UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

GUILHERME VIEIRA SCHWADE

Improving Black-Box Speech-to-Text Systems via Machine Learning Techniques

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Bruno Castro da Silva

Porto Alegre
June 2016

“The man who moves a mountain
begins by carrying away small stones.”
— CONFUCIUS

ABSTRACT

There are several ways a user can interact with a computer. Not every way is equally appropriate for all situations: when typing, a keyboard is more appropriate; a mouse, on the other hand, is a better fit in case the user needs to control the cursor with precision. In some complex systems, the user might need to execute several different tasks, and, therefore, might need different ways to interact with the system.

In order to simplify those interactions, the use of voice commands might be a good strategy, since they often allow the user to specify the task to be executed with a richer input vocabulary than that available via other, more standard input devices. However, the development of robust speech-to-text converters (SST converters) requires a lot of time and resources which development teams often do not have. There are widely-used SST converters available on the internet, such as the Web Speech API from Google; these systems are in a very advanced stage of maturity considering general context applications—for instance, when they are used to analyze terms and words that occur in day-to-day conversations. However, these systems are often not efficient when used to analyze *context-specific* terms, which occur only in particular systems or applications. Furthermore, these systems are usually black-box and cannot be modified or improved by developers who wish to use them to solve particular *specialized* speech-to-text problems.

To analyze possible solutions to this problem, we study the development of an additional layer of software, trained via machine learning techniques, to correct or adapt the imperfect translations generated by a black-box STT when applied to a specific domain. In particular, we propose and evaluate several machine learning solutions to improve a complex flight tickets management system to which we wish to add voice-control capabilities. In the first part of this work, we discuss our motivation and describe the domain where the proposed methods evaluated. After that, mathematical theoretical background is presented and we introduce possible solutions to the particular domain at hand. At the end, a critical analysis of the results is made and future work is discussed.

Keywords: Speech Recognition. Phonetic algorithm. Levenshtein distance. Machine learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ARD	Altea Reservation Desktop
HMM	Hidden Markov Models
KNN	K-Nearest Neighbors
ML	Machine learning
MNB	Multinomial naive Bayes
NN	Neural network
RNN	Recurrent Neural Networks
STT	Speech-to-text
WER	Word Error Rate

CONTENTS

1 INTRODUCTION

Voice recognition systems have a wide range of applications in the past few years, from systems to control cellphones, manage tasks in a schedule, enter queries in web search engines, etc. Machine learning techniques are often used to train these systems and result in high-accuracy models for predicting the most likely set of words/phonemes corresponding to a recorded sound file. Even though they have high accuracy when applied to general and broad problems or domains, they often fail if evaluated on tasks involving a narrow and specialized vocabulary.

Speech is arguably the most natural and simple medium of communication between humans. The speech recognition process, when performed by a machine, can be seen as the process of translating an acoustic signal—the voice signal—into a set of words and sentences that ideally matches the sentence that was pronounced by a speaker (THAYSE; BINOT, 1991). This recognition process is typically divided into different steps and it is often interpreted within the context of language models that represent one set of hypotheses for translating speech into text. These language models represent the result of analyses of a *particular* set of training data, corresponding to a particular set of people speaking about topics within a particular domain, and could therefore reflect only one specific context; if this model were to be applied to translate sentences of a different domain, it could produce erroneous translations.

There are currently several research groups and companies trying to address the problem of automatic translation. Most of them have access to large amounts of training data, which they use to produce models that are as general as possible. As two examples, we cite the work of (HANNUN et al., 2014), where the authors present the use of recurrent neural network (RNN; see chapter 2 for a discussion on neural networks). They claim to achieve a 16% prediction error, which means that their approach correctly translates 84% of the input voice recordings to text; this yields better performance than most commercial solutions when deployed in noisy solutions. According to authors, a training set of 5000 hours of speech, collected from 9600 different speakers, was used to train their network. (KOMBRINK et al., 2011) also describes the use of a RNN to implement a specialized language model constructed to tackle a different domain or problem: that of transcribing recordings of meetings.

Most commercial applications that may benefit from voice-recognition capabilities, however, depend on systems capable of recognizing domain-specific terms or acronyms.

Such terms and expressions are typically not understood by general-purpose speech-to-text conversion systems, since these are usually trained via supervised learning techniques and often do not have access to a sufficient amount of examples involving specialized vocabulary. In the following chapters, we refer to general-purpose STT systems as *black-box solutions*. To the best of our knowledge, no solutions currently exist that allow developers or users to modify general-purpose STT systems by including additional specialized words into their training set, in order to improve their performance on different domains. The alternative would be to use the available domain-specific training set to construct a new STT solution from scratch. This is usually not feasible since most companies that could benefit from domain-specific STTs do not have access to training sets as large as those available to companies that develop black-box STTs, such as Google and IBM. To address this problem, we propose and evaluate different machine learning techniques that compose an additional software layer capable of boosting the translation given by a black-box STT interpreter.

1.1 Motivation

This work is motivated by the importance of being capable of rapidly constructing domain-specific STTs in order to improve applications that depend on specialized terms. We first encountered this difficulty when developing a voice recognition feature for Amadeus, an European company specialized in creating IT solutions to the travel industry. This company wished to develop a voice recognition feature that could improve performance and usability of a large air ticket management system. It soon became clear that the available black-box STT systems could not meet all the performance requirements imposed by the company, and that it would be exceedingly expensive to build one from scratch. The former difficulty results from the fact that voice recognition methods usually assume a fixed language model (YOUNG et al., 1989). Such a model is responsible for restricting the set of possible valid translations and is typically constructed based on a training corpus representing a more general domain. It was not possible for the company to modify the language model used by black-box STTs, in order to adapt them to a more specific domain; this led to the interest in building incremental, machine learning-based solutions capable of *adapting* the translations provided by general-purpose STTs to narrower translation tasks.

A research group was assembled and started to develop a proof-of-concept solu-

tion to the problem. During this process, a set of research questions and desired features was constructed, all of which were relevant to the development of an adaptable solution capable of improving black-box STTs to address the problem described in the introduction. The research questions that need to be addressed to construct such a system are still relevant and serve as a starting point to this work; they are presented and discussed in Section 1.1.1.

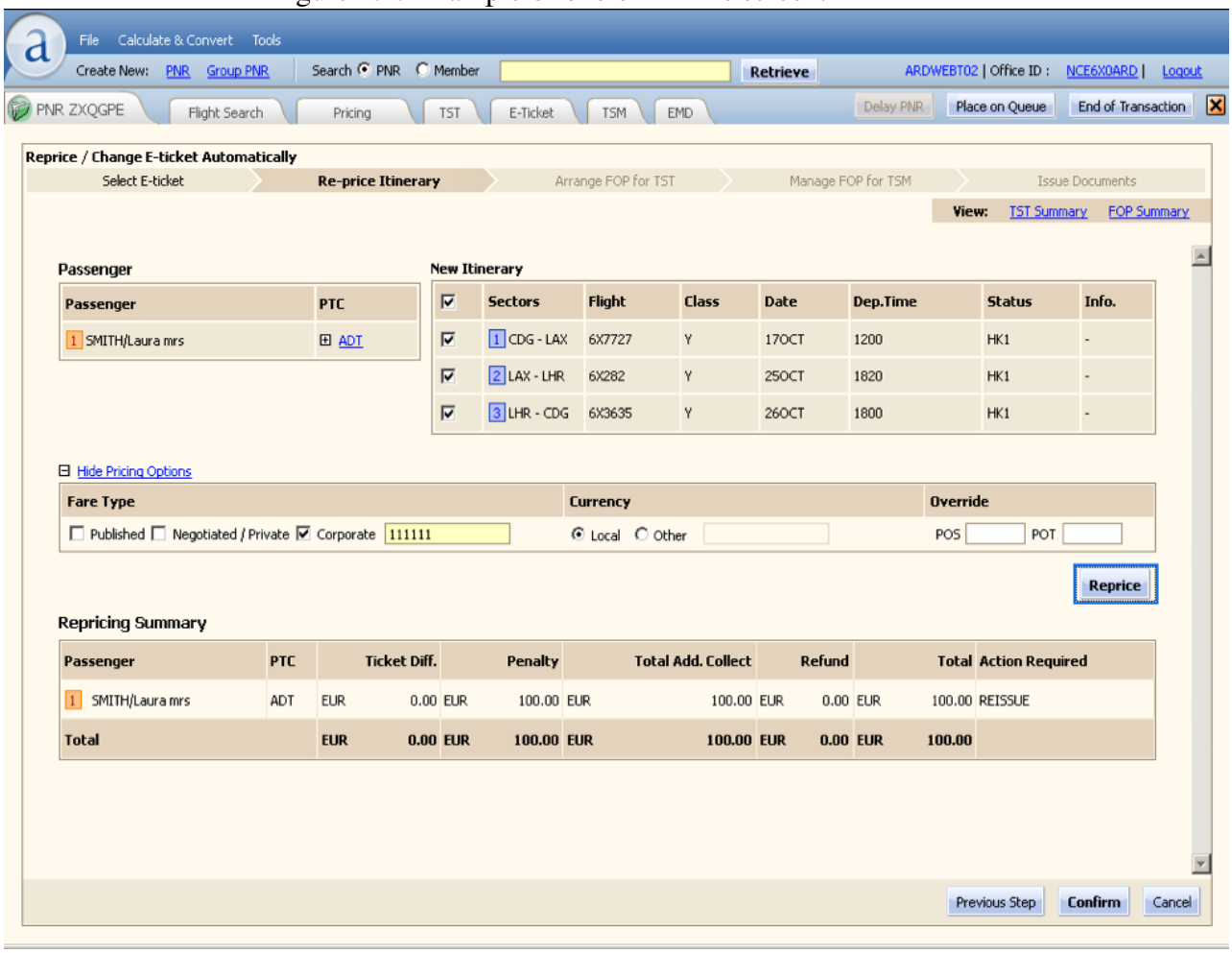
1.1.1 The OKARD Project

One of the most important products developed by Amadeus is the Altéa Reservation Desktop system, also known as ARD. ARD is a web-based reservation manager tool used by assistance services—call centers, city travel offices, etc.—of airlines companies. Altéa was designed to improve agent efficiency in terms of reservations, sales, and customer services. ARD differs from other typical solutions in the market by allowing the use of guided flows (which define the process by which users can interact with the system), configurable panels, and customisation options that enable the product to be tailored to specific client needs. A representative screen of ARD is shown in Figure 1.1. Due to the amount of features and their complexity, ARD screens usually contain many controls, tabs, and buttons, which negatively impact the system's accessibility and makes the training process of new users complex. One of the hypotheses made by Amadeus regarding how to best address this problem is that the software interface could become easier to use if voice-recognition capabilities were to be added to Altéa. That was the main motivation for the OKARD project, described in the next section.

The OKARD project had the objective to create a proof-of-concept application that could enable the user to control the ARD system using voice commands. The name OKARD is a reference to Google's voice system, in which the user launches the recognition system by saying "OK, Google". The Amadeus research group was not concerned, at the time, with measuring possible improvements in the usability of the system, but only with whether the proof-of-concept application could improve the recognition capabilities of the base black-box STT. In particular, it was not the objective of the project to have a full natural language processing system, but execute actions based on recognizing commands from a predefined list of specialized terms.

The commands typically used by an OKARD user involved common English terms, such as "Add" and "Save", and ARD specific terms, such as "XBAG"—which

Figure 1.1: Example of one of ARD's screen.



refers to “extra baggage service”. The idea was that by starting from simple commands like “Search flight” and “Close popup”, the user could chain commands into a more complex one, generating longer tasks such as “Search flight and close popup”. The system was also expected to be able to deal with tasks that required the pronunciation of commands that expect parameters, such as “Add passenger <name>”.

The base black-box STT interpreter used in the project was Google’s speech recognition webservice. The research team of Amadeus identified that when comparing Google’s translation of a voice recording with the correct sentence corresponding to it, in some situations a direct string comparison was completely incorrect. This occurred even when the sentence spoken by the user and the sentence recognized by Google were pronounced the same. With that in mind, the research group decided to compare the *sound* of the the sentence predicted by Google with known words belonging to the application’s vocabulary. This was achieved by using phonetic algorithms to transform a given text string into its phonetic encoding—how it would sound if pronounced by a person[1]. The team trained a neural network mapping the phonetic distances between spoken terms and known terms to probabilities of each one of those terms belonging to the spoken sentence. By comparing these phonetic encodings, the system could produce an ordered list of most likely commands that could explain the spoken sentence at hand.

To further narrow down the list of possible commands, the team used manual feedback from the user. Specifically, the system presented the top 5 different commands that were phonetically consistent with the spoken command, and the user could select the desired one—or none, if all proposed translations were inaccurate. If the the correct option was available and the user selected it, the system could use this information to grow the training set used by the neural network with a new example.

Even then, in some situations the system output was incorrect because the phonemes of separate spoken words could be combined by Google into a single word, or the phonemes of a single word could be split into many words. For instance, the spoken word “Redisplay” could be interpreted by Google as “Regis play”. This implied that the input phonetic patterns given as training examples to the neural network would be incorrect. To address this problem, the research team studied the use of manually-tuned algorithms as a pre-processing step to prepare the training set (see chapter 3.2). The final system had an accuracy of 75%; most the errors could be explained by relatively low performance of the pre-processing step; in this work we propose to automate such a process and improve its

[1] More details can be found in Section 2.3.

performance automatically via machine learning techniques.

1.2 General Objective

Although this research was motivated initially by Amadeus and by the specific requirements of ARD system, in this work we present methods and contributions that fit within a more general setting of the problem of improving the output of black-box STTs via machine learning techniques—so that its performance is improved when applied to domains with a specialized, narrower vocabulary. We aim to achieve this objective by collecting a relatively small amount of training data, representing ways in which users pronounce application commands, and using it to train a machine learning-based layer on top of output of a black-box interpreter, so as to improve its overall prediction accuracy.

2 THEORETICAL BACKGROUND

In this chapter we review the main machine learning methods that are relevant to the implementation of speech-to-text techniques, as well as to the development of the techniques are proposed in this work.

2.1 Boosting

The solution proposed in this work consists of composing prediction stages, each one improving the predictions made by previous one, in order to obtain a better final result. In machine learning, this approach is referred to as *Boosting*. According to (SCHAPIRE, 2003), boosting is based on the hypothesis that finding and composing many inaccurate rules of behavior is easier than finding a single highly accurate one. If a method that generates approximate prediction rules (called "weak" or "base" learners[1]) is available, and by assigning different weights to training examples, it is possible to iteratively produce new partial behavior rules that address the limitations of existing rules. At the end of the algorithm, several weak rules are combined into rule chain whose prediction power is expected to be much higher than that of any of the individual rules it composes.

