

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Out of Vocabulary Queries for Word Graph-based Keyword Spotting

Joan Puigcerver i Pérez

Directors:

Dr. Enrique Vidal Ruiz

Dr. Alejandro H. Toselli Rossi

February 6, 2014

Gràcies,
A la meva família, en especial als meus pares,
per animar-me a perseguir allò que vull.
A Anna que m'ha acompanyat, literalment,
per mig món i ha tingut la paciència de
suportar el meu estrés diari.
A Moisés per la seva confiança i
haver-me parlat d'aquesta temàtica
per primera vegada.
I, molt especialment, a Enrique i Alejandro
que m'han introduït al món de la recerca,
m'han guiat a través de tot aquest treball,
amb substancioses aportacions i consells, i
m'han donat llibertat per a interrompre'l,
quan estava a l'altra banda de l'Atlàntic,
i continuar-lo de nou a la meva tornada.



ABSTRACT

This master thesis presents different approaches that aim to give a solution to the search of out of vocabulary queries in Keyword Spotting applications based on Word Graphs.

Keyword Spotting refers to the problem of determining whether a given keyword is present in a collection of images. Typically, a score is computed for each keyword and line of each image, and pairs with a score surpassing a predetermined threshold are retrieved. This provides the user the ability to fine-tune the results of his/her searches.

Keyword Spotting systems based on Word Graph provide very fast lookup speeds and accurate results, in comparison to other alternatives which operate at a character level. The improvement in the accuracy is achieved by using lexical information that can be extracted from many sources freely available, and not only the given training data in images and their transcripts. However, they suffer from the problem that a null score is assigned to any word that was not part of the training data (out of vocabulary keyword).

Out of vocabulary words are an intrinsic problem of word-level language models and are present in many applications like Automatic Speech Recognition (ASR), Handwriting Text Recognition (HTR) and also in Keyword Spotting (KWS), which is studied here applied to handwritten line images.

This work contributes to the KWS field with different alternatives that enable a reasonable estimation of out of vocabulary keyword scores in Word Graph-based systems, providing the versatility that character-level models offer (being able to spot any keyword), without giving up on the benefits of using language models and the speed that Word Graphs provide.

Alternatives are presented at different abstraction levels, which go from very fast searches with moderate accuracy, to a very accurate approach at the expense of slower search speeds.

CONTENTS

Table of Contents	vii
1 Introduction	1
2 Handwriting Text Recognition	3
2.1 Pattern Recognition	3
2.2 Pattern Recognition for Handwriting Text Recognition	5
2.3 Hidden Markov Models	6
2.4 Language Models	12
2.5 Word Graphs	14
3 Keyword Spotting	21
3.1 Keyword Spotting for Handwritten Text Images	21
3.2 Filler approach	24
3.3 Word Graph approach	27
3.3.1 Frame-level word posterior	27
3.3.2 Fast computation of Frame-level Word Posteriors	28
4 Out of vocabulary Queries	31
4.1 Similarity measures among strings	32
4.1.1 Levenshtein Edit Distance	32
4.1.2 Stochastic Error Correcting Probability	33
4.2 Line-level smoothing	36
4.2.1 Levenshtein Distance-based smoothing	36
4.2.2 Stochastic Error Correcting-based smoothing	36
4.3 Frame-level smoothing	38
4.4 Filler-based scores	40
4.5 Combining scores with a Back-off heuristic	41
4.6 Computational issues	42

5 Experiments	47
5.1 Corpora	47
5.1.1 Cristo Salvador	47
5.1.2 IAM Database	50
5.1.3 Query keywords selection	52
5.2 Assessment metrics	53
5.2.1 Precision and Recall	53
5.2.2 Average Precision	55
5.2.3 Mean Average Precision	56
5.3 Description of the underlying HTR system	58
5.4 Baseline results	59
5.5 Smoothing methods	61
5.5.1 Line-level smoothing	63
5.5.2 Frame-level smoothing	65
5.5.3 Back-off combination	67
5.5.4 Summary results	70
5.6 Posteriogram compression	74
6 Conclusions	77
Bibliography	81
List of Figures	88
List of Tables	90

CHAPTER 1

INTRODUCTION

The usage of computers to preserve information from the passage of time and organize it to facilitate its access, has been one of their main applications since their very first days. Thanks to the evolution and improvement of technology, now we are able to store more digital information and access to it in a faster way and from almost everywhere around the world. However, for many centuries, handwriting has been the only way to preserve the human knowledge, besides oral communication. Precisely, storing information in books, manuscripts and other written forms, first handwritten and printed later, has been one of the main basis of all the human development around the world.

Unfortunately, books have not always been successful in their task of preserving the human knowledge. One only needs to remember the dramatic incidents that devastated the old Library of Alexandria and caused the irreparable loss of thousands of ancient works. Handwriting Text Recognition (HTR) was developed, among other reasons, to preserve the information contained in handwritten documents using computers, a support which has proven to be more reliable than analog supports. And event more importantly, to allow an easier and faster access to that information (since information is only useful if it can be accessed). Despite all the progress achieved in the recent years by the HTR community, this field has still many problems and shortcomings, basically in terms of final accuracy of the transcribed text. These limitations affect the spreading of HTR and prevent it to be used for many useful applications. The dominant approach to the state-of-the-art HTR and its limitations will

be explained in Chapter 2.

Keyword Spotting (KWS) applied on handwritten documents is one of the applications which is affected by the shortcomings of HTR. The aim of KWS is to locate a certain keyword within a collection of documents. In our scenario, we restrict to handwritten documents such as ancient books, logbooks, registries, etc. There are two main approaches to KWS: Query-by-Example and Query-by-String. In the first approach, the user provides an *example* of the keyword to search for, in the case of handwritten Keyword Spotting, that is an image containing the keyword to spot. The latter approach, however, does not require the user to have an existing example of the keyword to spot. Instead, the user only provides a string, which is the keyword to spot in the collection of documents. In Chapter 3, two models that try to solve Query-by-String KWS will be presented and compared in order to highlight their advantages and deficiencies.

One of these models uses lexical information about the language to boost the performance in Keyword Spotting tasks, however, it performs badly with keywords that were not observed during the training of the system (since no lexical information is known about them). This problem is referred to as the *out of vocabulary query* problem in this thesis, and will be more formally stated in Chapter 4. This chapter will also introduce several alternatives that try to provide a solution for this problem, which are the main contributions of this thesis.

The alternatives presented in Chapter 4 to solve the problem of *out of vocabulary queries* are tested experimentally in Chapter 5 using two different databases of handwritten text, with different properties and origins, which demonstrate the effectiveness of each of the proposed solutions.

The final chapter will present the conclusions of this work, together with some suggestions of future lines of research in the field of Keyword Spotting.

CHAPTER 2

HANDWRITING TEXT RECOGNITION

2.1 Pattern Recognition

Pattern Recognition (PR) is a branch of Machine Learning (ML) which aims to give computers the ability to discern among different objects in their environment. PR systems are able to perceive the environment from different sensors, such as photo or video cameras, microphones, laser sensors, temperature sensors, etc. Once the signals have been acquired, a PR system attempts to discover and to give a meaning to the different represented objects by those signals (the most typical task is to assign each of them to a category, i. e. classify them) [8].

PR systems are built using supervised learning, which tries to approximate the function $f : \mathcal{X} \rightarrow \mathcal{Y}$ mapping the signal space \mathcal{X} to the set of labels \mathcal{Y} . Supervised learning approximates the true and unknown function f by fitting an hypothesis function $h : \mathcal{X} \rightarrow \mathcal{Y}$ with a set of labeled examples $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)}) \mid (x, y) \in \mathcal{X} \times \mathcal{Y}\}$.

The signals from \mathcal{X} and the labels \mathcal{Y} are usually, and surely for all interesting applications, subject to noise. The types of noise depend on the application and may come from a huge set of sources. For instance, in a handwriting recognition application, many physical devices and information representation limitations introduce noise in \mathcal{X} , like artifacts due to the scanning resolution, lighting changes, color quantification, image compression, etc. Moreover, there are some changes in the representation

of a certain object that can be considered noise: the changes in the scripture among different writers, in the writing tools, or even in the mood of a single writer. This leads to the fact that a single abstract object may be represented by different elements of \mathcal{X} .

In addition, as mentioned before, \mathcal{Y} is also subject to some form of noise. In the first place, the labeling of an input signal may be wrong due to human error. But also, by the limitations of what context is captured and represented in \mathcal{X} , some elements of this set may have multiple instances with different associated categories, depending on the context.

In the presence of these kinds of noises, the mapping from \mathcal{X} to \mathcal{Y} that PR tries to infer is not strictly a function, according to the mathematical definition of *function*. Thus, PR usually adopts a probabilistic approach, where it tries to infer the unknown probability distribution $Pr(Y|X)$, where $Y \in \mathcal{Y}$ and $X \in \mathcal{X}$. This true conditional distribution is approximated by some distribution $p(Y|X, \mathcal{D})$. The form of $p(Y|X, \mathcal{D})$ is given by the used model (Gaussian distribution, Bernoulli distribution, Mixture Model, Neural Network, etc) and usually has associated some parameters which are fitted using the labeled training examples in \mathcal{D} .

If the real distribution $Pr(Y|X)$ is known, the label $\hat{y} \in \mathcal{Y}$ of an unseen example $x \in \mathcal{X}$ that minimizes the expected error is known as the Minimum Error Rate Classification [8], and it is given by:

$$\hat{y} = \underset{\forall y \in \mathcal{Y}}{\operatorname{argmax}} Pr(Y = y|X = x) \quad (2.1)$$

Substituting the real distribution by the fitted hypothesis given by the PR approach described before, results in:

$$\hat{y} \approx \underset{\forall y \in \mathcal{Y}}{\operatorname{argmax}} p(Y = y|X = x, \mathcal{D}) \quad (2.2)$$

Of course, depending on the modeling and the fitting of $p(Y|X, \mathcal{D})$, and the training data used, the final performance of the system may change dramatically.

2.2 Pattern Recognition for Handwriting Text Recognition

There are different HTR scenarios and all of them can be modeled as PR problems. In this work, we focus on a line-level HTR scenario. That is, we have an input image from a handwritten text line representing a sequence of words, which should be the output of the recognition step.

In this scenario, input images have usually different dimensions, both height and width. In order to deal with the different dimensions and to reduce the amount of noise present in the scanned images (e.g. color, rotation, skew, handwriting styles, etc), a sequence of d -dimensional vectors is extracted from each input image, known as the feature vectors of the image. Thus, the input signal space \mathcal{X} is in fact a space of variable-length sequences of d -dimensional vectors. An element in \mathcal{X} is denoted by $\mathbf{x} = \vec{x}_1, \vec{x}_2, \dots, \vec{x}_l$, where l is the length of the sequence \mathbf{x} .

There are many different approaches to extract a reasonable set of relevant features from an image: from heuristic approaches, like the gray and horizontal and vertical derivatives, to learning-based approaches, like PCA or Neural Network auto-encoders. A comprehensive set of feature extraction approaches can be found in [4, 36].

The same applies to the labels space \mathcal{Y} , since the output can be an arbitrary sequence of words. An output element in \mathcal{Y} is denoted by $\mathbf{y} = y_1, y_2, \dots, y_m$, where m is the length of the sequence \mathbf{y} .

The output sequence of words $\hat{\mathbf{y}}$ given by the HTR system, for an input image represented by \mathbf{x} , is given by the application of Equation (2.2) to this scenario, which results in:

$$\hat{\mathbf{y}} = \underset{\forall \mathbf{y} \in \mathcal{Y}}{\operatorname{argmax}} P(Y = \mathbf{y} | X = \mathbf{x}, \mathcal{D}) = \underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y} | \mathbf{x}) \quad (2.3)$$

Observe that $\operatorname{argmax}_{\forall \mathbf{y} \in \mathcal{Y}} P(Y = \mathbf{y} | X = \mathbf{x}, \mathcal{D})$ has been rewritten as $\operatorname{argmax}_{\mathbf{y}} P(\mathbf{y} | \mathbf{x})$, for sake of simplification. This widely used notation, will be used from now on, as long as it does not cause any confusion.

A priori, Equation (2.3) is a simple equation. However, in reality some problems arise which prevent to use it to directly solve the HTR problem and many related problems like Automatic Speech Recognition (ASR) and Machine Translation (MT).

First of all, modeling the conditional distribution $P(\mathbf{y}|\mathbf{x})$ is not trivial and the most predominant approaches in HTR and ASR do not model it directly, but apply the Bayes theorem to rewrite the previous equation as:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmax}} \frac{p(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{p(\mathbf{x})} = \underset{\mathbf{y}}{\operatorname{argmax}} p(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \quad (2.4)$$

In the previous equation, $p(\mathbf{x}|\mathbf{y})$ is modeled as a concatenation of Hidden Markov Models (HMMs) which captures the likelihood that some signal \mathbf{x} is the representation of some sequence \mathbf{y} . On the other hand, $P(\mathbf{y})$ is modeled through a Language Model (LM), which measures how likely is the sequence \mathbf{y} to be a valid sentence of the language. In the following sections, the basics of these two models will be explained, since they are fundamental for the understanding of the KWS approach followed in this thesis.

Figure 2.1 shows an overview of the stages of a regular HTR system: feature extraction, training and recognition modules, together with the information that each of these modules use and the information that they provide. Observe that, in the figure, the sequence of labels \mathbf{y} has been substituted by the sequence \mathbf{w} : this is just a matter of notation. Since we are addressing from now a concrete application of Pattern Recognition to recognize sequences of *words*, we denote such sequences as \mathbf{w} . This notation will be used the rest of this work.

2.3 Hidden Markov Models

A HMM is defined by a finite set of states, each of which has associated a continuous probability distribution of *observations*. Transitions among the states of the HMM are controlled by transition probabilities. These models are called *hidden* because only the sequence of emitted observations is known, but not the sequence of states that produced those observations. Depending on the nature of the observations, HMMs can be divided into three different groups:

Discrete: observations are vectors of symbols of a finite alphabet.

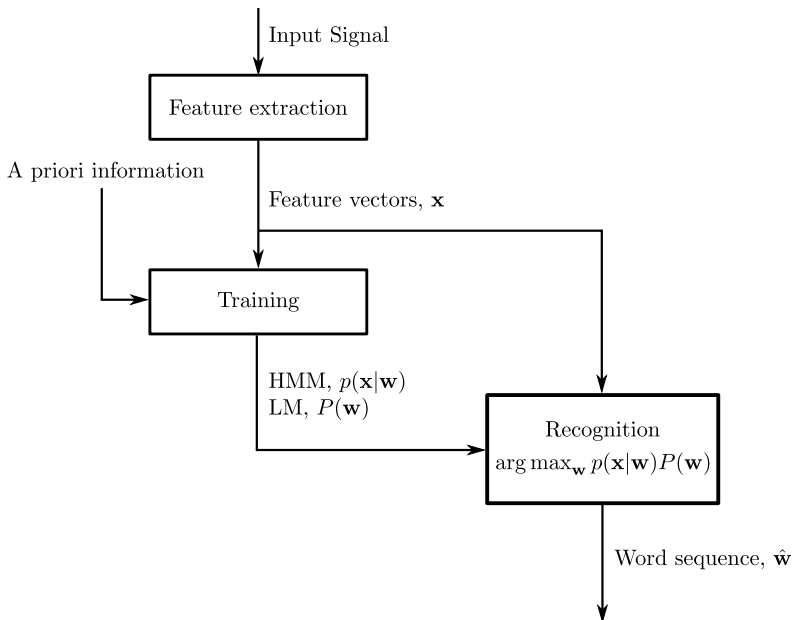


Figure 2.1: Overview of the stages of a HTR system.

Continuous: observations are vectors of elements in a continuous space, like \mathbb{R} .

Semi-continuous: observations are also vectors of symbols of a finite alphabet, but they are modeled using continuous probability density functions.

A continuous HMM, M , is a stochastic finite state machine (FSM) where each state from the HMM is said to *emit* a real-valued vector of observations at a certain discrete time point t . The emission of a vector is done following a certain probability distribution function. A sequence of observation vectors is emitted by a sequence of states of the HMM that are visited at different time points $1 \leq t \leq T$. The transition among states from time t to $t+1$ is ruled by a probability mass function. The sequence of observed vectors is denoted by $\mathbf{x} = \vec{x}_1, \dots, \vec{x}_T$, and the sequence of visited states by $\mathbf{z} = z_1, \dots, z_T$, where state z_t emitted the observation vector \vec{x}_t . Figure 2.2 shows an example of a HMM modeling a character from a handwritten text image.

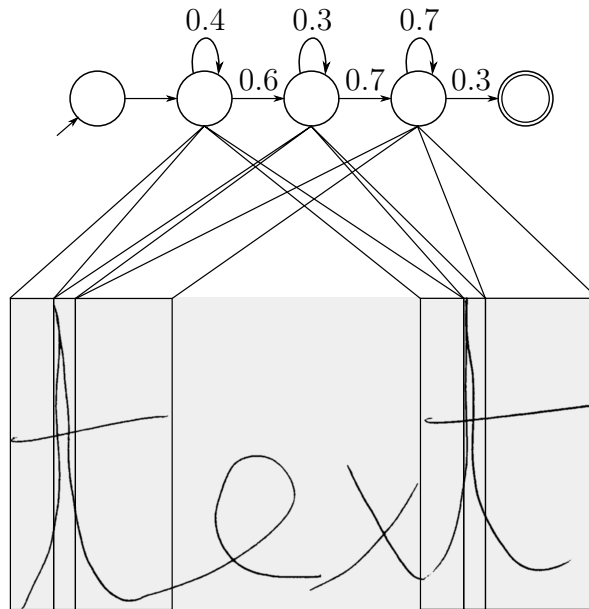


Figure 2.2: Illustration of a HMM modeling the character “t” of a handwritten text image where the text “text” is found. The boxed image regions are mapped to the state that emitted them.

Formally, a continuous HMM is defined by the tuple $(Q, q_0, q_{|Q|-1}, X, a, b)$.

- Q is the finite set of states. In order to refer to a particular state $q_i \in Q$, we will assume that Q is an enumerated set $\{q_0, \dots, q_{|Q|-1}\}$.
- q_0 is the initial state, $q_0 \in Q$.
- $q_{|Q|-1}$ is the final state, $q_{|Q|-1} \in Q$.
- X is the real d -dimensional space of observation, $X \subseteq \mathbb{R}^d$.
- a is the state-transition probability distribution. In theory, a transition into a state q_i at time t (i.e. $z_t = q_i$), may depend upon all the previous sequence of states z_1, \dots, z_{t-1} . However, for sake of simplicity and computational tractability, usually the *Markov assumption* is taken, which assumes that z_t is only conditionally dependent of z_{t-1} , that is:

$$P(z_t|z_1, \dots, z_{t-1}) = P(z_t|z_{t-1}) \quad (2.5)$$

Additionally, it is also assumed that this distribution does not depend on the actual time when the transitions take place. This is known as the *stationary assumption*, and is formalized as:

$$P(z_t = q_j|z_{t-1} = q_i) = P(z_{t+k} = q_j|z_{t+k-1} = q_i), k \geq 0 \quad (2.6)$$

Finally, for sake of the notation simplification and without any loss of generalization, it is assumed that q_0 is only visited in the first place, and $q_{|Q|-1}$ at the end. Thus, the actual sequence of visited states is assumed to be $q_0, z_1, \dots, z_T, q_{|Q|-1}$ with $z_i \in Q$.

