Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

A privacy-preserving AI-based Intent Recognition engine with Probabilistic Spell-Editing for an Italian Smart Home Voice Assistant

Relatore:
Chiar.mo Prof.
DANILO MONTESI

Presentata da:
PAOLA PERSICO

Sessione II
Anno Accademico 2022/2023

To my nephew Francesco

Sommario

Negli ultimi decenni, il mercato dei dispositivi per la Smart Home si è espanso notevolmente. Tra le varie interfacce che permettono di inviare comandi a questi dispositivi, è di particolare interesse quella fornita dagli assistenti virtuali, testuali e/o vocali, soprattutto in quanto capace di offrire più indipendenza alle persone con disabilità e alle persone anziane, gruppo in aumento significativo in Italia.

Purtroppo le soluzioni attuali sul mercato, come gli smart speaker, sono basate sull'invio dei comandi a server remoti, facendo sorgere preoccupazioni più o meno legittime riguardo la privacy. Le alternative open-source attualmente disponibili, di contro, sono poco accurate per la lingua italiana.

L'obiettivo di questa tesi è di sviluppare un nuovo motore di Intent Recognition, chiamato Converso, per assistenti domotici in lingua italiana che possono essere integrati in piattaforme locali come Home Assistant. Per raggiungere quest'obiettivo, è stato generato un dataset sintetico, pre-processato tramite embedding Word2Vec, per addestrare modelli di Machine Learning per la classificazione degli Intent e degli slot; inoltre, è stato sviluppato un algoritmo basato su N-grammi per correggere gli errori ortografici o di riconoscimento vocale.

L'agente di conversazione derivante, che si serve di una Support Vector Machine e non richiede alcuna connessione a server remoti, è stato valutato in un esperimento in condizioni realistiche, dimostrando un'accuratezza superiore al 60%.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the last decades, the market of *smart* appliances which can be monitored and controlled from a distance, such as light bulbs and thermostatic valves, has greatly expanded, becoming varied and accessible to a wider set of consumers.

Among the various interfaces which allow a user to send commands to these devices, one of the most interesting ones is the Virtual Assistant, which can let a user communicate to the device, or most commonly to the Home Automation platform which integrates the device, through text commands or voice requests in natural language.

This type of interface, in fact, can not only be useful for ordinary people who temporarily require a hands-free interaction, but it can also provide more independence to people with disabilities, such as visually impaired people, and to old people who are not familiar with graphic UIs, with the latter group increasing significantly in Italy [52].

Unfortunately, the available consumer solutions, such as Amazon Alexa's and Google Assistant's smart speakers, require the voice commands to be sent to their own servers. Beyond depending on a connection to the Internet, the fundamental issue of these *cloud* solutions is the problem of privacy. Even if, in theory, only voice signals following activation words, called *wake words*, are sent to the Internet, a significant fraction of the population does not desire to adopt Voice Assistants in their own home, due to privacy concerns about potential leakage and abuse of private conversations [53].

1.2 Goal

The goal of this research is to design a novel Intent Recognition engine for the Italian Language which can be integrated into a local privacy-preserving Virtual Assistant in order to monitor and control domotic appliances. The main parameters which are taken into account are accuracy, delay, level of privacy and resource consumption.

In order to achieve this goal, a synthetic dataset with text commands is generated to train several Machine Learning models to classify Intents, by first extracting relevant features using Word2Vec word embeddings. The model with the best accuracy is used, in conjunction with a speech correction algorithm based on N-grams, to convert the text command into machine commands which can be handled by the smart home platform.

Finally, the model is evaluated by developing an integration for Home Assistant, the most popular local smart home platform.

1.3 Thesis structure

The thesis is structured into the following chapters:

- Chapter 1, which provides an overview on Text Analysis, clarifying the core definitions and concepts, dealing with the historical evolution of the field and its main applications;
- Chapter 2, which introduces the most commonly used Text Analysis techniques deriving from the fields of Natural Language Processing, Information Retrieval and Machine Learning;

- Chapter 3, which deals with the main notions related to the Home Automation domain and to the Virtual Assistants interfaces, with a section about Home Assistant;
- Chapter 4, which introduces Converso, a new privacy-preserving Intent Recognition engine specifically designed for local smart home platforms;
- Chapter 5, which evaluates the performance of the engine through an experiment conducted in a real environment with a varied set of participants.

Chapter 2

Overview on Text Comprehension

This chapter will deal with the principles and history of Text Comprehension, clarifying differences and similarities compared to other related fields, and the main applications.

2.1 Basics

Text Comprehension, the process of creating a structured meaningful representation from text, usually relies on Text Analysis (TA), which can be seen as a special case of Data Analysis.

There are multiple definitions of Data Analysis: for some researchers it is a synonym of Knowledge Discovery from Data (KDD), the process of *"extracting knowledge from data"*, which also includes the preliminary data pre-processing activities and the final activities of evaluation and presentation of the results; for others, however, Data Analysis is only the sub-process of KDD which consists in the application of automatic methods for the discovery of patterns [1, p. 6].

Similarly, Text Analysis can also be seen both as a synonym of Knowledge Discovery from Text (KDT), i.e. the general process of extracting knowledge from textual data, and as a specific sub-process.

In the following work, the term Text Analysis will be used in the first sense, as textual data have peculiarities that strongly influence each activity of the process: textual data are in fact unstructured data, which are both generated and consumed by human beings [2, p. 48]. For this reason, Text Analysis does not only make use of the classic Data Analysis techniques, derived from disciplines such as Machine Learning (ML), Probability and Statistics, Information Theory and Databases; but it also makes use of techniques deriving from the fields of Library Science and Computational Linguistics.

In particular, Information Retrieval (IR) and Natural Language Processing (NLP) techniques can be used. IR aims to *"find material (usually documents) of an unstructured nature (usually text) that satisfies an information need from large collections (usually stored on computers)"* [3, p. 1]. NLP, on the other hand, aims to *"allow a computer to understand the meaning of a text in natural language"* [2, p. 39]. In both cases, the human component plays a significant role.

2.2 Historical background

In this section we will briefly see the path of Text Analysis from its early days to the state of the art.

1913 The mathematician Markov proposes the N-gram characters probabilistic model for language modeling [13].

1949 Claude Shannon is the first to generate N-gram word models for the English language [14].

1950s Alan Turing describes a method for determining whether a machine has the ability to exhibit intelligent behavior (the so-called *Turing Test*), identifying natural language understanding as one of the most significant factors [12]. The linguist Chomsky criticizes the N-gram models, slowing down the line of research [10, p. 883]. The US government begins investing in the field of Machine Translation [2, p. 42].

1960s Bar-Hillel's report [15] is published, prompting researchers to pursue more feasibile goals for the time. The first efficient algorithms for automatic *parsing* are developed [10, p. 920].

1970s - 1980s Salton et al. introduce the Vector Space Model [7], which becomes the most used model in Information Retrieval. The first Information Extraction algorithms [10, p. 884] are developed. The techniques at this time are mainly based on heuristic rules of logical inference [2, p. 42].

1990s - 2000s Inspired by the success of statistical approaches to speech recognition, researchers rediscover N-gram models [10, p. 883]. In the mid-1990s, with the spread of the Internet and Big Data, the use of Machine Learning models for Text Mining tasks becomes more and more common.

2010s - today The excellent results achieved by Artificial Neural Networks, combined with greater GPUs computational power, is directing research towards Deep Learning [17]. A commonly adopted approach nowadays consists in using neural networks to obtain vector representations of words which are capable of capturing their semantics (*word embedding*), as in the case of the Word2Vec models (developed by Google in 2013)[18], the ELMo model (developed by the Allen Institute for Artificial Intelligence and the University of Washington in 2018)[21] and the BERT model (developed by Google in 2018) [22].

2.3 Applications

As previously mentioned, Text Analysis consists in extracting knowledge from text, and the knowledge can be defined as a set of *patterns relevant to applications*. In this section we will overview some macro-problems that can be solved through Text Analysis techniques, and some specific applications.

2.3.1 Classification

Classification consists in labeling data using predefined categories. This can be especially useful for facilitating search (reducing the amount of data to take into account) and analysis (helping discovering correlations).

Some examples in the textual field are the subdivision of news into categories, such as ‘news’, ‘entertainment’ and ‘sport’, and the classification of scientific articles according to the field of research.

In addition to classifying texts according to the topics covered, it is also possible to classify them according to their style (for example, to trace the author or to filter spam emails), or according to the underlying *sentiment* (for example, to evaluate whether a review is positive or negative, or to analyze political trends without relying on official statistics).

2.3.2 Clustering

Clustering consists in grouping similar elements into *clusters.* In the Text Analysis field, elements can be entire documents, sentences, or single terms. Clustering can be useful to get an overview of textual data, and therefore to facilitate exploration.

There are two main types of clustering [28]:

- partitional clustering, which outputs disjoint clusters;
- hierarchical clustering, which outputs a tree of clusters.

Compared to classification, clustering does not use predefined categories, and it is more useful for applications that need more flexibility because of this. In fact, defining categories in advance when there are multiple levels of interpretation can be not only burdensome for large volumes of data, but also very complex.

Clustering is mainly used in recommendation systems, and in *topic analysis.* Recommender systems aim to offer to a specific user content that may interest them, and to achieve this they either group similar content (*content-based filtering*) or similar users (*collaborative filtering*) [2, p. 221]. As far as *topic analysis* is concerned, it aims to discover the latent themes of a set of documents and the degree of coverage of the themes in each document.

2.3.3 Summarization

Summarization consists in compressing text to speed up its reading. Like digital compression, summarization aims to lose as little information as possible given a certain level of size reduction.

There are two main methods of summarization [2, p.318]:

- selection-based method, which outputs a subset of relevant sentences from the original document;
- generation-based method, which outputs new sentences generated by a language model.

Summarization is especially useful for long documents, such as books, legal and biomedical documents, and scientific articles.

2.3.4 Information Extraction

Information Extraction (IE) is the conversion of a document from unstructured to structured. In particular, it is based on entity extraction (for example, names of people, names of companies) and relationships between entities (for example, who works in a given company). Typically, these activities are referred to as Named Entity Recognition (NER) and Relationship Extraction, respectively.

Information Extraction not only allows textual data to be represented in other forms, such as tables and graphs, facilitating search and navigation, but can also be used as a preliminary step for other applications that need a deeper understanding of the text documents.

Chapter 3

Text Comprehension Techniques

In this chapter we will examine some of the most commonly adopted methods in text understanding, and we will split them according to their field of origin for clarity of exposition. Depending on the specific application, these methods can be combined at different stages of the Text Comprehension process.

3.1 Natural Language Processing

As mentioned in the previous chapter, the field of NLP deals with natural language understanding. Natural languages, compared to formal languages such as programming languages, have some peculiarities that make them difficult to deal with [10, p. 861] [2, p. 41]:

- they cannot be characterized as a defined set of sentences, as they may not follow a precise grammar (for example, the sentence *the majority of Italians are brown-haired* violates the singular-plural agreement rule, but it is still understandable for a human being);

- they are ambiguous, at the level of: part-of-speech analysis (*fishing* can be both a noun and a verb), logical analysis (in *A man saw a girl with a* *telescope* it is not clear whether *with the telescope* the telescope belongs to the man or the girl), semantic analysis (in *Mario said goodbye to Luigi and took his things* it is not clear whether Mario took Luigi's things or his own);

- they may take the so-called '*common sense knowledge*' for granted (in the sentence *he wanted a healthier lifestyle and he stopped smoking* it is implied that smoking is harmful);
- they are constantly evolving (for example, through the introduction of neologisms such as *memes*).

3.1.1 Tokenization

The first step of NLP is generally tokenization, which consists in dividing the sequence of characters into minimal units of analysis called *tokens*. In the case of the English language, a basic *tokenizer* is the *whitespace tokenizer*, which delimits tokens based on spacing [2, p. 149]. To improve tokenization, also other separators can be taken into account, such as tabs or punctuation elements, and tokens that are part of contracted forms can be split.

Once the tokenization has been done, depending on the specific applications, other pre-processing techniques can also be applied, such as:

- normalization;
- lemmatization;
- stemming.

Normalization is the transformation of tokens into terms: the same term, understood as an information unit, in fact, can be written in different ways [3, p. 28]. Normalization may include: removing periods in acronyms (from *P.S.* to *PS*); hyphen removal (from *e-mail* to *email*); the expansion of abbreviations (from *approx.* to *approximately*); uppercase to lowercase conversion of characters (from *Nike* to *nike*). In any case, attention must be paid to the collisions that may arise; for example, the token *Polish* in the middle of a sentence probably means something relating to Poland, while *polish* probably means *to make something shine by cleaning it.*

Lemmatization, on the other hand, is the transformation of the *inflected forms* of words into lemmas, which correspond to the entries in the dictionary. During lemmatization, words that were grammatically inflected are brought back to their base form following the conventions of the language. For example, the verb *was* and the verb *is* can both be replaced with *be* [8].

Finally, stemming consists in reducing words to their roots, and then removing prefixes and suffixes. Some examples of stemming are the transformation of *consolation* into *consol* and of *digestive* into *digest.* A very popular stemming algorithm for the English language is the Porter2 English Stemmer [9], based on rules and exceptions.

3.1.2 N-gram language models

One of the major contributions of NLP research are language models, i.e. *'models that predict the probability distribution of linguistic expressions'*. These can be, for example, single characters, words, or sequences of words [10, p. 861].

Estimating the probabilities of every possible sequence of linguistic expressions is not feasible, as they are infinite, so the most used model is the N-gram model, which simplifies the problem through some independence assumptions.

A model is called a N-gram model when it is based on sequences of length N, but there are specific terms for sequences of unit length ('unigrams'), for sequences of length equal to two ('bigrams'), and for sequences of length equal to three ('trigrams').

Formally, an N-gram model is a Markovian process, i.e. a probabilistic model that respects Markov's assumption of dependence on the present state only; in the specific case of N-gram language model the present state is determined by the N - 1 linguistic expressions prior to the one under examination.

The table 3.2 shows the Markov assumptions for the unigram, bigram and trigram models.

Unigram	$P(w_i \vert w_{1:i-1}) = P(w_i)$
Bigram	$P(w_i \vert w_{1:i-1}) = P(w_i \vert w_{i-1})$
Trigram	$P(w_i \vert w_{1:i-1}) = P(w_i \vert w_{i-2:i-1})$

Table 3.1: Markov's assumptions applied to the N-gram models.

In the case of bigram word models, for example, we have:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{1:i-1}) = \prod_{i=1}^{N} P(w_i|w_{i-1})$$

Therefore, considering the sentence θ = 'I eat an apple', with a bigram model, we get:

$$P(\theta) = P(I)P(eat|I)P(one|eat)P(apple|one)$$

Given a text or set of texts, its language pattern can be estimated by estimating the probabilities of the N-grams through their normalized frequencies.