The first provable polynomial-time boosting algorithm was presented in (SCHAPIRE, 1990). In that work paper, the author presents a model where a weak learner needs only to perform slightly better than random guessing; by combining such learners, the overall prediction error can be made arbitrarily small. In (FREUND, 1995), the boosting technique was further improved via a much more efficient algorithm; a first experiment using it in the OCR (Optical character recognition) task was described in (DRUCKER; SCHAPIRE; SIMARD, 1993).

One of the most famous boosting algorithms is AdaBoost (FREUND; SCHAPIRE, 1997). Its pseudocode is shown in Figure 2.1. The input of the algorithm is a training set in the form (x_i, y_i), where x_i belongs to the feature space X and y_i is the label of x_i, drawn from the label set Y. The algorithm creates a sequence of weak learners in a series of rounds $t = 1, \ldots, T$. The objective is to maintain a distribution or set of weights over the training set. In Figure 2.1, $D_t(i)$ represents the distribution or importance of training example i in the round t; at each round, a base trainer h_t is trained to improve

[1] These are called *weak rules* because if used alone, they do not result in a good accuracy when making predictions.

Figure 2.1: AdaBoost Pseudocode

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

Source: (SCHAPIRE, 2003)

upon the predictions of the previous learner. All distributions or sample importance are set equally initially; the weights of misclassified examples are increased so that the algorithm is forced to focus on them in the next round.

The learner's job is to find a convenient classification $h_t : X \to \mathbb{R}$ for sample distribution D_t. In the case of a binary classifier, i.e., $Y = \{-1, +1\}$, the objective is to minimize the error presented in Equation 2.1. Once each h_t is learned, the combined output of AdaBoost is given by a linear combination of learners, weighted by respective weights α_t. Such weights are usually set according to rules as those proposed in (FREUND; SCHAPIRE, 1997) (see Equation 2.2). There are some generalizations of AdaBoost to work with multiclass classification, as shown in (FREUND; SCHAPIRE, 1997; SCHAPIRE; SINGER, 1999; SCHAPIRE, 1997).

$$\epsilon_t = Pr_{(i\ D_t)}[h_t(x_i) \neq y_i] \tag{2.1}$$

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \tag{2.2}$$

Our present work can be seen as an instantiation of the boosting framework. We assume a base weak learner given by an STT translator, which most likely performs well on day-to-day words and expressions (since it was trained on those) but poorly on context-specific terms. We propose to take its output prediction and use it as input to another machine learning-based prediction block that is trained to fix the errors made by the weak

STT learner. In other words, neither the STT by itself, nor the second machine learning block we are proposing, can be used alone to correctly and accurately translate domain-specific voice commands into one of the known commands list. We use the failures of the first prediction stage (the weak STT translator) to control the training of the second prediction state.

2.2 Machine Learning Methods

In this section we discuss basic machine learning methods that are useful both for understanding related work, and also the methods we propose in this work.

2.2.1 Multinomial Naive Bayes

Bayesian classifiers are probabilistic methods that impose assumptions about the data, thereby generating a probabilistic model that embodies them. After that, using a supervised approach, they take a training set of labeled examples to estimate the parameters linked to a particular probabilistic model. Unseen examples are classified into one of the many classes by using Bayes' rule; the output of the model is the class that most likely generated the example. The simplest among these models is called Naive Bayes model. An early description can be found in (DUDA; HART et al., 1973).

Bayes' rule, also known as Bayes' Theorem, is shown in Equation 2.3:

$$P(c_k|x) = P(c_k) \times \frac{P(x|c_k)}{P(x)} \tag{2.3}$$

where

$$P(x) = \sum_{k'=1}^{e_C} P(x|c_{k'}) \times P(c_{k'}) \tag{2.4}$$

When using the theorem to implement the Naive Bayes method, we assume that all the events fall into exactly one of e_C classes ($c_1, c_2, c_3, \ldots, c_{e_C}$), and assume that x represents all input features that characterize the event: $x = (x_1, x_2, \ldots, x_d)$. $P(c_k|x)$ is then the conditional probability that the event belongs to class c_k given that its input features are x .

The classification model is said to be optimal when we know the exact value of

$P(c_k|x)$. As this value most often cannot be known with perfect accuracy, the number of classification errors can be minimized by making the model return the class c_k for which $P(c_k|x)$ is highest. Since estimating $P(c_k|x)$ from data is difficult, Bayes' rule requires us to instead estimate $P(x|c_k)$. This, however, can be difficult given the huge number of possible values for $x = (x_1, x_2, \ldots, x_d)$. What is normally assumed is that the distribution of x conditional on c_k can be decomposed, as shown in Equation 2.5.

$$P(x|c_k) = \prod_{j=1}^{d} P(x_j|c_k) \tag{2.5}$$

In particular, the assumption made in Equation 2.5 is that the occurrence of x_i is statistically independent from the occurrence of any other x_j, given that the event belongs to class c_k. This simplifies Equations 2.3 and 2.4 to Equations 2.6 and 2.7.

$$P(c_k|x) = P(c_k) \times \frac{\prod_{j=1}^{d} P(x_j|c_k)}{P(x)} \tag{2.6}$$

where

$$P(x) = \sum_{k'=1}^{e_C} P(c_{k'}) \times \prod_{j'=1}^{d} P(x_{j'}|c_{k'}) \tag{2.7}$$

As the denominator in Equation 2.3 is the same for all classes, it can be suppressed. This indicates that the classification is given as the class whose value of $P(c_k) \times \prod_{j=1}^{d} P(x_j|c_k)$ is the highest among all.

One of the classic variants of Naive Bayes is the Multinomial Naive Bayes (MNB), which is used for multinomially distributed data. The distribution is parametrized by a vector $\theta_k = (\theta_{k1}, \ldots, \theta_{kn})$, one for each class k, and θ_{ki} is the probability of feature i appearing in a sample of class k. The parameters θ_{ki} are estimated by using a smoothed version of maximum likelihood equation shown in Equation 2.8, where N_{ki} is the number of times the feature i appears in samples of class k, N_k is the count of all features appearing in samples of class k, and n is the number of features; finally, α is used to solve the problems of features that are not present in the training set, in order to prevent zero probabilities from appearing in the computations.

$$\theta_{ki} = \frac{N_{ki} + \alpha}{N_k + \alpha n} \tag{2.8}$$

2.2.2 Neural Networks

An artificial neural network, or simply neural network (NN), is a mathematical model inspired by the behavior of the human brain (MCCULLOCH; PITTS, 1943). It is represented by a oriented graph where each node is called a neuron and is the smallest unity of a network, and each edge is called a synapse.

A neuron is a weighted sum of n inputs and an activation function φ, such as $\varphi(x) = \frac{1}{1+e^{-t}}$ or $\varphi(x) = \tanh(x)$. The activation function takes the weighted sum and computes the final output of the neuron. Complex networks can be composed of different parallel and serial connections of neurons where neuron has its own weights.

An example of a neural network architecture is called Feed Forward (FF) neural network. A Feed Forward network is a multilayer network where all the synapses are directed towards the output. The input data, in these networks, is passed through each layer and propagated up to the output layer. There exist several different algorithms to train such a network and estimate the optimal weights of each neuron, but one of the most famous is called Backpropagation.

The Backpropagation algorithm (BRYSON; DENHAM; DREYFUS, 1963) is a supervised learning technique that receives as input a training set and iterates over its elements in order to estimate the network's internal weights. Before the process begins, all weights are initialized with one of several strategies—e.g. random numbers drawn from a normal distribution, all zeros, or other custom initialization. After samples are propagated forward through the network, the calculated output is compared with the expected one using a given loss function—e.g. mean square error, categorical cross-entropy—and if the generated difference is not satisfactory, the error is propagated back to the previous layer and used to adjust its internal weights. After that, another iteration starts and the process is repeated until a stop criterion is met.

2.2.3 K-Nearest Neighbors

The K-Nearest Neighbors algorithm (KNN) is a non-parametric method normally used when there is little or no prior knowledge about the distribution of the data (FIX; JR, 1951). Its objective is to find the k points closest to a query point in an input space, typically using the Euclidean distance.

Let x_i be an input sample with p features $(x_{i1}, \ldots, x_{ip})$, and n be the total number

of samples ($i = 1, 2, \ldots, n$). The Euclidean distance between sample x_i and x_j is defined in Equation 2.9. The KNN algorithm then searches for the k points x_j which minimize the distance $d(x_i, x_j)$ to a given x_i. In a classification problem, the p input features could be seen as a vector of coordinates in a p-dimension space. KNN is then used to find the points whose input features are closest to a given test point and, through a simple voting system, define its class.

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \ldots + (x_{ip} - x_{jp})^2} \tag{2.9}$$

2.2.3.1 K-Nearest Neighbors with Dimensionality Reduction

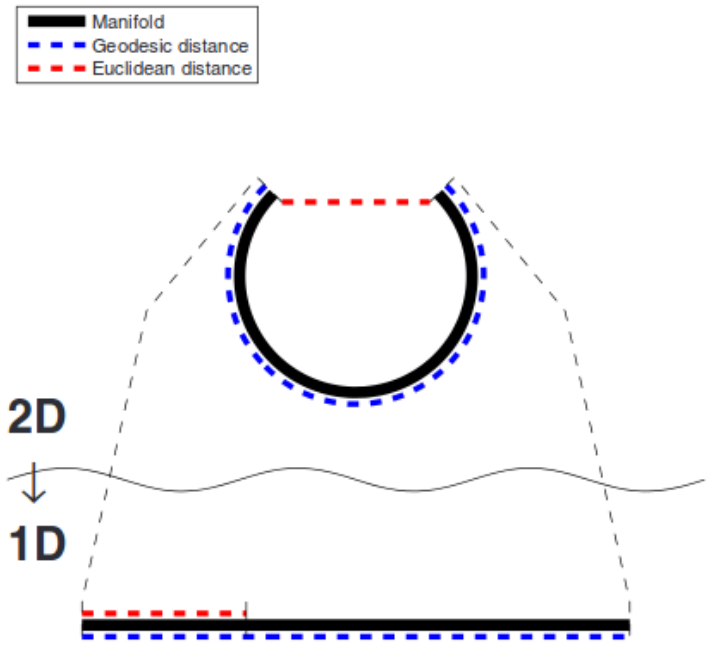
In (BEYER et al., 1999), the authors discuss the effect of the dimensionality of the samples on the nearest neighbor algorithm. According to them, as the vectors' dimensionality increases, the distances to the nearest point becomes equal to the distance to the farthest one. To address this problem, it is possible to use algorithms of dimensionality reduction before applying KNN.

A dimensionality reduction algorithm aims at modeling a high-dimensional dataset as a set of vector points lying close to a low-dimensional nonlinear manifold. Among all the classes of methods, there are those that seek to reduce the dimensionality by preserving distances in the original space; one of the possible criteria to be used in this case is to approximate distances in the original space by shortest paths in a graph, where the graph represents a discretized version of the high-dimensional surface where points lie.

In Figure 2.2 we present a problem where a curved line in a two-dimensional space needs to have its dimensionality reduced to one. Assume that distances between the original points *over the curve* should be preserved—this makes sense since the Euclidean distance between (for instance) the two endpoints of the curve does not in fact represent how close they are under the assumption that points need to necessarily lie over the curve. This problem could be solved by measuring or estimating the distance along the curve (blue dashed line in the figure); this curve is called a manifold. Such distance is also called the geodesic distance, by analogy with the curves of the Earth.

Most of the time the parametric equation of a manifold M is unknown, which makes it impossible to directly find a projection to a lower-dimensional space that minimizes the error in preserving geodesic distances (LEE; VERLEYSEN, 2007). However, some points of M may be known and may be used to approximate the geodesic distance as the length of the shortest path between points y_i and y_j, passing through points $y_k, y_l, \ldots$

Figure 2.2: Example of a 2D curved line reduction to 1D space, comparing the use of Euclidean distance and Geodesic distance.

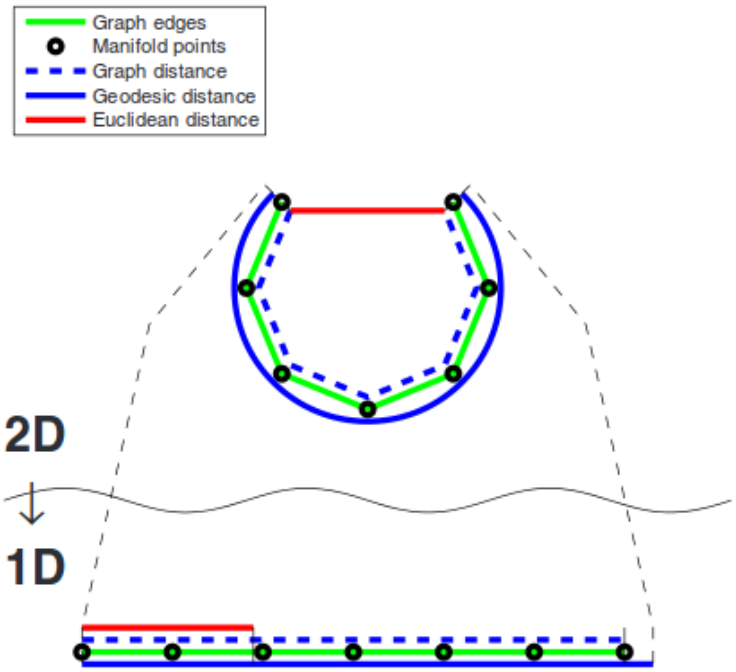


Source: (LEE; VERLEYSEN, 2007)

of M. As the manifold geometry needs to be respected, only small jumps between points are allowed, which means that points from the line/manifold need to be sampled with sufficient density, according to some given restriction. One example of restriction is the K-rule, where a point y_i is only considered to be connected (on the line/manifold) to its K closest points, where K is a constant. The lengths of the shortest paths between pairs of points are called, in the literature, graph distances. Figure 2.3 shows the earlier example, but now with a graph distance demonstration.