According to the previous restrictions, a is defined for each pair of states $(q_i, q_j) \in Q - \{q_{|Q|-1}\} \times Q - \{q_0\}$ as:

$$a(q_i, q_j) = P(z_t = q_j|z_{t-1} = q_i) \quad (2.7)$$

As a probability mass function, a must guarantee that:

$$\sum_{q_j \in Q - \{q_0\}} a(q_i, q_j) = 1, \forall q_i \in Q - \{q_{|Q|-1}\} \quad (2.8)$$

- b is the emission probability distribution. First of all, for sake of simplification, the initial and final states, q_0 and $q_{|Q|-1}$, are assumed not to emit any observation vector. Thus the sequence of observed states $q_0, z_1, \dots, z_T, q_{|Q|-1}$ of $T + 2$ elements, emits the observed sequence $\vec{x}_1, \dots, \vec{x}_T$ of T elements.

Additionally, an independence assumption is made for this conditional probability distribution. The observed feature vector \vec{x}_t at time t may depend upon all previously visited states z_1, \dots, z_t and observed vectors $\vec{x}_1, \dots, \vec{x}_{t-1}$. The *output independence assumption* assumes that \vec{x}_t only depends on the state at time t , that is, z_t .

$$p(\vec{x}_t|z_1, \dots, z_t, \vec{x}_1, \dots, \vec{x}_{t-1}) = p(\vec{x}_t|z_t) \quad (2.9)$$

Hence, b is defined for each pair $(q_i, \vec{x}) \in Q - \{q_0, q_{|Q|-1}\} \times X$ as:

$$b(q_i, \vec{x}) = p(\vec{x}_t = \vec{x} | z_t = q_i) \quad (2.10)$$

As a probability distribution function, b must guarantee:

$$\int_{\vec{x} \in X} b(q_i, \vec{x}) d\vec{x} = 1, \quad \forall q_i \in Q - \{q_0, q_{|Q|-1}\} \quad (2.11)$$

In ASR and HTR applications, the emission distribution function is usually modeled as a Gaussian Mixture Model (GMM), which is a weighted sum of K Gaussian distributions.

$$b(q_i, \vec{x}) = \sum_{k=1}^K \lambda_{i,k} b_k(q_i, \vec{x}) \quad (2.12)$$

where,

$$b_k(q_i, \vec{x}) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_{i,k}|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_{i,k})' \Sigma_{i,k}^{-1} (\vec{x} - \vec{\mu}_{i,k})\right) \quad (2.13)$$

- $\mu_{i,k}$ is the mean vector of the component k of state q_i .
- $\Sigma_{i,k}$ is the covariance matrix of the component k of state q_i .
- $\lambda_{i,k}$ is the weight of the component k of state q_i , and must satisfy:

$$\begin{aligned} \lambda_{i,k} &\geq 0 \\ \sum_{k=1}^K \lambda_{i,k} &= 1 \end{aligned}$$

Other distributions used in HTR or ASR applications are mixtures of Bernoulli distributions [18] or distributions estimated using Neural Networks [48, 10].

There are three main problems regarding the use of HMM for Pattern Recognition applications.

- **Evaluation:** To compute the probability of a sequence of observations \mathbf{x} , given the model M , i.e. $p(\mathbf{x}|M)$. This is simply known as the *emission probability* of a sequence. A naive computation of this probability entails an exponential cost with the length of the sequences, since the probability that each sequence of states gives to \mathbf{x} must be summed up to obtain the total probability, that is:

$$p(\mathbf{x}|M) = \sum_{\mathbf{z} \in Q^T} p(\mathbf{x}, \mathbf{z}|M) \quad (2.14)$$

The previous can be efficiently computed by using two central algorithms for HMM, both based on Dynamic Programming: the forward and backward algorithms. Any of these two algorithms can be used to compute the emission probability of \mathbf{x} and both have the same time and space computational costs. This reduces the computation of $p(\mathbf{x}|M)$ from a worst-case time complexity of $O(T \cdot |Q|^T)$ to simply $O(T \cdot |Q|^2)$, which can even be further reduced to $O(T \cdot |Q|)$, if certain topologies are assumed for the HMMs.

- **Decoding:** An other central problem related to HMMs is to compute the most likely sequence of states $\hat{\mathbf{z}}$ that emitted a sequence of observations \mathbf{x} , that is:

$$\hat{\mathbf{z}} = \operatorname{argmax}_{\mathbf{z} \in Q^T} p(\mathbf{x}, \mathbf{z}|M) \quad (2.15)$$

Viterbi algorithm is used to compute the previous probability efficiently. It is very similar to the forward algorithm and has the same time complexity $O(T \cdot |Q|^2)$, which can also be reduced to $O(T \cdot |Q|)$, under the same assumptions than forward and backward algorithms.

Viterbi algorithm is the most used algorithm in HMM-based recognition systems, since it is the one used each time an input sequence \mathbf{x} needs to be recognized into some text sequence $\hat{\mathbf{w}}$.

- **Learning:** The previous problems and algorithms assume that a HMM M is given, but when a HTR or ASR task is considered, the only data available at the beginning is a training data set \mathcal{D} , as

described in Section 2.2. The remaining problem is to optimize the transition and emission parameters of a HMM to optimize a certain criterion, based on the training data set \mathcal{D} .

The criterion used to optimize the HMMs parameters is the Maximum Likelihood, which maximizes the emission probability of the training data in \mathcal{D} . Baum-Welch algorithm, a form of Expectation-Maximization (EM) algorithm, is used to improve iteratively the parameters of a HMM until no improvements are observed. The algorithm reaches a local optimum typically with very few iterations, and the cost of each iteration is $O(N \cdot T \cdot |Q|^2)$, which is also reduced to $O(N \cdot T \cdot |Q|)$ under the same assumptions than the forward, backward and Viterbi algorithms.

In HTR and ASR applications, it is usually necessary to use a concatenation of C HMMs to model a whole word, since character HMMs (or phoneme HMMs, in the case of ASR) are used to model a whole word, which is the basic input unit. In this case, the “embedded training Baum-Welch” algorithm is used, which re-estimates the parameters of the composition of the C HMMs together. This algorithm allows to train the character models without any prior segmentation of the input sequences.

The details of the previous algorithms are omitted from this thesis, since they are not required to understand the fundamental contributions of this work. More detailed information about HMMs can be found in [20, 28, 43].

2.4 Language Models

A Language Models (LM) is used to model the prior probability of a sequence of words in a certain language. LMs are used in a wide variety of natural language processing applications such as MT, ASR, HTR or text analysis. These models attempt to capture the properties of a given language and are able to predict the next word in a word sequence, given the previous words. For a sequence of l words $\mathbf{w} = w_1, w_2, \dots, w_l$, the probability of such sequence $Pr(\mathbf{w})$ can be decomposed by using the Bayes Rule, which gives the next equation:

$$Pr(\mathbf{w}) = Pr(w_1, \dots, w_l) = Pr(w_1) \prod_{i=2}^l Pr(w_i | w_1, \dots, w_{i-1}) \quad (2.16)$$

Usually, the estimation of the probability of arbitrary sequences becomes intractable since the length of these sequences is unrestricted and many of them are not observed during the training of the LM. For any sequence not seen during training, the model will assign a null probability to it. Observe that, if $|V|$ is the number of words in a given vocabulary V , the number of different sentences that can be composed with l words is $|V|^l$. Thus, if one needs to accurately estimate the probability of sequences of length l , the number of training sequences grows exponentially with that length, which clearly becomes a problem when the length of the sequences is unrestricted, which is the case of HTR. In fact, this problem is one of the manifestations of the well-known *curse of dimensionality* phenomenon [3, 35], which is usually one of the main handicaps of many ML models.

In order to face the *curse of dimensionality* in Language Models, n -gram models are used. These models assume that only the $n - 1$ most recent words in a sequence are the ones that determine the probability distribution of the next word. That is, the probability of the word w_i is only conditionally dependent of $w_{i-n+1}, \dots, w_{i-1}$, which is called the context of w_i . Hence, $Pr(\mathbf{w})$ is approximated by:

$$Pr(\mathbf{w}) \approx Pr(w_1) \prod_{i=2}^{n-1} Pr(w_i | w_1, \dots, w_{i-1}) \prod_{i=n}^l Pr(w_i | w_{i-n+1}, \dots, w_{i-1}) \quad (2.17)$$

The Maximum Likelihood estimation of the conditional probability $Pr(w|\mathbf{v})$, given a set of sentences, is computed by:

$$P(w|\mathbf{v}) = \frac{C(\mathbf{v}w)}{C(\mathbf{v})} \quad (2.18)$$

where $C(\mathbf{v})$ is the number of times that the sequence \mathbf{v} has appeared in any of the sentences in the training data set.

However, n -gram models are still sensible to the *curse of dimensionality* when a large context is considered, and thus, many of the sequences encountered on test data may be unseen during training, causing that $P(w|\mathbf{v}) = 0$ in these cases. In aim to improve the estimation of such sequences, smoothing techniques are used to give some probability mass to these sequences. The two main smoothing methods are *interpolation* [20] and *back-off* [23].

In the experiments conducted in this work, the chosen smoothing method was the Kneser-Ney *back-off*, presented in [26], which is one of the most used techniques for n -grams smoothing. In the Kneser-Ney smoothing, a constant D is discounted from the n -gram count. The main idea is to use a modified probability estimate for lower order n -grams. Specifically, the probability for a lower order n -gram is taken to be proportional to the number of unique words that precede it in the training data. Further details of the Kneser-Ney algorithm are omitted, since they are not relevant for the understanding of the central contributions of this work. Nevertheless, more details about this method and others can be found in [6].

Language Models and HMMs can be both interpreted as Stochastic Finite State Machines (SFSM), which provide an easy integration into a single SFSM. A version of the Viterbi algorithm, described on section 2.3, can be used on the top SFSM to obtain the best recognition hypothesis. Figure 2.3 shows an scheme of this integration. Additional details can be found in [44, 50, 51].

2.5 Word Graphs

Viterbi decoding algorithm described in section 2.3 can be adapted to provided multiple decoding hypotheses with their associated probability, instead of the single best hypothesis. Furthermore, as it will be explained further, these multiple hypotheses can be represented in form of a graph, that, when normalized appropriately provides certain features very useful for the task of general handwriting recognition and particularly, keyword spotting for handwritten image lines.

A word graph (WG) is a labeled weighted directed acyclic graph (WDAG) whose edges are labeled with words and weighted with scores

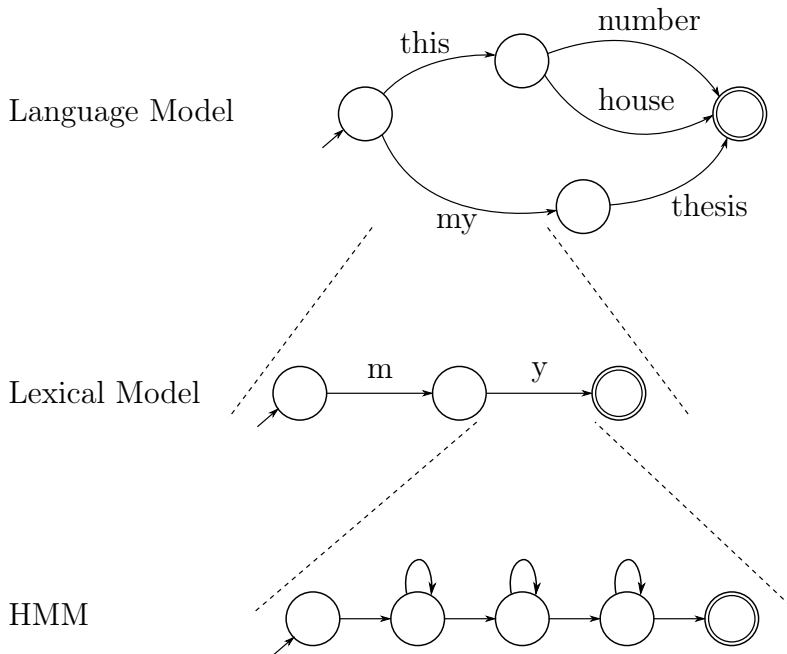


Figure 2.3: A hierarchical FSM constructed by integrating Character HMMs, Lexical Models and LMs.

derived from the HMM (likelihood) and N-gram (prior) probabilities computed during the line image decoding process. Formally, a word graph, G , is a tuple $(Q, V, E, q_1, F, \tau, \omega, \rho)$.

- Q is a finite set of nodes. Given that word graphs are DAG, a topological order on the nodes may be assumed. Each node q is labeled with its corresponding index in this order: $Q = \{1, 2, \dots, |Q|\}$.
- V is a vocabulary, a non-empty set of words.
- $E \subset Q \times Q$ is a finite set of edges connecting pairs of nodes in Q . Each edge is denoted by its starting and ending nodes: $e = (i, j), i, j \in Q, i < j$.
- $q_I \in Q$ is the initial node. For most cases, $q_I = 1$.
- $F \subset Q$ is the set of final nodes.

- $\tau : Q \rightarrow \{1, \dots, T\}$ is a function that associates each node with the index of a vector from the input sequence. In handwritten images, that is the horizontal position in the image. Usually, $\tau(q_I) = 1$ and $\tau(q) = T, \forall q \in F$.
- $\omega : E \rightarrow V$ is a function associating each edge with a word from the vocabulary. Given the edge $e = (i, j)$, $\omega(e)$ is a word hypothesis represented between the positions $\tau(i)$ and $\tau(j)$ of the input sequence. An alternative notation for ω will sometimes be used to use pair of nodes $Q \times Q$ as the domain, instead of edges E . That is, $\omega(i, j) = \omega(e)$, such that $e = (i, j)$.
- $\rho : E \rightarrow [0, 1]$ is a probability function for each edge. The meaning of this probability depends on the normalization of the word graph, but $\rho(e)$ typically refers to the probability of the hypothesis that $\omega(e)$ appears between the positions $\tau(i)$ and $\tau(j)$ of the decoded sequence. An alternative notation for ρ will sometimes be used to use pair of nodes $Q \times Q$ as the domain, instead of edges E . That is, $\rho(i, j) = \rho(e)$, such that $e = (i, j)$.

As mentioned before, word graphs encode multiple word sequences hypotheses of an image line. A word sequence is represented by the words given by the ω function across a *complete path* in the word graph: a path from the initial node to any of the final nodes. More formally, given a path $\phi = (e_1, e_2, \dots, e_l)$, where $e_i \in E, e_1 = (1, j_1), e_l = (i_l, j_l)$ and $j_l \in F$, the word sequence associated to this path is $\mathbf{w} = \omega(e_1) \cdot \omega(e_2) \cdots \omega(e_l)$.

As a matter of fact, each complete path in the word graph encodes a different alignment between a sequence hypothesis and different positions of the input sequence of feature vectors. In principle, since the same word sequence can be aligned in different ways to an input sequence, different paths on the word graph can actually represent the same word sequence, but with a different alignment to the input image.

Word graphs can be restricted so that they contain only a single alignment for each word sequence (typically, the one with the highest likelihood), and thus, only a complete path exists for each sequence of words \mathbf{w} . This class of word graphs are called *unambiguous* and from now on, we will assume that the considered word graphs are unambiguous in order to simplify the following expressions. The complete path encoding

the sentence \mathbf{w} will be denoted by $\phi_{\mathbf{w}}$ and the edges composing this path, will be denoted as $e \in \phi_{\mathbf{w}}$.

The joint probability of a word sequence \mathbf{w} and the input image \mathbf{x} can be approximated using the scores on the edges of a word graph.

$$p(\mathbf{w}, \mathbf{x}) \approx \prod_{e \in \phi_{\mathbf{w}}} \rho(e) \quad (2.19)$$

A naive computation of the previous equation can be prohibitively expensive in terms of computation time, due to the large number of sequence hypotheses encoded in word graphs. However, it can be very efficiently computed using Dynamic Programming by means of a forward (α) and backward-like (β) functions.

$$p(\mathbf{x}) \approx \beta(q_I) = \sum_{q \in F} \alpha(q) \quad (2.20)$$

where:

$$\alpha(q) = \begin{cases} 1 & q = q_I \\ \sum_{\forall i \in Q: (i, n) \in E} \alpha(i) \rho(i, n) & q \neq q_I \end{cases} \quad (2.21)$$

$$\beta(q) = \begin{cases} 1 & q \in F \\ \sum_{\forall j \in Q: (q, j) \in E} \rho(n, j) \beta(j) & n \notin F \end{cases} \quad (2.22)$$

Finally, the word sequence posterior probabilities $P(\mathbf{w}|\mathbf{x})$ can be approximately computed as:

$$P(\mathbf{w}|\mathbf{x}) = \frac{p(\mathbf{w}, \mathbf{x})}{p(\mathbf{x})} \quad (2.23)$$

Usually, algorithms used for extracting the word graph introduce a parameter to specify the maximum input degree of each word graph node, which limits the amount of information retained in the word graph. Additionally, beam-search and other pruning techniques can be applied to accelerate the Viterbi search through the word graph.

Figure 2.4 shows an example of a word graph for a text line image. The word graph represents a set of possible transcription hypotheses for the Spanish sentence “*antiguos ciudadanos que en Castilla se llamaban*”.

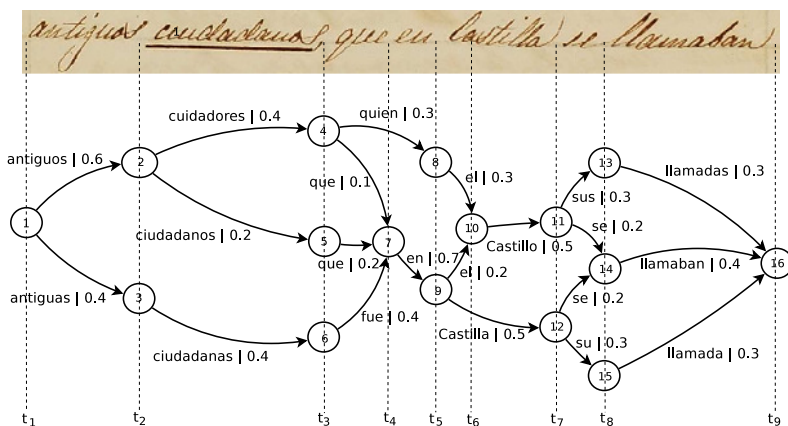


Figure 2.4: An example of a word graph for an image of the sentence “*antiguos ciudadanos que en Castilla se llamaban*”.

In the example figure, the word over an edge e is the value of $\omega(e)$ and the number that is drawn together the word is the value of $\rho(e)$. The t_i values that are written in the bottom of the image are the values of the function τ for different nodes (the nodes which are crossed by the vertical lines over each of the t_i values), that is: $\tau(1) = t_1, \tau(2) = \tau(3) = t_2, \dots, \tau(t_{16}) = t_9$.

Observe that each word sequence is represented by a single complete path in the word graph. There are 14 different sequences encoded in the previous word graph. Some of these sequences are listed below, together with their joint and conditional probabilities.