The table 3.2 shows the estimates for the unigram, bigram and trigram models.

Unigrams	$P(w_i) = \frac{count(w_i)}{\sum_w count(w)}$
Bigrams	$P(w_i \vert w_{i-1}) = \frac{count(w_{i-1:i})}{\sum_w count(w_{i-1},w)}$
Trigrams	$P(w_i \vert w_{i-2:i-1}) = \frac{count(w_{i-2:i})}{\sum_w count(w_{i-2:i-1},w)}$

Table 3.2: Estimation of probabilities for N-gram models

Wanting to estimate, for example, the conditional probability $P(apple|one)$

for a bigram model, one would have to normalize the number of occurrences of the bigram 'one apple' by the number of occurrences of 'one'.

Estimating probabilities using frequencies has its problems.

Assuming, for example, that a certain N-gram never appears in the corpus, its probability will be equal to zero. To mitigate this problem, a *smoothing* algorithm can be applied; *smoothing by linear interpolation*, for example, is a *backoff* model that combines unigram, bigram and trigram models.

Another problem that can arise is that of words not present in the vocabulary. One method that can be used to handle them is to add a new word to the vocabulary for unknown words - usually <*UNK*> - and, before building the model, to replace all the first occurrences of each word precisely with <*UNK*> [10, p. 863-864].

3.1.3 Hidden Markov Models

Like N-gram models, Hidden Markov Models (HMM) are Markov process models, with the difference that HMM have hidden states (properties) that cause observable events.

To understand the logic behind HMM, we can analyze how they can be used for POS (*Part-of-Speech*) tagging, an activity which consists in annotating words with the grammar role they cover within sentences (generally used in the field of NLP to reduce ambiguity due to homographies and polysemies). Based on the grammatical characteristics that depend on the language, a POS tagger will be able to classify a word as a noun, verb, adjective, article, etc.; as singular or plural; and so on. The Penn Treebank, an annotated corpus of more than 4.5 million words in American English, for example, uses 36 tags [29].

In the case of POS tagging, the hidden states are the tags, while the observable events are the words, and the goal is therefore to find the most probable sequence of tags that generated the sequence of words (*decoding*). If we have the probabilities of emission of the words given the tags, and the probabilities of transition between tags, it is possible to find the optimal sequence by applying the Viterbi algorithm, a dynamic programming method whose details are discussed in [30].

3.1.4 Probabilistic Context-Free Grammars

Another widely used model in the field of NLP derives from Chomsky's studies on grammars. Formally, a *grammar* is a *"set of rules which define a language as a set of allowed strings of words"* [10, p. 890]. Indeed, assuming that sentences have a syntactic tree structure, new sentences can be generated by defining rewriting rules on the basis of the different lexical and syntactic categories.

Among the various possible grammars, Probabilistic Context-Free Grammars (PCFG) represent a good trade-off between expressiveness and efficiency and are often used in NLP. As with the context-free grammars defined by Chomsky, each rule has only one non-terminal symbol on the left-hand side, while both terminal and non-terminal symbols can be found on the right-hand side; in addition to that, probabilistic grammars assign a probability to each string. Clearly, the symbols representing the lexical categories (the POS tags mentioned above) and those representing the syntactic categories (for example *NP* and *VP*, respectively noun phrase and verb phrase) constitute the non-terminal symbols , while the dictionary words are the terminal symbols.

An example of a PCFG production rule could be the following [10, p. 891]:

$$S \rightarrow NP\ VP\ [0.70] \mid S\ Conj\ S\ [0.30]$$

According to this rule, a sentence can be composed of a noun phrase followed by a verb phrase (with 70% probability), or it can be composed of two sentences joined by a conjunction (with 30% probability).

For the construction of the rules, the approach is generally to learn them from the data using an annotated corpus such as the aforementioned Penn Treebank.

The utility of PCFG depends on the applications: they can be useful for tasks such as *parsing*, i.e. for analyzing texts in order to obtain the structure of sentences, but they have the disadvantage, compared to the N-gram models, that they don't distinguish between non-terminal symbols belonging to the same lexical category.

3.2 Information Retrieval

Even though information retrieval is more about accessing data than it is about analytics, it does propose some basic ideas that can be used for the latter as well.

In order to satisfy the user's information need, expressed through a *query*, i.e. a sequence of keywords, Information Retrieval methods sort the documents of the collection according to their relevance.

3.2.1 Vector Space Model

One of the most common approaches to quantify the relevance of a document is to measure the similarity between the query and the document itself. In the Vector Space Retrieval model, both the document and the query - which can be seen as a short document - are represented as vectors of weights [7]. Once the vectors $\vec{q}$ and $\vec{d}$ have been defined, the *cosine similarity* can be used to measure their similarity, which corresponds to the scalar product of the normalized vectors:

$$similarity(\vec{q}, \vec{d}) = cos(\theta) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|} \tag{3.1}$$

where θ is the angle between $\vec{q}$ and $\vec{d}$ [3, p. 121].

3.2.2 Bag-of-Words

As far as the vectorization of documents is concerned, refinements of the Bag-of-Words (BoW) model are typically used in the context of Information

Retrieval, whereby a document can be represented as a multiset (*bag*) of the terms cointained in the document, ignoring their order. Therefore, considering each term as a dimension in the vector space, a weight can be associated to each of them on the basis of various factors. In the basic version of the BoW model, the weight associated with a term is equal to the *term frequency* (TF), i.e. the number of occurrences of that term in the document [2, p. 89].

3.2.3 TF-IDF

The major flaw of the BoW model is that it does not take into account the importance of words: words like 'the' or 'of' (the so-called '*stop words*') carry almost zero information, since they are very common.

If we consider the distribution of words in any natural language, this can be approximated by a Zipfian distribution (Zipf's law), whereby the frequency of a word is inversely proportional to its rank. In fact, by drawing the rank-frequency graph of the words of a corpus, a *power law* type distribution is obtained, whereby there are a few very frequent words, a relatively small group of words with an average frequency, and many rare words [4]. In figure 3.1 we can see Zipf's law applied to Wikipedia in 30 different languages: the linear trend that emerges from the log-log scale graph is a clear indicator of a power distribution.

Going back to the original problem, the BoW model can be improved by taking this distribution into account, favoring the rarer terms. One of the most used methods is TF-IDF, which consists in multiplying the *term frequency* by an importance factor: the *inverse document frequency* (IDF):

$$w = tf \cdot idf = tf \cdot log\frac{N+1}{df} \tag{3.2}$$

where N is the number of documents and df is the *document frequency*, i.e. the number of documents containing the term [2, p. 99].

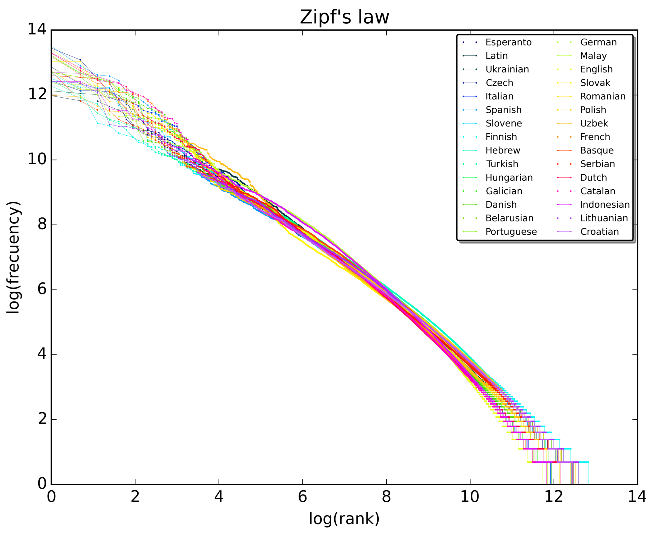


Figure 3.1: Rank-frequency distribution of the first 10 million words on Wikipedia [11].

3.2.4 Sub-linear transformations of TF

Another aspect to consider when representing a document is that the *term frequency* differences are much more significant in the lower ranges than in the higher ones: for example, it can be very important to know if the word ‘Covid’ occurs at least once; while knowing whether it appears 50 or 51 times doesn’t make much difference.

For this reason, a sub-linear transformation can be applied to the *term frequency*. Given f equal to the raw number of occurrences of the term in the document, some options can be:

- logarithmic transformation: $tf = log(1 + f)$
- double logarithmic transformation: $tf = log(1 + log(1 + f))$
- BM25 TF: $tf = \frac{(k+1)f}{f+k}$

Compared to the logarithmic transformations, the BM25 TF transformation has the advantage of being able to be controlled through the parameter

k: if it is equal to 0, a binary *term frequency* will be obtained (and, consequently, a bit vector representation), while as k increases we will tend towards a linear trend, with *k+1* as the upper limit [6].

3.2.5 Pivoted length normalization

Even applying a sub-linear transformation of the *term frequency*, there can be a bias towards long documents, as they contain more words. This bias may or may not be justified depending on the specific application. In the case of Information Retrieval, it may be useful to penalize (with caution) longer documents, because they may cover more topics and therefore require a greater effort from the user in order to satisfy his information needs.

One of the methods that can be used is the *pivoted length normalization*, which uses the average length of the documents (pivot) as a reference, penalizing the longer ones with respect to the pivot and rewarding the shorter ones. In practice, the following normalizer can be used:

$$1 - b + b\frac{|d|}{pivot}$$

where $|d|$ is the length of the document and b is a parameter between 0 and 1 that controls the degree of normalization [5].

3.2.6 Maximum TF normalization

Another approach that is sometimes used as an alternative to the sub-linear transformation combined with *pivoted length normalization* is *maximum TF normalization.* The goal of this technique is to mitigate the effect of words that are repeated over and over in a document. The normalization that is applied is the following:

$$tf = a + (1 - a)\frac{f}{fmax} \tag{3.3}$$

where *fmax* is the maximum raw frequency in the document considering all the terms and a is a *smoothing* parameter between 0 and 1 (when it is equal to 0.5, it is usually called *'Augmented Frequency'*) [3, p. 127].

3.3 Machine Learning

Another field that has generated some interesting techniques for Text Analysis is the field of Machine Learning (ML). The objective of Machine Learning is similar to the one of Data Mining, i.e. the recognition of patterns in data, but with the first expression we mean a specific approach, based on the automatic training of models through examples (inferential learning).

Therefore at the basis of learning there is a dataset, made up of n samples (also called 'observations'), described by a vector of attributes (also called 'variables' or 'features'). Attributes can be of various types: nominal/categorical, binary, ordinal, numeric.

The purpose of Machine Learning consists in the estimation of an output (also called 'target') starting from the values of the attributes. If the dataset that is used for model training also contains sample targets (also called 'labels'), the training is called *supervised*, otherwise it is *unsupervised.* The two main problems that can be solved with a supervised approach are regression (when the target is continuous) and classification (when the target is discrete), while the clustering problem (i.e. the partition of the dataset into groups such that the elements in the same group are similar and those in different groups are dissimilar) typically requires an unsupervised approach.

As for the supervised case, suppose we observe p variables $\vec{x}$ together with a target variable y. The latter is a function of $\vec{x}$:

$$y = f(\vec{x}) + \epsilon \tag{3.4}$$

where ϵ is the random error (independent of $\vec{x}$ and with zero mean) [23, pp. 28-29].

In this case, the training of the Machine Learning model consists in estimating the parameters of a function $\hat{f}$ such that $\forall(\vec{x}, y)$:

$$y \approx \hat{f}(\vec{x}) \tag{3.5}$$

With model evaluation, we can determine how close $\hat{y} = \hat{f}(x)$ is to y. To avoid *bias*, the evaluation cannot be performed using samples which are not seen in the training phase, so the dataset is initially divided into three subsets:

- *training set* (for the training phase);
- *validation set* (for hyperparameter calibration);
- *test set* (for evaluation).

If we have a high error on the training set, it means that the model is too simple and it *underfits*; if instead there is a low error on the train set but a high error on the test set, it means that the model is not able to generalize and it *overfits*.

3.3.1 K-Nearest Neighbours

K-Nearest Neighbors (K-NN) [24, pp. 39-42] is one of the simplest classification models. It consists in representing the samples as points in a p-dimensional space, and calculating the distance between the samples and the sample under examination, according to a distance metric, such as the Euclidean distance or the Manhattan distance.

Once the k closest samples have been identified, the output class is simply the most common class of those samples.

To find a good value of k, experiments can be performed by incrementing k from a value of 1 and selecting the value of k that results in the lowest error rate.

3.3.2 Support Vector Machine

Another one of the most used models in the field of text classification is the *Support Vector Machine* (SVM) model [24, pp. 340-353].

In the SVM model, as in the K-NN model, each observation is represented as a point in space. The goal is to find an optimal separation hyperplane between samples of different classes. Assuming we have a vector space of dimension p, a hyperplane of this space can be described by the following linear equation:

$$w_0 + \sum_{j=1}^{p} w_j x_j = 0 \tag{3.6}$$

Supposing we have two classes, with $y_i \in \{-1, 1\}$, we can define a separating hyperplane as a hyperplane for which the following property holds for each $i = 1...n$:

$$y_i(w_0 + \sum_{j=1}^{p} w_j x_{ij}) > 0 \tag{3.7}$$

Concretely, this constraint imposes that the points corresponding to the negative and positive class lie respectively below and above the hyperplane.

The problem with this definition is that there can be infinite hyperplanes that fulfill this constraint. Obviously, the most reasonable choice is a hyperplane that is as far away from the train set observations as possible. In the *Maximal Margin Classifier*, this is done by looking for the parameters $\beta_0, ...\beta_p$ which maximize a variable M called *margin*, with the constraint that all observations are at least at a distance M from the hyperplane:

$$y_i(w_0 + \sum_{j=1}^{p} w_j x_{ij}) > M \tag{3.8}$$

However, this optimization problem may not have a solution. To solve this issue, and to avoid overfitting due to *outliers*, some violations can be accepted during training, effectively defining a *soft margin.* The resulting model is a *Soft Margin Classifier*, also known as *Support Vector Classifier* (SVC).

To loosen the constraint, positive *slack variables* $\epsilon_1...\epsilon_n$ are added to the optimization problem, which, if greater than zero, indicate that the observa-

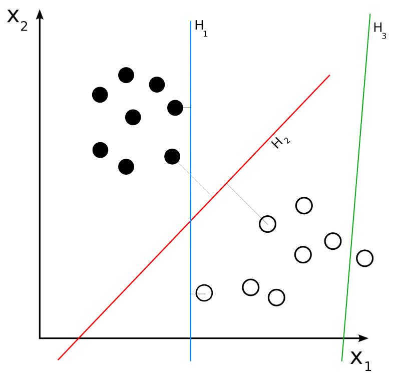


Figure 3.2: Comparison of 2D hyperplanes.. H_3 does not separate the two classes. H_1 separates the classes with a small margin, while H_2 is the separating hyperplane with maximum margin [27].

tion is on the wrong side of the margin (these points and the ones exactly on the margin are called *support vectors*):

$$y_i(w_0 + \sum_{j=1}^{p} w_j x_{ij}) > M(1 - \epsilon_i) \tag{3.9}$$

To control the number and severity of violations, the sum of *slack variables* is required to be less than a C hyperparameter.