The ISOMAP algorithm (TENENBAUM; SILVA; LANGFORD, 2000) is the simplest nonlinear dimension reduction method that preserves graph distances. The basic algorithm starts by determining the nearest neighbors of each point, when considering all points in a training set. Next, it creates a neighboring graph where each point is connected to its neighbors by edges weighted by the Euclidean distance between the nodes. Following, the shortest path between all points is computed using Dijkstra's algorithm, and their squares are stored in a matrix D which will be double centered to generate a gram matrix S. Finally, the spectral composition of S is computed in the form of $S = U\Lambda U^T$, and the new representation (in dimension P) of each point is obtained from the product shown in

Figure 2.3: Example of a 2D curved line reduction to 1D space, comparing the use of Euclidean distance, Geodesic distance and graph distance.



Source: (LEE; VERLEYSEN, 2007)

Equation 2.10.

$$\hat{X} = I_{P \times N} \Lambda^{\frac{1}{2}} U^T \tag{2.10}$$

Once the dimension of points is reduced, it is possible to apply the KNN method over the newly represented dataset.

2.3 Phonetic Algorithms

Another type of algorithm that is relevant for this work is the family of *phonetic algorithms*, which process words and return a more appropriate representation for them; specifically, one that represents/encodes how they are pronounced in the English language. By using such a representation, machine learning algorithms can, for instance, make better classification or prediction decisions based on the sound of words, and not necessarily how they are spelled.

In 1918, Margaret O'Dell and Robert C. Russel patented an algorithm called Soundex (KAUTH, 1973). This algorithm was first created to search English last names that sounded the same, but which could be spelled differently—for instance, SMITH and

SMYTH. Soundex produces an encoding for a string using six phonetic classes of human speech sounds (bilabial, labiodental, dental, alveolar, velar and glottal). This new representation of a word consists of the first letter of the word followed by three digits that together represent the phonetic class of that word.

A new algorithm, called Metaphone, was published in (PHILIPS, 1990). It was created based on observations of variations in the English spelling and pronunciation of a word to produce a more accurate encoding than Soundex. Although it was created with the goal of encode last names, Metaphone also addresses regular words—not just names. The transformation rules table used by Metaphone is shown in A.1.

2.4 String Comparison Algorithms

One possible way of comparing strings is by using string *distance algorithms*, which compute the dissimilarity between two words. Such a dissimilarity is typically defined as the minimum amount of actions necessary to transform one string into another. This type of method is important for this work because phonetic encodings can, themselves, be seen as strings; string comparison algorithms can, therefore, be used to compare phonetic encodings, thereby obtaining a dissimilarity metric over how words sound.

Published in 1966 by (LEVENSHTEIN, 1966), the Levenshtein distance measures the difference between two sequences of characters by calculating the minimum number of character's additions, deletions and substitutions required to transform one sequence into another.

Considering two strings s_a and s_b, whose lengths are, respectively, $|s_a|$ and $|s_b|$, the Levenshtein distance between them is $lev_{a,b}(|s_a|, |s_b|)$ and is defined in Equation 2.11. The term $1_{(a_i \neq b_j)}$ is the characteristic function which is equal to 0 when $a_i = b_j$ and is 1 otherwise. The i and j arguments of the function indicate, respectively, that the first i characters of the first string and the first j characters of the second string should be used

in the comparison.

$$\text{lev}_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{2.11}$$

From the Levenshtein distance, it is possible to derive the Word Error Rate (WER)—another metric that is commonly used in speech recognition and machine translation. The WER is basically the Levenshtein distance between a reference word and a test word, divided by the length of the reference word, as shown in Equation 2.12.

$$\text{WER} = \frac{\text{lev}_{(W_{ref},W_{test})}(|W_{ref}|,|W_{test}|)}{|W_{ref}|} \tag{2.12}$$

3 SPEECH-TO-TEXT METHODS

3.1 Main Approaches

Most STT converters are built on a multi-level structure, from acoustic processing to a pragmatic level, in which the possible resulting sentences associated with the acoustic level are analysed under the light of grammar constraints. In this type of multi-level structure, the acoustic signal is first discretized into samples in order to facilitate the numerical analysis. Speech production is a phenomenon that varies with the time (THAYSE; BINOT, 1991); for this reason, the signal is framed in intervals of typically 10ms, where it can be considered reasonably stationary. These frames are then processed to extract features—e.g. the energy of the signal in different frequency bands—which are saved in a coefficient vector, also known as an acoustic vector.

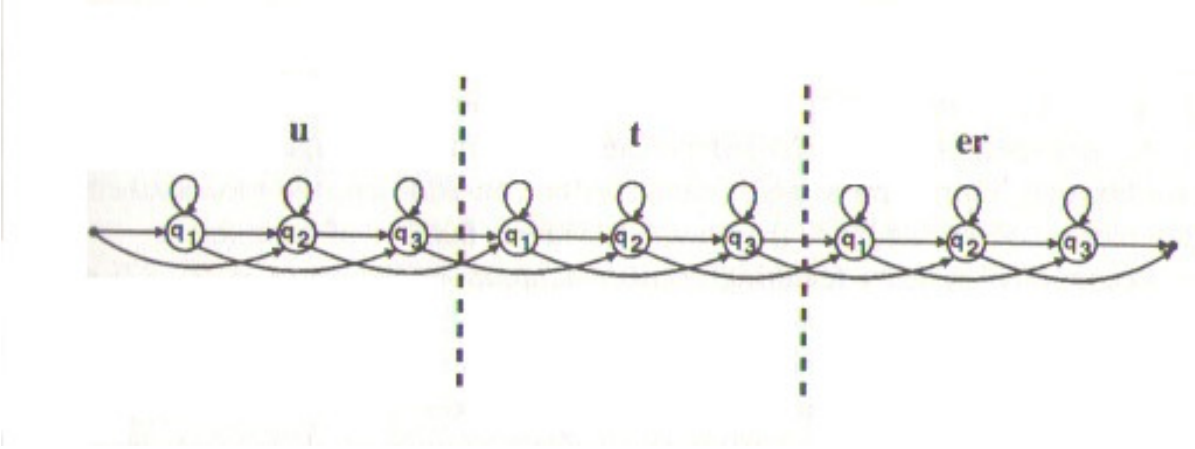
In a processing stage called acoustic decoding, a sequence of acoustic vectors are then transformed into a sequence of words according to an acoustic model. Due to the high degree of variability in a speaker's speech (speed and intonations, for example), most of the first STT systems were constructed using statistical methods—the most common approach being based on Markov chains. However, recent studies have shown that the use of deep learning networks could produce better results in this stage (HANNUN et al., 2014). We discuss more about both techniques in the following sections.

3.1.1 Hidden Markov Models

A Hidden Markov Model (HMM) consists of a sequence of finite states and transitions between them. HMMs are commonly used as phoneme models, where they try to represent the time-dependent characteristics of a single phoneme. Hidden Markov Models representing words or sequences of words, on the other hand, are obtained linking multiple phoneme models, as shown in Figure 3.1. For the intonation aspect of the problem, an emission probability is associated with each state and, whenever that state is visited, it produces a random acoustic vector according to the probability distribution associated to the state. In order to model speaking rates, each transition has an attached probability of occurring. In this way, slow pronunciations are characterized by a large number of loops, while transitions between non-adjacent states represent faster speech.

State transition probabilities are defined via a training procedure on a large num-

Figure 3.1: Phonetic model of the word 'utter': u, t, er



Source: (THAYSE; BINOT, 1991)

ber of collected sounds. After that, a model is generated using a database of sentences, or a sequence of phonemes. Having collected this data, the probabilities are re-estimated via training algorithms that maximizes specific optimization criteria (e.g. the Viterbi or Baum-Welch algorithm). Again, independently of the algorithm used, a large-scale database should be used in order to correctly estimate the state transition probabilities.

One more aspect that needs to be considered is whether the recognition will be speaker-dependent or speaker-independent. The former considers a specific model for each user, allowing the use of smaller databases but forcing every new speaker to pass through the entire training phase. For speaker-independent approaches, models are more sophisticated and it becomes crucial to have access to large and representative databases.

3.1.2 Neural Networks and Deep Learning

In (HANNUN et al., 2014), a speech recognition strategy is proposed using an end-to-end neural network-based approach called deep learning. Such approach performed better than the classical one described in the previous section, especially in noisy environments; furthermore, it was not based on hand-designed components constructed to model noise, reverberation and speaker variations. The proposed solution is based on a highly trained and optimized recurrent neural network that predicts the most likely transcriptions given a voice signal.

According to the authors, the produced transcriptions were, most of the time, exactly the same as the spoken sentence, and the errors produced by the network tended to be phonetically plausible regarding the English words. Transcription errors were often

produced when the interpreter tried to recognized words that were not well-represented (or were not present) in the training set. As the construction of a training set with all the possible pronunciations of each word in the English language is impractical, researchers integrated the network output to an N-gram language model. This type of model takes into account the previously classified terms when trying to classify the next term in a sentence. In other words, the model takes context into account when performing a transcription. The method takes N-1 previous terms into account: a 2-gram, or bigram, considers the last recognized term, a 3-gram considers the last two recognized terms, etc. The system, then, aims at finding the sequence of characters that explains the spoken sentence by trying to maximize a weighted sum of the probabilities outputted by the RNN. The weights used in the optimization criterion are set using cross-validation.

To train their models, (HANNUN et al., 2014) collected more than 5000 hours of read speech from 9600 speakers without background noise. As the objective of the proposed solution was also to improve accuracy in noisy environments, when compared to classical solutions, and since acquiring recordings under these situations is not necessarily easy, the authors generated artificial background noise by using audio superposition techniques. Basically, they used a speech audio track and a noise audio track and summed them up, adding reverberation, echoes and other damping techniques when necessary. To prevent the network from memorizing the noise and ignoring the original track, the researchers mixed small clips from different noise recordings before adding them up to the speech audio track. Another care taken by the researchers was to reproduce what is called as the *Lombard effect* (JUNQUA, 1993). This effect refers to when the speaker actively changes its pitch to overcome background noise. In order to replicate this effect in the training set, recorded speakers wore a headphone that would randomly produce loud noisy sounds which induced them to inflect their voices. The results obtained by using the technique and dataset presented in (HANNUN et al., 2014) outperformed other market interpreters available at the time under quiet and noisy backgrounds.

3.1.3 Language Models

Even when using sophisticated techniques to address the acoustic phase of transcription, the result produced by the model could still be different from what the speaker said. For instance, the sentence

I have two ice creams

could be translated to

Eye have to I screams.

To reduce transcription error, the use of a *language model* is necessary in order to introduce linguistic constraints to the results (YOUNG et al., 1989). Several categories of methods were proposed in the literature, raging from elementary ones to more sophisticated ones.

The more rudimentary methods are based on statistics of the language they are trying to analyze instead of language theory—such as grammars or syntactic structures. They assign higher probability to word sequences that are more common in the training corpus. This represents a major drawback since the training corpus must be statistically representative of the context that must be modeled.

Other language models were proposed in the literature, such as the Deterministic models. These models are based on formal theories of natural language. One example of those theories are formal languages, where rules are defined—usually in a form of a grammar—to describe acceptable syntactic structures that, though approximated, can cover large fragments of a natural language. Even though this can reduce the number of possible sentences that can be captured by the model, more restrictive models can also better lower the probability of erroneous recognitions.

A commonly used type of grammar, which usually forms the base of more complex models, is the context-free grammar. For instance, some examples of context-free grammars are *generalized phased structure grammars*, *lexical-functional grammars*, and *definite clause grammars*. They, among other benefits, provide semantic constrains based on the context of the natural language covered that helps to refuse meaningless syntactic correct sentences.

It is important to notice that higher quality language models can also contemplate correlations across the sentence margins, where *margins* represent the sentences surrounding the sentence being analyzed. In other words, these models can contemplate the *context* where the sentence occurs. As an example of this concept, consider sentences where there exists an anaphoric pronoun, such as “She” in

She is a good kid

whose meaning depends on information from adjacent sentences. For instance, the theory of representation of discourse takes these situations into account (KAMP; REYLE, 2013).

In this work we will also experiment the use of sentence margin information, introducing the information about adjacent words in the form of *context*.

3.2 Related Work

In this section we review work that is related specifically to the goal of improving base STT models so that they can be used to transcribe voice recordings originating from narrower, more specialized domains.

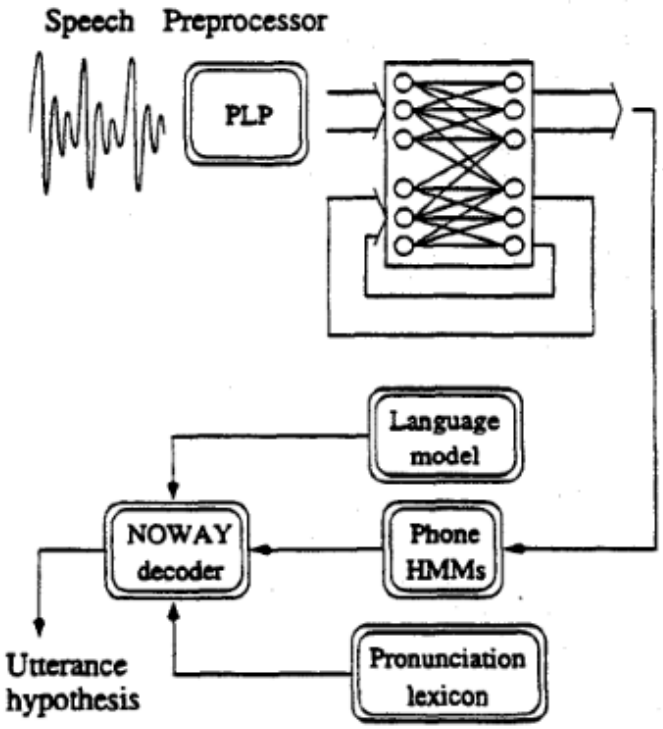
3.2.1 Boosting In Acoustic Models

When constructing traditional HMM systems, training data needs to be carefully selected, otherwise training time could be increased significantly but provide only marginal gains in performance. (COOK; ROBINSON, 1996) address this problem by using recurrent neural networks to generate the transitions probabilities and feed them to a HMM acoustic model. Figure 3.2 shows the general structure of the solution. The authors argue that the original boosting procedure is appropriate to static pattern recognition problems, but that it cannot be directly used in speech recognition problems due to the temporal problems linked to the speech recognition.