- $\mathbf{w}_1 =$ “antiguos cuidadores que en el Castillo sus llamadas”
 $p(\mathbf{w}, \mathbf{x}) = 0.6 \cdot 0.4 \cdot 0.1 \cdot 0.7 \cdot 0.2 \cdot 0.5 \cdot 0.3 \cdot 0.3 = 0.0001512$
 $P(\mathbf{w}|\mathbf{x}) = 0.0153771$
- $\mathbf{w}_2 =$ “antiguos cuidadores que en el Castillo se llamaban”
 $p(\mathbf{w}, \mathbf{x}) = 0.6 \cdot 0.4 \cdot 0.1 \cdot 0.7 \cdot 0.2 \cdot 0.5 \cdot 0.2 \cdot 0.4 = 0.0001344$
 $P(\mathbf{w}|\mathbf{x}) = 0.0136685$
- $\mathbf{w}_3 =$ “antiguos cuidadores que en Castilla se llamaban”
 $p(\mathbf{w}, \mathbf{x}) = 0.6 \cdot 0.4 \cdot 0.1 \cdot 0.7 \cdot 0.5 \cdot 0.2 \cdot 0.4 = 0.000672$
 $P(\mathbf{w}|\mathbf{x}) = 0.0683427$

- $\mathbf{w}_4 = \text{“antiguos cuidadores que en Castilla su llamada”}$
 $p(\mathbf{w}, \mathbf{x}) = 0.6 \cdot 0.4 \cdot 0.1 \cdot 0.7 \cdot 0.5 \cdot 0.3 \cdot 0.3 = 0.000756$
 $p(\mathbf{w}|\mathbf{x}) = 0.0768855$
- $\mathbf{w}_5 = \text{“antiguos cuidadores quien el Castillo sus llamadas”}$
 $p(\mathbf{w}, \mathbf{x}) = 0.6 \cdot 0.4 \cdot 0.4 \cdot 0.3 \cdot 0.5 \cdot 0.3 \cdot 0.3 = 0.000972$
 $p(\mathbf{w}|\mathbf{x}) = 0.0988528$

Once the word graph has been constructed, the Recursive Enumeration Algorithm (REA) [21] can be used to compute the n -best sequence hypotheses, according to the word graph.

CHAPTER 3

KEYWORD SPOTTING

3.1 Keyword Spotting for Handwritten Text Images

In Chapter 1, we already introduced the concept of Keyword Spotting (KWS) and what it tries to answer: to determine in which set of documents a keyword, given by the user, is present. Optionally, it may be also interesting to detect in which position of the document the keyword is represented.

While Keyword Spotting can be applied to many forms of input signals, like speech or optical text, in this work we restrict the domain to handwritten text images and from now, this scenario will be assumed.

Keyword Spotting systems are organized according to the type of query the user introduces to the system, which constitutes a first high-level taxonomy of such systems.

Query-by-Example: The query is given as an example image (or a few example images) that is matched with different regions from the collection of documents. The system will return the list of regions which it believes represent the same keyword as the given examples.

This approach can be implemented in such a way that it does not need any text block extraction or segmentation technique [39]. First, at a processing step a set of overlapping patches (represented by a bag-of-visual words) is extracted from the documents in the

database. In the spotting step, the example keyword is considered as a single patch, from which the previous bag-of-visual words is extracted. Then, using a similarity measure a list of patches from the database is retrieved and sorted according to this similarity. Finally, a voting scheme is used to find the regions of the document image having more accumulation of evidences that the user query is more probable to be present.

If the text is segmented into individual lines, it is better to extract a sequence of features from the handwritten text, using a small sliding window procedure. This allows to apply techniques designed to compare sequences. The first proposed method for handwritten KWS, which is one of the most used, follows this procedure [31]. It uses dynamic time warping (DTW) to define a distance measure between two sequences. However, DTW typically requires segmented word images, although some works avoid the word-segmenting requirement using DTW with line images [27]. Anyhow, DTW is not a learning-based approach, which limits its ability to generalize and results in one of its main shortcomings, considering the huge variability encountered in handwritten text. The other drawback of DTW-based methods is their computational cost. Recently, it has been proposed to combine a fast selection of template candidates with a learning-based version of DTW, which learns variations of the training sequences to improve generalization [40].

Query-by-String: The keyword is given as a string of symbols (characters). The previous approach requires a query example (or a set of queries) in order to find similar image regions in the database. In contrast, Query-by-String allows to search for any arbitrary word. Thus, the handwritten text has to be modeled using pattern recognition approaches like the ones described in Chapter 2.

Previous works use character-based HMMs and a Gaussian Mixture Model to test how well the queries can be explained with the emissions of the HMMs in the database [37]. Both models are trained on the same training data and the likelihood ratio of the keyword-HMM to the GMM filler model acts as the matching score.

An alternative model [12, 13], also based on HMMs, proposes to

train character-HMMs and use these models to assemble keyword HMMs for spotting words. This method works on unsegmented lines, and is one of the most popular methods. In Section 3.2, further details of this method will be discussed.

Recently, an approach based on word graphs constructed from HMMs and LMs has been presented [46]. This approach does not require any word or character segmentation either and, by using language models, permits to take into account the context of each spotted word, which yields in a better KWS accuracy. Moreover, this approach offers the possibility to build a hierarchical index to speed up the lookup times. However, the use of a word LM restricts the queries to a certain vocabulary. This approach is the center of this thesis work, which tries to provide solutions for this latter problem. The details of this method will be described in Section 3.3.

Finally, recurrent neural networks trained to predict the occurrence probability of each keyword character at each position in a text line, have been applied to KWS [15, 16]. Given a keyword, the most likely path that visits the corresponding characters in the correct order is then computed using dynamic programming.

Query-by-String is usually preferred over Query-by-Example, since users may not have an example of the keyword that they are looking for, but they can usually typeset it. Hereafter, we will assume the Query-by-String model when we refer to KWS, unless the opposite is specified.

As seen in the previous examples, the definition of “document” may vary among different levels of abstraction. For instance, in the scenario of KWS applied to HTR, a “document” can be anything from a single image line, to a whole page or manuscript. In some applications, may be enough to determine whether a certain keyword is present in a page, if the pages do not contain much text, other may require to determine in which exact line is the text written, and other may even require to determine the exact region within the line. From now on, we will assume the line-level KWS approach, from which all the theory of this work is developed.

KWS systems usually compute a score $S(\mathbf{x}, v)$ for each line \mathbf{x} in the database and the keyword v . Then, all lines with a score greater than or

equal to a certain confidence threshold T are retrieved. The threshold T allows the user to control the *precision-recall trade-off* for each query. In the Information Retrieval field, *precision* basically measures the portion of retrieved lines that are relevant and *recall* measures the portion of relevant lines that are retrieved.

For instance, suppose the user wants to find information about the H.M.S. Beagle from a collection of logbooks. It may look for the keyword “beagle” with a medium confidence threshold. The KWS system will then retrieve several lines: some of them relevant (“hits” or “true positives”), where the keyword “beagle” actually is present in the line, and some other will be misidentified lines, where the system spotted the “beagle” keyword but it does not actually appears (“false positives”). On the other hand, the system will probably miss also some lines where the keyword is written but a lower score is assigned (“false negatives”).

By decreasing the confidence threshold T , the user will start getting some of the missing true lines where “beagle” is present but also other lines where the system *thought* that “beagle” was present (i.e. *recall* increases). Increasing the confidence threshold has the opposite effect: some “false positives” will be discarded, but it is also likely that some “hits” are lost, since their score is not high enough (i.e. *precision* increases). When $T \rightarrow -\infty$, all lines are retrieved achieving the highest *recall*, but also, the lowest *precision*. When $T \rightarrow +\infty$, no line score exceeds the threshold and no results are retrieved for the query.

More details about the *precision* and *recall* measures will be introduced in Section 5.2. For now, the important factor is that by providing the score $S(\mathbf{x}, v)$, the system is able to provide the user a tool for customization, which enables to personalize for each query the minimum degree of accuracy that the user expects.

3.2 Filler approach

One of the most successful techniques to model the score $S(\mathbf{x}, v)$, for a given handwritten line image \mathbf{x} and a given keyword v , is known as the *Filler approach*. It was first used in the field of Automatic Speech Recognition [30, 24, 25]. The approach uses character-HMMs to build a *query-specific* model K_v for the keyword v and a *filler* or *garbage* model

F . It has been recently applied to handwritten KWS with considerable success [12, 13].

The filler HMM allows to recognize any sequence of characters. This is done using a composed HMM with an initial state linked to all the trained character HMMs in parallel and connecting each of them to a final state which has a loop-back to the initial state, in order to allow multiple strings with any arbitrary length to be decoded. The resulting HMM is depicted in Figure 3.1a.

The keyword HMM, shown in Figure 3.1b, allows the recognition of the particular queried keyword. It is constructed by the concatenation of the character HMMs forming the keyword surrounded by the space HMM and embedding the filler HMM at the beginning and the end of the combined HMM. The initial state is linked to the filler and the first character HMM of the keyword and the edges to the final state come from the second filler HMM and the last character HMM of the keyword. This allows to recognize the query at any position of the line, no matter if the line is formed by multiple words or if it is formed only by the keyword.

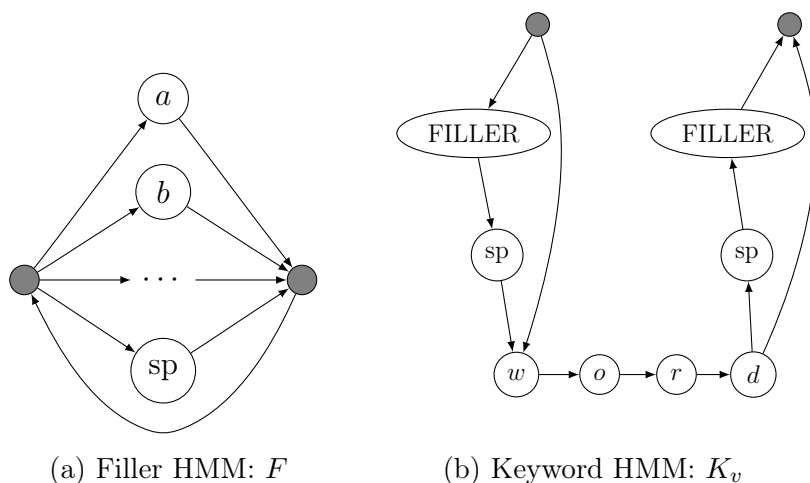


Figure 3.1: Compositions of HMMs used by the Filler approach.

The states of the character HMMs usually use GMMs emission distributions, as explained in Section 2.3.

In the spotting phase, a score $S_F(\mathbf{x}, v)$ is computed as described in Equation (3.1), where $p(\mathbf{x}|K_v)$ is the probability of the whole image \mathbf{x} given by the keyword HMM K_v , $p(\mathbf{x}|F)$ is the probability given by the filler HMM and L_v is the number of frames from the start to the end of the keyword v , according to the alignment given by K_v .

$$S_F(\mathbf{x}, v) := \frac{\log p(\mathbf{x}|K_v) - \log p(\mathbf{x}|F)}{L_v} \quad (3.1)$$

Intuitively, the probability given by the filler HMM will be greater than the one given by the keyword HMM, since the latter is a more restrictive modeling than the former. On the other hand, the probability computed by the keyword-specific model, K_v , will be greater on image lines that contain the modeled keyword than on image lines which contain other text, since the restrictions imposed in the keyword-specific model will dampen the probability given by the HMM. Thus, in the best case, the probability given by K_v would be similar to the given by F , if the keyword is present in line \mathbf{x} . If the keyword is not in the line image, $p(\mathbf{x}|K_v)$ will be typically much smaller than $p(\mathbf{x}|F)$. The difference of the logarithm of these two probabilities is used as the score, as shown in Equation (3.1). A formal justification of this score can be found in [13].

The main advantage of this model is that it does not require a word LM. This implies that any keyword can be spotted, even if that keyword was not part of the original training set. This provides the system with a huge versatility. However, LMs have been shown to be useful to boost the KWS accuracy and character LMs have been used together with the Filler approach [11]. Nevertheless, the main issue with the Filler-based approaches is that the lookup times are usually huge, since one Viterbi run has to be done for each keyword and for each line in the collection of documents.

This yields in a $O(|\mathcal{D}| \cdot |q| \cdot \gamma)$ running time for a single query v , where $|\mathcal{D}|$ is the number of lines in the collection of documents, $|q|$ is the number of characters of the query and γ is a constant that depends on the total number of states in all character HMMs in F and K_v and on the square of the number of character HMMs in F .

3.3 Word Graph approach

An alternative to diminish the computational cost of the search phase introduced by the Filler approach is the, so called, Word Graph approach. This approach uses the concept of word graphs introduced in Section 2.5. For each line in the collection, a word graph is computed and stored after the training phase. With the proper normalization, a score is obtained directly from the WG for each word in the training vocabulary. This score is stored during indexing. On the search phase, the only task to be done is to find in which WG the query keyword has a score that surpasses the confidence thresholds set by the user. The main publications presenting this method for handwritten KWS are [46, 47].

3.3.1 Frame-level word posterior

The first step to fully explain how this approach works is to introduce the *word posterior probability at frame level*, which accounts for the degree of uncertainty about a given keyword v being present in a specific horizontal position i within the line image, represented by the sequence of feature vectors $\mathbf{x} = \vec{x}_1, \vec{x}_2, \dots, \vec{x}_n$, where each vector \vec{x}_i corresponds to a frame of the image. This probability distribution can be marginalized considering all possible non-nested intervals containing the frame i , which gives Equation (3.2).

$$P(v|\mathbf{x}, i) = \sum_{k=1}^i \sum_{l=i}^n P(v, k, l|\mathbf{x}, i) \quad (3.2)$$

Applying Bayes rule to the previous result gives:

$$P(v|\mathbf{x}, i) = \sum_{k=1}^i \sum_{l=i}^n P(k, l|\mathbf{x}, i) P(v|\mathbf{x}, k, l, i) \quad (3.3)$$

Finally, assuming that the probability $P(k, l|\mathbf{x}, i)$ is distributed uniformly among all $[k, l]$ intervals containing i , and that $P(v|\mathbf{x}, k, l, i)$ is independent of i (since $i \in [k, l]$), gives the next equation:

$$P(v|\mathbf{x}, i) \approx \lambda \sum_{k=1}^i \sum_{l=i}^n P(v|\mathbf{x}, k, l); \quad \lambda = \frac{1}{i \cdot (n - i + 1)} \quad (3.4)$$

Observe that $P(v|\mathbf{x}, k, l)$ can be approximated by any state-of-the-art HTR engine, since it is the probability of the alignment of the word v to the image \mathbf{x} within the frames $[k, l]$. Obviously, the better the accuracy of the HTR engine is, the better frame-level word posterior estimation is achieved.

A naive computation of the previous equation, that could be done using a HTR engine for isolated words and all possible segmentation hypothesis of the image line, would entail a tremendous cost: at least $\Omega(|\mathbf{x}|^4)$. Nonetheless, this cost can be dramatically reduced if word graphs are used, as it will be explained in the next section.

The previous equation could be used to address KWS in a naive way: Given a keyword v to be spotted, the frame-level word posterior probabilities $P(v|\mathbf{x}, i)$ can be computed to make the frame i a spotting candidate, that would be filtered according to the given threshold T .

Nevertheless, the previous score gives a frame-level measure and lines have usually thousands of frames and such approach will be too fine-grained, since hundreds of spots would be given for each line. Thus, we aim to give a line-level global measure, without considering the specific position of the queried keyword v within the line image. In order to fulfill this requirement, the confidence score $S(\mathbf{x}, v)$ is defined as:

$$S_G(\mathbf{x}, v) := \max_{1 \leq i \leq n} P(v|\mathbf{x}, i) \quad (3.5)$$

Similar scores to the one described in the previous equation have been proposed in the past as good heuristics to obtain word and sentence recognition confidence measures in Machine Translation [49], Automatic Speech Recognition [52, 41] and Handwriting Text Recognition [42]. Once the line-level score $S_G(\mathbf{x}, v)$ is computed, lines with a score surpassing the threshold T , given by the user, are retrieved.

3.3.2 Fast computation of Frame-level Word Posteriors

It was pointed in the previous section that a naive computation of Equation (3.4) results in a huge cost. Here we summarize the main ideas presented in [46] that allow a fast computation of the previous equation by using word graphs.

In Equation (3.4), the frame-level word posterior probability $P(v|\mathbf{x}, i)$ is approximated using the alignment probability of the word v between frames k and l , i.e. $P(v|\mathbf{x}, k, l)$. Observe that this probability distribution can be approximated from the word graphs, using the fact that each edge (q, q') in the word graph has associated a start and end frames, corresponding to the values of the τ function for nodes q and q' , as explained in Section 2.5.

Using the fact that the sequence of character \mathbf{w} has a single complete path in the word graph G , we can compute the *edge posterior*, $\varphi(q, q')$, efficiently using a the forward and backward algorithms:

$$\varphi(q, q') = \sum_{\mathbf{w}:(q,q') \in \phi_{\mathbf{w}}} P(\mathbf{w}|\mathbf{x}) = \frac{\alpha(q) \cdot \rho(q, q') \cdot \beta(q')}{\beta(q_I)} \quad (3.6)$$

Observe that this defines a new normalized weight for each edge in the word graph. This weight function has the following properties, as it has been proved in [46]:

- Flow-preserving nodes:

$$\sum_{q' \in Q} \varphi(q', q) = \sum_{q'' \in Q} \varphi(q, q''), \quad \forall q \in Q \quad (3.7)$$

- Edge-posteriors are frame-level conditional distributions:

$$\sum_{(q,q') \in E: \tau(q) < i \leq \tau(q')} \varphi(q, q') = 1, \quad 1 \leq i \leq n \quad (3.8)$$

Given the previous normalization of the word graphs, then the frame-level word posterior probability $P(v|\mathbf{x}, i)$ can be approximated as:

$$P(v|\mathbf{x}, i) \approx \sum_{(q,q') \in E: \omega(q,q')=v, \tau(q) < i \leq \tau(q')} \varphi(q, q') \quad (3.9)$$

According to Equation (3.8), the previous Equation correctly defines a probability distribution with: $\sum_{v \in \Sigma^*} P(v|\mathbf{x}, i) = 1$.

The computation of $P(v|\mathbf{x}, i)$ can be performed efficiently by sequentially visiting the WG edges and updating, for edge (q, q') the value of

$P(v|\mathbf{x}, i)$. Let be $\omega(q, q') = v$, $\tau(q) = i$, $\tau(q') = k$, then the counter of frames $i < j \leq k$ for the keyword v are updated. When all edges have been visited, the absolute frequencies are normalized per frame.

The computing time is then proportional to the number of edges and the average length of an edge, i.e. $\tau(q') - \tau(q)$. As the authors of the previous computation explain, this is, in fact, $\Theta(\kappa \cdot |\mathbf{x}|)$, where κ is a relatively small constant that depends on the WG input degree.

The cost of the following steps determine the overall cost of computing $S_G(\mathbf{x}, v)$:

- word graph generation and normalization, $O(\Gamma \cdot |\mathbf{x}|)$.
- word graph-based computation of the frame-level word posterior probabilities $P(v|\mathbf{x}, i)$, $\Theta(\kappa \cdot |\mathbf{x}|)$.
- maximization step for the final calculation of $S_G(\mathbf{x}, v)$, $\Theta(|\mathbf{x}|)$.

