+=1
        distinct_words.add(token.text.lower())

    ttr_list.append(len(distinct_words)/words_count)

```

```
ttr_list1=[]
  for win in islice(rolling_window(span, 30), 30, len(span)+1):
    words_count=0
    distinct_words.clear()
    for token in win:
      words_count+=1
      distinct_words.add(token.text.lower())
    ttr_list1.append(len(distinct_words)/words_count)

mattr_list.append(sum(ttr_list1)/float(len(ttr_list1)))

r1w0f_list.append(repetitions_rate(span, 1, 0))
r1w1f_list.append(repetitions_rate(span, 1, 1))
r1w2f_list.append(repetitions_rate(span, 1, 2))
r2w0f_list.append(repetitions_rate(span, 2, 0))
r2w1f_list.append(repetitions_rate(span, 2, 1))
r2w2f_list.append(repetitions_rate(span, 2, 2))

p=pyphen.Pyphen(lang='it_IT')
syllables_count=0
for w in span:
  syllables_count+=len(p.inserted(w.text).split('-'))
if win_number>len(windows_times):
  syllables_per_min_list.append(average_window_time)
else:
  syllables_per_min_list.append(syllables_count*60/windows_times[
      ↪ win_number-1])

anaphora_rate_list1=[]
for w in islice(rolling_window(span, 30), 30, len(span)+1):
  anaphora_rate_list1.append(pronominal_anaphora_resolution_rate(w))
anaphora_rate_list.append(sum(anaphora_rate_list1)/float(len(
    ↪ anaphora_rate_list1)))

w2v_cosine_ontoscore_list1=[]
w2v_euclidean_ontoscore_list1=[]
w2v_coverage_list1=[]
for w in islice(rolling_window(span, 30), 30, len(span)+1):
  w2vco, w2veo, w2vc=w2v_ontoscore(w, model)
  w2v_cosine_ontoscore_list1.append(w2vco)
  w2v_euclidean_ontoscore_list1.append(w2veo)
  w2v_coverage_list1.append(w2vc)
w2v_cosine_ontoscore_list.append(sum(w2v_cosine_ontoscore_list1)/float
    ↪ (len(w2v_cosine_ontoscore_list1)))
```

```
w2v_euclidean_ontoscore_list.append(sum(w2v_euclidean_ontoscore_list1)
    ↪ /float(len(w2v_euclidean_ontoscore_list1)))
w2v_coverage_list.append(sum(w2v_coverage_list1)/float(len(
    ↪ w2v_coverage_list1)))

wordnet_ontoscore_list1=[]
wordnet_coverage_list1=[]
i=0
cw=[]
wno=0
cache={}
for w in islice(rolling_window(doc, 5), 5, len(doc)+1):
  wno, wnc, cw=wordnet_ontoscore(w, synsets_list, i, cw, wno, cache)
  i+=1
  wordnet_ontoscore_list1.append(wno)
  wordnet_coverage_list1.append(wnc)
wn_ontoscore=0
for e in wordnet_ontoscore_list1:
  wn_ontoscore+=e
wn_ontoscore/=len(wordnet_ontoscore_list1)
wordnet_ontoscore_list.append(wn_ontoscore)
wordnet_coverage=0
for e in wordnet_coverage_list1:
wordnet_coverage+=e
wordnet_coverage/=len(wordnet_coverage_list1)
wordnet_coverage_list.append(wordnet_coverage)

wl=[]
for w in span:
  if not w.is_stop:# and w.text in model.wv.vocab:
    wl.append(w.text)
g=textacy.network.terms_to_semantic_network(wl, normalize='lower')
nl=list(g.nodes)
for i in range(len(nl)):
  n1=nl[i]
  if n1 in model.wv.vocab:
    for n2 in nl[i+1:]:
      if n2 in model.wv.vocab and model.wv.similarity(n1, n2)>=0.5:
        g.add_edge(n1, n2)
l=nx.pagerank_numpy(g).values()
pagerank_list.append(sum(l)/float(len(l)))
l=nx.betweenness_centrality(g).values()
betweenness_list.append(sum(l)/float(len(l)))
l=nx.eccentricity(g).values()
```

```
    eccentricity_list.append(sum(l)/float(len(l)))
    l=nx.eigenvector_centrality_numpy(g).values()
    eigenvector_list.append(sum(l)/float(len(l)))
    l=nx.average_neighbor_degree(g).values()
    neighbor_degree_list.append(sum(l)/float(len(l)))
    average_shortest_path_list.append(nx.average_shortest_path_length(g))
    l=nx.degree_centrality(g).values()
    degree_list.append(sum(l)/float(len(l)))
    degree_assortativity_list.append(nx.degree_assortativity_coefficient(g
        ↪ ))
    diameter_list.append(nx.diameter(g))
    average_clustering_list.append(nx.average_clustering(g))

print('Saving on file')
wb=load_workbook('results.xlsx')
s=wb.get_sheet_by_name('sheet1')
for ar, nr, mnr, pr, vr, maar, manr, mamnr, mapr, mavr, ttr, mattr, r1w0f
    ↪ , r1w1f, r1w2f, r2w0f, r2w1f, r2w2f, anaphora,
    ↪ w2v_cosine_ontoscore, w2v_euclidean_ontoscore, w2v_coverage,
    ↪ pagerank, degree, betweenness, eccentricity, eigenvector,
    ↪ average_shortest_path, diameter, average_clustering,
    ↪ syllables_per_minute, average_confidence in zip(adj_rate_list,
    ↪ noun_rate_list, modified_noun_rate_list, pron_rate_list,
    ↪ verb_rate_list, maar_list, manr_list, mamnr_list, mapr_list,
    ↪ mavr_list, ttr_list, mattr_list, r1w0f_list, r1w1f_list,
    ↪ r1w2f_list, r2w0f_list, r2w1f_list, r2w2f_list, anaphora_rate_list
    ↪ , w2v_cosine_ontoscore_list, w2v_euclidean_ontoscore_list,
    ↪ w2v_coverage_list, pagerank_list, degree_list, betweenness_list,
    ↪ eccentricity_list, eigenvector_list, average_shortest_path_list,
    ↪ diameter_list, average_clustering_list, syllables_per_min_list,
    ↪ windows_average_confidences_list):
    print('appending row')
    s.append([ar, nr, mnr, pr, vr, maar, manr, mamnr, mapr, mavr, ttr,
        ↪ mattr, r1w0f, r1w1f, r1w2f, r2w0f, r2w1f, r2w2f, anaphora,
        ↪ w2v_cosine_ontoscore, w2v_euclidean_ontoscore, w2v_coverage,
        ↪ pagerank, degree, betweenness, eccentricity, eigenvector,
        ↪ average_shortest_path, diameter, average_clustering,
        ↪ syllables_per_minute, average_confidence, 'alzheimer', 'alz1'])
wb.save('results.xlsx')
wb.close()
```