To address this problem the authors created and evaluated a series of different boosting procedures for use with speech data. The first proposed solution (*boost1*) starts by training a network on a randomly-selected subset of the training set and estimates the frame recognition error rate on the rest of the sentences. Then, it creates a training set which is composed of sentences which produced the highest recognition error rate and of sentences on which the STT system performed well; this dataset is then fed to to a second network, which focuses on improving prediction accuracy on these harder-to-transcribe samples. The second proposed system (*boost2*) also trains a network with a randomly subset of data but then feeds to a second network only the samples with highest error rates in the first network. The last approach proposed by the author (*boost3*) uses a NN to produce the probabilities needed to define the stochastic connections of an HMM. This HMM is then used to transcribe the sentences and only the ones with higher error rates are sent to train a second network.

As stated before, in Chapter 2, the original boosting framework uses a simple

Figure 3.2: Hybrid Connectionist-HMM Speech Recognition System



Source: (COOK; ROBINSON, 1996)

voting scheme to combine the outputs of a chain of weak learners; however, as explained in (COOK; ROBINSON, 1996), the authors wished to interpret the outputs of the NN's as posterior probabilities, having to use, instead, linear combination to combine the neural networks outputs, as shown in Equation 3.1. In the Equation, $y_i^{(k)}$ is the i^{th} output of the k^{th} neural network and all the βs sum up to one (i.e. $\sum_{k=1}^{K} \beta_k = 1$) and are non-negative (i.e. $\beta_k \geq 0$). In this work, the authors used a simple average $\beta_k = 1/K$.

$$y_i = \sum_{k=1}^{K} \beta_k y_i^{(k)}. \tag{3.1}$$

The results obtained in (COOK; ROBINSON, 1996) were compared both with the classical structure of speech recognition (*baseline* with no boosting) and with random selection of new training data. Table 3.1 shows the WER the methods produced.

3.2.2 The OKARD Project

A more recent approach aiming at improving weak STTs via machine learning techniques is the OKARD project. The OKARD project was developed in 2015 in Sophia

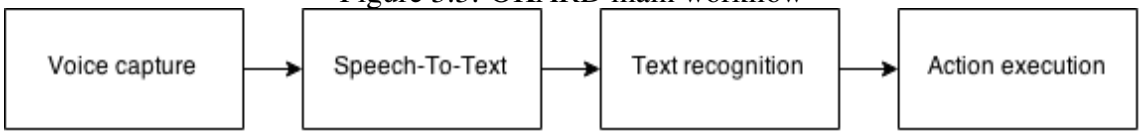
Table 3.1: Comparison of the different boosting procedures in acoustic models, as proposed by (COOK; ROBINSON, 1996)

Boosting procedure	*Word Error Rate*	*Improvement over baseline*
baseline	11.2%	-
random	10.4%	7.1%
boost1	9.1%	9.8%
boost2	9.5%	15.2%
boost3	10.4%	7.1%

Source: (COOK; ROBINSON, 1996)

Antipolis, France, by the Amadeus company. As discussed earlier, the main objective of this project was to propose a voice-controlled feature for ARD that could improve the performance of operators and also system accessibility; the research team tried to achieve this goal by using a machine learning technique to improve the base STT transcriptions originating from a black-box interpreter. Figure 3.3 shows the main work flow used in the solution.

Figure 3.3: OKARD main workflow

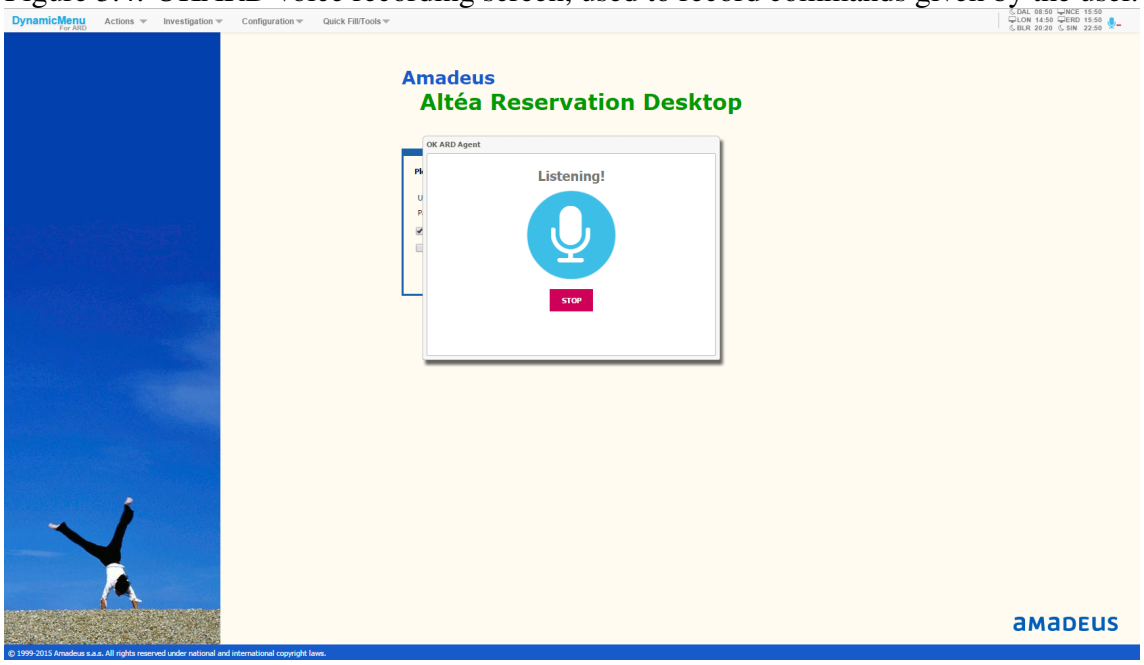


The black-box STT interpreter used by OKARD was the Google's speech recognition webservice. The main arguments in favor of this choice was the reputation of Google's recognition system and their well-designed communication API, which allowed the OKARD research team to quick prototype different proof-of-concept applications. Figure 3.4 shows an example of the OKARD recording application, waiting for the user to say a command.

After the capture of voice recordings and translation, the string outputted from the base STT was encoded into a metaphone, in order to bring it to the phonetic domain. This decision showed to be well-suited for the problem, since experiments made during the execution of the project showed that when preparing inputs to the learning method a phonetic comparison between the desired voice command and the translation produced much better results than a direct comparison between strings.

The OKARD recognition process was implemented using a fully-connected feed-forward network with one input layer, one output layer and a single hidden layer. The hidden layer had 20 neurons, all of them using a sigmoid function as the activation function. The recognition process was made phoneme by phoneme, and the input features presented to the network were the Levenshtein distances between the current phoneme

Figure 3.4: OKARD voice recording screen, used to record commands given by the user.



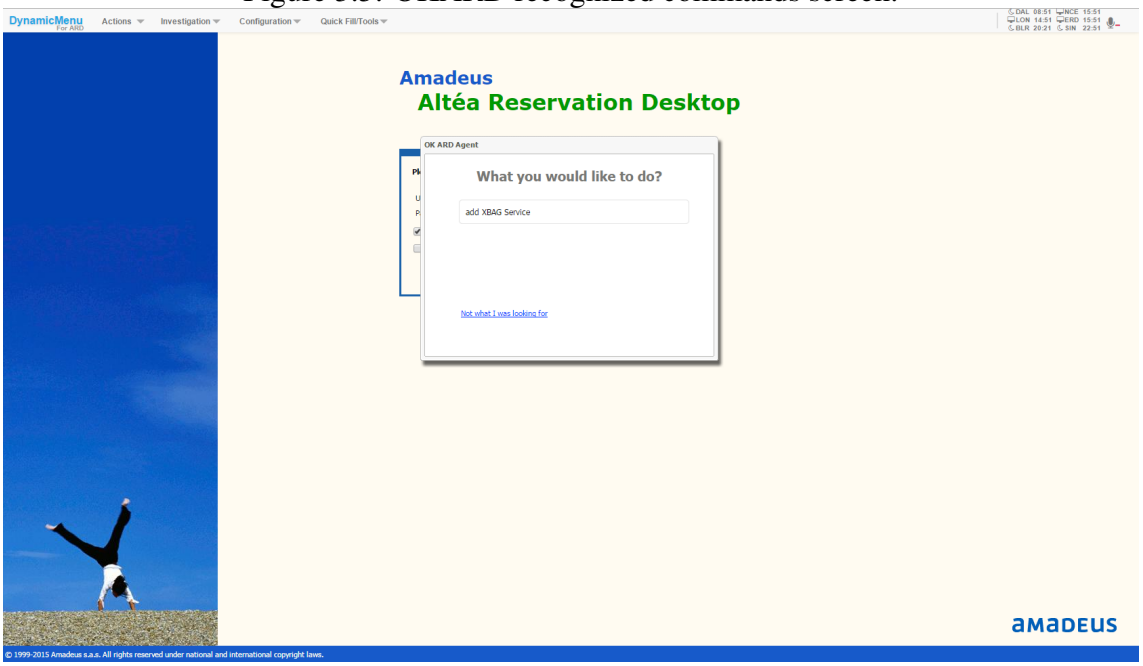
being processed and all known phonemes in the system's vocabulary. In total, there were 28 known words. The network's output was composed of one neuron per known term—28 in total—and corresponded to a floating point number between 0 and 1. The system would then select only the outputs with values higher than a manually-tuned acceptance threshold[1].

For each recognized phoneme, the system would filter the list of possible commands consistent with the sound and, in the end, present to the user all the commands whose terms were above the acceptance threshold. Figure 3.5 shows an example of the OKARD options screen; once the user selected an option, the corresponding example (i.e., the example corresponding of the the list of phonemes and the recognized term) would be included into the training set. This was done so that by retraining the network, recognition accuracy could be improved which would (hopefully) reduce the number of candidate options presented to the user in future interactions.

As explained before, unknown terms (or less commonly used sentence structures) can make a general-purpose STT produce incorrect transcriptions. Most of the time, the sound of a word could get mixed with the sounds of the next one, thereby producing a prediction of a completely different word. To address that problem, the OKARD team decided to create an algorithm that could automatically aggregate and separate phonemes

[1] In this work we extend the basic OKARD architecture and describe, in Chapter 4, a method for removing the dependency on this parameter.

Figure 3.5: OKARD recognized commands screen.



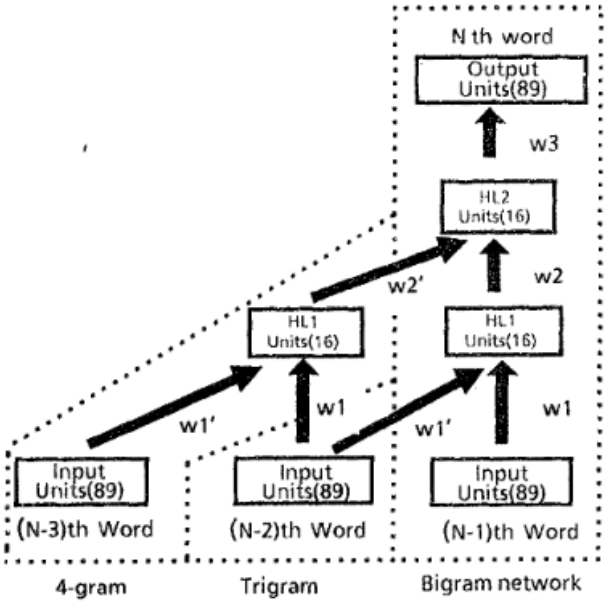
into groups representing the most likely word they correspond to, thereby deciding how the output from the base STT should be broken-down. Again, this algorithm was based on thresholds manually-tuned in order to decide whether a given analysis strategy was the right one.

At the end of the project, the best accuracy obtained by the proposed boosting mechanism was of 75%. The research team came to the conclusion that the main (but not only) cause of such result were the manually-tuned parameters used to define both the acceptance threshold of the NN outputs and also how to break down the input into a sequence of phonemes.

3.2.3 NETGram

Recognizing English words is a difficult task if one uses only their acoustic characteristics. Because of that, word recognition systems often use linguistic information, such as word categories. As discussed in (BROWN et al., 1992), traditional voice-recognition systems often use probabilistic methods such as the n-gram language model, which treat two sequences as being equivalent if they end in the same $n-1$ words; i.e, for $k \geq n$, $P\left(\omega_k|\omega_1^{k-1}\right)$ is equal to $P\left(\omega_k|\omega_{k-n+1}^{k-1}\right)$. More concretely, the parameter n indicates the the order of the Markov model associated with the process—in particular, it is a $n-1$ order Markov process.

Figure 3.6: NETGram structure for 4-gram, trigram, and bigram networks.



Source: (NAKAMURA et al., 1990)

(NAKAMURA et al., 1990) present a method called NETGram; NETGram is a word category predictor that uses neural networks that, according to the authors, has comparable accuracy to the traditional n-gram approach, but requiring fewer training data. The authors proposed the use of a feed-forward neural network with two hidden layers for dealing with the bigram case. As n increases, a new input block is created and is fully connected to the first hidden layer of a basic 2-gram network. The hidden layers have 16 neurons each. Figure 3.6 shows the structure of the 4-gram, 3-gram, and 2-gram networks.

In (NAKAMURA et al., 1990) the authors also discuss the amount of memory required by traditional probabilistic methods. As these systems work based on table-lookup, a trigram stochastic method would have to store their probabilities in a $89 \times 89 \times 89 = 704969$ size table—in case there were 89 word categories. The NETGram system, on the other hand, only has to store its free parameters—weights and biases of the neural network; in the case of a trigram network, that means $(89 + 89) \times 16 + 16 \times 16 + 16 \times 89 + 121 = 5193$ parameters.

Note, finally, that the amount of parameters of a network influences the amount of training data needed to train those models. If the model has fewer parameters, it may in theory be trained with fewer example. Using 512 training sentences, NETGram shows a recognition rate of 86.3%, while the n-gram statistical model obtained 85.5%.

4 A ML FRAMEWORK TO IMPROVE BLACK-BOX STT RECOGNITION

In this chapter we introduce three alternative frameworks for achieving the goal described in Section 1.2. The first section of this chapter explains the data flow, input features, and other hypothesis employed in the development of the proposed techniques. The remaining sections introduce different ways of solving the problem, presenting pros and cons of each specific technique and explaining how it may be used to implement a type of Boosting. In the subsequent chapters of this work (Chapters 5 and 6) we evaluate the performance of each proposed technique when applied to a STT problem involving the OKARD system.