Given that, the final cost of $S_G(\mathbf{x}, v)$ is $O((\Gamma + \kappa) \cdot |\mathbf{x}|)$, where both Γ and κ depend on the input degree of the word graph. This may become too expensive for large collections of documents and/or dense word graphs. Nevertheless, this cost is incurred only during the document collection processing needed to create the indexes. Once the indexes have been created, the lookup time is extremely fast.

OUT OF VOCABULARY QUERIES

As mentioned in Section 3.3, one of the main benefits of using word graphs for KWS is that, by using contextual information, a better accuracy is achieved in front of the Filler model, which is a lexicon-agnostic approach. Word graphs assume that a vocabulary of words is given, which can be used to create an index that allows for fast searches. However, the use of word Language Models has a non-trivial implication: all words out of that vocabulary are not part of the modeled language, and thus, their prior probability is zero. The words which are not part of a Language Model are called out of vocabulary (OOV) words.

For any query consisting in an OOV keyword, the word graph approach presented in the previous chapter will assign a null score to any line in the database. Only when the user sets the threshold to the minimum value, the system will report some line. Unfortunately, all lines will be reported in this scenario (since all of them have the same null score), which clearly limits the usability of the system for such queries, since it may be inadmissible to the user to review all the image lines, specially if he or she is working with a huge collection of documents.

In further sections we will present different alternatives that try to give a reasonable score for every possible keyword, in aim to make the WG-based Keyword Spotting system more robust and flexible.

Most approaches presented here are based in the same idea: When an OOV keyword is presented to the system to be spotted, its score will be based on the score of *similar* in-vocabulary keywords, indexed by the system. We will refer to this procedure as the score *smoothing* of an

OOV keyword.

First, two alternative *similarity* measures are presented. The first one is the well-known Levenshtein edit distance. The second one, is a probability distribution based on Stochastic Error Correcting, that we call *confusion probability*. These alternative measures are presented in 4.1.

Then, we propose different *smoothing* techniques at different stages of the spotting step. In Section 4.2, two line-level smoothing approaches will be presented that try to use the line-level score $S_G(\mathbf{x}, v)$ computed for the indexed keywords using the word graph approach.

Section 4.3 will present a method which is done at each frame by smoothing $P(v|\mathbf{x}, i)$ (see Section 3.3.1), using more available information than the previous methods, but requiring more computation.

The last smoothing technique uses the line score $S_F(\mathbf{x}, v)$, computed by the Filler model, to estimate the score of OOV keywords, given that this model is able to give a reasonable score for any keyword.

Finally, an heuristic approach to combine the smoothed score for out of vocabulary queries and in-vocabulary queries will be presented (Section 4.5) and the different time and space requirements of the presented smoothing techniques will be commented in Section 4.6.

4.1 Similarity measures among strings

4.1.1 Levenshtein Edit Distance

The Levenshtein edit distance is defined as the minimum number of insertion, deletion and substitution (edition) operations that are required to transform a string u into v [29]. For instance, in order to transform the string “handwriting” into “handwritten”, at least, three of these edit operations are required:

1. Replace the second “i” by “t”:
 $handwriting \rightarrow handwritng$
2. Replace the second “n” by “e”:
 $handwritng \rightarrow handwritteg$

3. Replace the last “g” by “n”:
handwritteg → *handwritten*

The Levenshtein distance is a well-known distance measure among strings which can be computed using the following recursive equation, for strings $u = u_{1,m} = u_1, \dots, u_m$ and $v = v_{1,n} = v_1, \dots, v_n$:

$$d(u_{1,m}, v_{1,n}) = \begin{cases} 0 & n = m = 0 \\ n & n > 0 \wedge m = 0 \\ m & n = 0 \wedge m > 0 \\ \min \begin{cases} d(u_{1,m-1}, v_{1,n}) + 1 \\ d(u_{1,m}, v_{1,n-1}) + 1 \\ d(u_{1,m-1}, v_{1,n-1}) + \delta(u_m, v_n) \end{cases} & n > 0 \wedge m > 0 \end{cases} \quad (4.1)$$

where the δ function is defined as:

$$\delta(a, b) = \begin{cases} 1 & a \neq b \\ 0 & a = b \end{cases} \quad (4.2)$$

The previous equation allows for an efficient implementation using dynamic programming with a time complexity of $O(|u| \cdot |v|)$.

The problem that presents the use of the Levenshtein distance for the particular problem of KWS, is that it is blind to the morphological structure of the symbols that compose both string u and v . Thus, the Levenshtein distance between “hello” and “halo” is the same as “hello” and “hZlXo” (2 edit operations are required), but the latter transformation is very unlikely to happen in regular applications using natural English language.

4.1.2 Stochastic Error Correcting Probability

In order to cope with the problem presented by the Levenshtein distance, we define an alternative similarity measure, based on the Stochastic Error Correcting approach [1]. The key idea is to compute a *confusion probability* $P(u|v)$ which measures how likely is the HTR system to detect v instead of u , where $u \in \Sigma^*$ and $v \in V$, the vocabulary used for training. Following the ideas proposed by the previously cited paper,

we model the conditional distribution $P(u|v)$ using a Stochastic Finite-State Machine (SFSM) for each keyword $v \in V$. Given the keyword $v = v_{1,n} = v_1, \dots, v_n$, a SFSM $G = (Q, \Sigma, \epsilon, q_0, q_n, E, \rho)$ is defined:

- $Q = \{q_0, \dots, q_n\}$ is the set of $n + 1$ states of the SFSM.
- Σ is the alphabet of the string u and v , that is, the set of characters from which all possible strings u and v are constructed.
- ϵ is the null symbol. A string composed only by the null symbol is called empty string.
- q_0 is the initial state of G.
- q_n is the final state of G.
- $E \subset Q \times Q \times (\Sigma \cup \{\epsilon\})$ is the set of edges connecting the states of the SFSM with a symbol in $\Sigma \cup \{\epsilon\}$. There are edges connecting consecutive states for each symbol $b \in \Sigma \cup \{\epsilon\}$, which represent the substitution and deletion operations of symbols in v . Moreover, there are edges forming loops for each state $q_i \in Q$ and each symbol $b \in \Sigma$, which represents the insertion of a new symbol in u .
- $\rho : \Sigma \times \Sigma \rightarrow \mathbb{R}$ is the weight function, which represent the probability of transforming a symbol a into a symbol b . Suppose that $v_{i+1} = a$, then the edge from node q_i to q_{i+1} and labeled with symbol b , has associated the weight $\rho(v_{i+1}, b) = \rho(a, b)$.

Observe that a SFSM defined in such way can accept any string in Σ^* . Figure 4.1 shows the SFSM defined for the string “aab”.

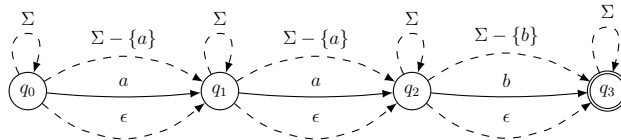


Figure 4.1: SFSM used to compute $P(u|aab), \forall u \in \Sigma^*$. Erroneous edit operations are represented by dashed lines.

The ρ function is estimated from a set of pairs of strings using the Levenshtein distance alignment among those pairs. Since the final objective is to account for the *confusion probability* of the HTR engine used by the KWS system, a validation dataset is recognized using the trained HTR engine, and the frequencies of each edit operations are computed between the best decoding of the HTR system and the ground truth of the validation set, that is, how many times the symbol a was transformed into b . We will refer to this frequency as $C(a, b)$. The absolute frequency $C(a, b)$ can be divided by the number of times that the symbol a was transformed, i.e. $C(a)$. This is the Maximum Likelihood Estimation of the conditional distribution $P(b|a), \forall a, b \in \Sigma \cup \{\epsilon\}$:

$$P(b|a) = \frac{C(a, b)}{C(a)} \quad (4.3)$$

Once these conditional probabilities have been estimated for each pair of symbols a and b , the function ρ of the SFSM is defined as follows:

$$\rho(a, b) = \begin{cases} 0 & a = b = \epsilon \\ P_{\text{Ins}} \cdot P(b|a) & a = \epsilon \\ (1 - P_{\text{Ins}}) \cdot P(b|a) & a \neq \epsilon \end{cases} \quad (4.4)$$

where P_{Ins} is the probability of the insertion operation, that is:

$$P_{\text{Ins}} = \frac{\sum_{b \in \Sigma} C(\epsilon, b)}{\sum_{a', b' \in \Sigma \cup \{\epsilon\}} C(a', b')} \quad (4.5)$$

The weights defined in such way, allow us to use the defined SFSM in order to estimate the conditional distribution $P(u|v)$, for a given keyword v , using a recursive function $\alpha(i, j)$, based on the Forward algorithm. The function $\alpha(i, j)$ defines the probability of being at state q_i and having observed the sequence u_1, \dots, u_j . It is defined as follows:

$$\alpha(i, j) = \begin{cases} 1 & i = j = 0 \\ \alpha(i-1, j) \cdot \rho(v_i, \epsilon) & i > 0 \wedge j = 0 \\ \alpha(i, j-1) \cdot \rho(\epsilon, u_j) & i = 0 \wedge j > 0 \\ \alpha(i-1, j) \cdot \rho(v_i, \epsilon) + \\ \alpha(i, j-1) \cdot \rho(\epsilon, u_j) + \\ \alpha(i, j) \cdot \rho(v_j, u_j) & i > 0 \wedge j > 0 \end{cases} \quad (4.6)$$

The, the *confusion probability*, $P(u|v)$ is given by:

$$P(u|v) = (1 - P_{\text{Ins}}) \cdot \alpha(|v|, |u|) \quad (4.7)$$

The previous equation can be computed efficiently using a dynamic programming approach, which results in a $O(|u| \cdot |v|)$ time complexity, the same as the Levenshtein distance algorithm described above.

4.2 Line-level smoothing

The approaches presented here smooth the score of an out of vocabulary keyword u directly, based on the scores of similar keywords in the vocabulary. Hence, the smoothing is done for each OOV query u and each line \mathbf{x} in the database.

4.2.1 Levenshtein Distance-based smoothing

The first heuristic algorithm is based on the Levenshtein distance, presented in Section 4.1.1 to model the similarity between two keywords u and v . Using this distance metric d , the following smoothed score is defined for any keyword u :

$$\tilde{S}_L(\mathbf{x}, u) := \max_{v \in V} S_G(\mathbf{x}, v)^{1-\alpha} \cdot e^{-\alpha d(u,v)} \quad (4.8)$$

The α parameter is used to adjust the weight of the score of the in-vocabulary keyword v and the similarity measure between the two keywords u and v . In this definition, α must be in the range $[0, 1]$. Observe that the minimum distance between two strings is zero, when both strings are identical. Thus, by using the negative exponential we ensure that the factor multiplying $S_G(\mathbf{x}, v)$ has a maximum value equal to 1. On the other hand, when the Levenshtein distance tends to infinity, the exponential value tends to zero. The score of a keyword u is then restricted to the range $[0, 1]$.

4.2.2 Stochastic Error Correcting-based smoothing

The second smoothing approach focused on the line-level uses a more refined similarity measure between a OOV keyword u and a in-vocabulary

keyword v , which takes into account the morphological structure of both u and v , using the *confusion probability*, previously defined in Section 4.1.2.

Given the distribution $P(u|v)$, defined for each keyword v in the training vocabulary V , the following smoothed score is computed for any out of vocabulary keyword $u \in \Sigma^*$:

$$\tilde{S}_C(\mathbf{x}, u) := \max_{v \in V} S_G(\mathbf{x}, v) \cdot P(u|v)^\alpha \quad (4.9)$$

Same as before, the parameter α is used to adjust the weight of the scores of the in-vocabulary keywords v and the similarity measure between the keywords u and v . Here, α must be in the range $[0, \infty)$ in order to ensure that the smoothed score is in the range $[0, 1]$.

This smoothing approach suffers from a well-known problem when modeling $P(u|v)$ with a SFSM: when the length of v is increased, the maximum of the distribution $P(u|v)$ decreases. This behavior is not desirable, since the score $\tilde{S}_C(\mathbf{x}, u)$ happens to be highly dependent of the length of both u and v , no matter which line is considered. We observed that long keywords tended to have much lower scores systematically, and thus, we decided to change the way of smoothing $\tilde{S}_C(\mathbf{x}, u)$, so that the length of the keyword does not affect this score that much. We define the similarity factor $f(u, v)$ as:

$$f(u, v) = \frac{P(u|v)}{\max_{u' \in \Sigma^*} P(u'|v)^\beta} \quad (4.10)$$

The parameter β is used to tune the effect of the length-correction and it is determined experimentally using a validation set. This parameter can take any value in the range $[0, \infty)$, in order to keep consistent the following score.

Then, the score definition in Equation (4.9) is redefined as:

$$\tilde{S}'_C(\mathbf{x}, u) := \max_{v \in V} S_G(\mathbf{x}, v) \cdot f(u, v)^\alpha \quad (4.11)$$

Observe that the factor $f(u, v)$ is in the range $[0, 1]$. Hence, the rules applied to the parameter α for Equation (4.9) are also applied here, in order to make sure that $\tilde{S}'_C(\mathbf{x}, u)$ is in the range $[0, 1]$.

4.3 Frame-level smoothing

A lower-level approximation to the score smoothing of out of vocabulary queries is adopted here. The word graph frame-level word posterior distribution, defined in Equation (3.9), will be smoothed in order to give a probability to any keyword $u \in \Sigma^*$, based on the frame-level word posterior of the keywords present in the word graph, $v \in V_{G(\mathbf{x})}$, and the confusion probability $P(u|v)$. From now on, we will refer to the word graph frame-level word posterior distribution as $P(v|\mathbf{x}, i)$.

Now, the conditional distribution $\tilde{P}(u|\mathbf{x}, i)$ is defined, for any keyword $u \in \Sigma^*$. This can be marginalized among all keywords $v \in V_{G(\mathbf{x})}$, which results in the following equation:

$$\tilde{P}(u|\mathbf{x}, i) = \sum_{v \in V_{G(\mathbf{x})}} P(u, v|\mathbf{x}, i) = \sum_{v \in V_{G(\mathbf{x})}} P(v|\mathbf{x}, i) \cdot P(u|\mathbf{x}, i, v) \quad (4.12)$$

By assuming that u is conditionally independent of \mathbf{x} and i , given v , then Equation (4.12) is approximated as:

$$\tilde{P}(u|\mathbf{x}, i) \approx \sum_{v \in V_{G(\mathbf{x})}} P(v|\mathbf{x}, i) \cdot P(u|v) \quad (4.13)$$

It has been proved in the original literature that $P(v|\mathbf{x}, i)$ and $P(u|v)$ are well-defined distributions with the following properties:

$$P(v|\mathbf{x}, i) = 0, \forall v \in \Sigma^* - V_{G(\mathbf{x})} \quad (4.14)$$

$$0 \leq P(v|\mathbf{x}, i) \leq 1, \forall v \in V_{G(\mathbf{x})} \quad (4.15)$$

$$\sum_{v \in \Sigma^*} P(v|\mathbf{x}, i) = 1 \quad (4.16)$$

$$0 \leq P(u|v) \leq 1, \forall u \in \Sigma^* \quad (4.17)$$

$$\sum_{u \in \Sigma^*} P(u|v) = 1 \quad (4.18)$$

Trivially, it can be proven that $\tilde{P}(u|\mathbf{x}, i)$ is in the range $[0,1]$. Moreover, we prove that $\tilde{P}(u|\mathbf{x}, i)$ is also a well-defined and normalized con-

ditional distribution:

$$\begin{aligned}
 \sum_{u \in \Sigma^*} \tilde{P}(u|\mathbf{x}, i) &= \\
 \sum_{u \in \Sigma^*} \sum_{v \in V_G(\mathbf{x})} P(v|\mathbf{x}, i) \cdot P(u|v) &= \\
 \sum_{v \in V_G(\mathbf{x})} \sum_{u \in \Sigma^*} P(v|\mathbf{x}, i) \cdot P(u|v) &= \\
 \sum_{v \in V_G(\mathbf{x})} P(v|\mathbf{x}, i) \sum_{u \in \Sigma^*} P(u|v) &= \\
 \sum_{v \in V_G(\mathbf{x})} P(v|\mathbf{x}, i) &= 1
 \end{aligned} \tag{4.19}$$

Finally, following the definition of the line score in Equation (3.5), the smoothed score for a keyword u in a image line \mathbf{x} is given by:

$$\tilde{S}_I(\mathbf{x}, u) := \max_{1 \leq i \leq |\mathbf{x}|} \tilde{P}(u|\mathbf{x}, i) \tag{4.20}$$

This smoothing technique is also affected by the problem presented in Section 4.2.2, caused when modeling $P(u|v)$ using a SFSM. In order to face this issue, a similar heuristic to the one described in Section 4.2.2 has been introduced into Equation (4.13), resulting in the definition of $F_S(u|\mathbf{x}, i)$:

$$F_S(u|\mathbf{x}, i) = \sum_{v \in V_G(\mathbf{x})} P(v|\mathbf{x}, i) \cdot \frac{P(u|v)}{\max_{u' \in \Sigma^*} P(u'|v)^\beta} \tag{4.21}$$

If one takes into account the properties of $P(u|v)$, it can be proven than the summation $\sum_{u \in \Sigma^*} F_S(u|\mathbf{x}, i)$ only depends on the words in-

cluded in the word graph of the given image line \mathbf{x} :

$$\begin{aligned}
 & \sum_{u \in \Sigma^*} F_S(u|\mathbf{x}, i) = \\
 & \sum_{u \in \Sigma^*} \sum_{v \in V_G(\mathbf{x})} P(v|\mathbf{x}, i) \cdot \frac{P(u|v)}{\max_{u' \in \Sigma^*} P(u'|v)^\beta} = \\
 & \sum_{v \in V_G(\mathbf{x})} \frac{P(v|\mathbf{x}, i)}{\max_{u' \in \Sigma^*} P(u'|v)^\beta} \sum_{u \in \Sigma^*} P(u|v) = \\
 & \sum_{v \in V_G(\mathbf{x})} \frac{P(v|\mathbf{x}, i)}{\max_{u' \in \Sigma^*} P(u'|v)^\beta} = Z_{\mathbf{x}, i}
 \end{aligned} \tag{4.22}$$

Then, the posterior distribution $\tilde{P}'(u|\mathbf{x}, i)$ is defined as:

$$\tilde{P}'(u|\mathbf{x}, i) = \frac{1}{Z_{\mathbf{x}, i}} \cdot F_S(u|\mathbf{x}, i) \tag{4.23}$$

Finally, this smoothed frame-level word posterior probability is used to compute the finale line score for image \mathbf{x} and keyword u , as usual:

$$\tilde{S}'_I(\mathbf{x}, u) := \max_{1 \leq i \leq |\mathbf{x}|} \tilde{P}'(u|\mathbf{x}, i) \tag{4.24}$$

The parameter β is used to tune the length-correction heuristic. Observe that if $\beta = 0$, then Equation (4.23) reduces to Equation (4.13).

4.4 Filler-based scores

Here we propose to take advantage of the capability of the Filler model to deal with out of vocabulary queries, and we propose a very simple smoothing approach based on using the line scores computed by the Filler model, $S_F(\mathbf{x}, u)$ in order to estimate the score of such queries.

Since the scores produced by the Filler model are usually in a logarithmic scale in the range $(-\infty, 0]$, they are converted to the same range than the ones given by the word graph method using the following equation:

$$S'_F(\mathbf{x}, u) = \exp(S_F(\mathbf{x}, u)) \tag{4.25}$$

assuming that natural logarithms were used in the computation of $S_F(\mathbf{x}, u)$.