Using sophisticated techniques, one can reformulate the optimization problem so that it can be solved simply by calculating the scalar products between the observations.

The *Support Vector Machines* are the generalization of the SVC: instead of the dot product, in fact, other similarity *kernel* functions can be used. Indeed, if the data is not linearly separable, a nonlinear *decision boundary* can be obtained by mapping the points to a multidimensional space, using, for example, the polynomial *kernel*.

Since these models are of binary type, in the presence of an arbitrary number $k \geq 2$ of classes it is necessary to train more classifiers, adopting one of the following approaches:

- *One-Versus-One* (OVO), implementing a voting system with $\frac{k(k-1)}{2}$

classifiers;

- *One-Versus-All* (OVA), implementing only k classifiers and searching for the class with the highest confidence level.

3.3.3 Bayesian models

Bayesian models are probabilistic models, based on the assumption that reality is governed by probability distributions.

The simplest Bayesian model is the *Naive Bayes* (NB) model, a *Maximum Posterior* (MAP) classifier.

The goal is to find the class for which the conditional probability is maximum [25, pp. 177-178]:

$$k^* = \operatorname*{argmax}_{k_i \in K} P(k_i|x_t) \tag{3.10}$$

where K is the set of classes and x_t is a new sample.

By Bayes' theorem, we have:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \tag{3.11}$$

where:

- $P(A|B)$ is the *a posteriori probability*;
- $P(B|A)$ is the *likelihood*;
- $P(A)$ is the *a priori probability.*

Therefore we have:

$$k^* = \operatorname*{argmax}_{k_i \in K} \frac{P(k_i)P(x_t|k_i)}{P(x_t)} = \operatorname*{argmax}_{k_i \in K} P(k_i)P(x_t|k_i) \tag{3.12}$$

Assuming that the variables $x_{j,t}$ are independent - and therefore their joint probability is equal to the product of the individual probabilities - we have:

$$k^* = \underset{k_i \in K}{\operatorname{argmax}} P(k_i) \prod_{j=1}^{p} P(x_{j,t}|k_i) \tag{3.13}$$

In the case of qualitative variables, these probabilities can be easily estimated from the dataset by calculating the relative frequencies, while in the case of quantitative variables it can be assumed that each variable follows a different distribution whose parameters can be estimated using *Maximum Likelihood Estimation* (MLE).

The *Gaussian Naive Bayes* model, for example, assumes that the variables are governed by normal distributions, and it estimates the mean and variance for each class.

3.3.4 Artificial Neural Networks

Artificial Neural Networks (ANN) are regression and classification models inspired by neuroscience.

The basic unit of ANNs is the neuron, which performs two fundamental operations: a weighted sum of the inputs (2.1) and the application of an activation function (2.2).

$$z = \sum_{i=0}^{n} w_i x_i \tag{3.14}$$

$$a = g(z) \tag{3.15}$$

where x is the vector of inputs (with a bias input $x_0 = 1$), w is the vector of weights associated with the inputs, and g is a nonlinear activation function. Figure 3.3 shows some of the most used activation functions.

There are two basic types of [10, p.729] networks:

- *feed-forward* networks, whose connections between units are in one direction only;

- recurrent networks (RNN), whose outputs can become inputs, thus supporting short-term memory.

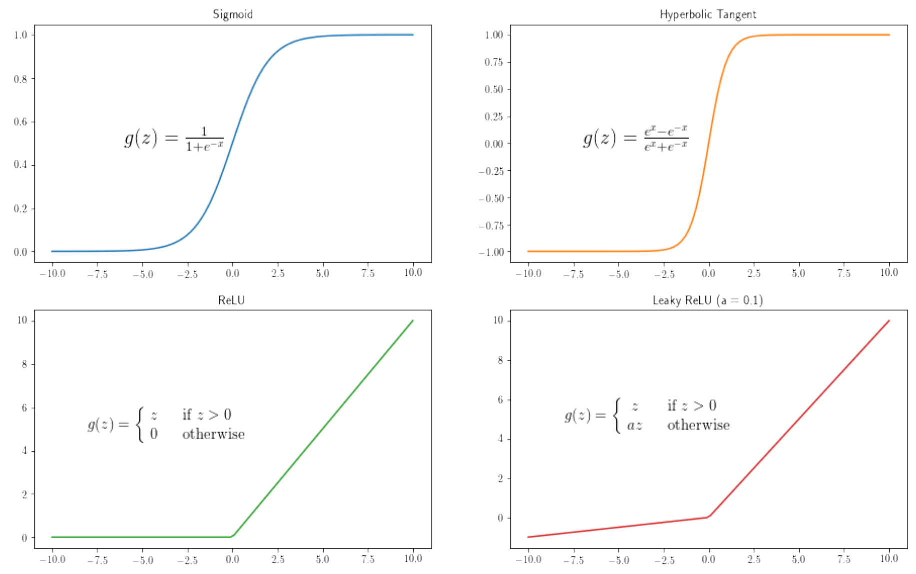


Figure 3.3: Commony used activation functions: Sigmoid, Hyperbolic tangent, ReLU, and Leaky ReLU.

The choice of network type usually depends on the task: for example, feed-forward networks are suitable for simple classification problems, while RNNs can be used to generate sequential data.

The base class of *feed-forward* networks is that of *Multilayer Perceptron* (MLP), in which there are several layers in sequence, each composed of units working in parallel, and the outputs of each layer become the inputs of the next level.

All units other than output units are called *hidden units*, and, if every unit in one level is connected to all other units in the next level, the network is said to be *fully connected*.

The width of the network is defined as the maximum number of *hidden units* in a single level, while the depth of the network is defined as the total number of levels; in the case of neural networks with many levels, we call it *Deep Learning*.

The learning consists in the estimation of the weights of the network, in order to determine a composite function that solves the given problem.

To do this, we try to optimize a given objective function.

In the case of regression, the mean squared error (*Mean Squared Error*, MSE) can be used as a cost function.

In the case of multiclass classification, on the other hand, one may wish to maximize the *likelihood* of the target class for each sample [26, p. 209]:

$$P(y|x,w) = \prod_{n=1}^{N}\prod_{k=1}^{K}\hat{y}_{nk}^{y_{nk}} \tag{3.16}$$

where y_{nk} is a bit equal to 1 only if class k is associated with sample n, and $\hat{y}_{nk}$ is the probability of class k for sample n estimated via the network.

Since the product of a large number of probabilities tends to zero and since maximization problems are generally reduced to minimization problems, the goal becomes the minimization of the *Negative Log Likelihood* (NLL), which corresponds to *Multiclass Cross-Entropy*:

$$J(w) = -lnP(y|x,w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_{nk}ln(\hat{y}_{nk}) \tag{3.17}$$

To minimize the cost function J, variants of the iterative Gradient Descent algorithm are typically used. The latter allows to find the local minima of the function starting from a random point and following each time the direction opposite to that indicated by the gradient, until convergence [10, p.719]:

$$w_l = w_l - \alpha\frac{\delta J(w)}{\delta w_l} \tag{3.18}$$

where l is the level, and α is the *learning rate* which determines the length of the steps.

To calculate the gradient at each level, the *backpropagation* method is used, based on the rule of the chain of derivatives, starting from the output level and going backwards.

In addition to Gradient Descent, other optimizers potentially faster in achieving convergence can be used, such as, for example, *Stochastic Gradient Descent* (SDG) which trains the network with a subset (*minibatch*) of samples at a time [10, p.720].

3.3.5 Word2Vec

Word2Vec is a set of models to generate word embeddings (mapping of words into vectors of real numbers) developed by Google in 2013 [18]. Embeddings generated by Word2Vec allow semantic analysis, since similar words have similar embeddings. Moreover, taking into account different types of similarity, these embeddings can be used in vector arithmetic: for example, vec(*Germany*) + vec(*capital*) is close to vec(*Berlin*) [19] [20]. Instead of creating a sparse representation as one-hot encoding models, Word2vec creates a dense representation, assuming that each element of the vector is associated to a concept (e.g. gender, nationality, and so on).

Word2Vec learning is based on the Distributional Hypothesis formulated by the linguist John Rupert Firth: "You shall know a word by the company it keeps" [31].

Word2Vec includes two models, both based on two-layer networks:

- Skip-Gram (SG), which predicts the context given a word as input;
- Continuous Bag-of-Words (CBOW), which predicts a word given the context as input.

Skip-Gram works by training a network to solve the task of predicting context words given an input word (center word). The network computes probabilities related to how likely it is find each vocabulary word nearby the center word, i.e. $p(o|c)$. To be more precise, SG is a network with a single hidden layer (with as many units as the chosen number of dimensions) which considers as training samples the word pairs (represented as one-hot encodings). The network tries to maximize the probability of the words being in the same context, which is defined using the softmax function to get values between 0 and 1 for each word that together sum up to 1:

$$p(o|c) = \frac{e^{u_o^T v_c}}{\sum_{w=1}^{W} e^{u_w^T v_c}} \tag{3.19}$$

where W is the number of words in the vocabulary, u_w^T is the context embedding of word w and v_w is the context embedding of word w 3.4.

Actually, the goal is just to learn the weights of the hidden layer, which are the word vectors we are looking for.

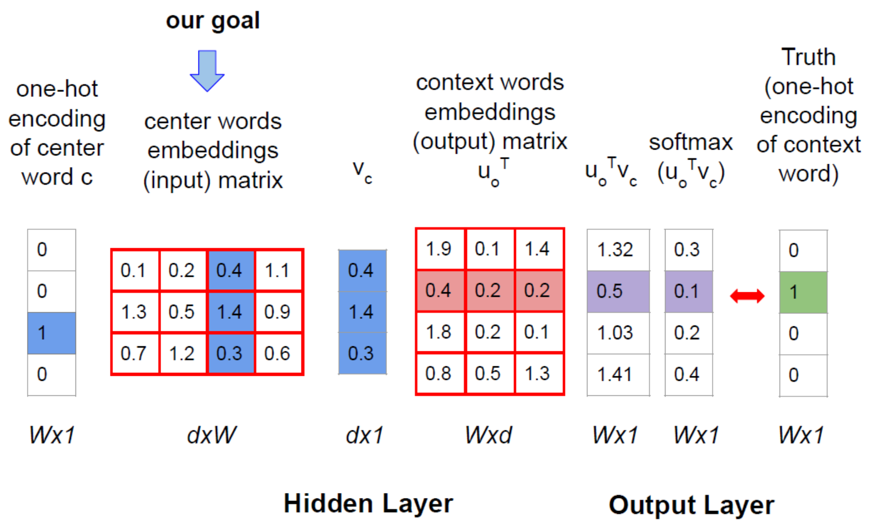


Figure 3.4: Skip-Gram model

Continuous Bag-of-Words is similar to Skip-Gram, but, since it uses many context words as inputs, it averages their embeddings in the hidden layer. SG is a little bit slower than CBOW, because it trains the network with a single context word at a time, but it works better with uncommon words.

Chapter 4

State of the Art of Virtual Assistants for Domotic Appliance Control

This chapter will introduce the core concepts and motivations of Home Automation and Virtual Assistants, dealing with available consumer hubs on the market and the most popular open-source alternative: Home Assistant.

4.1 Home Automation

Home Automation, also referred to as *domotics* from the Latin word *domus* meaning *house*, is the field of building systems capable of monitoring and controlling home appliances in order to improve quality of life.

These systems can be classified according to several parameters, such as topology and communication channel.

Depending on topology, there can be *centralized* systems with a hub or *distributed* systems with many smart devices. Each structure has its advantages and disadvantages: centralized systems are easier to control and manage, but distributed systems are more robust since there is not a single point of failure. In reality, the most common solution is to create a system which combines the advantages of each topology, by having smart devices which can *also* be integrated into a central hub.

Depending on communication channel, there can be wired systems, such as the most common KNX standard implementation which connects devices through a twisted pair bus, or *wireless* systems, which connect devices through a radio-frequency standard, such as Wi-Fi, Zigbee, Bluetooth or Z-Wave. Wired system have the advantage of being generally faster and almost completely immune to interference, but are more expensive and typically require a technician intervention for installation and upgrade. However, it is important to notice that there is a great variance even between wired protocols: Bluetooth is slower than Wi-Fi; Wi-Fi is slower and requires more power consumption than Zigbee and Z-Wave; all devices except Wi-Fi devices require an hub in order to be connected to the Internet and become an actual Internet-of-Things (IoT) device.

As far as appliances are concerned, the most common ones are:

- lightning devices, such as light bulbs, led strips and light switches;
- heating, ventilation and air conditioning (HVAC) devices, such as thermostats, thermostatic valves and fans;
- security devices, such as alarms, locks, contact sensors, motion sensors and security cameras;
- environmental sensors, such as temperature and humidity sensors, air quality sensors, smoke and gas leakage detectors;
- energy devices, such as plugs and meters.

Home Automation systems can be helpful in improving safety, energy efficiency, making home management easier, especially for elderly people and people with disabilities, thus increasing their independence.

4.2 Virtual Assistants

Interaction between humans and machines has evolved significantly over time and different styles of interaction have emerged to improve the efficacy of the dialogue, taking into account the features of the specific application, and to eventually meet the user's needs.

Between these styles, Natural Language appears to be the most attractive, since it is usually more immediate: for example, it does not require users to remember commands (as the Command Line Interface style) nor to navigate through long sequences of options (as the Menu style) [32]. However, as we have seen in 3.1, Natural Language is ambiguous by nature, so an NLP-based interaction is most effective in limited domains, where context can help in generating precise instructions for the computer.

Virtual Assistants (VA) are software which take as input a text/voice request and execute the requested task. While the most basic VAs make use of the command-line interaction style inheriting its lack of flexibility, the most advanced ones can be quite complex.

4.2.1 Components

The main component of a Virtual Assistant is the Intent Recognition component. If the input is in vocal form, a Speech Recognition component is also needed. The Speech Synthesis component is optional in each case.

A Voice Assistant is a special kind of VA which is composed of all three components.

Speech Recognition

The Automatic Speech Recognition (ASR) component is in charge of Speech-to-Text (STT): it translates the spoken language command (an acoustic signal) into a text command.

Most speech recognizers are made up of four components [10, pp. 913-918]:

- a signal processor, an Analog-to-Digital Converter (ADC) which samples and quantizes the acoustic signal and computes relevant frequency *features* over time slices called *frames*;
- a phone model, an Hidden Markov Model which describes a *phone* (distinct speech sound) as three states: onset, middle and end;
- a word-pronunciation model, a transition model whose states are the phone models;
- a language model, typically a N-gram model.