4.1 Solution Overview

As discussed in Section 3.2.2, one of the main assumptions regarding the cause of low accuracy in the OKARD project is the use of manually-tuned algorithms to control the translation analysis. The process of correcting the output of a base STT system cannot be effectively done by analyzing individual phonemes or individual words, since *1)* the identification of domain-specific terms and sentences may depend on their contexts; and *2)* the phonemes outputted by a general-purpose STT system, as a response to a given recorded audio, may incorrectly combine the sounds of separate words and merge them as if they were a single word. This means that the transcribed output generated by the black-box system may be interpreted as a *sequence* of phonemes that are, in some situations, incorrectly grouped. As an example of an incorrect grouping, consider a situation where a user says “Add remark” and the STT system transcribes this sentence as “A dream arc”. At a first glance, both sentences have nothing in common, but by analysing their metaphone encodings (*AT RMRK* and *A TRM ARK*, respectively) it becomes clear that they share the same phonemes—-except that they were incorrectly grouped. During the analysis of a given input, then, the system does not know *a priori* how to group those phonemes in a way that correctly detects whether they belong to a single word. The solution selected by the OKARD research team, as described in the last chapter, was to use a hand-tuned algorithm that could evaluate different candidate grouping of the phonemes before feeding them to the neural network.

In this work, we propose to replace this manually-tuned approach used by OKARD in the control of the input analysis by a self-trained one. The basic idea is that instead of

having transcription errors caused by inaccurately-tuned procedures for breaking down or grouping phonemes, a machine learning decision system will dictate the grouping strategy by which the machine learning method will consume input items/phonemes (from a sequence of inputs) in order to try to better transcribe a given spoken sentence. The system that implements this decision is depicted in Figure 4.1 and is composed of two main decision blocks; we call these the Decision 1 and Decision 2 blocks (or stages), respectively.

The first decision block, *Decision 1* (depicted in blue), is responsible for controlling the transcription analysis. It acts as a state machine that decides how to group phonemes before moving on to the next decision stage (*Decision 2*). In other words, this stage is trained via a supervised learning technique to decide whether *1)* the current phoneme P_i should be classified by Decision 2, and the system should process the next term in line (action *Next*); *2)* the current phoneme should be classified by Decision 2 but kept in the system's input since it is likely relevant for classifying the next term (action *Stay*); or *3)* the current phoneme should be concatenated with the next phoneme (P_{i+1}) before being sent to be classified by Decision 2 (action *Aggregate*). The Decision 1 stage can also decide to *Reject* a phoneme, in case it suspects that the phoneme should have been aggregated into a previous term, which was already analyzed by Decision 2. In this case, the transcription process identified an inconsistency and rejects the input. It is also important to clarify that the mapping between Google's STT and correct terms is implemented by the Decision 2 stage; Decision 1 is only concerned with identifying a correct phoneme grouping strategy.

The second decision block, *Decision 2* (depicted in red), corresponds to the second decision made by our overall architecture. It consists of a term classifier. Concretely, it receives the same inputs as the Decision 1 stage and indicates as output to which known term this phoneme belongs. This is functionally similar to the prediction algorithm used in the OKARD project, but in our proposed method it is implemented differently: new activation functions, different neural network architectures, and different representations for the input features.

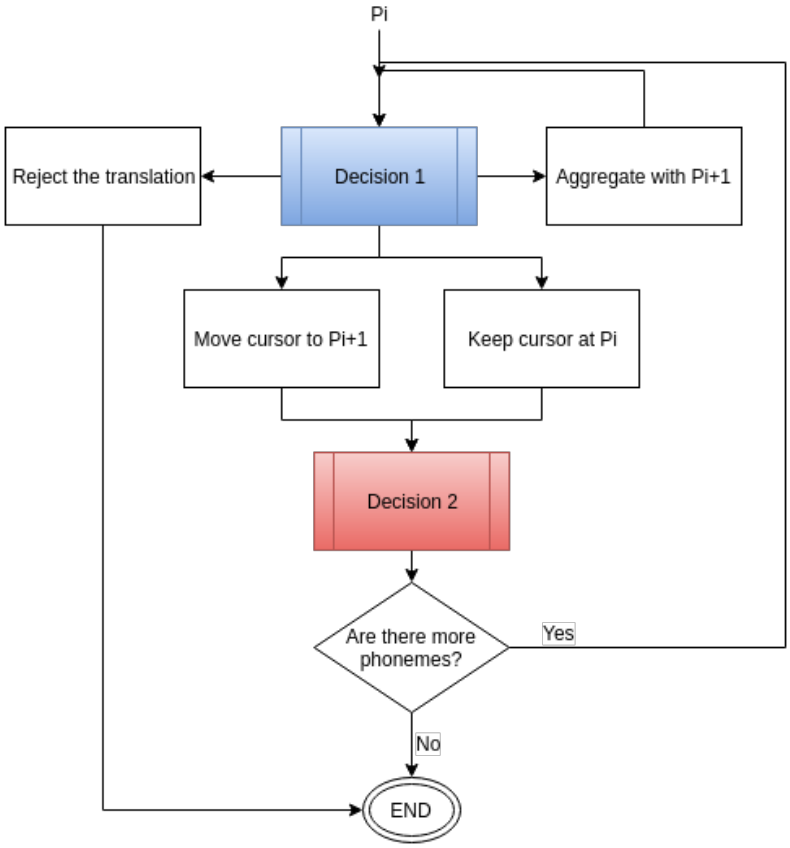
There are several strategies that could be used to develop each of these decision layers, or decision blocks. In the following sections we will discuss some of them and later, in Chapter 6, we show an empirical performance evaluation corresponding to using different algorithms for Decision 1 and Decision 2. As described in Chapter 5, we then use those results to choose which combination of algorithms to use to construct the most

accurate overall method for improving a base weak STT. The final results obtained are discussed in Chapter 6.

In this work we implement the decision stages of the method via a set of features similar to those used in OKARD. Specifically, the main input features used correspond to numerical comparisons of the current phoneme being processed to all the known words. This time, however, inspired by the NETGram idea (explained in Section 3.2) we extend the network by adding information about the previous terms given to the network, so as to serve as contextual information. The contextual input features are implemented via a binary vector (with size equal to of the number of terms in the domain vocabulary); all entries are zero, except the one corresponding to the last identified term presented to the network. A comparison of the impact of using contextual information in our prediction algorithm it is presented in the following chapters.

Finally, we will also discuss how to more effectively use the available training data for the implementation of our strategy. Usually, as discussed earlier, a learner in the boosting chain pays higher attention to examples that were misclassified by previous weak learners. One of our interests in this work is to check whether presenting *only* misclassified examples to latter learners in the boosting chain improves performance, or whether all examples (though possibly with different weights) should be presented.

Figure 4.1: Dataflow used to implement our proposed boosting algorithm.



4.2 Multinomial Naive Bayes

In order to evaluate different machine learning algorithms as ways of implementing the decision stages of our architecture, we first analyzed the use of a probabilistic classifier. As explained in Section 2.2.1, Multinomial Naive Bayes (MNB) is a supervised method that uses Bayes' theorem to classify new examples.

According to (RENNIE et al., 2003), even though this algorithm is often used with word vector counts—discrete data—term frequency-inverse document frequency (a continuous value) also works well in practice. (MCCALLUM; NIGAM et al., 1998) discuss the effectiveness, both in terms of accuracy and time efficiency, of the resulting Naive Bayes algorithm (now applicable over non-discrete data, despite the simplified assumptions it makes).

Taking this into account, we believe that the use of this method to implement the decision stages of our architecture may outperform others concerning the training time. On the other hand, we anticipate that the method could have as shortcoming the fact that it does not deal well with weakly represented cases in the training set, requiring, perhaps,

more samples then we dispose. Also, the non-determinism of Google's speech recognition webservice (discussed in more details in the next chapter) may impose problems as it would demand more samples to better characterize the learning domain at hand (KIM et al., 2006).

4.3 Feed-Forward Neural Networks

The use of a connectionist cognitive model is currently in vogue because of the recent successes of deep learning techniques. This suggests that using neural networks as a machine learning method to boost STT translations may be a good idea. Its parametric configuration enables us not only to correctly classify unseen examples, but also to save memory, as the training set can be discarded once the parameters are trained.

Even though it was the center of OKARD project, we believe that the use of various types of neural networks can be investigated in more depth when designing a solution to the problem being studied. In the work developed at Amadeus, for instance, a careful study about the impact of different properties of the NN, such as the number of hidden layers and neurons, was not made; similarly, that work also did not evaluate the use of different activation functions in order to compare the accuracy results obtained by them.

4.4 K-Nearest Neighbors

In contrast to the neural networks, we also decided to test the use of a non-parametric approach to implement the decision states of our architecture, since it is conceptually simpler than parametric approaches such as NNs. The K-Nearest Neighbors algorithm is a method that fits well with this requirement, since there is no model to be learned, and hence the algorithm is simpler to be applied. The idea here is to assume that the training and test data are vectors lying in a high-dimensional space; under this assumption, for each test data, we will compute their K-nearest neighbors and, via a simple voting system, decide the class (i.e. the identity of the term or phoneme) corresponding to current test input.

As explained in Chapter 2, for high-dimensional spaces, Euclidean distance loses its relevance: the distance to the nearest neighbors gets close to the distance of the farthest one. To address this problem, we are going to evaluate using a dimensionality reduction

algorithm in the training and test sets, as a pre-processing step, before running the KNN algorithm.

The method chosen to perform this reduction was ISOMAP. Such a method is fast and tends to preserve the graph distance[1] between them. Since we believe that the input points in our training set lie on a non-linear high-dimensional curve, it could be beneficial to apply a ISOMAP reduction. On the other hand, if this assumption is incorrect, then using an approach based on preserving graph distances for the dimensionality reduction may produce even worse results than applying KNN directly. In the experiments to be discussed in the following chapters, we also experiment with different amounts of neighbors considered by the ISOMAP procedure, and also with the target dimension for the reduction.

[1] More on ISOMAP and graph distance in Chapter 2.

5 EXPERIMENTAL METHODOLOGY

In this chapter we describe the methodology employed to develop the empirical analyses of the techniques proposed in Chapter 4. We discuss how the base STT method was structured in terms of available commands, then discuss how data was collected, and present the ways in which each method was be evaluated—how performance is measured and how we quantify whether the method succeeded in the task at hand.

5.1 Domain

All the experiments in this work were executed using the same list of commands and terms. This list was assembled in order to have a sufficient heterogeneous set with a combination of commonly used terms (e.g. “Add”, “Open” and others), and application-specific ones (e.g. “PNR”, “FOP” and “XBAG”). We also defined some commands with the multiple terms, and others as single-term commands. The list of commands can be seen in Table 5.1.

Table 5.1: Command list used in our experiments.

Number	*Command*
1	Add remark
2	Search flight
3	Focus section
4	Remove passenger
5	Add XBAG service
6	Go to FOP
7	Issue ticket
8	Open PNR
9	Open TST
10	Redisplay PNR
11	Quit
12	Save

From the command list, it is possible to extract the Table 5.2, where we show all distinct terms in it and compute their metaphone encoding.

Table 5.2: Terms list and their metaphone encoding.

Number	*Term*	*Metaphone*
1	add	AT
2	remark	RMRK
3	search	SRX
4	flight	FLT
5	focus	FKS
6	section	SKXN
7	remove	RMF
8	passenger	PSNKR
9	xbag	SPK
10	service	SRFS
11	go	K
12	to	T
13	fop	FP
14	issue	AS
15	ticket	TKT
16	open	APN
17	pnr	NR
18	tst	TST
19	redisplay	RTSPL
20	quit	KT
21	save	SF

5.2 Collected Training Data

As stated earlier, one of the motivations for developing this work is to implement a voice-command feature without the necessity of large amounts of data for creating an accurate STT from scratch. For this reason, the recordings and translations used in these experiments are not many and, therefore, try to replicate *only* the situation pictured in the early chapters—one where we try to improve a weak STT by training a second layer of prediction from a limited number of examples.

That being said, it is not one of the objectives of this work to propose a solution that will necessarily be 100% accurate—especially considering the possible need to train the stages of our framework with limited data. We may, in fact, be satisfied with a solution that has decent (but not perfect) accuracy, as long as it is sufficiently good to implement a proof-of-concept application demonstrating that the approach is promising; from there, the architecture could be improved with more data and better design decisions.

We recorded 13 different subjects, men and women, all of them Brazilians and with different levels of accent when speaking English. Those subjects recorded—under negligible background noise environment— all of the commands in Table 5.1; these re-

sulted in a total of 156 different audios.

We chose to use as base weak learner the black-box STT provided by Google's speech recognition webservice. An interesting study made by (ADORF, 2013) shows that on a word level, Google correctly recognized 74% of 11540 words, but when entire sentences were used, only 21% of 1444 spoken ones were correctly recognized. However, that study counted as correctly recognized sentences only those where the STT interpreter outputted the *exact same sentence as expected*, ignoring the phonetic domain—which are considered in our work. Another important point shown in (ADORF, 2013) and also in our own experiments, is that the chosen STT interpreter is not deterministic. The same sound can be played several times and different translations can be produced. That being said, each one of the recordings in our training set was played and translated by the base interpreter 5 times, given us a total of 780 translations.

As explained in Chapter 4, the input features used in both decision stages are the distances from the metaphone encoding of a given term t_i, as outputted by the STT interpreter, and the metaphones of all known terms in Table 5.2. Note that these inputs also include contextual information about the last term recognized by the network (as explained in Chapter 4). As we have two stages in the recognition process (one which controls the iteration over phonemes and one which tries to classify them into terms), we analyzed every sample translation in the dataset and manually *tagged* (or labeled) them, word by word, with the correct decisions of decision stages 1 and 2 of the architecture. An extract of the tagged/labeled samples in our training set can be seen in Tables 5.3 and 5.4. The former table shows a list of 7 examples where a command was spoken and its respective translation, as outputted by Google's STT; the latter table shows the result of the manual tagging/labeling process. We will use some of the examples in these tables to explain, in what follows, how our manual tagging process was performed.

Table 5.3: Extract from the constructed training set. In these examples, we show the command that was spoken by the user (*Spoken command* column) and the respective output generated by Google's STT (*Translation from Google's STT* column).

	Spoken command	*Translation from Google's STT*
1	search flight	search flight
2	remove passenger	remove passenger
3	add xbag service	add Xbox service
4	add remark	adrimar
5	add remark	agri-mark
6	redisplay PNR	British play piano
7	add remark	Andrew Marc

Table 5.4: Extract from the training data manually tagged with the expected correct outputs of Decision 1 and Decision 2.