In fact, this approach is not an actual smoothing, since we are just using the Filler model to compute the scores of any event (in and out of vocabulary) and, thus, we are not taking advantage of the benefits that the WG-based KWS provides (lexicon-awareness and much faster time responses).

However, in the following section, we will present a combination heuristic for taking advantage of the lexicon knowledge of word graphs for in-vocabulary keywords, and the capability of the Filler model to serve out of vocabulary queries.

4.5 Combining scores with a Back-off heuristic

In previous sections, we have seen many approaches in order to smooth the score of out of vocabulary queries. The previous smoothing techniques are defined for any keyword $u \in \Sigma^*$, including the in-vocabulary keywords whose score is already well-modeled by the WG-based KWS approach.

We observed that using directly the smoothed scores for any keyword, no matter if it is a in or out of vocabulary keyword, does not result in the best performance.

For instance, the score assigned to relevant events by some of the presented methods (Sections 4.2.2, 4.3 and 4.4) have very different distributions depending on the length of the keyword, which affects the performance. We tried to provide an heuristic solution for this issue, as explained in the respective sections, however, the problem is not completely solved and better results can be achieved, if the scores given by the word graph and the presented smoothed scores are combined in a more elaborated way.

During experimentation, we observed that the following heuristic works very well for combining in-vocabulary and out of vocabulary scores:

$$S(\mathbf{x}, u) := \begin{cases} S_G(\mathbf{x}, u) & u \in V_{G(\mathbf{x})} \\ \tilde{S}(\mathbf{x}, u)^\eta & \text{otherwise} \end{cases} \quad (4.26)$$

where η , that we call *score scaling factor*, is a parameter that controls the weight of the out of vocabulary scores. It is adjusted experimentally using a validation set, as usual.

Additionally, $\tilde{S}(\mathbf{x}, v)$ can be any of the of the presented smoothing approaches:

- Levenshtein Distance-based smoothing, $\tilde{S}_L(\mathbf{x}, u)$, presented in Section 4.2.1.
- Stochastic Error Correcting-based smoothing, $\tilde{S}_C(\mathbf{x}, u)$ and $\tilde{S}'_C(\mathbf{x}, u)$, presented in Section 4.2.2.
- Frame-level smoothing, $\tilde{S}_I(\mathbf{x}, u)$ and $\tilde{S}'_I(\mathbf{x}, u)$, introduced in Section 4.3.
- Filler-based smoothing, $S'_F(\mathbf{x}, u)$, presented in Section 4.4.

This heuristic is similar to the *back-off* smoothing technique used in N -gram language models, where lower-order N -grams are used to estimate the probabilities of unseen events during training [23].

This similarity suggests that perhaps other combination techniques based on interpolation could also work well in this scenario. However, we have not explored this path.

Finally, observe that the smoothed frame-level word posterior probability $\tilde{P}(v|\mathbf{x}, i)$, presented in Section 4.3, could also be combined (by means of *back-off* or interpolation) with the word posterior given by the word graph model (Section 3.3.1), instead of directly combining the scores at line-level. It remains to be explored also if this is a better alternative.

4.6 Computational issues

Three different levels of smoothing have been presented in the previous sections, each of them entailing very different costs. While high-level smoothing techniques, proposed in Section 4.2, are relatively fast, low-level methods are much more slow, like the Filler-based score.

Regarding the first level smoothing (line level), both presented approaches have the same time complexity. A naive implementation of the line-level smoothing methods would have a time complexity of $O(|u| \cdot$

$L \cdot |V| \cdot |\mathcal{D}|$), where $|u|$ is the length of the query, $L = \max_{v \in V} |v|$ is the length of the longest indexed keyword, V is the number of indexed keywords, and $|\mathcal{D}|$ is the number of indexed documents.

Even if the information of frequent non-indexed keywords is cached, computing the distance or *confusion probability* between a new query u and all the indexed keywords would be prohibitive, for large data sets. However, different methods have been presented in the literature in order to reduce such operations based on common prefixes of the indexed keywords [5, 2], which dramatically reduce the time needed for computing these similarity measures.

The frame-level smoothing presents an additional problem, which is that it adds an additional linear dependency on the number of frames of the indexed documents. Then, if $F = \max_{\mathbf{x} \in \mathcal{D}} |\mathbf{x}|$ is the maximum length of the indexed image lines, a trivial implementation of this smoothing method would require about $O(|u| \cdot L \cdot |V| \cdot |\mathcal{D}| \cdot F)$ time steps.

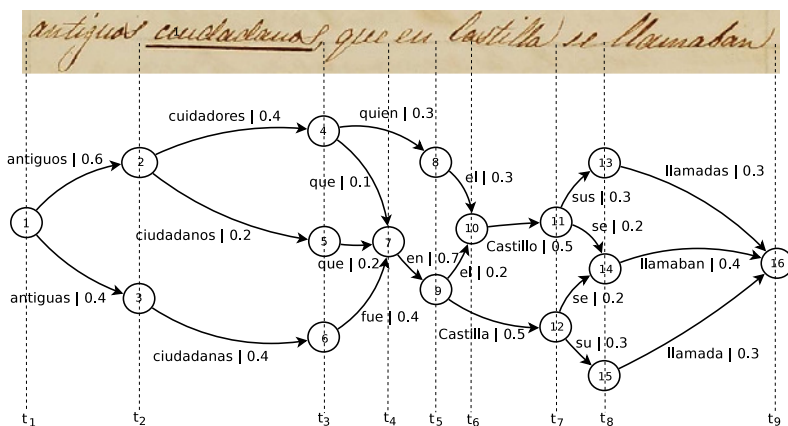
Of course, the same time reductions explained before could be used to reduce the cost of computing the *confusion probability* among the query and all indexed keywords.

Moreover, the maximization operation stated in Equation (4.13) which results in the linear dependence of F does not actually need to involve all frames of the line image, since for many of them, the frame-level word posterior will be identical. Observe that two frames that cut the same set of edges will have the same frame-level word posterior distribution. Since the set of cutting edges can only change when a new node is introduced in the word-graph, then the frame-level word posterior distribution will only change between two frames i and j , if and only if, a node q has a time-stamp $\tau(q)$ in the range $i < \tau(q) < j$.

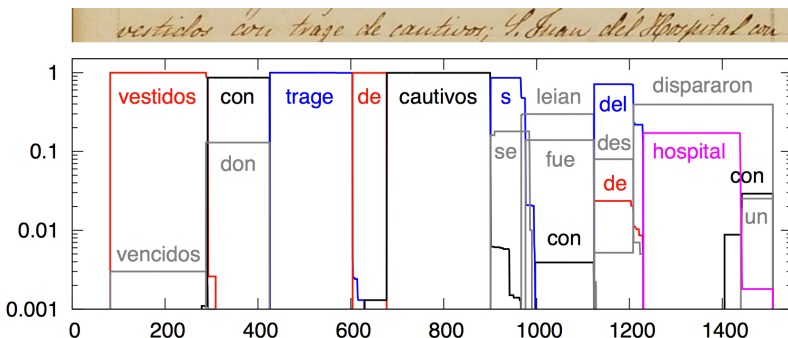
For instance, it can be easily seen in Figure 4.2a that any frame i between t_1 and t_2 will have the same word posterior, i.e. $P(\text{antiguos}|\mathbf{x}, i) = 0.6$ and $P(\text{antiguos}|\mathbf{x}, i) = 0.6$, $t_1 < i < t_2$.

Figure 4.2b shows an example of a posteriorgram, where it is clearly shown that many columns of the histogram have exactly the same distribution, and thus, can be compressed into a single column in order to compute the maximization operation explained before.

Even for dense word graphs, the number of frames required to perform the maximization operation is small: It typically decreases from 1000-2000 frames, which is the typical length of the feature sequences, to 20-40



(a) Example of a word graph. Frames between t_i and t_{i+1} have the same posterior.



(b) Example showing a posteriorgram distribution. Columns of the histogram with the same distribution can be compressed.

Figure 4.2: Examples showing that the compression of posteriorgrams is possible.

frames. This gives not only a huge speed-up in the lookup times, but also in the space requirements of this smoothing methods, since it needs to store not only the scores associated to each indexed keyword and image line, but the whole (compressed) posteriorgram of each line.

Finally, Filler-based scores are the ones that require most time-expensive computations. For one million images, for instance, a single keyword query could require days or weeks of intensive computing. However, as

shown experimentally, using the back-off heuristic with this method, results in the best performance for out of vocabulary queries.

CHAPTER 5

EXPERIMENTS

5.1 Corpora

5.1.1 Cristo Salvador

The Cristo Salvador corpus (CS) is a XIX century Spanish manuscript provided by the the Biblioteca Valenciana Digital (BIVALDI)¹.

The manuscript contains handwritten text of a single writer and was scanned at 300dpi. The corpus suffers the typical degradation problems present in legacy documents [7], such as smear, significant background variations, drastic illumination changes, spots caused by the humidity, ink spots, etc. Moreover, this manuscript includes words with different sizes and style, underlined words, etc. All these degradation problems make the accurate recognition of this document difficult to achieve.

The CS corpus is relatively small collection composed of 50 text page images in color. Figure 5.1 show some examples of the page images. Figure 5.2 contains a detailed view of a page image region.

The page images were preprocessed and lines were automatically segmented as described in [38]. The results of this automatic processing were visually inspected and the segmentation errors were manually corrected, resulting in 1,172 text line images. The transcriptions of the corresponding lines contain 10,860 running words with a vocabulary of 3,287 different words.

¹ <http://bv2.gva.es>



Figure 5.1: Some page examples of the “Cristo-Salvador” corpus.

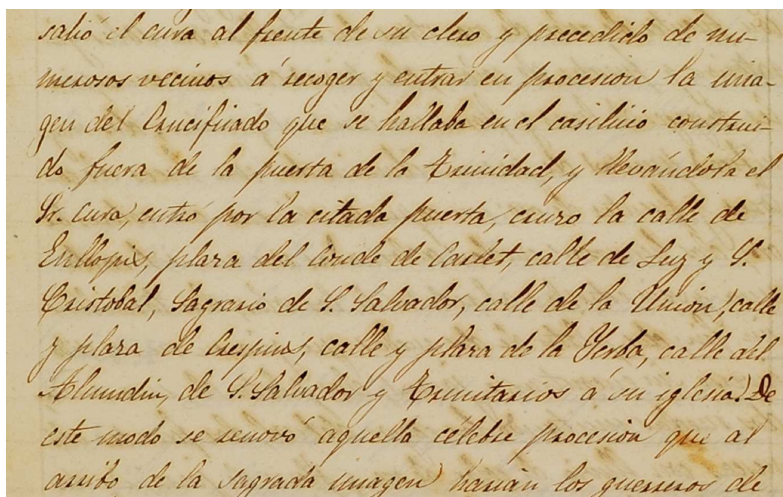


Figure 5.2: Detailed view of a “Cristo-Salvador” page image.

Two different partitions are defined for this dataset. The first one, called the “page” partition, consists of 491 test line images, corresponding to the last ten lines of each document page, and 681 training line images (the rest of lines). The second one, the “book” partition, consists of

497 line images belonging to the last 21 pages of the collection, and the remaining 675 lines are used for training (corresponding to the first 29 pages). In the experiments conducted in this work, we use the “book” partition. Additional details of the training and test sets are depicted in Table 5.1. This table shows the statistics of the data used for the HMM and LM training.

This database does not provide a default validation set and this is needed to tune some of the parameters of the different algorithms needed to perform HTR and KWS. In order to do so, the training set have been divided into 10 sub-partitions to perform 10-fold cross-validation to adjust the required parameters. As usually done in cross-validation, each of these sub-partitions is used for testing while the remaining 9 are used for training. Table 5.2 shows the averaged statistics among the 10 cross-validation partitions.

	Training	Test
Lines	675	491
Running chars	35,176	25,189
Char set size	53	52
Running words	6,227	4,691
Word set size	2,474	1,879

Table 5.1: Basic statistics of the “Cristo-Salvador” used partition.

	CV Train	CV Test
Avg. Lines	607.5	67.5
Avg. Running Characters	31,366.8	3,485.2
Avg. Running Words	5,603.3	621.7
Avg. Character Lexicon	78	78
Avg. Word Lexicon	2,072.6	354.3

Table 5.2: Basic averaged statistics of the “Cristo-Salvadors” cross-validation partitions.

It is important to remark that this corpus has quite a small training

ratio of words (roughly 2.5 training running words per lexicon-entry). Hence, it is expected that the n -gram language models suffer from overfitting, which will increase the errors in the recognition task.

5.1.2 IAM Database

The IAM database (IAMDB) was compiled by the Research Group on Computer Vision and Artificial Intelligence (FKI) at the Institute of Computer Science and Applied Mathematics (IAM) in Bern (Switzerland). The database as of October 2002 is described in [34]. It is publicly accessible and freely available upon request for non-commercial research purposes.

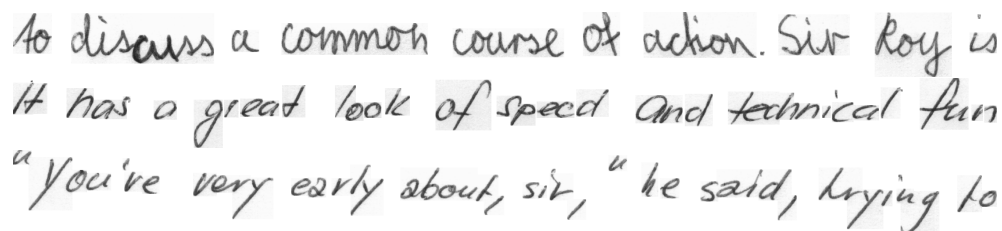
The IAMDB images correspond to handwritten texts copied from the Lancaster- Oslo/Bergen (LOB) corpus [17, 22], which is formed approximately by 500 printed electronic British English texts of about 2,000 words each and about one million total running words. The LOB corpus contains text from diverse categories: editorial, reportage, religion, skill, etc.

In order to create the IAMDB corpus, text from the LOB corpus was split into fragments of 3-6 sentences, with at least 50 words each. Different persons were asked to write several sentences by hand. No restrictions were imposed on the writing style or the type of pen and thus, very different styles and sizes are present. Handwritten texts were scanned at 300dpi, quantized to 256 gray levels and lossless compressed with PNG.

The latest version of IAMDB (version 3.0) is composed of 1,539 scanned text pages, handwritten by 657 different authors. Each author only participates in one of the partition sets: training (283 writers), validation (56 writers) or test (161 writers). We use a line-level partition of the IAMDB (there are sentence, line and isolated words versions of the IAMDB). Some examples of the IAMDB lines are shown in Figure 5.3.

Line detection and extraction, as well as (manually) detecting sentences boundaries, was carried out by the IAM institute [33]. We use a version of this latest IAMDB release frequently used in previous KWS publications [45, 15, 16]. The information summarizing the used version can be found in Table 5.3.

Instead of using only the IAMDB training partition to build the n -



to discuss a common course of action. Sir Roy is
 It has a great look of speed and technical fun
 "You're very early about, sir," he said, trying to

Figure 5.3: Line examples of the IAM database.

	Train	Validation	Test
Running chars	269,270	39,318	39,130
Char set size	72	69	65
Running words	47,615	7,291	7,197
Lexicon size	7,778	2,442	2,488
Lines	6161	920	929

Table 5.3: Basic statistics of the IAMDB corpus.

gram language model, three text corpora have been used: the LOB corpus (except the sentences included in the IAMDB test set), the Brown corpus and the Wellington corpus. The Brown [14] and Wellington [19] are both comparable to the LOB corpus in size and content, but the former is written in American English and the latter in New Zealand English. Table 5.4 shows the basic information of these corpora. In order to reduce the huge vocabulary present in those text corpora, only 20K words were used to build the final vocabulary and train the language models.

	LOB	Brown	Wellington
Lines	52,676	49,362	56,745
Running Characters	5,803,916	5,582,023	6,055,820
Running Words	1,119,904	1,045,213	1,144,401
Lexicon	52,724	53,115	58,919

Table 5.4: Basic statistics of the LOB, Brown and Wellington corpora.

5.1.3 Query keywords selection

In order to carry out the experiments for this thesis we used the validation lexicon when performing (cross-)validation of the models and the test test lexicon for the final evaluation.

Some previous publications used the training lexicon instead [46]. However, by using the training lexicon one is excluding the effect of out of vocabulary keywords.

An argument against using the validation and test lexicon is that all query keywords are *relevant*, since all of them will appear in some of the documents to be spotted, while using the training lexicon provides some *non-relevant* keywords.

Then, one may think that by testing only on *relevant* keywords, one might get good results by automatically assigning high scores to any keyword, while this would not work if both relevant and non-relevant keywords have to be spotted.

However, the previous will only be true if most of keywords are relevant in most of the documents, i.e. we have plenty of *relevant events*. But that is very unlikely in real scenarios (with tens of thousands of documents and just a few thousands of keywords) and is not true either for the databases used in this thesis.

Observe that many keywords are only relevant for a couple of lines, so assigning a high score, no matter which line is considered will hurt the performance of the keyword spotting system.

Table 5.5 summarizes the selected query set used in each corpus. This table clearly shows that out of vocabulary keywords are an important issue for keyword spotting. In the Cristo-Salvador database, more than 30% of the relevant events involve out of vocabulary keywords. Even when a huge external text corpus is used to build the language models, as is the case of IAM database, the out of vocabulary keywords are involved in more than 14% of the relevant events.

It is important to remark that, in order to carry out the KWS experiments, punctuation marks and diacritics were ignored from both databases. Additionally, the LM of the CS database ignores the text capitalization. Table 5.5 takes into account these facts.

	CS		IAMDB	
	Validation	Test	Validation	Test
Lines	67.5	491	920	929
Queries	354.3	1,671	2,134	2,209
OOV Queries	163.6	1,051	435	437
Events	23,915.3	820,461	1,963,280	2,052,161
OOV Events	11,043	516,041	400,200	405,973
Relevant Events	590.8	4,346	3,384	3,446
Relevant OOV Events	167.1	1,341	497	496

Table 5.5: Basic statistics of the selected query keywords for CS and IAMDB. Data from the validation CS set is averaged across the 10 partitions.

5.2 Assessment metrics

In order to assess the performance of the proposed smoothing approaches for keywords spotting, we use the Average Precision (AP) and Mean Average Precision (MAP), based on the concepts of Precision and Recall.

These metrics are widely used in the Keyword Spotting community and others like Information Retrieval, Classification, etc.

5.2.1 Precision and Recall

In the field of Information Retrieval, when a set of documents is retrieved for a query, *precision* measures the fraction of the retrieved documents that were relevant for the given query (that is, the given keyword was actually present in the retrieved documents).

On the other hand, *recall* measures the fraction of relevant documents that were retrieved.

Relevance may be a subjective concept in many fields of Information Retrieval, like search engines. However, in the field of KWS, *relevance* is a well defined concept: a document is relevant for a given query, if the queried keyword is present (written) in the given document.