The probabilities of the phone models and the word-pronunciation models are acquired from a corpus of speech, while those of the language model are acquired from a corpus of written text.

With these components, a speech recognizer can compute the most likely command through Bayes' rule:

$$\underset{word_{1:t}}{\operatorname{argmax}} P(word_{1:t}|sound_{1:t}) = \underset{word_{1:t}}{\operatorname{argmax}} P(sound_{1:t}|word_{1:t})P(word_{1:t}) \quad (4.1)$$

In the last few years, a new end-to-end approach based on Transformers has emerged, which is more flexible than HMM. While the signal processing step is similar, the acoustic and language models are replaced by encoders and decoders made up of Recurrent Neural Networks (RNN) with an attention mechanism [48]. An example of open-source transformer-based ASR system is OpenAI's Whisper, trained on 680,000 hours of multilingual supervised data, which approaches human accuracy [49].

Intent Recognition

The Intent Recognition component is in charge of translating the textual command into a command (*Intent*) which matches the user intentions and is understandable by the machine.

The two main approaches which can be adopted to reach this goal are:

- the rule-based approach, which makes use of regular expressions or equivalent matching techniques;
- the Machine Learning approach, which treats the Intent Recognition problem as a classification problem.

In addition to the Intent, this component also extracts additional information from the input: the slot-filling process identifies relevant slot labels (such as *area*) and slot values (such as *kitchen*).

Speech Synthesis

The Speech Synthesis component is in charge of Text-to-Speech (TTS): it generates human speech from text, and it can be used to provide a feedback about the operation or the result of the query.

To accomplish this task, the engine pre-processes the text (for example, normalizing numbers and abbreviations) and assigns phonetic transcriptions to the tokens. By using these annotations, the synthesizer can generate the speech by using pre-recorded samples.

4.3 Proprietary Voice Assistants for Home Control

The most widespread VAs with home control capabilities are Amazon Alexa, Google Assistant and Siri.

Amazon Alexa, first released in 2014, is embedded on a wide range of smart devices, such as the popular smart speakers Amazon Echo Dot and Amazon Echo Plus. It can perform several basic tasks, which can be extended by installing applications called *Skills*.

Google Assistant, first released in 2016 as Android application, is now available on numerous other devices as well, such as the Google Nest speakers. The platform which is in charge of linking all Google's components for home control is called *Google Home*.

Siri, the VA released by Apple Inc. in 2011, is only available on Apple devices, such as iPad, iPhone, and HomePod speakers. The framework which allows users to use Siri for home control is called *HomeKit*.

As far as performance is concerned, a study conducted by Loup Ventures has compared Google's, Amazon's and Apple's VAs; in the *command* category, the most accurate VA appears to be Siri (85% accuracy), followed by Google Assistant (73% accuracy) and Amazon Alexa (68% accuracy) [59].

For each one of the VAs, access to the Internet is required for ASR. Once the smart speaker recognizes the *wake word*, it turns on a LED indicator and streams the voice command to the cloud in order to understand it and execute it, as shown in Figure 4.1.

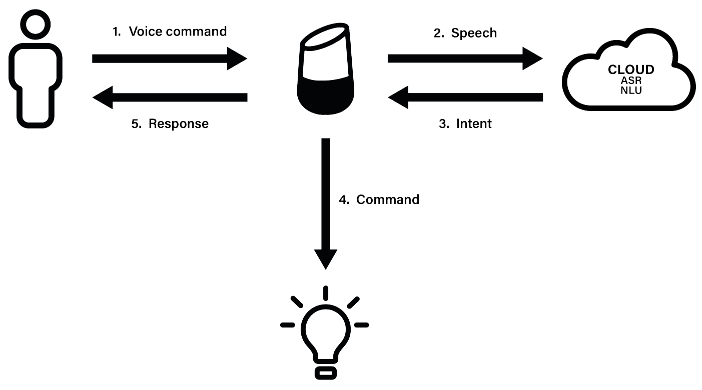


Figure 4.1: Cloud-based Voice Assistant architecture.

Several privacy concerns have emerged over the years. Amazon, Google and Apple have admitted in 2019 that human workers have been regularly listening to VAs recordings [60], and there have been numerous cases in which smart speakers have been accidentally triggered [61]. The companies have tackled the issue by establishing the possibility for the user to delete the recordings and to opt-out of recording retention.

Nevertheless, privacy is "the right of a user to determine the degree to which they are *willing* to share their personal information with others" [62], thus the user's perception, which is also influenced by trust, cannot be neglected.

4.4 An open-source alternative: Home Assistant

Home Assistant is an open-source platform which allows to build a home automation system and run it on a local server to preserve privacy.

It was first released in 2013 by Paulus Schoutsen, and it has now become one of the most active projects on GitHub.com, with more than 3000 contributors [36].

The exact number of users of Home Assistant cannot be known, but its developers estimate from usage reports that there are more than 200,000 active installation currently with an increasing trend 4.2.

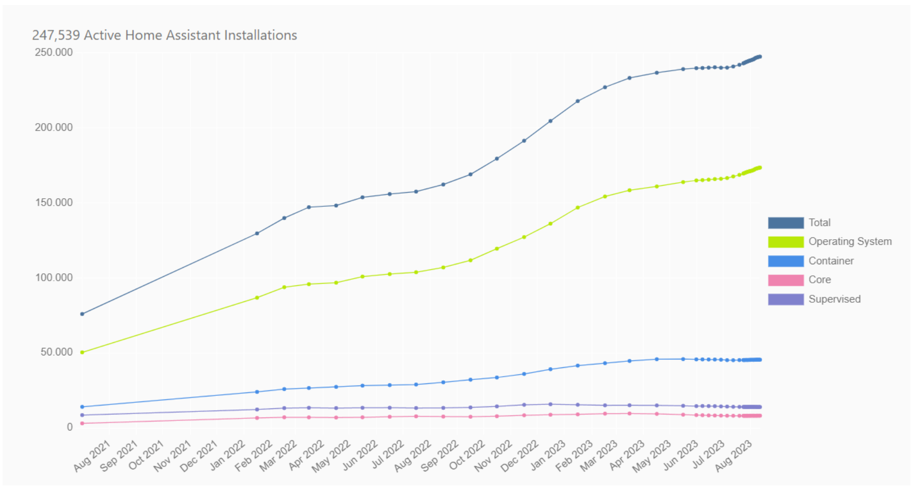


Figure 4.2: Home Assistant installations [34].

The most popular board for the system is the microprocessor Raspberry Pi, which is designed to be an always-on system, has a low energy consumption, a small form factor, and is cheaper than most computers with comparable specifications. Its latest version, Raspberry Pi 4, comes with four variants ranging from 1GB RAM to 8GB RAM.

4.4.1 Core concepts

Home Assistant has a layered architecture:

- *Home Assistant Core* is the main application, which is in charge of monitoring events and entity state changes, calling services, handling the UI, and so on;
- *Home Assistant Supervisor* is the layer in charge of running and updating Core and Add-ons and managing backups;
- *Home Assistant Operating System* (HAOS, formerly known as HassIO) is an operating system built on Linux specifically to run Home Assistant with minimal effort.

Home Assistant Core is built incrementally on *integrations*, which provide devices and services. For example, the *light* integration allows to track light bulbs and exposes services to control them (e.g. turn on/off, change color, and so on). There are four main types of integrations:

- integrations which define a domain (e.g. *light*, *fan*, *lock*);
- integrations which provide devices interacting with external platforms (e.g. *MQTT*);
- integrations which represent virtual data (e.g. *input_boolean*);
- integrations which provide sequences of actions (e.g. *automation*).

All relevant information for the end-user is in the form of *entities*, whose states (e.g. *on*) and attributes (e.g. *last changed*) can be used to create YAML scripts and automations (scripts with specific triggers). Internally, the system stores information about devices, entities, and user-defined *areas* (location of devices) in specific registries. Home Assistant Core currently includes more than 2000 built-in integrations, and can also be extended with Python-written custom integrations.

4.4.2 Assist

Home Assistant provides a Virtual Assistant called Assist (released in January 2023 for the "Year of the Voice" project [51]) capable of accessing device information and calling services, which lets users input text commands or voice commands. There are several ways to input voice commands:

- through a computer microphone accessed by the browser;
- through an Android/iOS smartphone microphone accessed by the Home Assistant app;
- through a WearOS smartwatch microphone;
- through ESP32 boards equipped with a microphone (self-built or pre-assembled as the M5Stack Atom Echo)

The idea is to have multiple *voice satellites* which can be activated (currently pressing a button, in the future, according to developers, through wake word detection) in several areas of the house where they can be useful.

The Assist Pipeline is in charge of linking the STT component, the Intent Recognition component, the Intent execution component, and the TTS component 4.3. Each part, except the Intent execution component, is customizable by the end user: it is possible to connect external voice services (e.g. Whisper as STT service and Piper as TTS service) through the Wyoming protocol, as well as using custom conversation agents.

Conversation integration

The default conversation agent provided by Home Assistant relies on an Intent Recognition engine called *hassil*. Hassil is a match-based engine: it

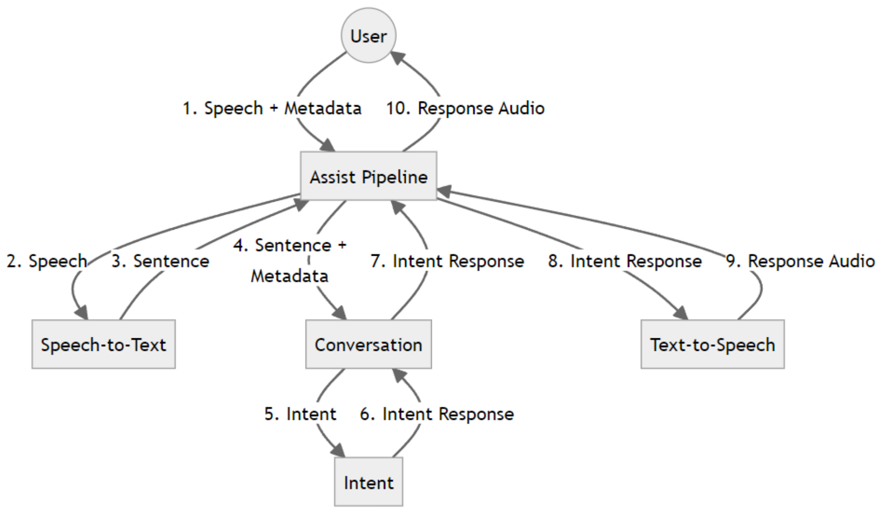


Figure 4.3: Assist Architecture [33].

makes use of YAML sentence templates provided by the Home Assistant community to recognize the intent and to fill the slots for that specific intent by extracting data using positional information. For example, a small fragment for the *HassTurnOn* intent is the following:

```
language: en
intents:
  HassTurnOn:
    data:
      - sentences:
        - "(open | raise) [the] {cover_classes:device_class}
          in <area>"
      slots:
        domain: cover
      response: cover_device_class
```

The parentheses and the pipe operator define alternative text parts and allow the sentence to start with *open* or *raise*; the square brackets introduce the optional text sequence *The*; the curly brackets introduce a placeholder for a slot value; the colon defines the possible input values and the corresponding output values for the slot (e.g. maps the cover class *doors* to the device class *door*); the angle brackets are linked to expansion rules defined elsewhere which in this case replace ⟨*area*⟩ with *[The]* {*area*}.

In addition to the intent, the slot labels and the slot values, the sentence template can also contain a response type to make it possible to generate a response which is as pertinent as possible to the request, which is especially useful for query requests such as "How many doors are open?". Response templates make use of the Jinja2 templating engine which manipulates the matched entities.

Home Assistant currently supports intents in table 4.4.2 for the Italian language.

Intent	Slots
HassTurnOn	name, area, domain, device_class
HassTurnOff	name, area, domain, device_class
HassGetState	name, area, domain, device_class, state
HassLightSet	name, area, brightness, color

The *name*, *area*, *domain* and *device_class* slots are self-explanatory, while the *state* slot is used to store question information which have to be used as filters for the query (e.g. if the input is "Is the kitchen light on?" the slot will contain the value 'on').

Not all slots need to be filled in order to handle the Intent: it is possible to identify a device/entity or a set of devices using: only their name; only their area; their area and name/domain/device class; only their domain; their domain and device class.

Table 4.4.2 shows the possible responses for each intent. The responses for *HassTurnOn*, *HassTurnOff* and *HassLightSet* are all variants of feedback about the action which has been carried out (e.g. "Turned on light" or "Brightness changed to 100%"). Unlike these, the responses for *HassGetState* are the result of a query:

- *one* is the response type for questions about the state of a single entity;
- *one_yesno* is the response type when asking whether a single entity is in a *specific* state;
- *all* is the response type when asking whether every entity is in a specific state;
- *any* is the response type when asking whether there is at least one entity in a specific state;
- *which* is the response type when asking to list the entities in a specific state;
- *how_many* is the response type when requesting the total count of entities in a specific state;

Intent	Responses
HassTurnOn	default, lights_area, fans_area, cover_device_class, cover_device_class_area
HassTurnOff	default, lights_area, fans_area, cover_device_class, cover_device_class_area
HassGetState	one, one_yesno, all, any, which, how_many
HassLightSet	brightness, brightness_area, color, color_area

Chapter 5

A new Intent Recognition engine: Converso

This chapter will introduce a novel Intent Recognition engine for the Italian Language which can be integrated into Home Assistant in order to control and monitor home appliances.

5.1 Design

The main requirements for this Voice Assistant are three:

- high accuracy ($> 60\%$);
- low delay (median value < 5s);
- privacy-preservation (complete control of data).

Assuming all household members and visitors can be trusted, the choice of a local platform like Home Assistant automatically fulfills the privacy requirement, since the system can work without connecting to any cloud service: no leakage, no data breaches and no misuse are possible without access to the local server. Figure 5.1 shows the architecture of the system.