	Term	*Context*	*Decision 1*	*Decision 2*
1	search	-	Next	search
	flight	search	Next	flight
2	remove	-	Next	remove
	passenger	remove	Next	passenger
3	add	-	Next	add
	Xbox	add	Next	xbag
	service	xbag	Next	service
4	adrimar	-	Stay	add
	adrimar	add	Next	remark
5	agri-mark	-	Stay	add
	agri-mark	add	Next	remark
6	British	-	Aggregate	-
	play	-	Reject	-
	Britishplay	-	Next	redisplay
	piano	redisplay	Next	pnr
7	Andrew	-	Stay	add
	Andrew	add	Aggregate	-
	Marc	add	Reject	-
	AndrewMarc	add	Next	remark

In the extract shown in Table 5.4 it is possible to observe a great variety of cases in the system. There are three possible cases:

1. when the output generated by Google correctly guesses the number of terms and the terms themselves;
2. when the output generated by Google correctly guesses the number of terms but translates at least one term incorrectly;
3. when Google guesses the number of terms incorrectly—thereby merging two spoken words into a single one that is phonetically similar, or splitting a spoken word into more than one word.

Consider translation 3, where the user spoke “Add Xbag Service”. This was incorrectly transcribed by Google as “Add Xbox Service”. Because of that we will add three examples to the training set: one specifying that when Google produces the term “Add” and the system has no contextual information (it is the first word of the sentence), Decision 2 should map it to “Add” (since a correction is not needed) and that Decision 1 should read the next output of the STT (in this case, “Xbox”). A second training example is added, specifying that when Google produces the term “Xbox” and the system *has* contextual information (that the previous term was “Add”), that Decision 2 should correct

the term to “Xbag” and Decision 1 should read the next output of the STT (in this case, “Service”). A final training example is added to the dataset, specifying that when Google produces the term “Service” with contextual information (specifically, that the previous term was “Xbag”), that Decision 2 should map it to “Service” (since no corrections are needed) and Decision 1 should read the next output of the STT—in this case, none.

Now consider a more complicated case—translation 4—where the user spoke “Add Remark” but it was incorrectly transcribed by Google as “Adrimar”. We will add two examples to the training set: one specifying that when Google produces the term “adrimar” and the system has no contextual information (it is the first word of the sentence), Decision 2 should map it to “add” (since a correction to Google’s output is needed) and Decision 1 should *not* move to the next output of the STT (i.e., it should *Stay* in the same input word). A second training example is added, specifying that when the system processes the term “adrimar” and it *has* contextual information (that the previously classified term was “add”), Decision 2 should correct it to “remark” and Decision 1 should be read the next word in output of the STT—in case of translation 4, no words are left to be analyzed.

Finally, consider the case of translation 6, where the user spoke “Redisplay PNR” but it was incorrectly transcribed by Google as “British play piano”. We will add four examples to the training set: one specifying that when Google produces the term “British” and the system has no contextual information (it is the first word of the sentence), Decision 1 should aggregate the corresponding phoneme to the next one in the input, and Decision 2 should not be executed. The reason for this is that we want to teach the system that it is unlikely that a valid command would start with the word “British”, and that it should therefore aggregate the corresponding phoneme to the next one in line—in hopes of generating a more likely phoneme that could correspond to the first word for a command. Furthermore, the first training example also specifies that Decision 2 should not be executed, since for this specific term (“British”) the system just concluded that its phoneme is most likely part of a larger word, and therefore should not be individually mapped to a term. A second example is added, specifying that if the term “play” is processed without any contextual information, that it most likely corresponds to an error—it most likely corresponds to a phoneme that in reality belongs to the previous word. In this case, we tag the example with information so that Decision 1 simply discards the term (action *Reject*), and Decision 2 does not try to map it to a known term. A third training example is added, which specifies what to do when the term “Britishplay” is encountered.

Note that this term was generated by the aggregation process performed by our system, based on the previously seen phonemes, in order to fix the incorrect phoneme separation produced by Google's base STT. We tag/label this example in order to teach the system that Decision 2 should map "Britishplay" to "redisplay", and that Decision 1 should move to the next output of the STT—in this case, "piano". Finally, a fourth training example is added, which specifies that when the system encounters the term "piano" with contextual information (specifically, that it follows the term "redisplay"), it should be mapped by Decision 2 to "pnr", and that Decision 1 should move to the next word in output of the STT—in case of translation 6, no words are left to be analyzed.

After we completed the the manual tagging process of the training examples, the training set was ready to be used. With all the words directly extracted from Google's transcriptions, plus the training examples manually generated by the tagging process, the complete training set contained 1855 training samples.

5.3 Evaluation Methods

As discussed in Chapter 4, our proposed solution is composed of two decision stages. Therefore, the experiments conducted here are split into ones that separately evaluate the learned solutions for Decision 1 and Decision 2. Each decision will be tested by using the learning methods described previously; in our final test, we evaluate the combination of the best performing selected methods (and their hyper-parameters) in order to implement an overall composite solution to the transcription problem at hand.

In order to properly evaluate each proposed solution in both decision stages, and in particular to prevent overfitting, we employed a 10-folds cross-validation technique. *Cross-validation* consists in a technique for properly evaluating the generalization capability of a learning method, and works by dividing the set of examples into k randomly selected subsets (or *folds*); it then executes the training process k times, selecting, at each time, a different fold for testing purposes and training the model with the remaining data. The average performance on each of the k runs is then computed, together with its standard deviation. This process—of evaluating the generalization of the model when constructed under different training sets—helps to avoid overfitting, particularly in scenarios where artificially high performance may be achieved by constructing a model that fits too closely a particular set of training elements. Although in this work we use the number of folds most frequently advised in the literature ($k = 10$; (RODRIGUEZ;

PEREZ; LOZANO, 2010; KOHAVI et al., 1995)), we also experimented with different number of folds. Smaller numbers of folds typically result in performance estimates with small variance but strong bias, while larger numbers typically showed small bias but large variance.

In the following experiments (i.e. when evaluating a particular algorithm as a way of implementing either Decision 1 or Decision 2), every fold is used once as test fold, in order to measure the performance obtained by using the remaining data as training examples. We repeat this process for each fold and the average performance and corresponding standard deviation are computed and analyzed. To evaluate the impact of using different algorithm hyper-parameters (e.g. different number of hidden layers in a neural network, input data representation, activation functions, etc) we test several values and analyze their resulting performances. Our work focuses on measuring, in particular, the *overall error rate* resulting from the use of the selected learning methods, but a discussion about training time and execution speed is also presented. For the specific case of evaluating the generalization performance of neural networks, extra care was taken in order to prevent overfitting. All the tests were conducted using the Neural Network toolbox of MATLAB, which uses an early stopping technique[1] when training neural networks.

Once the best learning methods for implementing Decision 1 and Decision 2 and their respective hyper-parameter configurations are defined, an accuracy test is executed, using the complete system combining both decision stages. This is evaluated in terms of its error rate over the collected samples.

[1]More information about this technique can be found in (PRECHELT, 1998).

6 RESULTS

In the first two sections of this chapter we will present the performance results when applying the algorithms discussed in Chapter 4 to each of the decision blocks. We also present different variations of their hyper-parameters and configurations, thereafter comparing error rates in the classification process. Finally, in the last section, we present the results of the entire composed transcription solution, when using the best selected algorithms and their corresponding hyper-parameters.

6.1 Performance of the Decision 1 Stage

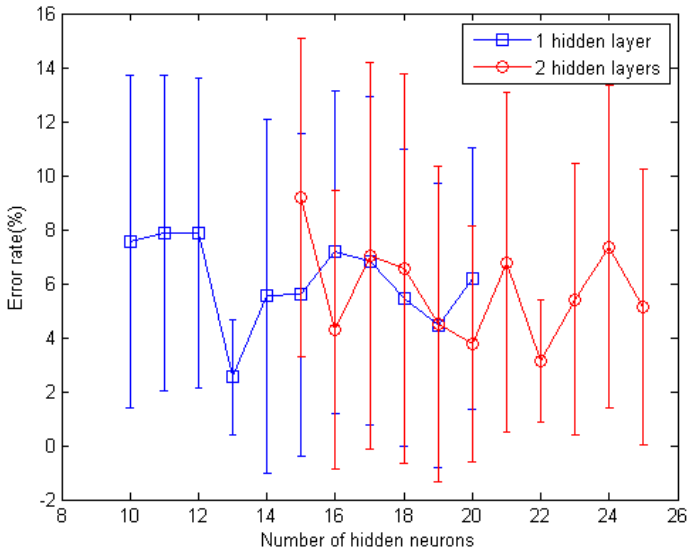
This section is dedicated to evaluating the performance obtained when testing different algorithms to implement the Decision 1 stage. Table 6.1 shows the results obtained when using the MNB algorithm.

Table 6.1: Decision 1 mean error rate if using Multinomial Naive Bayes.

Mean Error Rate(%)	*Standard Deviation (%)*
16.26302	1.77324

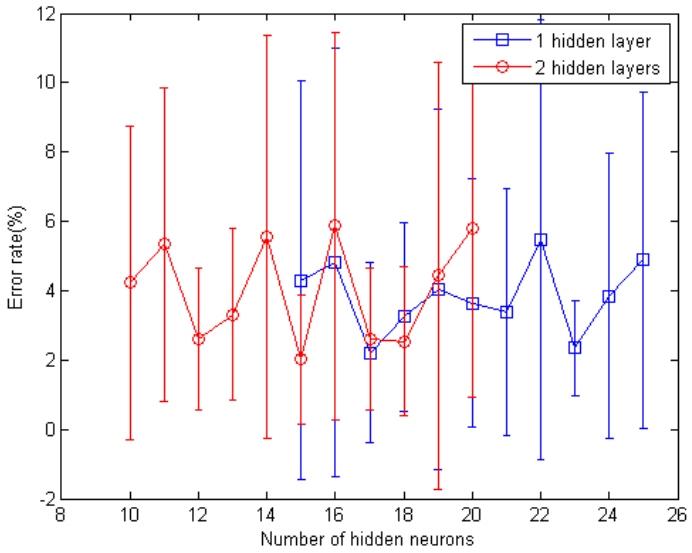
When implementing the Decision 1 using Neural Networks, we evaluated two different activation functions in order to identify the corresponding best performing network architecture. Figure 6.1 shows this the performance of networks that use a logistic function as the activation function—the same activation function used in the OKARD Project. In that figure, the blue curve and red curve represent, respectively, the error rates of networks that use one or two hidden layers, as we vary the number of neurons in each layer. We started by evaluating the error rate of a broad range of neurons, but at a coarse level—by varying the number of neurons in large increments. After this broader-scope pre-experiment was done, we then identified the most promising regions (i.e., the ones with lowest overall error rate) are focused our analysis on those region—those are precisely the regions shown in Figure 6.1.

Figure 6.1: Decision 1 error rate as a function of the number of hidden neurons in a NN using logistic function as activation function.



A second experiment was performed using the hyperbolic tangent activation function $y = \tanh(x)$, and the initialization of the weights of the NN as described in (GLOROT; BENGIO, 2010). Different network configurations (i.e., networks with different numbers of hidden layers and neurons per layer) were evaluated. A graph depicting the error rate of the resulting networks, as a function of the number of hidden neurons, is presented in Figure 6.2. As before, we identified the most promising range of neurons to test by first conducting a wide-scope, coarse analysis of the error rate resulting from the use of different numbers of neurons.

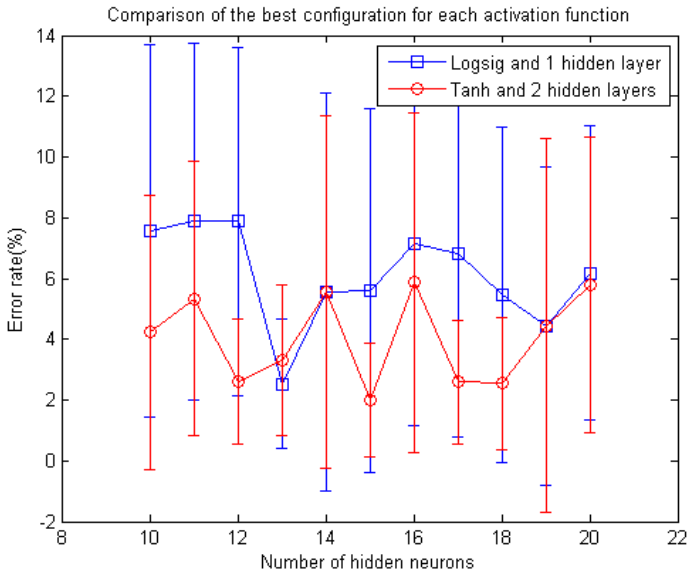
Figure 6.2: Decision 1 error rate as a function of the number of hidden neurons in a NN using $\tanh(x)$ as activation function.



By analyzing Figures 6.1 and 6.2, we note that the error rate does not seem to depend significantly on the number of hidden layers—especially when we consider the large error bars associated with the mean rates shown in the figure. We can, however, focus our analysis on the knees in the graph (i.e., the regions where the error rate decreases significantly), which gives us a better idea of the best number of hidden neurons to use in each network structure. Figure 6.3 compares networks that *1)* use the logistic function as the activation function and that have one hidden layer (i.e., the best configuration according to the results in Figure 6.1); and *2)* which use $\tanh(x)$ as the activation function and that have two hidden layers (i.e., the best configuration according to the results in Figure 6.2); in particular, Figure 6.3 shows the resulting error rate of the network as a function of the number of neurons per layer.