For a particular KWS system, suppose that \mathcal{D}_v is the complete set of relevant indexed documents, for a query keyword v , and \mathcal{R}_v is the set of

retrieved documents for the query v . The set of retrieved and relevant documents, also referred as “hits”, is denoted by $\mathcal{H}_v = \mathcal{D}_v \cap \mathcal{R}_v$. These sets define the precision π_v and recall ρ_v , for the keyword v , as:

$$\pi_v = \frac{|\mathcal{H}_v|}{|\mathcal{R}_v|} \quad (5.1)$$

$$\rho_v = \frac{|\mathcal{H}_v|}{|\mathcal{D}_v|} \quad (5.2)$$

When a threshold τ is used to filter the results of the KWS system based on a confidence measure, the precision and recall metrics are defined for a particular threshold value. If the set of retrieved documents for a query v and threshold τ is denoted by $\mathcal{R}_v(\tau)$, and the set of “hits” is represented by $\mathcal{H}_v(\tau) = \mathcal{D}_v \cap \mathcal{R}_v(\tau)$ then precision and recall are defined as functions of the threshold τ .

$$\pi_v(\tau) = \frac{|\mathcal{H}_v(\tau)|}{|\mathcal{R}_v(\tau)|} \quad (5.3)$$

$$\rho_v(\tau) = \frac{|\mathcal{H}_v(\tau)|}{|\mathcal{D}_v|} \quad (5.4)$$

Figure 5.4 shows an example of the sets that define the precision and recall metrics. In the figure, D refers to the total collection of indexed documents.

Typically, when a low threshold is used, a high recall and low precision are achieved. In the extreme case, when all documents are retrieved the recall achieves its maximum value $\rho_v = 1$, and the precision its minimum $\pi_v = \frac{|\mathcal{D}_v|}{|D|}$.

On the other hand, when a high threshold is used, a low recall and high precision are achieved. Observe that the extreme case in which no documents are retrieved, the recall is $\rho = 0$ but the precision is not well-defined, since $|\mathcal{R}_v(\tau)|$ is zero.

The previous definitions assumed a single query v . When a set of queries Q is used, precision and recall measures are defined as:

$$\pi(\tau) = \frac{\sum_{v \in Q} |\mathcal{H}_v(\tau)|}{\sum_{v \in Q} |\mathcal{R}_v(\tau)|} \quad (5.5)$$

$$\rho(\tau) = \frac{\sum_{v \in Q} |\mathcal{H}_v(\tau)|}{\sum_{v \in Q} |\mathcal{D}_v(\tau)|} \quad (5.6)$$

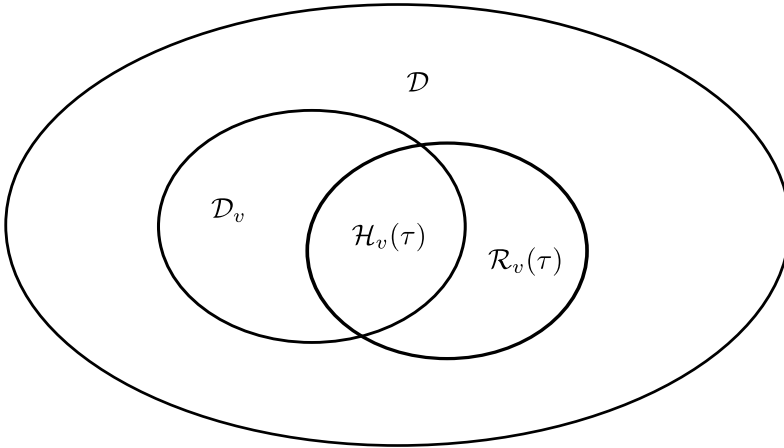


Figure 5.4: Venn diagram showing the sets that define Precision and Recall.

5.2.2 Average Precision

By changing the threshold value τ one gets different pairs of recall and precision values, which allow to plot the *recall-precision* curve. In this curve, the recall is plotted on the x -axis and the precision in the y -axis.

The area under the recall-precision curve is known as the Average Precision (AP), and is a widely used assessment metric that combines both Precision and Recall. It is commonly accepted that the larger the AP, the better the KWS system performs.

As mentioned before, precision is ill-defined in some extreme cases, the recall-precision curve does not always present a monotonically decreasing curve for increasing recall values [9]. To surpass these problems, the *interpolated precision* is usually used in the literature, instead.

The interpolated precision π' at a certain recall level ρ , denoted as $\pi'(\rho)$, is defined as the highest precision found for any recall value greater or equal to ρ :

$$\pi'(\rho) = \max_{\rho' \geq \rho} \pi(\rho') \quad (5.7)$$

Figure 5.5 shows an example of a recall-precision curve (dashed line) and the recall-precision curve using interpolated precision (solid line). From now on, we will assume that the interpolated precision is used,

when referring to Average Precision.

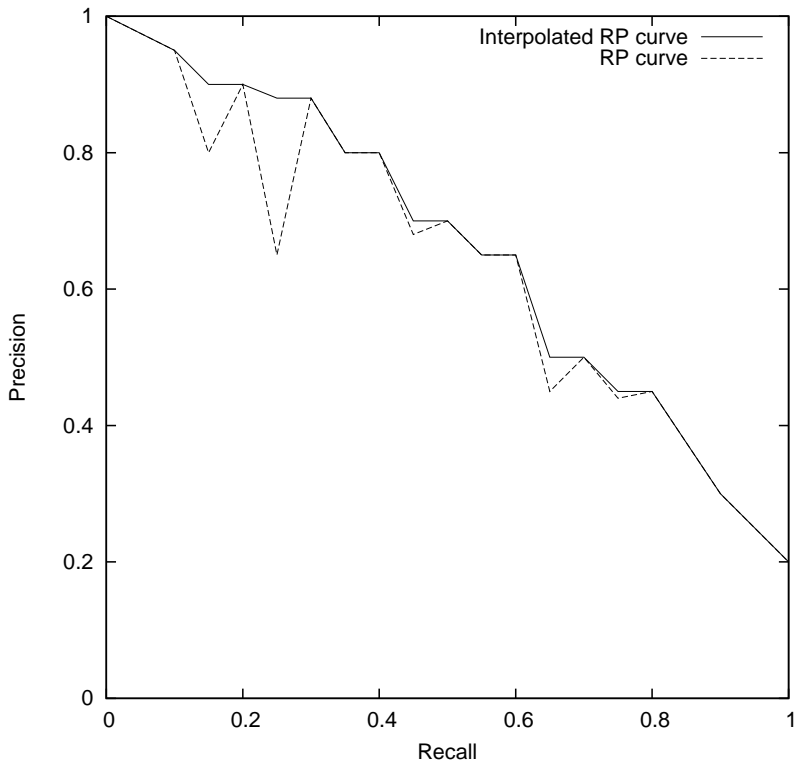


Figure 5.5: An example of a Recall-Precision curve (dashed) and the RP curve with interpolated precision (solid).

5.2.3 Mean Average Precision

The recall-precision curve and the Average Precision are usually computed for a set of queries, as explained before. However, in the literature there is also an alternative metric that consists in computing the area under the recall-precision curves for each keyword separately and then averaging all of them. This metric is known as the Mean Average Precision (MAP) [32].

For both AP and MAP the value of the score which is assigned to each event is not important, the only important factor that determines

the area under the PR curve is the relative ordering between relevant and non-relevant results. As long as most of the relevant results have higher scores than the non-relevant results, both metrics will be similar.

However, there is an important difference between MAP and AP. The latter computes the area under the recall-precision curve when all queries are considered simultaneously, while the former only considers queries one at a time. Then, when the score assigned to an event highly depends on the keyword, very different values for the AP and MAP may be obtained.

Table 5.6 shows an example of this phenomenon. Observe that the scores of the keyword “K1” are systematically higher than the scores of the keyword “K2”. If the MAP metric is considered, which computes the area under the PR curve for each keyword separately, the performance is perfect since the relevant events have greater scores for both keywords, thus a MAP of 1.0 is achieved.

On the other hand, if all events are considered together, as the AP metric does, the performance is considered worse, since the relevant event of keyword “K2” has a lower score than the non-relevant events of keyword “K1”. Then, the achieved AP has a value of 0.75.

Document	Keyword	Relevant	Score
D1	K1	1	1
D2	K1	0	0.1
D3	K1	0	0.1
D1	K2	1	0.05
D2	K2	0	0
D3	K2	0	0
AP = 0.75		MAP = 1.0	

Table 5.6: Example showing the difference between AP and MAP.

There is some controversy in the Information Retrieval community regarding this phenomenon. Some authors defend that the AP metric should be used, since it demands a consistent and robust threshold which assigns high scores for relevant events and low scores for non-relevant ones, no matter which keyword or document is considered. Additionally,

the MAP metric is not well-defined when the considered queries include non-relevant keywords (those which are not present in any document).

On the other hand, other authors defend that the MAP metric should be used since it is more similar to the user point-of-view, because users do not search for all keywords at the same time, but they perform individual queries. Moreover, very few search engines let the user to adjust the threshold, but provide a sorted list of retrieved documents, where the events with the highest score are situated in the top positions of the results list.

During the development of this thesis, we decided to measure both AP and MAP in order to have a broader view of the performance of the proposed algorithms.

5.3 Description of the underlying HTR system

As mentioned in Chapter 3, KWS systems are built on top of a HTR engine. In this section we discuss the underlying HTR systems used for keyword spotting. We use the standard HMM-based approach for HTR.

Once the text document images have been segmented into lines, several preprocessing steps are carried out to normalize the image (slope, slant and size normalization) and a sequence of feature vectors for each line image. In the case of the CS database, the images are processed using a sliding window approach where each original image line is segmented into $r = 16$ rows and a number of columns, c , is chosen so that the number of rows $\frac{r}{c}$ is three times the original line image aspect ratio. For each cell, the average gray-level and the horizontal and vertical gradients of the gray-level are used as features, thus each line is represented by a sequence of c 48-dimensional vectors (frames). More details of this feature extraction process can be found in [44, 38].

On the other hand, for the IAM database, the carried image normalization preprocessing and the used features are presented in [33]. Same as before, a sliding window scheme is used here, but nine features are extracted from each frame: the number of black pixels in the window, the center of gravity, the second order moment of the window, the position of the upper and lower contour, the gradient of this contour, the number

of black-white transitions in vertical direction and the number of black pixels contained between the upper and lower contour.

Once the the image features have been extracted, 78 character-HMMs were trained for the CS database. The number of states for each character-HMM was determined experimentally using force-alignment, as described in [43]. The average number of states per character was 14, with 16 Gaussian components for each state. These parameters were tuned for the cross-validation sets.

A bi-gram language model was trained from the transcriptions and smoothed using the standard Kneser-Ney back-off technique [26]. The LM was estimated only using the transcriptions of the training set.

Finally, the word insertion penalty and the grammar scale factor parameters were set to 80 and -160, respectively. They were also tuned using the cross- validation sets.

Regarding the IAM database, 72 character-HMMs were trained. The number of states was also variable and was adjusted using the validation set. The number of states per character-HMM varies from 2 to 28 states, with 12 Gaussian components per state.

The bi-gram language model was trained on the LOB-Brown-Wellington corpora described in Section 5.1.2. The resulting language model was smoothed using also the standard Kneser-Ney back-off.

In this case, the word insertion penalty and the grammar scale factor parameters were set to 28 and -17, respectively. Both adjusted on the validation set of the IAMDB.

The software used to train the underlying handwriting text recognition system is the widely used Hidden Markov Model Toolkit (HTK)² [53].

5.4 Baseline results

In this section we describe the baseline results achieved by the Filler-based and the WG-based KWS systems described in Section 3.2 and Section 3.3, respectively.

The baseline results for the CS database are shown in Table 5.7. Table 5.8 shows the baseline results for IAMDB. For the word graph-based

² <http://htk.eng.cam.ac.uk/>

KWS approach, different WG input degrees were tried. The results are given for each tried density, d . In the case of the CS database, the results on the Validation columns are the average of the AP and MAP achieved by the 10 partitions defined in Section 5.1.1. The confidence interval is computed at 95% of significance.

	AP		MAP	
	Validation	Test	Validation	Test
Filler	0.784 ± 0.015	0.642	0.859 ± 0.016	0.739
WG, $d = 1$	0.495 ± 0.016	0.389	0.389 ± 0.007	0.198
WG, $d = 3$	0.617 ± 0.010	0.536	0.458 ± 0.004	0.275
WG, $d = 5$	0.630 ± 0.010	0.550	0.464 ± 0.004	0.281
WG, $d = 10$	0.632 ± 0.010	0.554	0.473 ± 0.005	0.287
WG, $d = 20$	0.634 ± 0.010	0.556	0.477 ± 0.005	0.290
WG, $d = 40$	0.634 ± 0.010	0.556	0.480 ± 0.006	0.290

Table 5.7: Baseline AP and MAP for CS.

	AP		MAP	
	Validation	Test	Validation	Test
Filler	0.548	0.467	0.737	0.665
WG, $d = 1$	0.555	0.510	0.567	0.524
WG, $d = 3$	0.705	0.659	0.677	0.639
WG, $d = 5$	0.717	0.674	0.687	0.661
WG, $d = 10$	0.728	0.683	0.700	0.675
WG, $d = 20$	0.731	0.689	0.707	0.684
WG, $d = 40$	0.733	0.691	0.712	0.688

Table 5.8: Baseline AP and MAP for IAMDB.

The previous tables show the role of the WG input degree in the performance of the WG-based method. While the AP and MAP increase with the input degree of the WG, the differences are not significant for $d = 10, 20, 40$, as the confidence intervals of the validation AP and MAP show for the CS database. For the IAM database, no confidence inter-

vals could be computed, but it is also seen that the improvement in the assessment metrics saturates for high-dense word graphs.

Also, the importance of the out of vocabulary keywords and the language model can be observed. In the CS database, the Filler model outperforms the WG-based approach with a high margin, specially in the MAP metric. On the other hand, the WG-based model is much better than the Filler model in the IAM database.

This is due to the fact that the CS language model was trained using very few data (only the transcriptions from the training set), whereas the IAMDB language model was trained using a huge external corpus. Also, many of the queries in the CS database were out of vocabulary (Table 5.5), while in the IAMDB scenario this percentage was not so elevated (see Figure 5.6, which shows the distribution of the AP among the test keywords in each database). This explains why AP and MAP are much better in the IAMDB for the WG, in comparison to the CS database.

On the other hand, the reason that explains why the Filler behaves poorly in the IAMDB, in comparison to the CS, is that the latter case was written only by a single writer, and thus the variability that has to be modeled by the GMMs and HMMs is much lower in the second case, in comparison to the first. The WG-based approach is capable of handle this variability by using the information provided by the well-estimated LM. Moreover, the Filler approach suffers from a problem similar to the one present in the introduced smoothing techniques that used the stochastic error correcting approach: the score assigned to an event highly depends on the length of the keyword. These two effects explain why the AP of WG-based model outperforms the Filler approach, while the MAP is similar. Figure 5.7, which shows the distribution of the scores for relevant events, can help to understand these issues.

5.5 Smoothing methods

In this section we explain the experiments performed to assess the improvements achieved by using the different smoothing methods presented in Chapter 4. The section is divided in four subsections: the first one describes the results of the line-level smoothing methods, the second one comprises the results of the frame-level methods, the third one explains

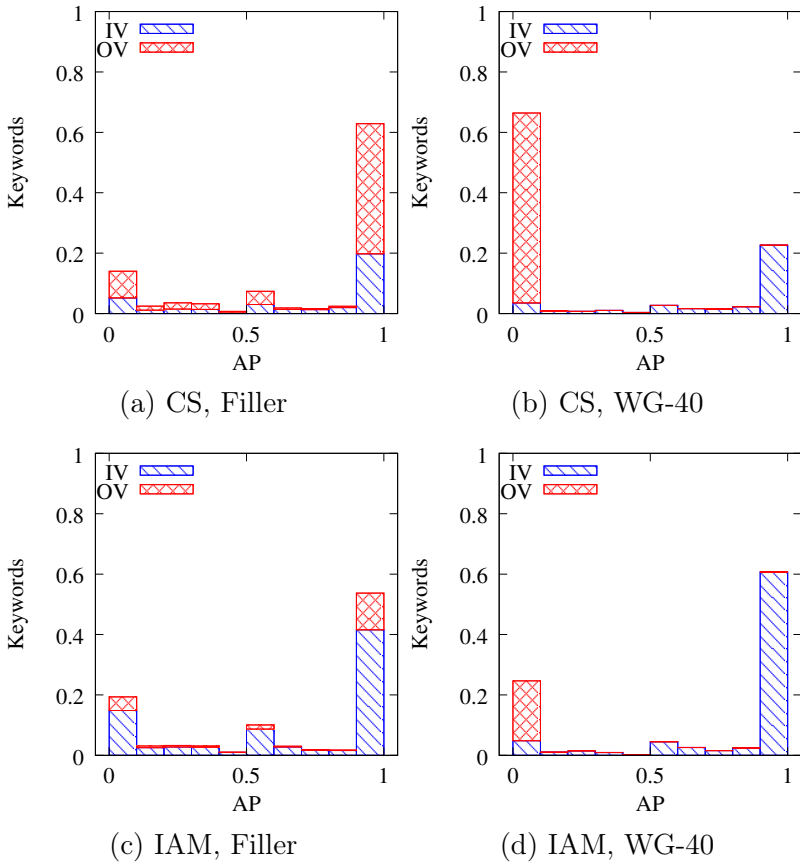


Figure 5.6: Histogram of the AP of test keywords for CS and IAMDB using the baseline Filler and WG models.

the experiments and performance of the previous methods combined with the baseline WG scores, using the Back-off approach described in Section 4.5, including the results of the Back-off method based on the Filler scores. Lastly, all results are summarized in the final subsection.

Since most of the proposed methods involve different hyper-parameters that need to be tuned in order to achieve a proper performance, we restrict the further experiments to word graphs with a maximum input degree of 40. Thus, the following results must be compared to the last rows of Table 5.7 and Table 5.8, which show the results of the baseline

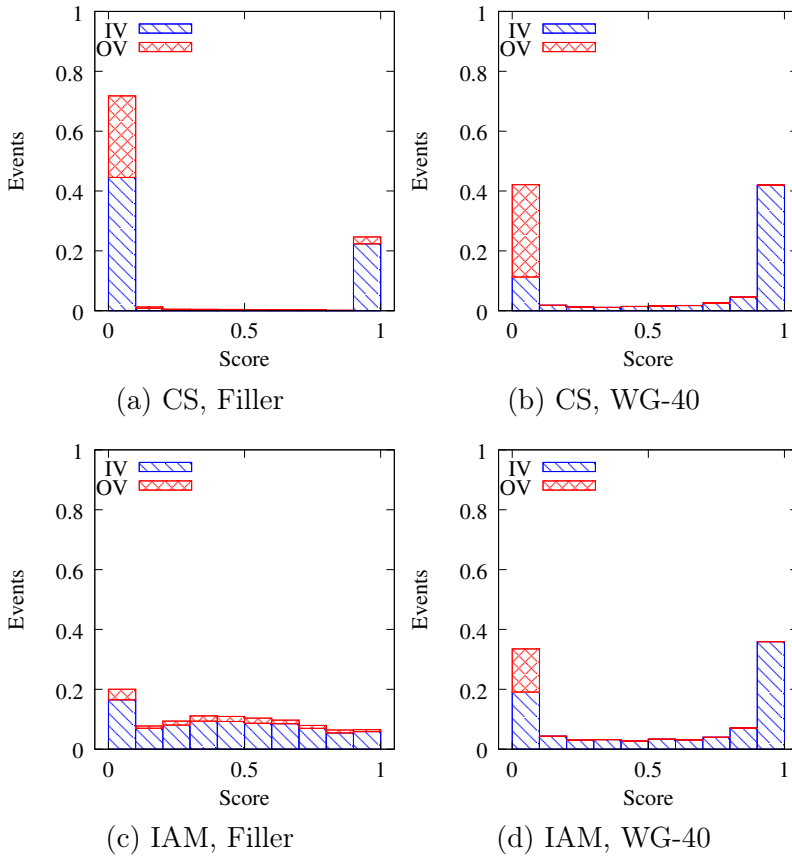


Figure 5.7: Histogram of the score of test relevant events for CS and IAMDB using the baseline Filler and WG models.