However, the choice of a local system introduces a new requirement: low resources consumption (less than 8 GB RAM). In fact, even if the main

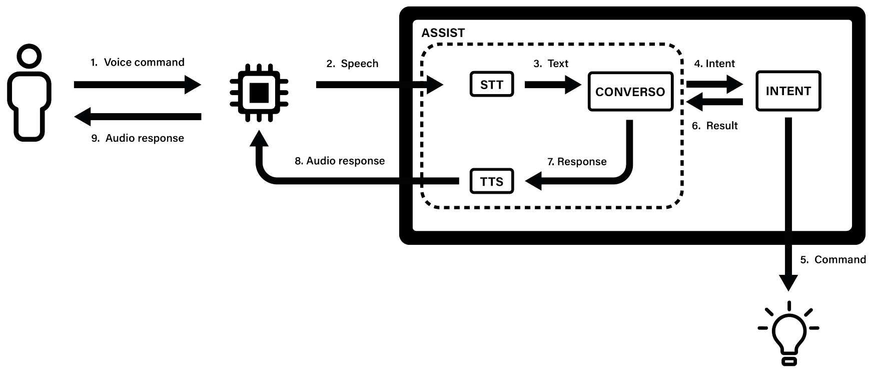
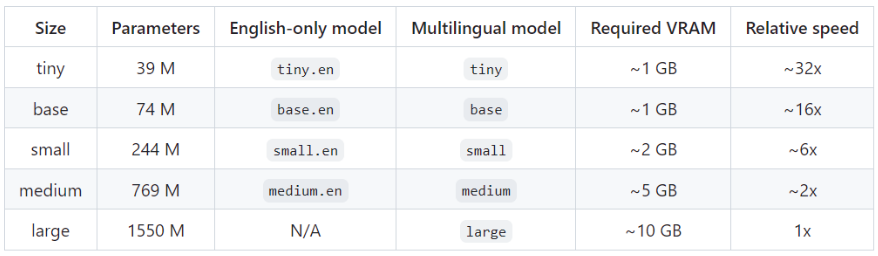


Figure 5.1: Converso-based Voice Assistant architecture.

bottleneck of the Voice pipeline is Speech Recognition, using low resource and faster STT models (such as Whisper *base* instead of Whisper *large* 5.2) results in less accurate output, which has to be handled and possibly corrected.



Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~32x
base	74 M	`base.en`	`base`	~1 GB	~16x
small	244 M	`small.en`	`small`	~2 GB	~6x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x

Figure 5.2: Whisper models [35].

5.1.1 Spelling Correction

Speech Recognition is prone to errors related to:

- environmental noise, due to music, people, loud appliances, and so on;
- bad sound quality, due to low quality/distant microphones;
- out-of-vocabulary words, for example due to the use of custom names;

- homophones, which are different words with similar phonemes;
- lack of pauses in speech, which can result in segmentation and coarticulation errors;
- pronunciation defects, for example due to dialect variations.

There exists many techniques to reduce these errors, among which the most flexible one is post-editing, which entails correcting the output text instead of the ASR algorithm [50]. In fact, these type of techniques can also be used for written commands which can contain spelling errors as well.

The main components of a post-editing algorithm are [55]:

1. error detection;
2. generation of candidates for correction;
3. selection of the best candidate.

Error detection

The error detection step is the simplest one. First, the input command is decomposed into unigrams and bigrams. Then, the algorithm identifies as possible errors the unigrams which are not included in the domain vocabulary, as well as bigrams whose probability is below a certain threshold γ.

The domain vocabulary is defined using all words which appear in the synthetic dataset defined in subsection 5.1.2, extended with user-defined custom entities' tokens.

The probability of each bigram is accessed from a pre-computed dataset derived from bigrams frequency lists extracted from the Italian WaCky corpus, which was built using the Italian Wikipedia and a 2 billion word corpus built by crawling .it web pages [54]. In order to reduce RAM usage, only bigrams with frequency greater than 10 are taken into account in the pre-computed dataset, resulting in 6,862,893 bigrams.

If the bigram is not included in the dataset, there are two cases: if the bigram is part of a custom name, the algorithm returns the maximum probability computed among all bigram probabilities; if the bigram is not part of a custom name, the algorithm returns the minimum probability instead.

Candidates generation

The generation of possible candidates for the correction of an error is performed by taking into account the trade-off between output accuracy and computational efficiency. For this reason, the possible candidates are generated in three different stages: if the best candidate in the current stage does not meet the requirements, then the algorithm continues the search by generating the candidates for the following stage.

Assuming that domain-specific unigrams and less edited N-grams have a greater probability, the candidates for each stage are:

1. 1-edit corrections which result in unigrams included in the domain vocabulary;
2. 2-edit corrections which result in unigrams included in the domain vocabulary and 1-edit corrections which result in unigrams included in the WaCky vocabulary;
3. 2-edit corrections which result in unigrams included in the WaCky vocabulary.

The possible corrections are generated by computing *close* strings, where closeness is defined through a metric called *edit distance*, which is the minimum number of operations required to transform one string into the other. In this work we use the Damerau-Levenshtein distance [57], which allows 4 types of operations:

- insertion of a character;
- deletion of a character;

- replacement of a character with another character;
- transposition of two adjacent characters.

The characters we consider are all the letters of the Latin alphabet plus the space character, which is useful to correct segmentation errors.

Finally, candidates which contain unigrams not included in the WaCky-derived unigram frequency list (containing the top 352,142 unigrams) are filtered out.

Candidate selection

The best candidate at each stage is selected using N-gram language models, in order to make use of context information. As seen in section 3.1.2, using a bigram model, the probability of a sentence can be computed as a product of conditional probabilities:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{1:i-1}) = \prod_{i=1}^{N} P(w_i|w_{i-1})$$

Each conditional probability can be estimated through frequency normalization:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1:i})}{\sum_w count(w_{i-1}, w)}$$

In this application, $count(w_{i-1:i})$ is the frequency of the bigram $w_{i-1}w_i$ which can be directly extracted from the WaCky dataset, while we approximate the denominator as the frequency of w_{i-1} for efficiency reasons.

As before, the frequency of custom N-grams is assumed to be the same as the most frequent N-grams, while the frequency of out-of-vocabulary N-grams is assumed to be the same as the less frequent N-gram, in order to avoid division by zero.

The conditional probability is also smoothed using linear interpolation:

$$P(w_i|w_{i-1}) = (1 - \lambda)P(w_i|w_{i-1}) + \lambda P(w_i)$$

Moreover, since the product of many probabilities can cause underflow, we transform probabilities into log-likelihoods:

$$P(w_{1:N}) = \prod_{i=1}^{N} P(w_i|w_{i-1}) = \sum_{i=1}^{N} log[P(w_i|w_{i-1})]$$

Thus:

$$P(w_{1:N}) \approx \sum_{i=1}^{N} log[(1-\lambda)\frac{count(w_{i-1}w_i)}{count(w_{i-1})} + \lambda\frac{count(w_i)}{\sum_{j=1}^{N} w_j}]$$

Therefore, the best candidate at each stage is the candidate which results in the greatest likelihood using bigrams.

Finally, assuming that a wrong correction is worse than no correction, the best candidate replaces the error only if a fixed improvement threshold β, which is proportional to the stage number and normalized by the sentence length, is reached.

5.1.2 Intent Recognition and Slot Labelling

The Intent Recognition problem can be seen as a multi-class classification problem, thus it can be faced adopting a Machine Learning approach. The possible intents taken into consideration are the following:

- *HassTurnOn*, which is used to turn on/open entities in the on/off domains;
- *HassTurnOff*, which is used to turn off/close entities in the on/off domains;
- *HassGetState*, which is used to ask questions about the state of entities in the on/off domains;
- *HassLightSet* which is used to change the brightness/color of a light;

- *HassClimateGetTemperature*, which is used to get the current temperature detected by a sensor of an appliance in the *climate* domain (such as a thermostat);

- *HassClimateSetTemperature*, which is used to modify the target temperature of a *climate* appliance.

As far as slot-filling is concerned, an hybrid approach is adopted. Slots whose value can be directly extracted from the text, such as area, name brightness and color, are handled in section 5.1.3. Machine Learning is used to classify all other slots: domain; device class; response; state.

The domain can be: *light*, *fan*, *cover*, *climate*, or *default*. The first three are considered on/off domains, and *default* means that the domain cannot be derived from the text, as in the case of turning on an entity by using its name. The domain slot has to be filled for the *HassTurnOn*, *HassTurnOff* and *HassGetState* intents.

The device class slot is currently used by Home Assistant only to disambiguate between different entities in the *cover* domain, and it can have one the following values: *door*; *window*; *blind*; *curtain*; *awning*; *garage*; *gate*; *shade*; *shutter*. The device class slot has to be filled for the *HassTurnOn*, *HassTurnOff* and *HassGetState* intents, but only if the domain is *cover*.

The response slot can have one of the following values depending on intent type:

- if *HassTurnOn*/*HassTurnOff*: *default*, *lights_area*, *fans_area*, *cover_area*, *cover*, *cover_device_class*, *cover_device_class_area*, *cover_garage*;

- if *HassGetState*: *one*, *one_yesno*, *any*, *all*, *which*, *how_many*;

- if *HassLightSet*: *brightness*, *brightness_area*, *color*, *color_area*;

- if *HassClimateGetTemperature*: *default*;

- if *HassClimateSetTemperature*: *default*.

The state slot, described in 4.4.2, can have value *on*, *off* or *none* (if the response is *one*). The state slot has to be filled only for the *HassGetState* intent.

Dataset generation

In absence of available data for the Italian language, a synthetic dataset with a large number of text commands is used as training dataset. The dataset is generated through a Feature Based Grammar, which takes into account the Syntactic Agreement, marking nouns and verbs as plural or singular; marking nouns as feminine or masculine; marking masculine nouns according to their article (in the Italian language, singular masculine nouns can be preceded by *il* or *lo*).

The grammar consists of productions inspired by the Home Assistant sentence templates [37].

The starting productions are the following:

```
S -> Intent
Intent -> HassTurnOn | HassTurnOff | HassGetState | HassLightSet |
          HassClimateGetTemperature | HassClimateSetTemperature
HassTurnOn -> Light_TurnOn | Fan_TurnOn | Cover_Open |
              Entity_TurnOn
HassTurnOff -> Light_TurnOff | Fan_TurnOff | Cover_Close |
               Entity_TurnOff
HassGetState -> Cover_Get | Entity_Get
HassLightSet -> Light_SetBrightness | Light_SetColor
HassClimateGetTemperature -> Climate_Get
HassClimateSetTemperature -> Climate_Set
```

The *Entity_TurnOn*, *Entity_TurnOff* and *Entity_Get* are related to sentences addressing the entity by name.

Productions related to the *cover* domain (an on/off domain) use the verbs *to open* and *to close* instead of *to turn on* and *to turn off* and differentiate between interior covers and exterior covers in order not to generate illogical sentences such as *“the garage in the kitchen”*.

Since the Italian language has a word order which is less strict than the English language, also valid permutations are added, such as:

```
Light_SetColor ->  Set Color WhereOf Onto ColorValue |
                   Set Onto ColorValue Color WhereOf
Light_SetColor ->  Change Color WhereOf Into ColorValue |
                   Change Into ColorValue Color WhereOf
WhereOf -> In Area | Of Area
```

Leaf values are Italian tokens, for example:

```
TempUnit -> ‘gradi’ | ‘gradi celsius’ | ‘gradi centigradi’
```

In order to automatically annotate each sentence with the intent, the slot values and the response type, the grammar tree is parsed and subtrees generated by filtering node labels are analyzed.

Finally, since the actual voice commands do not always contain stop-words (e.g. ‘turn kitchen lights on’) as users can prioritize speed in respect to grammar rules, the dataset is extended with commands without stop-words.

Data preprocessing

Each text command in the dataset is preprocessed through the following pipeline:

1. normalization;
2. tokenization;
3. stop-word removal;
4. feature extraction.

Since the last step entails using FastText, an open-source library for word embeddings, the first two steps are carried out consistently with the same algorithm which was used to tokenize FastText training data, included in Europarl Preprocessing Tools [39] [40]. This algorithm:

- lowercases all words;
- adds spaces around special characters (except spaces, dots, commas, apostrophes, backticks and dashes);
- keeps sequences of dots united;
- adds spaces around commas, except if the comma is between numbers;
- splits contractions (e.g. converts *isn't* to *isn' t*);
- tokenizes the text using a whitespace tokenizer.

In addition, it converts numbers from word form (e.g. *four*) to digit form (e.g. *4*).

Stop-word removal does not always result in better results with word-embeddings, as it can lead to loss of information, so models are tested with and without stopwords.

Feature extraction, as mentioned before, is performed using FastText. FastText includes models derived from Word2Vec, which enrich the word representations by taking into account the internal structure of words: each word is represented by the sum of the vector representations of its (bag of characters) N-grams. The model for the Italian language is trained on Common Crawl and Wikipedia using CBOW with a window size of 5 and character 5-grams [41].

In order to convert each command to an array of floating points, each token of the command is converted to a 100-dimensional vector and the average vector is computed.

To speed up training, each attribute is standardized by scaling to unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where μ is the mean value of the training samples and σ is the standard deviation of the training samples.

Finally, since datasets can be unbalanced and this could affect training, an undersampling technique is used to mitigate the problem by randomly removing samples in the majority class.

Model training

A different dataset is created for each label. The *Response* label is split into three different sub-labels (*ResponseHassTurn*, *ResponseLightSet* and *ResponseGetState*), since the possible responses depend on Intent. The datasets do not include examples whose target value for the label can be inferred from other target values:

- *Response* is always equal to *default* for *HassClimateGetTemperature* and *HassClimateSetTemperature*;
- *Domain* is always equal to *light* for *HassLightSet* and equal to *climate* for *HassClimateGetTemperature* and *HassClimateSetTemperature*;
- *DeviceClass* is only relevant if *Domain* is equal to *cover*;
- *State* is only relevant for *HassGetState* if *Response* is not equal to *one*

A splitting technique is applied to each dataset. Table 5.1 shows the splitting ratios adopted.

Train Set	Validation Set	Test Set
72%	18%	10%

Table 5.1: Datasets splitting ratios.

In addition to random shuffling, stratification is applied in order to have the same proportion of target values in each subset as the original dataset.

The proposed models are SVM (with linear and polynomial kernel), Gaussian Naive Bayes, KNN and Multi-Layer Perceptron (MLP). A grid search is performed in order to find the best hyperparameters for each model: for SVM, the possible values for C taken into account are 0.1, 1, 10, 100; with a degree of 2 or 3 in the case of polynomial kernel; for KNN, the number of neighbors taken into account are 1, 2, 3 and 4; for MLP, the possible values for α are 0.0001 and 0.05, with one (100 neurons) or two (100 and 10 neurons) hidden layers.

In order to obtain reliable error values, a 5-fold cross-validation is used: the dataset is split in 5 different ways and the average accuracy is used to tune the hyperparameters. K-fold cross-validation is usually implemented with k=5 or k=10 [24, p.184], in this case the smallest value is used due to the size of the dataset.

5.1.3 Information Extraction

A different approach is used to fill the slots whose value can be directly extracted from the text.

Firstly, the input text is preprocessed following the Europarl tokenization algorithm described previously, then unigrams, bigrams and trigrams are generated and converted to sentence embeddings computing the average of the vectors of each token.

In the case of area and name, the exposed entities, i.e. entities names, ids, and aliases for the entities made available for the Voice Assistant by the user through the UI, are converted to sentence embeddings in the same way, then the cosine similarity between each N-gram and exposed entity is computed. If the similarity is above a certain threshold, the area/name entity is added as slot value.

In the case of number extraction, such as brightness and temperature, a different approach is used. First, the text is parsed in order to find numeric modifiers (*nummod*), then related numeric modifiers and conjuncts (*conj*) are considered potential decimal parts to add to the number.