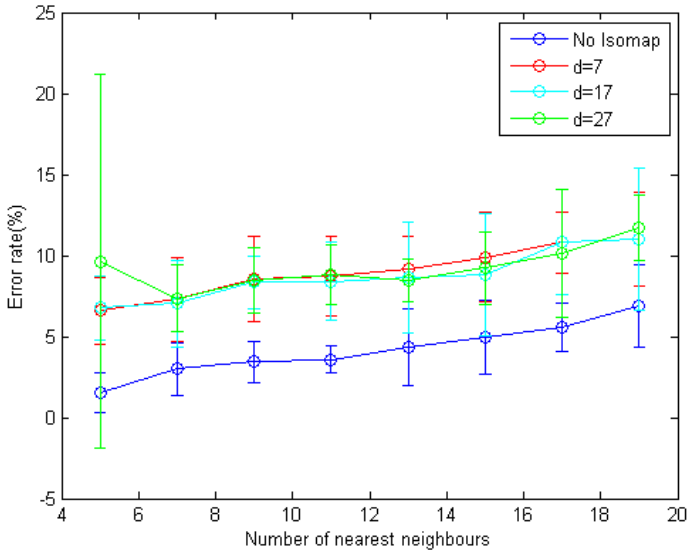
We compared the performance of the neural network architectures described above with a non-parametric method in order to implement Decision 1. In our work we chose to evaluate a combination of ISOMAP algorithm and KNN. As discussed earlier, the KNN algorithm can be configured regarding the number K of neighbors to consider in the voting algorithm. It is also possible to configure the target dimensionality to which we wish to transform the training points via the dimensionality reduction scheme implemented by ISOMAP. We designed experiments that use three selected target dimensions; for each one, we reduce the dimensionality of the points used as inputs to KNN and then compute the error rate of the resulting model as a function of K. Figure 6.4 show the corresponding

Figure 6.3: Decision 1 error rate as a function of the number of hidden neurons in a NN using the best configurations of each activation function.



results. Each curve in this figure is related to the error rate obtained by using a different target dimension d, and a last curve represents the use of KNN when no dimensionality reduction is performed (i.e., when ISOMAP is not used as a pre-processing step to KNN).

Figure 6.4: Decision 1 error rate as a function of the number of nearest neighbors in KNN and different target dimensions (d) for ISOMAP.



We were surprised that the direct use of KNN outperformed the combination of KNN with ISOMAP/dimensionality reduction. Our original hypothesis was that the use of Euclidean distances—the base of standard KNN—in such a high dimension training set would return unpredictable or low-performance results. However, that was not the

case, as shown in Figure 6.4; in that figure, the curve indicating the use of KNN without ISOMAP corresponds to the best results. We believe that the question of why dimensionality reduction did not improve the results of KNN should be addressed more carefully in future work, by performing more simulations and testing variations on both algorithms—ISOMAP and KNN—in order to be able to draw any final conclusions. In Figure 6.4, the best error rate obtained by the use of KNN was close to 1.5%, in particular by applying KNN directly to the data points. **Since KNN yielded better results to implement Decision 1 than NN or MNB, it was selected to be part of the first stage of final solution architecture being proposed in the work**.

6.2 Performance of the Decision 2 Stage

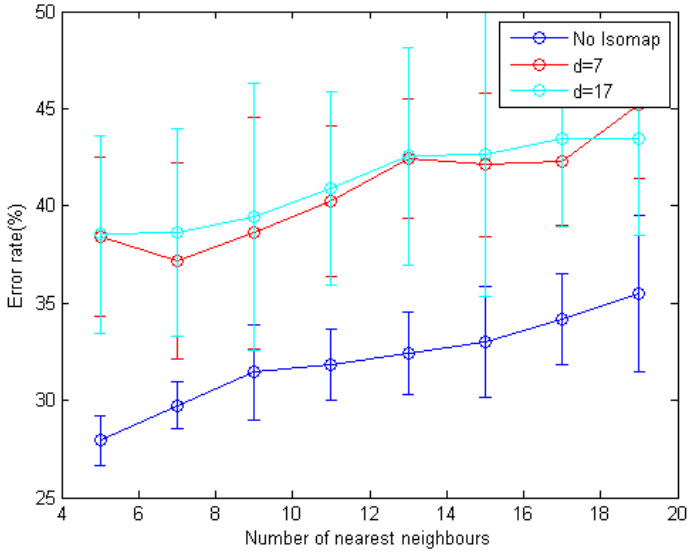
This section is dedicated to discussing performance results obtained when testing different algorithms to implement Decision 2 in our architecture. Table 6.2 shows the classification performance of Decision 2when using the MNB algorithm to process input examples.

Table 6.2: Decision 2 mean error rate if using Multinomial Naive Bayes.

Mean Error Rate(%)	*Standard Deviation (%)*
17.2526	3.539403

Again, the MNB algorithm did not perform well. We also evaluated the composition of ISOMAP and KNN, as discussed in Section 4.4; result is shown in Figure 6.5. These figures present the same type of analysis as explained in the last section. This time, however, the performance of KNN was not as good as in Decision 1, which reinforced our hypothesis that the results obtained in Decision 1 when using ISOMAP & KNN were only due the nature of the training set used in this experiment.

Figure 6.5: Decision 2 error rate as a function of the number of nearest neighbors in KNN and different target dimensions (d) for ISOMAP.



When evaluating the use of NNs to implement Decision 2, the same methodology used previously for Decision 1 was followed. Both activation functions—i.e. the logistic and hyperbolic tangent functions—were evaluated separately using different network structures. Figure 6.6 shows the error rate obtained when using the logistic activation function and Figure 6.7 shows the results obtained when using the hyperbolic tangent activation function; both graphs depict error as a function of the number of neurons in each hidden layer of the network.

Figure 6.6: Decision 2 error rate as a function of the number of hidden neurons in a NN using logistic function as activation function.

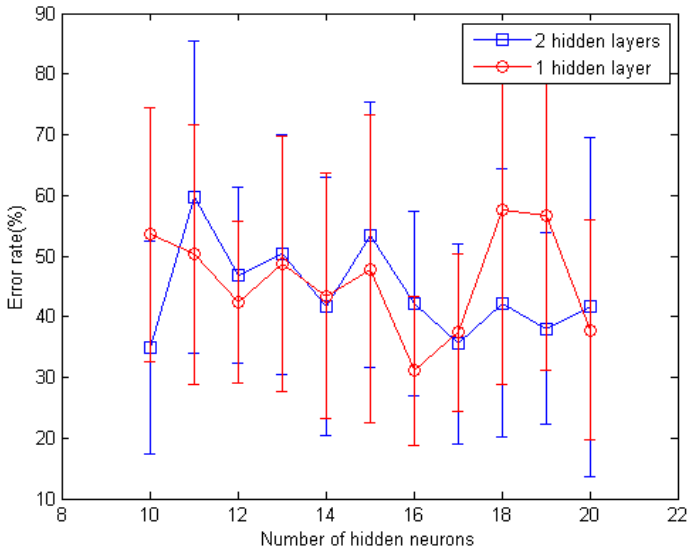
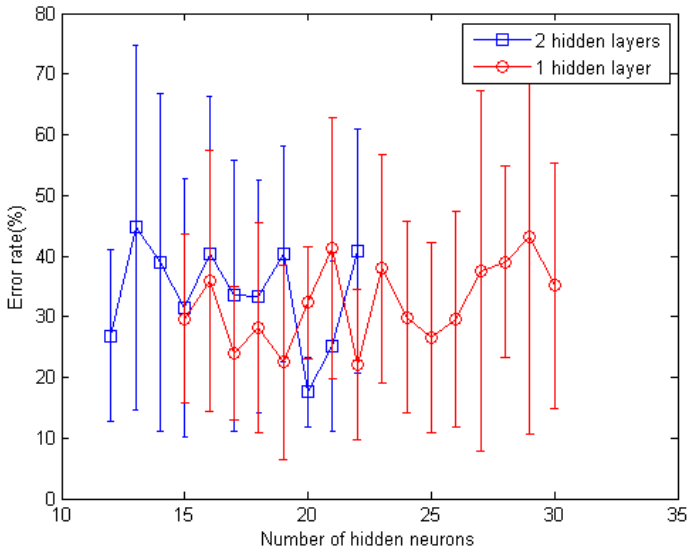
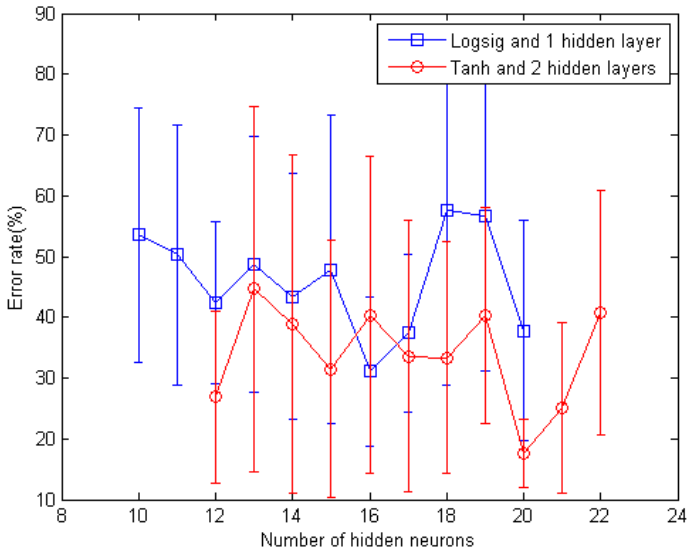


Figure 6.7: Decision 2 error rate as a function of the number of hidden neurons in a NN using $\tanh(x)$ as activation function.



We identify the best performing architectures (namely, single-layer logistic network and double-layer hyperbolic tangent and compared their error rates as a function of the number of neurons in each hidden neuron. The results are shown in Figure 6.8. The difference in the error rates achieved by these networks followed our expectations: the logistic activation function—depicted in the blue curve—performed poorly once again, while the use of $\tanh(x)$ as activation function resulted in a lower error rate. This finding is consistent with those described in the literature; experiments made by (KARLIK; OLGAC, 2011) and (KALMAN; KWASNY, 1992), for instance, show that the hyperbolic tangent activation function typically performs well in practice (in terms of accuracy) when compared to other activation functions, such as the logistic function.

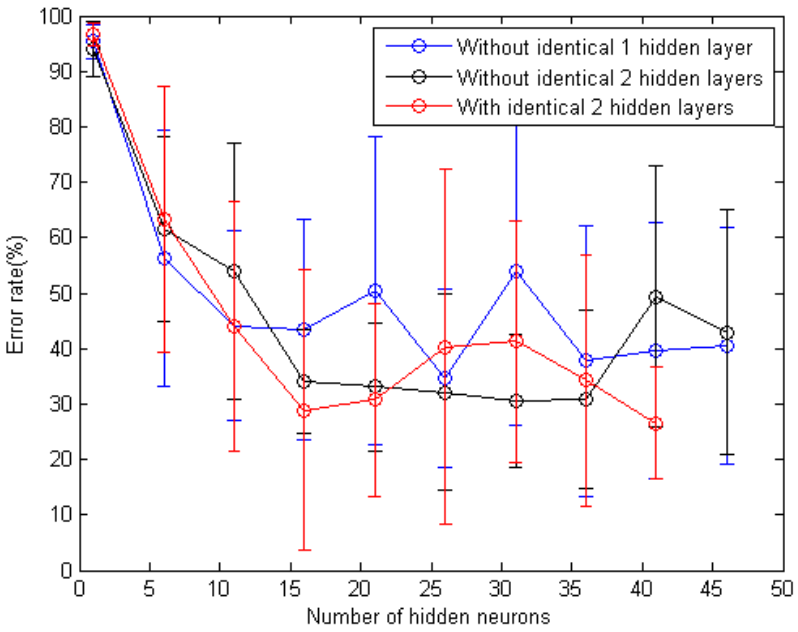
Figure 6.8: Decision 2 error rate as a function of the number of hidden neurons in a NN using the best configurations of each activation function.



As described before, classic boosting algorithms normally weight training examples in a way that weak learners down the chain of predictors tend to focus their prediction power on examples which were misclassified by the former weak learners. As our first weak learner is implemented by Google's black-box STT and the error is boolean—i.e. a term is either translated correctly or incorrectly—we decided to test whether to include samples that are correctly classified by the base STT system in the training set of our boosting system. Intuitively, it would make sense not to include them, since this would allow the system to focus only on outputs whose classification needs to be improved or corrected. On the other hand, removing these samples from the training set could imply a lower number of training examples—and examples, even if not incorrectly classified by Google's weak learner, could still contain useful information for the boosting procedure. This latter intuition is confirmed by the analysis shown in Figure 6.9, which shows clearly that the suppression of the identical translations is not beneficial to the whole system[1].

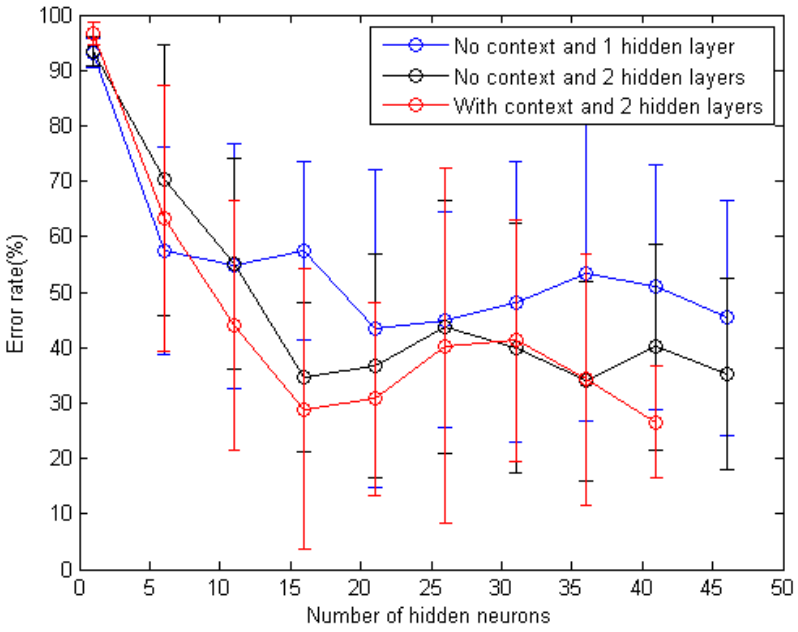
[1]In this experiment, we are just testing the suppression of the identical translations in Decision 2 stage. As the performance observed is worse than the using the whole training set, we can conclude that this would also be bad to the whole solution (composition of both decision blocks), since this methodology of passing the wrong translations to Decision 2 implicitly assumes that the Decision 1 block could perfectly identify those mistranslations and send them to Decision 2; but we directly compare the expected output, not using a procedure trained by machine learning.

Figure 6.9: Boosting NN (Decision 2) trained only with examples incorrectly predicted by the weak learner, compared with a Boosting NN trained with the entire set of examples. The blue curve shows the performance when using one hidden layer and only the errors made by Google. The black curve shows the performance when using two identical hidden layers and only the errors made by Google. The red curve shows the performance when using two identical hidden layers and all the data. All scenarios uses $\tanh(x)$ as activation function.