WG method with a density of 40.

5.5.1 Line-level smoothing

In this subsection we report the results achieved by all smoothing methods described in Section 4.2. These results are shown in Table 5.9 and Table 5.10, which correspond to the results on the CS and IAM databases, respectively.

In order to adjust the hyper-parameters in each of the smoothing equations, the validation partitions of each database were used and a

grid search was performed. The hyper-parameters were adjusted for each method and database independently.

For the Levenshtein Distance-based smoothing method (see Section 4.2.1), the optimal value of α was found exploring values in the range $[0, 1]$ with 0.1 steps. The results of this method are tagged with the keyword “Levenshtein” in the tables.

For the Stochastic Error Correcting-based (SEC) smoothing technique (see Section 4.2.2), which involves two parameters, α and β , a similar grid search was performed. The value of α was optimized for values in the range $[2^{-3}, 2^4]$ with exponential increments $(2^{-3}, 2^{-2}, \dots)$. On the other hand, the value of β was determined in the range $[0, 2.5]$ with 0.1 increments. Observe that if β is zero, the SEC method with length correction (SEC-LC) is equivalent to the basic SEC method, with no length correction.

	AP		MAP	
	Validation	Test	Validation	Test
Levenshtein ($\alpha = 0.8$)	0.659 ± 0.011	0.567	0.700 ± 0.006	0.443
SEC ($\alpha = 0.5$)	0.617 ± 0.014	0.544	0.684 ± 0.008	0.450
SEC-LC ($\alpha = 1, \beta = 1$)	0.660 ± 0.012	0.573	0.711 ± 0.007	0.462

Table 5.9: Best AP and MAP achieved using the three proposed line-level smoothing methods on the CS database.

	AP		MAP	
	Validation	Test	Validation	Test
Levenshtein ($\alpha = 0.9$)	0.735	0.694	0.798	0.761
SEC ($\alpha = 1$)	0.710	0.665	0.795	0.760
SEC-LC ($\alpha = 2, \beta = 1.2$)	0.735	0.691	0.794	0.761

Table 5.10: Best AP and MAP achieved using the three proposed line-level smoothing methods on the IAM database.

Table 5.9 and Table 5.10 show that the method that usually outperforms the others across all configurations is the Stochastic Error Correcting-based method, with length correction. However, the differences between

this method and the Levenshtein Distance-based smoothing are not significant in the CS validation sets (with 95% confidence) and they are probably not significant either for the other scenarios (CS test set, and IAM validation and test sets).

5.5.2 Frame-level smoothing

In this section we present the performance achieved by the frame-level smoothing methods described in Section 4.3. Validation and test results for each database are shown in Table 5.11 and Table 5.12, for CS and IAMDB respectively.

In order to adjust the hyper-parameter β in the frame-level smoothing with length correction, the validation partitions of each dataset were used and a grid search was performed. The value of β was adjusted for each dataset independently. The range of explored values for β was $[0, 2.5]$ with 0.1 increments. Observe that if β is zero, the length correction factor is ignored, and then Eq. 4.23 is the same as Eq. 4.13. This means that the frame-level smoothing with length correction is a more general smoothing than the version without length correction.

	AP		MAP	
	Validation	Test	Validation	Test
Pstgram	0.605 ± 0.015	0.532	0.718 ± 0.008	0.471
Pstgram-LC ($\beta = 1.5$)	0.612 ± 0.016	0.532	0.713 ± 0.008	0.463

Table 5.11: Best AP and MAP achieved using the two proposed frame-level smoothing methods on the CS database.

	AP		MAP	
	Validation	Test	Validation	Test
Pstgram	0.709	0.663	0.795	0.761
Pstgram-LC ($\beta = 2.1$)	0.719	0.672	0.794	0.754

Table 5.12: Best AP and MAP achieved using the two proposed frame-level smoothing methods on the IAM database.

Here, the differences in AP between the version with length correction and the version without length correction are subtle. Moreover, they are slightly worse than the line-level smoothing approaches presented before. However, the MAP of the frame-level methods is generally better than the MAP of the line-level approaches.

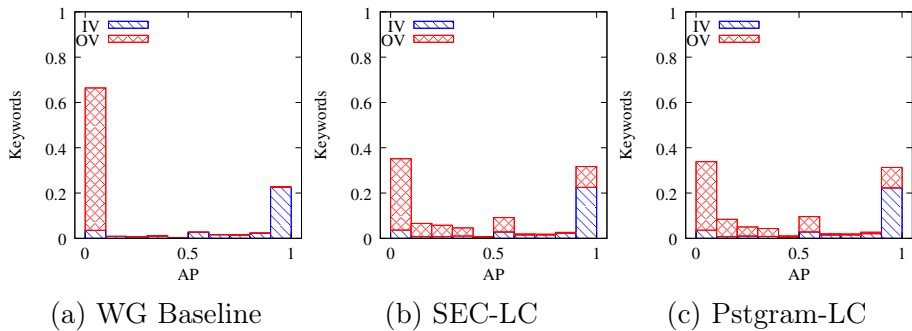


Figure 5.8: Histogram of the distribution of the AP among the CS test keywords achieved using the baseline scores and the line-level and frame-level smoothed scores.

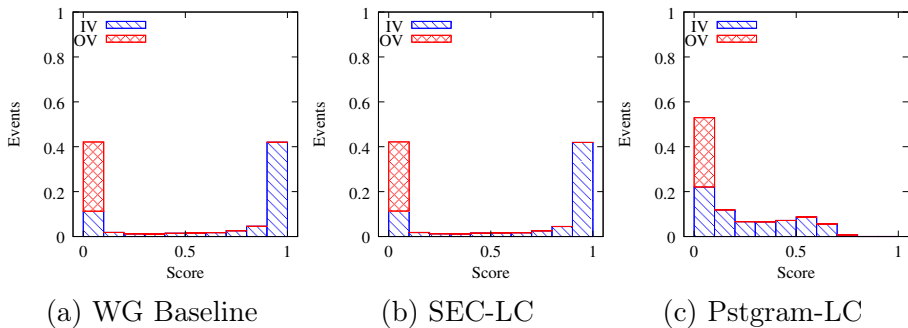


Figure 5.9: Histogram of the distribution of the scores among the CS test events given by the baseline WG, the line-level and frame-level smoothing methods.

Figure 5.8 shows the distribution of the AP among the CS test keywords for the baseline WG method, the “SEC-LC” line-level method (see

Table 5.9) and the “Pstgram-LC” frame-level method (see Table 5.11). As Fig. 5.8 shows, the smoothing methods assign higher scores for relevant events than non-relevant events, and that is why the MAP increases significantly for both smoothing methods, respect the baseline.

On the other hand, Figure 5.9 explains the results on the AP. For the line-level smoothing with length correction, the score assigned to an in-vocabulary keyword will not usually change. Observe that, if the confusion probability is well modelled, the value $\max_{u' \in \Sigma} P(u'|u)$ will typically correspond to $P(u|u)$. Thus, the score given by the word graph of a in-vocabulary keyword will be multiplied by 1, in Eq. 4.11, and, thus, it typically won't change. For the out of vocabulary methods, the scores will be typically very low, since the confusion probability will have a small value, give any in- vocabulary keyword. This explains why the distributions of the scores of the baseline system and the smoothed scores with Eq. 4.11 are so similar.

However, for the frame-level smoothing, the frame-level word posterior of a in-vocabulary keyword is also interpolated across the frame-level word posterior and confusion probabilities of *all* keywords in the word graph, and thus, even for in-vocabulary keywords, the smoothed frame-level word posterior probabilities will be smaller (since they have to sum up up to 1 for any string in Σ^*), and this translates to a smaller score, since the score is just the maximum of the frame-level word posterior of a given keyword, across all frames. This effect can be seen in Figure 5.9, where the “Pstgram-LC” figure shows that no event has a score higher than 0.8. This ultimately can cause a drop in the AP, when the scores of non-relative events have a higher score than other relative events, even if the events are due to different keywords.

The previous fact suggests that it is probably better to smooth only the scores of out of vocabulary keywords, and use the scores for in-vocabulary keywords as provided by the word graph. This combination scheme is what we refer as the *Back-off heuristic*, which is experimented in the following section.

5.5.3 Back-off combination

In this section we present the results of using the Back-off combination heuristic, introduced in Section 4.5, with all the previous smoothing

methods and the Filler score, provided by the baseline Filler model.

The value of the *score scaling factor* η , for the out of vocabulary scores, is determined using the validation partition of each corpus. All values in the range $[0, 3]$ with increments of 0.1 points were explored for each smoothing method, database and partition independently. It was assumed that the optimal value of this hyper-parameter is independent of the values of the parameters intrinsic in each smoothing method. This may not be true, but the number of combinations of parameters that would be required to explore, if this independence assumption is not taken, would be too high in some cases (observe that the number of experiments that have to be done with grid search grows exponentially with the number of hyper-parameters to tune). Thus, the best values of the intrinsic parameters of each method were determined by the results of the previous section, and then the previous exploration was performed to tune η .

Table 5.13 and Table 5.14 show the results of the Back-off heuristic on CS and IAM, respectively, using the different techniques of obtaining scores for out of vocabulary queries, presented in this thesis.

	AP		MAP	
	Validation	Test	Validation	Test
Levenshtein ($\eta = 2.6$)	0.689 ± 0.012	0.574	0.707 ± 0.006	0.450
SEC ($\eta = 0.4$)	0.688 ± 0.011	0.568	0.695 ± 0.008	0.455
SEC-LC ($\eta = 0.4$)	0.696 ± 0.012	0.585	0.713 ± 0.008	0.464
Pstgram ($\eta = 0.4$)	0.689 ± 0.012	0.581	0.719 ± 0.008	0.472
Pstgram-LC ($\eta = 0.4$)	0.690 ± 0.012	0.581	0.718 ± 0.008	0.466
Filler ($\eta = 0.05$)	0.838 ± 0.009	0.725	0.872 ± 0.008	0.766

Table 5.13: Best AP and MAP achieved with the Back-off combination heuristic for different smoothing methods, CS database.

If Table 5.13 and Table 5.14 are compared with the results from the previous section, it is clearly shown that smoothing the scores of all events, including in-vocabulary, which are already well-modelled by the WG-approach, is not the best thing to do. Those smoothing methods should be used only for out of vocabulary keywords, instead, and then

	AP		MAP	
	Validation	Test	Validation	Test
Levenshtein ($\eta = 4$)	0.743	0.698	0.797	0.760
SEC ($\eta = 0.4$)	0.745	0.702	0.794	0.760
SEC-LC ($\eta = 0.3$)	0.745	0.700	0.793	0.760
Pstgram ($\eta = 0.4$)	0.745	0.702	0.795	0.762
Posteriorgram-LC ($\eta = 0.4$)	0.746	0.702	0.795	0.761
Filler ($\eta = 0.5$)	0.820	0.769	0.863	0.822

Table 5.14: Best AP and MAP achieved with the Back-off combination heuristic for different smoothing methods, IAM database.

combined with the in-vocabulary scores, provided by the WG baseline with the Back-off heuristic. As we commented in Section 5.5.2, this is due to the fact that the proposed smoothing methods tend to produce scores for relevant and non-relevant events closer to zero, which can hurt the AP. Figure 5.10 shows that the Back-off heuristic is not affected by this issue.

Observe that the distribution of the scores of in-vocabulary events is the same as the baseline method, which is well distributed close to the *ideal* distribution where most of the relevant events have a score near to zero. On the other hand, the score of the out of vocabulary events is distributed in a different way depending on the smoothing method, being the “Filler Back-off” combination, the one that provides a closer distribution to the *ideal* one.

Figure 5.11 shows the importance of *scaling* the scores in the Back-off heuristic to improve the AP. It shows the distribution of the relevant scores for the baseline methods, the scaled Filler scores and the scores provided by the Back-off heuristic, using the scaled Filler scores. Observe that the Back-off heuristic distributes the in- vocabulary scores as the “WG Baseline” and the OOV scores as the “Filler Scaled”. In terms of MAP, the scaling has obviously no effect since the relative ordering of relevant and non-relevant events due to the same keyword is not altered.

The previous tables and figures show that best thing to do, in order to achieve the highest AP and MAP scores, is to use the Filler scores for

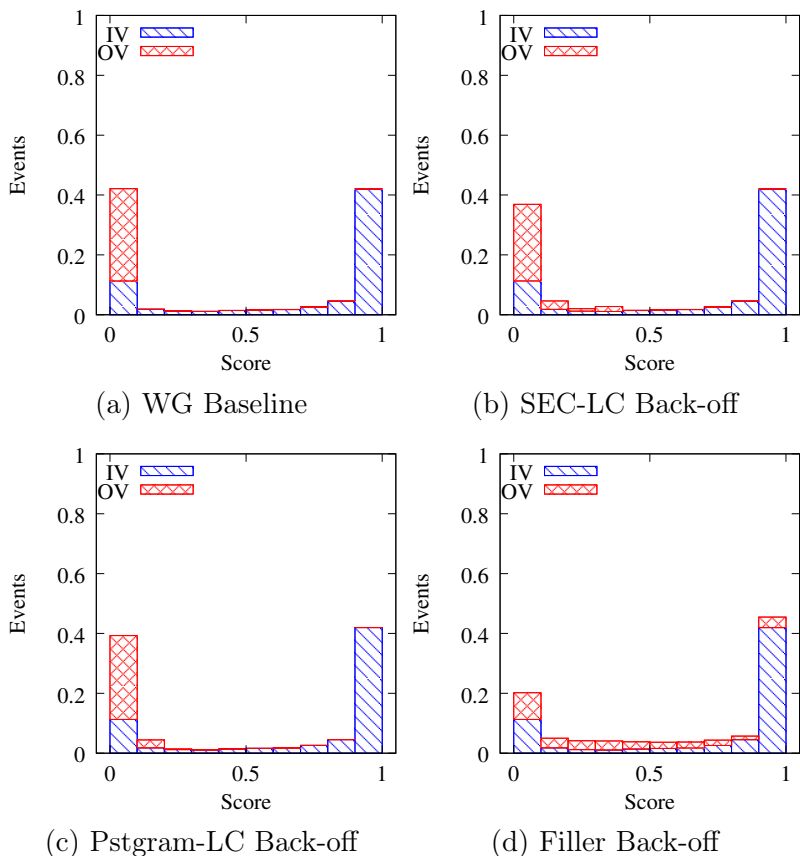


Figure 5.10: Histogram of the scores distribution for relevant events in the CS test partition, using different Back-off heuristics.

out of vocabulary keywords combined with the pure WG-based scores for in-vocabulary keywords. These excellent results prove that the proposed Back-off heuristic successfully combines the benefits of the WG-based and Filler-based KWS systems.

5.5.4 Summary results

Here we summarize the AP and MAP in the test set of each database for all the previous smoothing methods and the baseline results for the

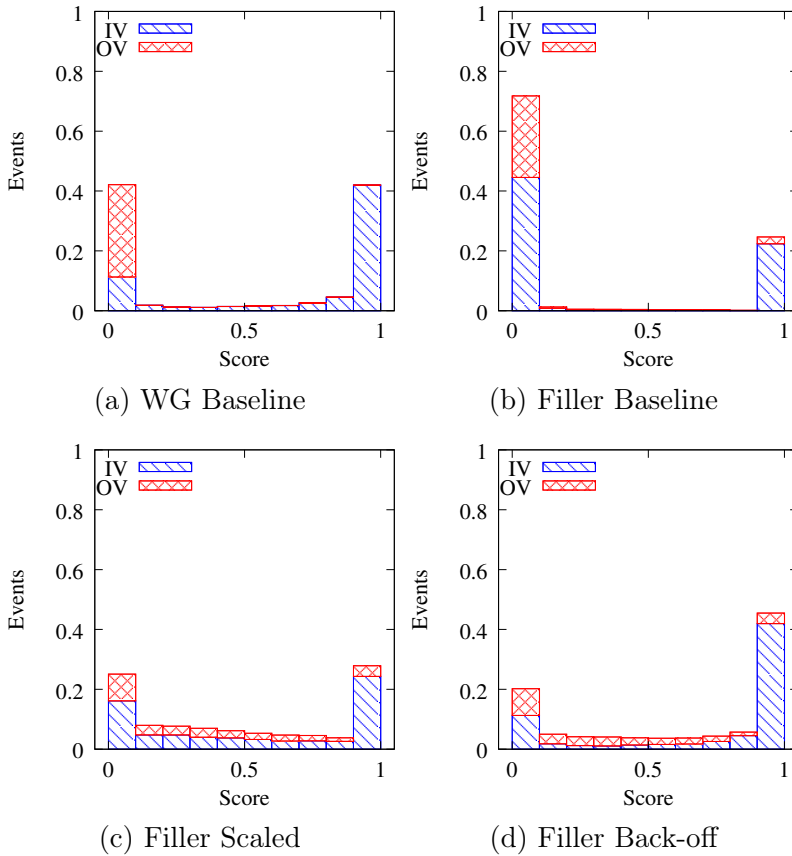


Figure 5.11: Histogram of the scores distribution for relevant events in the CS test partition, using different KWS approaches. Observe that the Back-off heuristic distributes the in-vocabulary scores as the “WG Baseline” and the OOV scores as the “Filler Scaled”.

WG and the Filler models. The rows in Table 5.15 are divided in four blocks. The first block corresponds to the baseline methods, the second block refers to the line-level methods (including their back-off versions), the frame-level smoothing methods are situated in the fourth block and, finally, the filler back-off model is situated on the final group.

For each group and each column (metric and database), the winning

method is emphasized. In the cases where two methods achieved the same performance for a particular dataset-metric pair, both methods are emphasized. Finally, the global best method for each dataset-metric pair is marked in bold (the “best” for each group have been determined using the validation partition).

	CS		IAMDB	
	AP	MAP	AP	MAP
Baseline Filler	<i>0.642</i>	<i>0.739</i>	0.467	0.665
Baseline WG	0.556	0.290	<i>0.691</i>	<i>0.688</i>
Levenshtein	0.567	0.443	0.694	<i>0.761</i>
SEC	0.544	0.450	0.665	0.760
SEC-LC	0.573	0.462	0.691	<i>0.761</i>
Levenshtein Back-off	0.574	0.450	0.698	0.760
SEC Back-off	0.568	0.455	<i>0.702</i>	0.760
SEC-LC Back-off	<i>0.585</i>	<i>0.464</i>	0.700	0.760
Pstgram	0.532	0.471	0.663	0.761
Pstgram-LC	0.532	0.463	0.672	0.754
Pstgram Back-off	<i>0.581</i>	<i>0.472</i>	<i>0.702</i>	<i>0.762</i>
Pstgram-LC Back-off	<i>0.581</i>	0.466	<i>0.702</i>	0.761
Filler Back-off	0.725	0.766	0.769	0.822

Table 5.15: Summary table showing the test AP and MAP on the CS and IAM databases for all the presented methods and the original baselines.