Finally, the color extraction is a simple hash-table search.

5.2 Implementation

The implementation consists of three packages: the *word2vec* package; the *intent_recognition* package; and the *converso* package. Relevant snippets of code can be found in Appendix B.

5.2.1 Word2Vec

The Word2Vec package handles the word embeddings, providing useful methods to convert tokens into their vector representation, to compute cosine similarity between text sequences, and to retrieve the most similar words to a given word.

As previously stated, the word embeddings are provided by the FastText library (specifically, the 6.74 GB *cc.it.300.bin* file). In order to reduce RAM usage, the model which is used in the training stage and in the final application is a pre-computed model derived from the original one by reducing it to 100 dimensions and saving it in word2vec binary format - resulting in a 783 MB file - using Gensim library [58]:

```
import fasttext
import fasttext.util
from gensim.models import KeyedVectors

fasttext.util.download_model('it', if_exists='ignore')
ft = fasttext.load_model('cc.it.300.bin')
fasttext.util.reduce_model(ft, 100)
ft.save_model('cc.it.100.bin')
m = gensim.models.fasttext.load_facebook_model('cc.it.100.bin')
m.wv.save_word2vec_format('gensim_cc.it.100.bin', binary=True)
```

5.2.2 Intent Recognition

The Intent Recognition package is used to generate and preprocess the dataset, to train the models, to evaluate them and save the best ones.

The synthetic dataset is generated through the *FeatureGrammar* of NLTK (Natural Language Toolkit) library [42]. The grammar consists of 202 productions, resulting in 33,874 commands, extended through NLTK stop-word removal to a total of 42,106 unique commands.

The tokenization is applied consistently with Europarl processing tools, using regex expressions and the NLTK *WhitespaceTokenizer*.

Each dataset is saved as a *.csv* file and handled as Pandas [43] dataframe. The details of each dataset - before balancing - can be found in Appendix A.

The models are trained and evaluated through Scikit-learn library [45], with cross validation performed using *GridSearchCV*, while the balancing is carried out using Imbalanced-learn library [38].

In order to save the models for re-use, the Joblib library [44] is used.

5.2.3 Converso

The Converso package can be installed as an Home Assistant integration and acts as conversation agent.

It includes the *ConversoAgent* class, which is a subclass of *AbstractConversationAgent*, and overrides the *_recognize* method.

The package also includes the *SpeechCorrector* class and the *IntentRecognizer* class.

As far the correction is concerned, NLTK *ngram* module is used, with $\lambda = 10^{-6}$, $\gamma = 10^{-8}$ and $\gamma = 10^{-8}$ and $\beta(i, N) = \frac{10 \cdot 2^{i-1}}{N}$ where i is the stage and N is the sentence length. The candidates are generated using a modified version of Norvig's implementation of string editing [56], not allowing the deletion and replacement of numeric characters. Conditional probabilities are cached in order to improve performance.

The *IntentRecognizer* class is in charge of using the *Intent Recognition* package to use the models and to extract the slots values in order to build a list of matched entities.

Spacy's library [63], and specifically the *it_core_news_sm* model with its Dependency Parser [64], is used for number extraction, as well as the Num2words library [65] in order to extract numbers from words (e.g. *three* to *3*) and vice versa.

Chapter 6

Experimental Results

This chapter will deal with the evaluation of the Converso engine. As in similar studies [66], models are first tested on the synthetic dataset, in order to find the best models to integrate into the final application, then the latter is tested in a realistic setting with human participants.

6.1 Synthetic dataset

The best models selected through cross-validation for Intent Recognition are shown in Table 6.1.

Label	Model	Hyperparameters
Intent	Linear SVC	C=10
	Linear SVC (without stop-words)	C=100
	Polynomial SVC	C=10, degree=2
	Polynomial SVC (without stop-words)	C=100, degree=2
	MLP	alpha=0.05, hidden_layer_sizes=(101,)
	MLP (without stop-words)	alpha=0.0001, hidden_layer_sizes=(101,)
	KNN	n_neighbours = 3
	KNN (without stop-words)	n_neighbours = 1

Table 6.1: Best configurations of models for Intent Recognition.

As far as performance is concerned, as it can be seen from Figure 6.1, accuracy is close to 100% for each model except Gaussian Naive-Bayes.

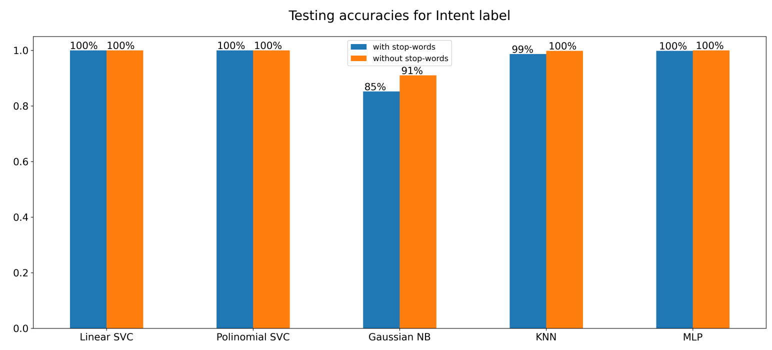


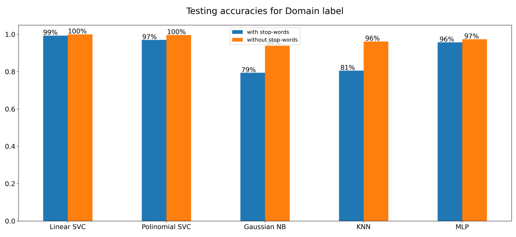
Figure 6.1: Testing accuracies for Intent.

Regarding slot filling, Table 6.2 shows the best models for the Domain, Device Class and State labels, while Figure 6.2 compares the accuracies of the models. As it can be seen, stop-word removal leads to an increase in accuracy and SVC with a linear kernel is the most accurate model, with a 100% score for each slot.

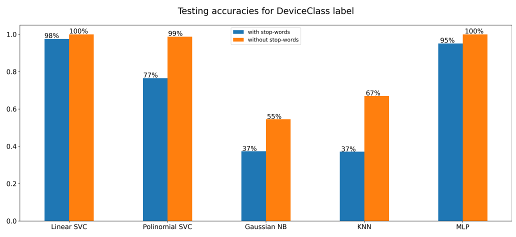
Finally, Table 6.3 shows the best models for Response Recognition, with the corresponding accuracies in Figure 6.3. As before, stop-word removal leads to an increase in accuracy, with the notable exception of the label ResponseHassGetState: in this case, stop-words such as *quale* (*which*) and *quanti* (*how many*) carry crucial information about the request. SVC with a linear kernel is still the most accurate model.

6.2 Experiment

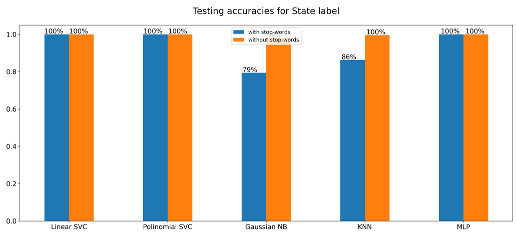
The Home Assistant integration is tested in a realistic scenario to gather more useful insights about the performance, since the usage of synthetic datasets could lead to overfitting.



(a) Domain label.

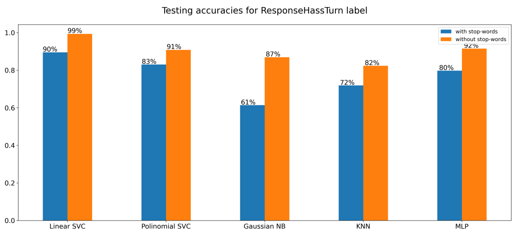


(b) Device Class label.

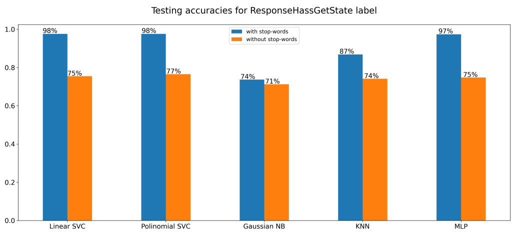


(c) State label.

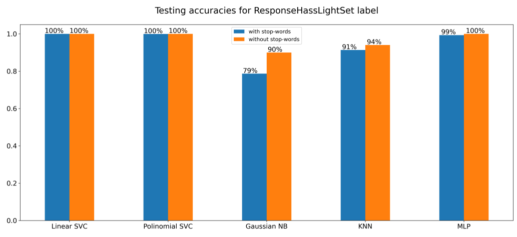
Figure 6.2: Testing accuracies for each slot.



(a) ResponseHassTurn label.



(b) ResponseHassGetState label.



(c) ResponseHassLightSet label.

Figure 6.3: Testing accuracies for each response type.

Label	Model	Hyperparameters
Domain	Linear SVC	C=10
	Linear SVC (without stop-words)	C=1
	Polynomial SVC	C=100, degree=3
	Polynomial SVC (without stop-words)	C=100, degree=2
	MLP	alpha=0.05, hidden_layer_sizes=(100,10)
	MLP (without stop-words)	alpha=0.0001, hidden_layer_sizes=(100,10)
	KNN	n_neighbours = 1
	KNN (without stop-words)	n_neighbours = 2
DeviceClass	Linear SVC	C=1
	Linear SVC (without stop-words)	C=0.1
	Polynomial SVC	C=100, degree=3
	Polynomial SVC (without stop-words)	C=100, degree=3
	MLP	alpha=0.05, hidden_layer_sizes=(100,10)
	MLP (without stop-words)	alpha=0.05, hidden_layer_sizes=(100,)
	KNN	n_neighbours = 1
	KNN (without stop-words)	n_neighbours = 1
State	Linear SVC	C=0.1
	Linear SVC (without stop-words)	C=0.1
	Polynomial SVC	C=10, degree=3
	Polynomial SVC (without stop-words)	C=1, degree=3
	MLP	alpha=0.05, hidden_layer_sizes=(100,)
	MLP (without stop-words)	alpha=0.0001, hidden_layer_sizes=(100,)
	KNN	n_neighbours = 1
	KNN (without stop-words)	n_neighbours = 1

Table 6.2: Best configurations of models for Slot Filling.

6.2.1 Hardware

The Home Assistant integration is tested using a development container and Docker [46] [47] on a machine with a 2.90 GHz AMD processor, allocating 4 cores and 8 GB RAM. In addition to the Home Assistant container, a container for Whisper and one for Piper are also instantiated.

The satellite device which is used for the experiment is the M5Stack Atom Echo, an inexpensive and small programmable smart speaker equipped with a ESP-PICO-D4 chip and a SPM1423 PDM microphone 6.4. The device can be integrated easily into Home Assistant through the ESPHome add-on and activated by pressing the button and pressing it again to send the command to the local server.

Label	Model	Hyperparameters
ResponseTurn	Linear SVC	C=1
	Linear SVC (without stop-words)	C=1
	Polynomial SVC	C=100, degree=3
	Polynomial SVC (without stop-words)	C=100, degree=3
	MLP	alpha=0.0001, hidden_layer_sizes=(10,10)
	MLP (without stop-words)	alpha=0.05, hidden_layer_sizes=(100,10)
	KNN	n_neighbours = 1
	KNN(without stop-words)	n_neighbours = 1
ResponseGetState	Linear SVC	C=1
	Linear SVC (without stop-words)	C=0.1
	Polynomial SVC	C=100, degree=2
	Polynomial SVC (without stop-words)	C=10, degree=2
	MLP	alpha=0.0001, hidden_layer_sizes=(100,10)
	MLP (without stop-words)	alpha=0.0001, hidden_layer_sizes=(100,)
	KNN	n_neighbours = 1
	KNN (without stop-words)	n_neighbours = 3
ResponseLightSet	Linear SVC	C=0.1
	Linear SVC (without stop-words)	C=0.1
	Polynomial SVC	C=100, degree=3
	Polynomial SVC (without stop-words)	C=10, degree=2
	MLP	alpha=0.05, hidden_layer_sizes=(101,4)
	MLP (without stop-words)	alpha=0.0001, hidden_layer_sizes=(101,)
	KNN	n_neighbours = 1
	KNN (without stop-words)	n_neighbours = 1

Table 6.3: Best configurations of models for Response Recognition.

6.2.2 Data collection

Ten participants, who have never used virtual assistants to control smart home appliances, are selected for the experiment. They are all different from each other in respect to:

- gender: 5 women, 4 men, 1 non-binary person;
- age: ranging from 22 to 65 years old;
- geographic origin: 3 from Emilia-Romagna, 3 from Piedmont, 3 from Campania, 1 from Marche.

The experiment is conducted in a controlled indoor environment, one participant at a time. The first phase of the experiment consists in system

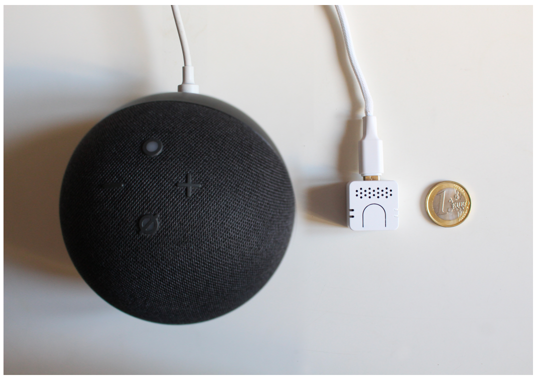


Figure 6.4: M5Stack Atom Echo size comparison. On the left, an Amazon Echo Dot 5th Gen.; on the right, a 1 euro coin.

setup: each participant has to describe their house, defining areas and devices (lights, fans, HVAC devices, covers and switches), naming each entity. The second phase consists in letting the participants imagine their houses were transformed into a smart home, describing the available features of the system. In this phase, periphrasis are used in order not to influence the lexicon and the syntax of the commands: for example, one of the prompts for the intent *HassLightSet* is “You can control how much light is emitted”.

The speech recognition engine which is used is Whisper *base-int8* with a beam size of 2 candidates. The distance from the microphone ranges from 0.5 meters to 2 meters; 2/3 of the experiments are conducted in a silent environment, while 1/3 of the experiments includes a 50dB-70dB background conversation noise.

A dataset with 360 uttered commands with the corresponding Whisper output commands is created and manually annotated, according to the predefined system setup.

Some takeaways from this first part of the experiment are:

- naming conventions differ greatly between participants: for example, a participant has chosen to name all lights *luce* (light), while other have chosen to name them including their location in the name;

- only few participants have included articles and prepositions in their commands;
- the ASR system recognizes commands with longer pauses between words better than rushed commands.

6.2.3 Evaluation

The gathered commands are processed with both the default Hassil-based conversation agent and the Converso agent, in order to compare their performance over three different types of input: the raw speech, the automatically corrected speech, and the text command.