In our experiments, the Decision 2 procedure is implemented similarly to how the neural network was constructed in OKARD Project—i.e. the term classifier; the main (but not only) difference is the use of contextual information—the last identified term—as an additional input feature. Although we believe that the poor accuracy obtained in OKARD was mainly due to the manually tuned algorithms and thresholds used in the process, we wanted to compare the use of the contextual information against a similar NN trained with no context information. Figure 6.10 shows that contextual information is very important to the term classification problem, since the error rates obtained in this case are lower than those obtained by using a NN that sees no context.

Figure 6.10: Decision 2 error rate as a function of the number of hidden neurons in a NN using $\tanh(x)$ as activation function. The blue and black curves show the error rates obtained by a NN without contextual information with one and two hidden layers, respectively. And the red one shows the performance of a NN with contextual information.



6.3 Performance of the Combined Selected Decision Algorithms

In this section we present the performance of the proposed boosted STT classifier—i.e. the composition of both decision stages—when tested on the collected dataset. As explained before, the experimental analysis performed in the previous section suggests that the second decision stage could be implemented either via a neural network with one single hidden layer, or via one network composed of two hidden layers of same size.

In the following experiment we measure, besides the overall error rate of an algorithm, the *timeout rate*. This corresponds to the fraction of prediction errors that were caused by samples which, when processed by the composite decision system, cause it to go into an infinite loop. This is caused by the Decision 1 stage repeatedly outputting a *Stay* command, which keeps the transcription analysis from moving to subsequent phonemes. In our implementation we placed a 10-second timeout to ensure that the training process does not freeze in case one of those errors occurs.

As we can see, the final composite architecture that performs the best on the available training set is the one where Decision 1 is implemented using KNN, and Decision 2 is implemented using a NN with $tanh(x)$ as activation function and two hidden layers.

Table 6.3: Error rate of the final composite architecture when using KNN with $k = 5$ for Decision 1 block, and a NN with $\tanh(x)$ as activation function and two identical hidden layer of size 20 to implement Decision 2. Note that the error rate presented was calculated so as to includes samples which, when processed by the system, incurred in a timeout.

Error Rate(%)	*Timeout Rate(%)*
17	31

This means that, after exploring a possible space of learning algorithms, and by evaluating different combinations of algorithms to automate the two decision stages of our process, we can draw the following conclusions:

1. considering that there are 4 possible outputs in Decision 1 and 21 possible outputs in Decision 2, our solution outperforms a random estimator by 73% and by 75%, respectively;
2. **we designed a new solution that improves the accuracy of the previous system (i.e., the OKARD project) by up to 8%;**[2]
3. **the final combined architecture proposed in this work improves the performance resulting from the direct use of the black-box Google STT up to 49%.**[3]

[2]Even though this is apparently a relatively small improvement, we note that during the execution of the OKARD project the authors made no effort to prevent overfitting—which suggest that the actual performance of OKARD was probably way worse than what was originally estimated, and that our gains may be significantly larger than 8%.

[3]In our experiments, Google had a 64% error rate when trying to directly transcribe the audio files.

7 CONCLUSION

We presented a machine learning-based method for constructing an additional prediction layer capable of reducing the mistranslations of a general-purpose black-box STT, so that it performs better when used to transcribe domain-specific sentences. Our empirical results show that the use of a non-parametric model—i.e. the K-Nearest Neighbors algorithm—to control the iteration over input phonemes produced by the weak STT system, and the use of a Feed-Forward neural network to classify those phonemes into known words, results in very high prediction performance (in the order of 1% prediction error). Importantly, we managed to achieve this level of accuracy even though the amount of training data available would *not* be sufficient for creating a complete STT interpreter from scratch; i.e., a boosting architecture, as the one proposed here, is justified.

Three decisions made in this work enabled us to obtain better overall accuracy in the final system, when comparing our results to those obtained in the OKARD project. The first one that contributed to the improvement is the use of a different activation function ($\tanh(x)$) than the one used originally, in particular when implementing the neural network responsible for classifying a phoneme into one of the known terms. The second contributing decision is the use of adjacent words as additional context information to the network, which showed itself useful for lowering the recognition error rate. The third contributing factor is the replacement of the manually-tuned algorithm originally used in OKARD project by a machine learning method to control the processing of input samples during the transcription analysis process.

By analyzing the results obtained from the combination of ISOMAP and KNN in the last chapter, it becomes clear that dimensionality reduction using graph distance preservation does not perform well on our data. Given the limited amount of time available to execute this research, we did not evaluate other dimensionality reduction strategies that could possibly improve the results, if used as pre-processing steps before a non-parametric method (such as KNN) was used to implement the second decision stage. We do believe, however, that this matter needs to be further investigated.

We should also point out that although the use of Feed Forward neural networks produced a satisfactory error rate in our experiments, we also believe that the use *recurrent neural networks* may be valuable to the problem. Our hypothesis is that it could directly (due its its recurrent nature) contextual information, without the need to explicit additional inputs to the network. The fact that the dimensionality of the input presented to the

network would be lower, we expect that this could reduce even more the necessary amount of training data.

Overall, we note that we have achieved by general objective set for this work—to reduce the error rate of a general-purpose voice-recognition system, when evaluated over domain-specific terms, by using machine learning to construct a boosting-based prediction system. We also believe that this same architecture could be used in similar voice-recognition applications, given only a fixed command list and a relatively small amount of training data.

7.1 Future Work

Given the relatively limited amount of time available to develop this work, several important questions remain unanswered and could be further investigated. Beside the ones described in the previous section, we also highlight that a graphical user interface could have been designed and used to implement some kind of feedback system (as was available in OKARD). In particular, despite the low error rates we obtained here, the overall prediction system could benefit from receiving feedback from user when a term is misclassified. This could generate a new training example that could be used to retrain the Decision 2 block in order to improve the classification model and, consequently, the recognition of various voice commands.

Another unanswered question is regarding the impact of the number of domain-specific terms that we wish to include in the vocabulary of the system. In this work, the list of known terms was composed of 21 words—this was due to the requirements set by the particular application at hand: a flight reservation system with a small set of voice commands that could be used to improve the speed with which users interact with the system. It remains to be experimentally studied what would be the impact of larger vocabularies in the sample complexity of the boosting architecture proposed in this work.

REFERENCES

ADORF, J. Web speech api. **KTH Royal Institute of Technology**, 2013.

BEYER, K. et al. When is "nearest neighbor" meaningful? In: **Database theory—ICDT'99**. [S.l.]: Springer, 1999. p. 217–235.

BROWN, P. F. et al. Class-based n-gram models of natural language. **Comput. Linguist.**, MIT Press, Cambridge, MA, USA, v. 18, n. 4, p. 467–479, dec. 1992. ISSN 0891-2017. Available from Internet: <http://dl.acm.org/citation.cfm?id=176313.176316>.

BRYSON, A. E.; DENHAM, W. F.; DREYFUS, S. E. Optimal programming problems with inequality constraints. **AIAA journal**, v. 1, n. 11, p. 2544–2550, 1963.

COOK, G.; ROBINSON, T. Boosting the performance of connectionist large vocabulary speech recognition. In: IEEE. **Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on**. [S.l.], 1996. v. 3, p. 1305–1308.

DRUCKER, H.; SCHAPIRE, R.; SIMARD, P. Boosting performance in neural networks. **International Journal of Pattern Recognition and Artificial Intelligence**, World Scientific, v. 7, n. 04, p. 705–719, 1993.

DUDA, R. O.; HART, P. E. et al. **Pattern classification and scene analysis**. [S.l.]: Wiley New York, 1973.

FIX, E.; JR, J. L. H. **Discriminatory analysis-nonparametric discrimination: consistency properties**. [S.l.], 1951.

FREUND, Y. Boosting a weak learning algorithm by majority. **Information and computation**, Elsevier, v. 121, n. 2, p. 256–285, 1995.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. **Journal of Computer and System Sciences**, v. 55, n. 1, p. 119 – 139, 1997. ISSN 0022-0000. Available from Internet: <http://www.sciencedirect.com/science/article/pii/S002200009791504X>.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **International conference on artificial intelligence and statistics**. [S.l.: s.n.], 2010. p. 249–256.

HANNUN, A. et al. Deep speech: Scaling up end-to-end speech recognition. **arXiv preprint arXiv:1412.5567**, 2014.

JUNQUA, J.-C. The lombard reflex and its role on human listeners and automatic speech recognizers. **The Journal of the Acoustical Society of America**, Acoustical Society of America, v. 93, n. 1, p. 510–524, 1993.

KALMAN, B. L.; KWASNY, S. C. Why tanh: choosing a sigmoidal function. In: IEEE. **Neural Networks, 1992. IJCNN., International Joint Conference on**. [S.l.], 1992. v. 4, p. 578–581.

KAMP, H.; REYLE, U. **From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory**. [S.l.]: Springer Science & Business Media, 2013.

KARLIK, B.; OLGAC, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. **International Journal of Artificial Intelligence and Expert Systems**, v. 1, n. 4, p. 111–122, 2011.

KAUTH, D. E. **The art of computer programming: Volume 3/Sorting and searching**. [S.l.]: Reading MA, Addison-Wesley, 722p, 1973.

KIM, S.-B. et al. Some effective techniques for naive bayes text classification. **Knowledge and Data Engineering, IEEE Transactions on**, IEEE, v. 18, n. 11, p. 1457–1466, 2006.

KOHAVI, R. et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: **Ijcai**. [S.l.: s.n.], 1995. v. 14, n. 2, p. 1137–1145.

KOMBRINK, S. et al. Recurrent neural network based language modeling in meeting recognition. In: **INTERSPEECH**. [S.l.: s.n.], 2011. p. 2877–2880.

LEE, J. A.; VERLEYSEN, M. **Nonlinear dimensionality reduction**. [S.l.]: Springer Science & Business Media, 2007.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. In: **Soviet physics doklady**. [S.l.: s.n.], 1966. v. 10, n. 8, p. 707–710.

LEWIS, D. D. Naive (bayes) at forty: The independence assumption in information retrieval. In: **Machine learning: ECML-98**. [S.l.]: Springer, 1998. p. 4–15.

MCCALLUM, A.; NIGAM, K. et al. A comparison of event models for naive bayes text classification. In: CITESEER. **AAAI-98 workshop on learning for text categorization**. [S.l.], 1998. v. 752, p. 41–48.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

NAKAMURA, M. et al. Neural network approach to word category prediction for english texts. In: **Proceedings of the 13th Conference on Computational Linguistics - Volume 3**. Stroudsburg, PA, USA: Association for Computational Linguistics, 1990. (COLING '90), p. 213–218. ISBN 952-90-2028-7. Available from Internet: <http://dx.doi.org/10.3115/991146.991184>.

PHILIPS, L. Hanging on the metaphone. **Computer Language**, v. 7, n. 12 (December), 1990.

PRECHELT, L. Automatic early stopping using cross validation: quantifying the criteria. **Neural Networks**, Elsevier, v. 11, n. 4, p. 761–767, 1998.

RENNIE, J. D. et al. Tackling the poor assumptions of naive bayes text classifiers. In: WASHINGTON DC). **ICML**. [S.l.], 2003. v. 3, p. 616–623.

RODRIGUEZ, J. D.; PEREZ, A.; LOZANO, J. A. Sensitivity analysis of k-fold cross validation in prediction error estimation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 32, n. 3, p. 569–575, 2010.

SCHAPIRE, R. E. The strength of weak learnability. **Machine learning**, Springer, v. 5, n. 2, p. 197–227, 1990.

SCHAPIRE, R. E. Using output codes to boost multiclass learning problems. In: **ICML**. [S.l.: s.n.], 1997. v. 97, p. 313–321.

SCHAPIRE, R. E. The boosting approach to machine learning: An overview. In: **Nonlinear estimation and classification**. [S.l.]: Springer, 2003. p. 149–171.

SCHAPIRE, R. E.; SINGER, Y. Improved boosting algorithms using confidence-rated predictions. **Machine learning**, Springer, v. 37, n. 3, p. 297–336, 1999.

TENENBAUM, J. B.; SILVA, V. D.; LANGFORD, J. C. A global geometric framework for nonlinear dimensionality reduction. **science**, American Association for the Advancement of Science, v. 290, n. 5500, p. 2319–2323, 2000.

THAYSE, A.; BINOT, J. **From natural language processing to logic for expert systems: a logic based approach to artificial intelligence**. [S.l.]: Wiley, 1991. (Logic based approach to artificial intelligence). ISBN 9780471924319.

YOUNG, S. et al. High level knowledge sources in usable speech recognition systems. **Communications of the ACM**, ACM, v. 32, n. 2, p. 183–194, 1989.

ZOU, Y. et al. Mariana: Tencent deep learning platform and its applications. **Proceedings of the VLDB Endowment**, VLDB Endowment, v. 7, n. 13, p. 1772–1777, 2014.

APPENDIX A — METAPHONE TRANSFORMATION RULES

Table A.1: Metaphone transformation rules table.

Character	*Encoding*	*Observation*
B	B	unless at the end of a word after "m" as in "dumb".
C	X	if -cia- or -ch-.
	S	if -ci-, -ce- or -cy-.
	K	otherwise, including -sch-.
D	J	if in -dge-, -dgy- or -dgi-.
	T	otherwise.
F	F	
G		silent if in -gh- and not at end or before a vowel in -gn- or -gned- (also see dge etc. above).
	J	if before i or e or y if not double gg.
	K	otherwise.
H		silent if after vowel and no vowel follows.
	H	otherwise.
J	J	
K		silent if after "c".
	K	otherwise.
L	L	
M	M	
N	N	
P	F	if before "h".
	P	otherwise.
Q	K	
R	R	
S	X	(sh) if before "h" or in -sio- or -sia-.
	S	otherwise.
T	X	(sh) if -tia- or -tio-.
	0	(th) if before "h".
		silent if in -tch-.
	T	otherwise.
V	F	
W		silent if not followed by a vowel.
	W	if followed by a vowel.
X	KS	
Y		silent if not followed by a vowel.
	Y	if followed by a vowel.
Z	S	
Double letters	Drop 2nd letter	except for double "c".
Vowels	Droped	Except if they are the first letter
Kn-, Gn- Pn, Ae- or Wr-	Drop first letter	Only if is the beginning of the string
X-	Change to "s"	Only if is the beginning of the string
Wh-	Change to "w"	Only if is the beginning of the string

Source: (PHILIPS, 1990)