During the processing, the maximum memory usage of the platform is 6.1 GB, while processor usage is always below 50%.

As far as speed is concerned, the total median delay, defined as the time between the end of the utterance and the response, is less than 3 seconds in each case. The delay is mainly affected by spelling correction, as it can be seen from Figure 6.5.

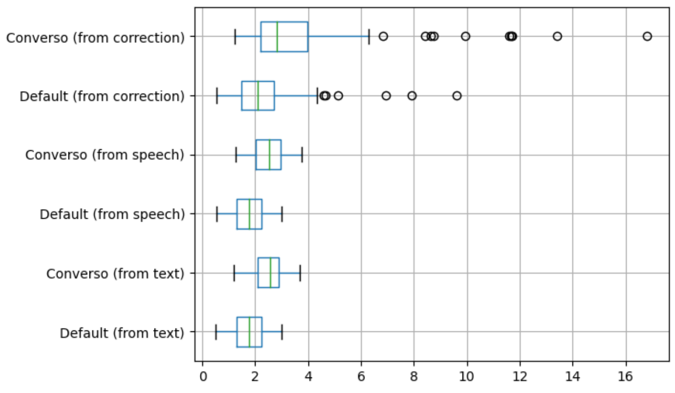


Figure 6.5: Delay of response since command utterance (s).

Regarding spelling correction, its performance can be evaluated through a metric called Word Error Rate (WER), defined as the sum of the word substitutions, deletions and insertions applied by the algorithm, normalized by the number of words in the reference text. Figure 6.6 shows the effect on WER of the spelling correction algorithm applied to the output of STT. Even though some inputs are too noisy to be corrected (even by humans), the spelling correction algorithm helps in shrinking WER; for example, the text: “Pegni lucci camera dune.” is correctly corrected to “spegni luce camera 2”, reducing WER from 0.75 to zero.

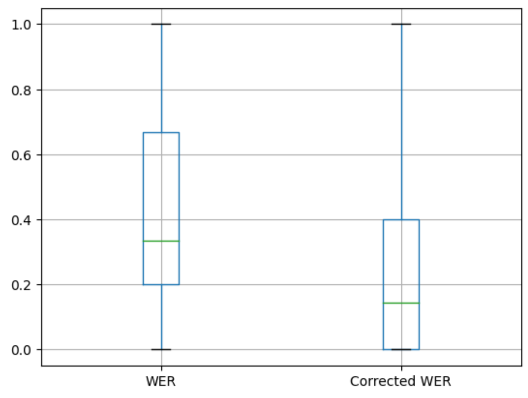


Figure 6.6: Comparison of Word Error Rates of raw STT and spelling corrected STT.

As for overall accuracy, Figure 6.7 shows a comparison of the distributions of correct action/responses and errors. As expected, the best results are achieved with textual commands, and ASR correction increases accuracy for speech commands. Comparing the two agents, Converso outperforms Hassil on all inputs; reaching an accuracy of 59%-64% on speech (depending on response, e.g. the agent could carry out the correct action but reply with an incomplete feedback). However, Converso yields more incorrect actions and responses in respect to Hassil, due to the former’s flexible approach.

In addition, Figure 6.8 shows the Confusion Matrix for the Intent label (the most important labels, since it affects all other classifications) for each conversation agent and input, which highlights that TurnOn and TurnOff are the hardest classes to distinguish, since their commands are similar and a one-word misspelling can overturn the meaning.

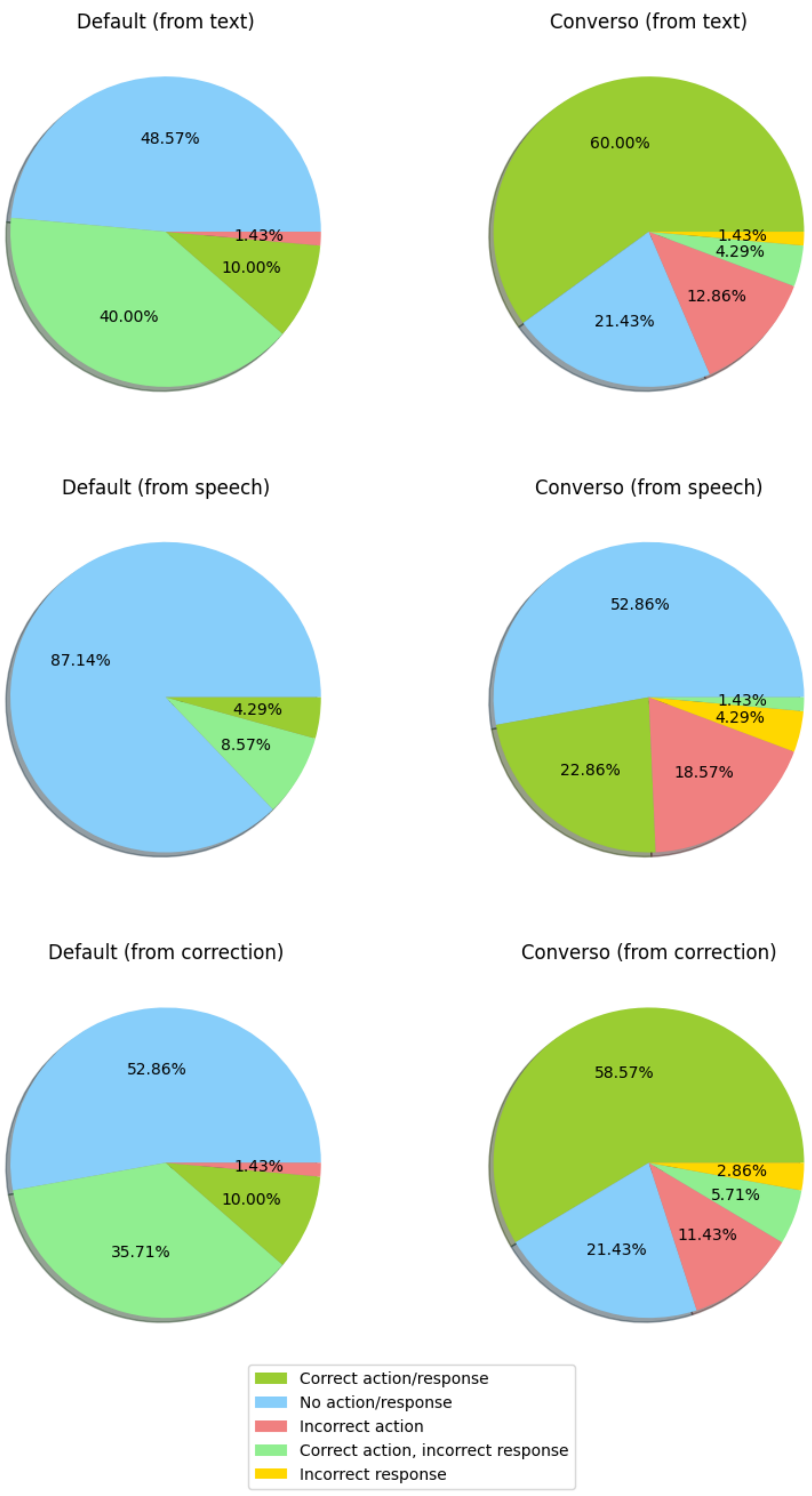


Figure 6.7: Comparison between the default Home Assistant conversation agent and Converso agent.

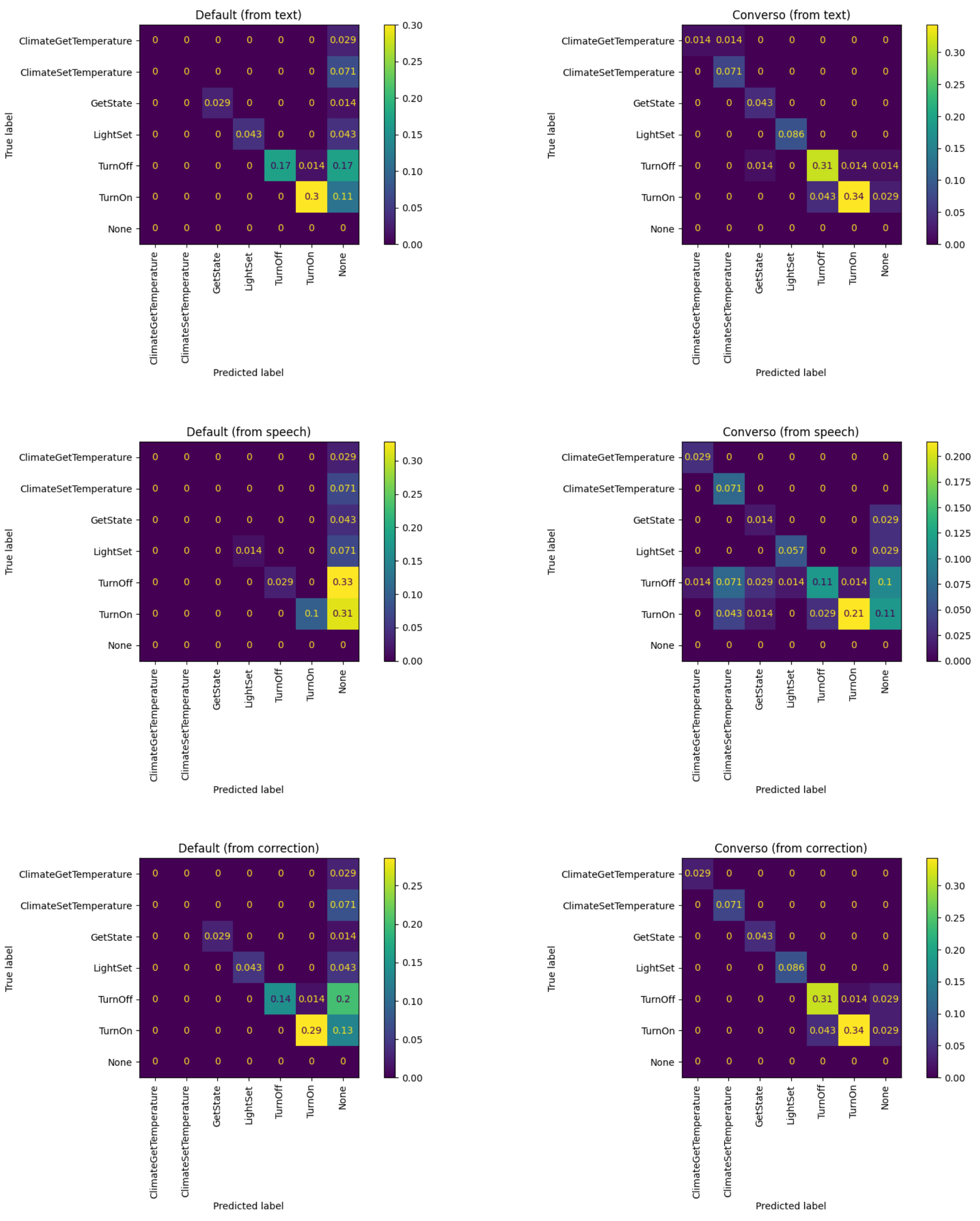


Figure 6.8: Normalized Confusion Matrices for the Intent label.

Chapter 7

Conclusions

The aim of this study was to design an Intent Recognition engine for a privacy-preserving local Virtual Assistant for the control of Smart Home appliances in Italian language.

In order to achieve this goal, techniques from several fields of study were used: a Context-Free Grammar was used to generate a synthetic dataset; N-grams were used for spelling correction; the Vector Space Model was used to extract relevant features, specifically Word2Vec embeddings, from text and to compare sequences of tokens; Support Vector Machines were used for Intent and slots classification.

Finally, the engine, called Converso, has been integrated into the most popular local home control platform, Home Assistant, which preserves privacy by not sending any data to external servers. The Virtual Assistant has been tested in a real life setting with 10 participants, for a total number of 360 commands. Converso has scored an accuracy above 60%, which outperforms Home Assistant's rule-based conversation agent, offering users more flexibility when they issue commands.

It would be desiderable that subsequent research focused on limiting memory usage further and on improving spelling correction performance, which is not perfect and also appears to be the main cause of delay. The main challenge is the trade-off between resource usage and performance: for example, generating 3-edits candidates for correction could result in a greater accuracy at the cost of a longer delay in command execution.

A possible alternative method could leverage the statistical distribution of ASR errors, but more data has to be gathered for the Italian language in the Smart Home domain in order to adopt this approach. A greater number of samples could also be used to reduce model overfitting.

However, this research has shown that the development of a privacy-preserving Voice Assistant for home control is feasible, and hopefully this can result in helping a greater number of people be more independant and comfortable in the future.

Appendix A

Datasets

Dataset	Class	Samples
Intent	HassTurnOn/HassTurnOff	761
	HassGetState	4,556
	HassLightSet	13,515
	HassClimateGetTemperature	1,201
	HassClimateSetTemperature	22,073
		42,106
Domain	default	90
	light	462
	fan	1,194
	cover	4,084
	climate	248
		6,078
State	off/on	2,257
		4,514
DeviceClass	door/window/blind/curtain/shade/shutter	644
	awning	54
	garage	120
	gate	46
		4,084
rn	default	
	lights_area	102
	cover_device_class_area	612
	cover_device_class	98
	fans_area	348
	cover_garage	28
		1,216
ResponseGetState	one	42
	one_yesno	1,150
	any	1,014
	all	574
	which	1,214
	how_many	562
		5,648
ResponseLightSet	brightness	1,386
	color	297
	brightness_area	9,744
	color_area	2,088
		13,515

Appendix B

Code

The full code is available at `https://github.com/paolapersico1/converso`. This appendix contains the most relevant snippets of code.

```python
def pipeline():
  """Generate Intent Recognition models."""
  df = load_synthetic_dataset()
  w2v = Word2Vec(dim=WORD2VEC_DIM)

  for label in (
    "Intent","Domain","DeviceClass","State",
    "ResponseHassTurn","ResponseHassLightSet",
    "ResponseHassGetState",
  ):
    ld = create_label_datasets(label,df)
    best_models = {}
    for without_sw in (True,False):
      x_current = ld["Text"].to_numpy().reshape(-1, 1)
      if label.startswith("Response"):
        y_current = ld["Response"]
      else:
        y_current = ld[label]

      x_train, x_test, y_train, y_test = train_test_split(
        x_current, y_current, test_size=0.10,
        random_state=42,stratify=y_current,
    )

    current_bests = generate_best_models(
      x_train, y_train, x_test, y_test, label,
      w2v, without_sw
    )
    best_models.update(current_bests)

```

```python
def preprocess_text(text):
  """Tokenize the text."""
  text = text.lower()
  text = text.strip()
  text = " " + text + " "
  text = re.sub(r"([^\w\s.\'\`,\-])", r" \1 ", text)

  text = re.sub(r"\.([\.]+)", r"DOTMULTI\1", text)
  while "DOTMULTI." in text:
    text = re.sub(r"DOTMULTI\.([^\.])", r"DOTDOTMULTI \1",
 text)
    text = re.sub(r"DOTMULTI\.", r"DOTDOTMULTI", text)

  text = re.sub(r"([^0-9]),([^0-9])", r"\1 , \2", text)
  text = re.sub(r"([0-9]),([^0-9])", r"\1 , \2", text)
  text = re.sub(r"([^0-9]),([0-9])", r"\1 , \2", text)
  text = text.replace("`", "'")
  text = text.replace("''", ' " ')
  text = re.sub(r"([^\w])[']([^\w])", r"\1 ' \2", text)
  text = re.sub(r"([^\w])[']([\w])", r"\1 ' \2", text)
  text = re.sub(r"([\w])[']([^\w])", r"\1 ' \2", text)
  text = re.sub(r"([\w])[']([\w])", r"\1' \2", text)
  text = text.lstrip()
  text = text.rstrip()
  text = re.sub(r"([0-9])\.([0-9])", r"\1FRACDOT\2", text)
  text = text.replace(".", " .")

  while "DOTDOTMULTI" in text:
    text = text.replace("DOTDOTMULTI", "DOTMULTI.")

  text = text.replace("DOTMULTI", " .")
  text = text.replace("FRACDOT", ".")

  tk = WhitespaceTokenizer()

  sequence = [str(NUMBER_DICT.get(token, token))  for token
   in tk.tokenize(text)]

  return tk.tokenize(text)

def mean_embedding(w2v, token_list):
  """Return the average embedding over a list of tokens."""
  result = np.mean(
    [
      w2v.word2vector(token.strip(" ’"))
      for token in token_list
      if token.strip(" ’") in w2v.w2v_model
    ],
    axis=0,
  )
  return result
```

```
def grid_search(x_trainval, y_trainval, clf, params, w2v,
   without_sw):
  """Perform grid search with different parameters."""
  rs = RandomUnderSampler()
  embedder = W2VTransformer(w2v, with_sw=(not without_sw))

  pipeline = Pipeline(
    [
      ("sampling", rs),
      ("embedding", embedder),
      ("scaler", StandardScaler()),
      ("clf", clf),
    ]
  )

  gs = GridSearchCV(
         pipeline,params,cv=5,n_jobs=1,
         return_train_score=True,verbose=3,
  )
  gs.fit(x_trainval, y_trainval)

  return pd.DataFrame(gs.cv_results_), gs.best_estimator_
```

```
def detect_errors(self, sequence):
  """Detect possible errors in a sequence."""
  errors = []
  for bigram in list(ngrams(sequence, 2)):
    prob = self.prob(bigram)
    if prob < self.plausibility_threshold:
      bigram_as_str = " ".join(bigram)
      errors.append(bigram_as_str)

  errors = errors + [
    unigram for unigram in sequence if unigram not in self.domain_vocab
  ]

return errors

def conditional_prob(self, left, right):
  """Compute conditional probability with raw count."""
  if (left, right) in self.cached_cond_prob:
    result = self.cached_cond_prob[(left, right)]
  else:
    num = self.raw_count((right, left))
    den = self.raw_count(right)
    cond_prob = float(num) / den
    result = (1 - self.lambda1) * cond_prob + self.lambda1 * self.prob(left)
    self.cached_cond_prob[(left, right)] = result

  return result

```

```python
def sentence_LL(self, sequence):
  """Compute the bigram log likelihood of a sequence."""
  ll = 0
  n_grams = list(ngrams(sequence, 2))
  for pre_word, post_word in n_grams:
    cond_prob = self.conditional_prob(post_word, pre_word)
    ll = ll + math.log(cond_prob)

  return ll

def generate_ngram_candidates(self, ngram, sentence, vocab,
    edits_num, edits):
  """Generate possible error corrections for ngrams."""
  candidates: list[Edit] = []

  for edit in self.legit_edits(edits, vocab):
    p_sequence = self.tk.tokenize(sentence.replace(ngram,
    edit))
    p_sequence = list(pad_both_ends(p_sequence, n=2))
    bigram_ll = self.sentence_LL(p_sequence)
    candidates.append(
      Edit(
        previous=ngram, new=edit,
        edits=edits_num, bigram_ll=bigram_ll
      )
    )
  return candidates
```

Bibliography

[1] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. Data Mining: Concepts and Techniques (3rd. ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[2] ChengXiang Zhai and Sean Massung. 2016. Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining. Association for Computing Machinery and Morgan and Claypool.

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. Introduction to Information Retrieval. Cambridge University Press, USA.

[4] Stephen Fagan, Ramazan Gençay. 2010. An introduction to textual econometrics, in Ullah, Aman; Giles, David E. A. (eds.), Handbook of Empirical Economics and Finance, CRC Press, pp. 133–153.

[5] Singhal, C. Buckley, and Mandar Mitra. 1996. Pivoted document length normalization. In Proc. of the 19th Annual International ACMSIGIR Conference on Research and Development in Information Retrieval, SIGIR 96, pp. 21–29, New York. DOI: 10.1145/243199.243206.

[6] Stephen Robertson, Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. Found. Trends Inf. Retr. 3, 4, 333–389.

[7] G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. Commun. ACM 18, 11 (Nov. 1975), 613–620. https://doi.org/10.1145/361219.361220

[8] Gian Luigi Beccaria. 2004. Dizionario di linguistica, ed. Einaudi.

[9] Martin Porter, Richard Boulton. 2002. The English (Porter2) stemming algorithm. Last access: 19 August 2022. Url: `http://snowball.tartarus.org/algorithms/english/stemmer.html`.

[10] Stuart Russell, Peter Norvig. 2009. Artificial Intelligence: A Modern Approach (3rd. ed.). Prentice Hall Press, USA.

[11] Sergio Jimenez. 2015. A plot of the rank versus frequency for the first 10 million words in 30 Wikipedias (dumps from October 2015) in a log-log scale. Last access: 20 August 2022. Url: `https://en.wikipedia.org/wiki/Zipf%27s_law#/media/File:Zipf_30wiki_en_labels.png`.

[12] Alan M. Turing. 1950. Computing machinery and intelligence. Mind, 59, pp. 433–460.

[13] A. A. Markov. 1913. Example of a statistical investigation of the text of Eugene Onegin illustrating the dependence between samples in chain. Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Academie Imperiale des Sciences de St.Petersbourg), 7, pp. 153–162.

[14] C. E. Shannon. 1948. A mathematical theory of communication. The Bell System Technical Journal, vol. 27, no. 3, pp. 379-423.

[15] Y. Bar-Hillel. 1960. The Present Status of Automatic Translation of Languages. Adv. Comput., 1, pp. 91–163.

[16] Bert F. Green, Alice K. Wolf, Carol Chomsky, Kenneth Laughery. 1961. Baseball: an automatic question-answerer. In Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference

(IRE-AIEE-ACM '61 (Western)). Association for Computing Machinery, New York, NY, USA, pp. 219–224.

[17] Yoav Goldberg. 2016. A Primer on Neural Network Models for Natural Language Processing. Journal of Artificial Intelligence Research. 57, pp. 345–420

[18] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. ICLR Workshop.

[19] Thomas Mikolov, Ilya Sutskever, Kai Chen, G. S. Corrado, Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 26.

[20] T. Mikolov, W.T. Yih, G. Zweig. 2013. Linguistic Regularities in Continuous Space Word Representations. NAACL HLT.

[21] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 2227–2237.

[22] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv abs/1810.04805

[23] T. Hastie, R. Tibshirani, J. Friedman. 2001. The Elements of Statistical Learning. New York, NY, USA: Springer New York Inc.

[24] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. 2014. An Introduction to Statistical Learning: with Applications in R. Springer Publishing Company, Incorporated.

[25] Thomas M. Mitchell. 1997. Machine Learning (1st. ed.). McGraw-Hill, Inc., USA.

[26] Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[27] Cyc. 2008. Svm separating hyperplanes. Last access: 10 September 2022. Url: `https://commons.wikimedia.org/wiki/File:Svm_separating_hyperplanes.png`.

[28] Abhishek Kaushik, Sudhanshu Naithani. 2016. A Comprehensive Study of Text Mining Approach. International Journal of Computer Science and Network Security. p. 69.

[29] Mitchell P. Marcus, Mary Ann Marcinkiewicz, Beatrice Santorini. 1993. Building a large annotated corpus of English: the penn treebank. Comput. Linguist. 19, 2, 313–330.

[30] Daniel M. J. H. Jurafsky. Speech and Language Processing 3rd Edition Draft. Last access: 18 October 2022. Url: `https://web.stanford.edu/~jurafsky/slp3/`.

[31] J. R. Firth. 1957. Studies in Linguistic Analysis. Wiley-Blackwell.

[32] A. Dix, J. Finlay, G. Abowd, R. Beale. 2003. Human Computer Interaction Third Edition. Prentice Hall.

[33] Home Assistant. Voice in Home Assistant. Last access: 11 August 2023. Url: `https://developers.home-assistant.io/docs/voice/overview/`

[34] Home Assistant. Home Assistant Analytics. Last access: 11 August 2023. Url: `https://analytics.home-assistant.io/`

[35] OpenAI. Whisper. Last access: 13 August 2023. Url: `https://github.com/openai/whisper`

[36] Home Assistant. Home Assistant. Last access: 13 August 2023. Url: `https://github.com/home-assistant/core`

[37] Home Assistant. Home Assistant intents. Last access: 18 August 2023. Url: `https://github.com/home-assistant/intents/tree/main/sentences/it`

[38] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: a python toolbox to tackle the curse of imbalanced datasets in machine learning. J. Mach. Learn. Res. 18, 1 (January 2017), 559-563.

[39] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. 2018. Learning Word Vectors for 157 Languages. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

[40] P. Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In MT summit, volume 5.

[41] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics 2017; 5 135-146

[42] NLTK Project. 2023. Natural Language Toolkit. Last access: 22 August 2023. Url: `https://www.nltk.org/`

[43] NumFOCUS, Inc. 2023. Pandas. Last access: 22 August 2023. Url: `https://pandas.pydata.org/`

[44] Joblib Development Team. 2022. Joblib: running Python functions as pipeline jobs. Last access: 22 August 2023. Url: `https://joblib.readthedocs.io/`

[45] Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. JMLR 12, pp. 2825-2830.

[46] Home Assistant. 2023. Set up Development Environment. Last access: 22 August 2023. Url: `https://developers.home-assistant.io/docs/development_environment/`

[47] Docker. 2023. Docker: Accelerated Container Application Development. Last access: 22 August 2023. Url: `https://www.docker.com/`

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000-6010.

[49] Alec Radford Alec, Jong Kim, Tao Xu, Greg Brockman, Christine McLeavey, Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision.

[50] Y. Bassil, P. Semaan. 2012. ASR Context-Sensitive Error Correction Based on Microsoft N-Gram Dataset. ArXiv, abs/1203.5262.

[51] Home Assistant. 2023. Year of the Voice - Chapter 1: Assist. Last access: 22 August 2023. Url: `https://www.home-assistant.io/blog/2023/01/26/year-of-the-voice-chapter-1/`.

[52] Istat. 2023. Una popolazione che invecchia. Last access: 22 August 2023. `https://www.istat.it/demografiadelleuropa/bloc-1c.html#:~:text=Cominciamo%20col%20guardare%20l'evoluzione,5%20punti%20percentuali%20(p.p.)`.

[53] Lydia Manikonda, Aditya Deotale, and Subbarao Kambhampati. 2018. What's up with Privacy? User Preferences and Privacy Concerns in Intelligent Personal Assistants. In Proceedings of the 2018

AAAI/ACM Conference on AI, Ethics, and Society (AIES ’18). Association for Computing Machinery, New York, NY, USA, 229-235. https://doi.org/10.1145/3278721.3278773.

[54] M. Baroni, S. Bernardini, A. Ferraresi and E. Zanchetta. 2009. The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora. Language Resources and Evaluation 43(3): 209-226.

[55] Youssef Bassil and Paul Semaan. 2012. ASR Context-Sensitive Error Correction Based on Microsoft N-Gram Dataset. Journal of Computing, Vol.4, No.1, January 2012.

[56] Peter Norvig. 2016. How to Write a Spelling Corrector. Last access: 22 August 2023. Url: `https://norvig.com/spell-correct.html`

[57] Fred J. Damerau. 1964. A technique for computer detection and correction of spelling errors. Commun. ACM 7, 3 (March 1964), 171-176. https://doi.org/10.1145/363958.363994.

[58] R. Rehurek, P. Sojka. 2011. Gensim: python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2).

[59] Gene Munster, Will Thompson. 2018. Annual Smart Speaker IQ Test. Last access: 18 September 2023. Url: `https://deepwatermgmt.com/annual-smart-speaker-iq-test/`

[60] P. Cheng, U. Roedig. 2022. Personal Voice Assistant Security and Privacy? A Survey. Proceedings of the IEEE, vol. 110, no. 4, pp. 476-507.

[61] Lea Schönherr, Maximilian Golla, Thorsten Eisenhofer, Jan Wiele, Dorothea Kolossa, Thorsten Holz. 2020. Unacceptable, where is my privacy? Exploring Accidental Triggers of Smart Speakers.

[62] R. W. Shirey. 2007. Internet Security Glossary Version 2, Aug. 2007. Last access: 18 September 2023. Url: `https://rfc-editor.org/rfc/rfc4949.txt.`

[63] M. Honnibal, I. Montani, S. Van Landeghem, A. Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python. https://doi.org/10.5281/zenodo.1212303

[64] Cristina Bosco, Simonetta Montemagni, and Maria Simi. 2013. Converting Italian Treebanks: Towards an Italian Stanford Dependency Treebank. In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, pages 61-69, Sofia, Bulgaria. Association for Computational Linguistics.

[65] Virgil Dupras. 2023. num2words. Last access; 22 September 2023. Url: `https://github.com/savoirfairelinux/num2words`

[66] Thierry Desot, Stefania Raimondo, Anastasia. Mishakova, François Portet, Michel Vacher. 2018. Towards a French Smart-Home Voice Command Corpus: Design and NLU Experiments. 21st International Conference on Text, Speech and Dialogue TSD 2018, Sep 2018, Brno, Czech Republic. pp.509-517.

Acknowlegments

I would like to thank all the people who contributed to this research with their active support.

First of all, I would like to thank my Supervisor, Prof. Danilo Montesi, not only for sharing his expertise and precious suggestions, but also for his patience and encouragement.

Special thanks go to the participants of the final experiment, who have kindly offered their time in order for me to gather useful data.

Finally, I must thank the people who supported and motivated me during dark times: my mother Maria, my sister Celestina, my significant other Giulia and her parents Marco and Cristina. Without them, I would not be here writing this thesis.

My sister once told me: "There is no problem without a solution", and all the people who shared this journey with me have taught me that this statement is true, as long as you have the motivation and courage to identify the right problem.