If one takes into consideration that, most of the line-level and frame-level methods had a confidence interval around 0.015 (95% confidence) in the validation set of the CS database, one realizes that the differences between these two approaches are subtle. Anyhow, the absolute winner above any method is the Filler back-off combination (where the WG scores are used for in-vocabulary keywords and a scaled version of the Filler scores are used for out of vocabulary keywords). However, one must realize that this comes with the price of running the expensive computation needed by the Filler model.

Finally, comparing the best smoothing model in each group and dataset-metric pair with the baseline WG model, which is the one sensible

	CS		IAMDB	
	AP	MAP	AP	MAP
Baseline WG	0.556	0.290	0.691	0.688
Line-Level	0.585 (5.2%)	0.464 (60.0%)	0.702 (1.6%)	0.761 (10.6%)
Frame-Level	0.581 (4.5%)	0.472 (62.8%)	0.702 (1.6%)	0.762 (10.7%)
Filler Back-off	0.725 (30.3%)	0.766 (164.1%)	0.769 (11.3%)	0.822 (19.5%)

Table 5.16: Summary table showing the best AP and MAP achieved for each smoothing level and the relative improvement respect the baseline WG method.

to out of vocabulary words, results in Table 5.16. The relative increase of each metric is shown in parentheses. This table is useful to show how much is expected the system to improve using the best method of each smoothing level.

Table 5.16 shows that the smoothing methods are specially useful when the number of expected out of vocabulary events is high (as is the case of CS). But even when the numbers of out of vocabulary events is not so high (IAMDB), the proposed line-level and frame-level smoothing methods provide moderate improvements on AP, and very good improvements on MAP. And the Filler Back-off combination provides even more excellent results for MAP and significant improvements in AP.

The recall-precision curves of the best method for each level on each dataset, summarized in Table 5.16 are shown in Figure 5.12. This figure summarizes all the previous discussion and clearly shows that any type of smoothing helps the WG-based KWS to improve. The best results are achieved in both datasets with the Filler Back-off combination, which clearly outperforms any of the other methods.

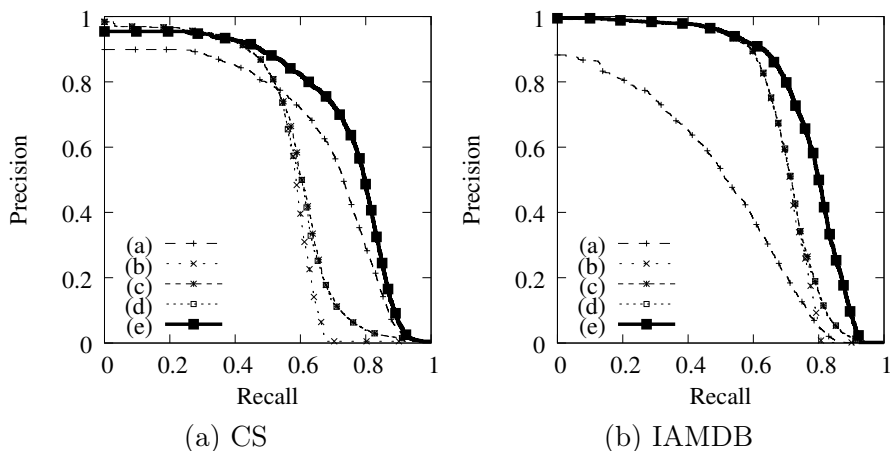


Figure 5.12: Recall-Precision curves of the best methods on the CS and IAMDB test partitions. Curves: (a) Baseline Filler (b) Baseline WG (c) SEC-LC Back-off (d) Pstgram-LC Back-off (e) Filler Back-off.

5.6 Posterigram compression

Finally, we measured how much the posterigrams are compressed by taking into account the fact that many frames will have exactly the same word-posterior distribution, and thus, not all of them are required in order to perform the frame-level smoothing (recall Section 4.6). Different word graph input degrees were considered. The results for the test partitions of CS and IAMDB are shown in Table 5.17.

The previous table shows that the compression achieved depends on the word graph density, as explained in Section 4.6. For the smallest word graphs, posterigrams are reduced to 1.5% (CS) and 0.8% (IAMDB) of their original size. For the largest explored input degree, they are reduced to 20.6% (CS) and 2.4% (IAMDB), respect the uncompressed posterigrams.

We already pointed in Section 5.3 that the language models of the CS dataset were underestimated, due to the lack of training data. This directly affects the weights of the arcs in the word graph. Having a lower quality system implies that the system is less confident about the

	CS	IAMDB
# Frames	1659.0	1439.2
$d = 3$	24.2	11.0
$d = 5$	43.8	13.6
$d = 10$	97.2	18.9
$d = 20$	196.7	26.0
$d = 40$	342.3	34.1

Table 5.17: Average number of frames and resulting number of frames after compression, for the test partition of the CS and IAM databases.

decoding, resulting, at the end, in higher perplexities at frame level, which prevents from a compression as effective as the one achieved in the IAM database.

CHAPTER 6

CONCLUSIONS

In this work different methods are studied to handle out of vocabulary queries when a word graph-based keyword spotting system is used. WG-based KWS systems (presented in Section 3.3) have the shortcoming that assign a null score to any keyword that was not part of the data from which the language model was trained. The amount of out of vocabulary words depends on the particular task, but for all the considered datasets this was not negligible.

Other models based on the modeling of the scores at a character-level have much better performance for the OOV keywords, as the Filler model presented in Section 3.2. However, character-level models are usually much more expensive in terms of computation time required to perform a search.

Thus, we aimed to provide the WG-based approach with the versatility of the character-level models but providing also reasonable lookup times to the user.

In Chapter 4 we presented several smoothing techniques that estimate the score of an out of vocabulary keyword based on its similarity among the keywords present in a word graph, and the score assigned to each of these present keywords. The proposed smoothing techniques are applied at two levels, one is the *line-level*, where the scores of the in-vocabulary keywords are directly used. The other proposed level, the *frame-level*, goes one step below and tries to smooth the *frame-level word posterior probability* at each frame, since this is directly used to compute the final score of a keyword for a line image.

We have seen during experimentation that there is not a significant difference, in terms of MAP, between the line-level and the frame-level smoothing methods. However, a significant difference was observed for AP (see Section 5.5.1 and Section 5.5.2). As we commented, this is due to the fact that the proposed frame-level tends to decrease excessively the score of in-vocabulary keywords.

We decided then to use an heuristic approach where the smoothing is only applied to out of vocabulary keywords, while in-vocabulary keywords use the score directly given by the baseline WG-based system. We called this method “Back-off heuristic”, since it reminds to the Back-off smoothing used for Language Modeling (see Section 4.5).

The Back-off method, once a certain parameter has been tuned, showed to be the key to significant improvements over the baseline AP and MAP (Section 5.5.3). Using this method, in combination to the smoothing techniques presented before, we observed a relative improvement of about 4.8% in the AP of the baseline WG-based system, for a dataset where the number of out of vocabulary events represents a big proportion of the queries. It also improved the baseline results on 1.6% relative points for a dataset where the number of out of vocabulary events is smaller. On the other hand, the MAP increases 60.4% relative points in the first task, and 10.6% in the second one (see Section 5.5.4).

Taking advantage of the ability of the Filler to give reasonable scores for out of vocabulary queries, and the ability of the word graphs to provide very good scores for in-vocabulary scores, we applied the same Back-off heuristic using the Filler scores for OOV keywords. This combination scheme clearly outperformed all the previous proposed smoothing techniques. The AP relatively improved 30.3% points for the first dataset, and 11.3% for the second one, with less out of vocabulary events. Moreover, the MAP improved 164.1% relative points in the first task and 19.5% in the second one, respect the original WG-based baseline (see Section 5.5.4).

Analyses of the computational cost of each proposed solution have been proposed in Section 4.6, where it is shown that the deeper level of smoothing we use, the more expensive results the lookup time (being the line-level the shallowest, and the Filler the deepest one).

This suggests that a hierarchy of solutions for out of vocabulary queries could be provided to the user, so he or she can decide if they

need a very accurate response, that would require maybe several minutes to be responded, or they prefer a fast search which will be completed in just a few seconds, at the expense of the accuracy of the responses.

Several lines of future work can be devised from this master thesis. First, better smoothing approaches at line-level or frame-level should be explored, in order to try to make smaller the gap between these levels, which provide a very fast lookup time, and the Filler Back-off model. Although, it is clear that models that do not rely on a word vocabulary, will always be less sensible to out of vocabulary events.

Finally, given that the Back-off heuristic gave such good results, other combination schemes should be explored, like interpolation of scores. For instance, the scores could be interpolated among different KWS approaches like WG, Filler and BLSTM. In the case of in-vocabulary keywords, this will not present any problem: Filler and BLSTM approaches are widely more expensive than the WG approach, but for in-vocabulary keywords, the score can be computed during indexing time. In the case of out of vocabulary keywords, the lookup speed would be obviously slower, so new ways of speeding up these models should be explored.

BIBLIOGRAPHY

- [1] J.-C. Amengual and E. Vidal. On the estimation of error-correcting parameters. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 883–886, 2000.
- [2] R. Baeza-Yates and G. Navarro. Fast approximate string matching in a dictionary. In *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pages 14–22. IEEE, 1998.
- [3] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.
- [4] H. Bunke and P. S. Wang. *Handbook of character recognition and document image analysis*. World Scientific, 1997.
- [5] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, Apr. 1973.
- [6] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–393, 1999.
- [7] F. Drira. Towards restoring historic documents degraded over time. In *Document Image Analysis for Libraries, 2006. DIAL'06. Second International Conference on*, pages 8–pp. IEEE, 2006.
- [8] R. O. Duda and P. E. Hart. *Pattern classification and scene analysis*. John Wiley, New York, 1973.
- [9] L. Egghe. The measures precision, recall, fallout and miss as a function of the number of retrieved documents and their mutual interrelations. *Information Processing & Management*, 44(2):856–876, 2008.

- [10] S. Espana-Boquera, M. J. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez. Improving offline handwritten text recognition with hybrid HMM/ANN models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(4):767–779, 2011.
- [11] A. Fischer, V. Frinken, H. Bunke, and C. Y. Suen. Improving HMM-based keyword spotting with character language models. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 506–510. IEEE, 2013.
- [12] A. Fischer, A. Keller, V. Frinken, and H. Bunke. HMM-based word spotting in handwritten documents using subword models. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3416–3419. IEEE, 2010.
- [13] A. Fischer, A. Keller, V. Frinken, and H. Bunke. Lexicon-free handwritten word spotting using character HMMs. *Pattern Recognition Letters*, 33(7):934 – 942, 2012. Special Issue on Awards from ICPR 2010.
- [14] W. Francis and H. Kucera. Manual of information to accompany a standard corpus of present- day edited american english, for use with digital computers., 1979. Cited by 0000.
- [15] V. Frinken, A. Fischer, and H. Bunke. A novel word spotting algorithm using bidirectional long short-term memory neural networks. In *Artificial Neural Networks in Pattern Recognition*, pages 185–196. Springer, 2010.
- [16] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A novel word spotting method based on recurrent neural networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):211–224, 2012.
- [17] R. Garside, G. Leech, and T. Váradi. Manual of information for the lancaster parsed corpus. *Bergen, Norway: Norwegian Computing Center for the Humanities*, 1995.
- [18] A. Giménez, I. Khoury, and A. Juan. Windowed bernoulli mixture HMMs for arabic handwritten word recognition. In *Frontiers in*

-
- Handwriting Recognition (ICFHR), 2010 International Conference on*, pages 533–538. IEEE, 2010.
- [19] J. Holmes, G. Johnson, and B. Vine. *Guide to the Wellington corpus of spoken New Zealand English*. School of Linguistics and Applied Language Studies, Victoria University of Wellington, 1998.
- [20] F. Jelinek. *Statistical methods for speech recognition*. MIT press, 1997.
- [21] V. M. Jiménez and A. Marzal. Computing the k shortest paths: A new algorithm and an experimental comparison. In *Algorithm engineering*, pages 15–29. Springer, 1999.
- [22] S. Johansson, G. Leech, and H. Goodluck. *Manual of information to accompany the Lancaster-Oslo/Bergen corpus of British English, for use with digital computers*. Department of English, University of Oslo, 1978.
- [23] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.
- [24] J. Keshet, D. Grangier, and S. Bengio. Discriminative keyword spotting. *Speech Communication*, 51(4):317–329, 2009.
- [25] H. Ketabdar, J. Vepa, S. Bengio, and H. Bourlard. Posterior based keyword spotting with a priori thresholds. 2006.
- [26] R. Kneser and H. Ney. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 181–184. IEEE, 1995.
- [27] A. Kolcz, J. Alspector, M. Augusteijn, R. Carlson, and G. V. Popescu. A line-oriented approach to word spotting in handwritten documents. *Pattern Analysis & Applications*, 3(2):153–168, 2000.
- [28] K.-F. Lee. *Automatic Speech Recognition: The Development of the Sphinx Recognition System*, volume 62. Springer, 1989.

- [29] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.
- [30] E. Lleida, J. B. Mariño, J. M. Salavedra, A. Bonafonte, E. Monte, and A. Martinez. Out-of-vocabulary word modelling and rejection for keyword spotting. In *EUROSPEECH*, 1993.
- [31] R. Manmatha, C. Han, and E. M. Riseman. Word spotting: A new approach to indexing handwriting. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on*, pages 631–637. IEEE, 1996.
- [32] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.
- [33] U.-V. Marti and H. Bunke. Using a statistical language model to improve the performance of an HMM-based cursive handwriting recognition system. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):65–90, 2001.
- [34] U.-V. Marti and H. Bunke. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.
- [35] K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012.
- [36] M. Pastor i Gadea. *Aportaciones al reconocimiento automático de texto manuscrito*. PhD thesis, Universitat Politècnica de València, 2007.
- [37] J. A. Rodríguez-Serrano and F. Perronnin. Handwritten word-spotting using hidden markov models and universal vocabularies. *Pattern Recognition*, 42(9):2106–2116, 2009.
- [38] V. Romero. *Multimodal Interactive Transcription of Handwritten Text Images*. PhD thesis, Universidad Politècnica de Valencia, 2010. Advisors: Enrique Vidal and Alejandro H. Toselli.

-
- [39] M. Rusinol, D. Aldavert, R. Toledo, and J. Lladós. Browsing heterogeneous document collections by a segmentation-free word spotting method. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 63–67. IEEE, 2011.
- [40] R. Saabni and A. Bronstein. Fast key-word searching via embedding and active-dtw. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 68–72. IEEE, 2011.
- [41] A. Sanchis, A. Juan, and E. Vidal. A word-based naïve bayes classifier for confidence estimation in speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(2):565–574, 2012.
- [42] L. Tarazón, D. Pérez, N. Serrano, V. Alabau, O. R. Terrades, A. Sanchis, and A. Juan. Confidence measures for error correction in interactive transcription handwritten text. In *Image Analysis and Processing-ICIAP 2009*, pages 567–574. Springer, 2009.
- [43] A. H. Toselli. *Reconocimiento de Texto Manuscrito Continuo*. PhD thesis, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 2004. Advisor(s): Dr. E. Vidal and Dr. A. Juan (in Spanish).
- [44] A. H. Toselli, A. Juan, J. González, I. Salvador, E. Vidal, F. Casacuberta, D. Keysers, and H. Ney. Integrated handwriting recognition and interpretation using finite- state models. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(04):519–539, 2004.
- [45] A. H. Toselli and E. Vidal. Fast HMM-Filler approach for key word spotting in handwritten documents. In *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 501–505, 2013.
- [46] A. H. Toselli, E. Vidal, V. Romero, and V. Frinken. Word-graph based keyword spotting and indexing of handwritten document images. Technical report, Universidad Politécnica de Valencia, 2013.

- [47] A. H. Toselli, E. Vidal, V. Romero, and V. Frinken. Word-graph based keyword spotting for handwritten document images. Under review, 2013.
- [48] E. Trentin and M. Gori. A survey of hybrid ANN/HMM models for automatic speech recognition. *Neurocomputing*, 37(1):91–126, 2001.
- [49] N. Ueffing and H. Ney. Word-level confidence estimation for machine translation. *Computational Linguistics*, 33(1):9–40, 2007.
- [50] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines-part i. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7):1013–1025, 2005.
- [51] E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite-state machines-part ii. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7):1026–1039, 2005.
- [52] F. Wessel, R. Schluter, K. Macherey, and H. Ney. Confidence measures for large vocabulary continuous speech recognition. *Speech and Audio Processing, IEEE Transactions on*, 9(3):288–298, 2001.
- [53] S. J. Young and S. Young. The HTK hidden markov model toolkit: Design and philosophy. *Entropic Cambridge Research Laboratory, Ltd*, 2:2–44, 1994.

LIST OF FIGURES

2.1	Overview of the stages of a HTR system.	7
2.2	Illustration of a HMM modeling the character “t” of a handwritten text image where the text “text” is found. The boxed image regions are mapped to the state that emitted them.	8
2.3	A hierarchical FSM constructed by integrating Character HMMs, Lexical Models and LMs.	15
2.4	An example of a word graph for an image of the sentence “ <i>antiguos ciudadanos que en Castilla se llamaban</i> ”. . . .	18
3.1	Compositions of HMMs used by the Filler approach. . . .	25
4.1	SFSM used to compute $P(u aab), \forall u \in \Sigma^*$. Erroneous edit operations are represented by dashed lines.	34
4.2	Examples showing that the compression of posteriorgrams is possible.	44
5.1	Some page examples of the “Cristo-Salvador” corpus. . .	48
5.2	Detailed view of a “Cristo-Salvador” page image.	48
5.3	Line examples of the IAM database.	51
5.4	Venn diagram showing the sets that define Precision and Recall.	55
5.5	An example of a Recall-Precision curve (dashed) and the RP curve with interpolated precision (solid).	56
5.6	Histogram of the AP of test keywords for CS and IAMDB using the baseline Filler and WG models.	62
5.7	Histogram of the score of test relevant events for CS and IAMDB using the baseline Filler and WG models.	63
5.8	Histogram of the distribution of the AP among the CS test keywords achieved using the baseline scores and the line-level and frame-level smoothed scores.	66

5.9	Histogram of the distribution of the scores among the CS test events given by the baseline WG, the line-level and frame-level smoothing methods.	66
5.10	Histogram of the scores distribution for relevant events in the CS test partition, using different Back-off heuristics.	70
5.11	Histogram of the scores distribution for relevant events in the CS test partition, using different KWS approaches. Observe that the Back-off heuristic distributes the in-vocabulary scores as the “WG Baseline” and the OOV scores as the “Filler Scaled”.	71
5.12	Recall-Precision curves of the best methods on the CS and IAMDB test partitions. Curves: (a) Baseline Filler (b) Baseline WG (c) SEC-LC Back-off (d) Pstgram-LC Back-off (e) Filler Back-off.	74

LIST OF TABLES

5.1	Basic statistics of the “Cristo-Salvador” used partition. . .	49
5.2	Basic averaged statistics of the “Cristo-Salvadors” cross-validation partitions.	49
5.3	Basic statistics of the IAMDB corpus.	51
5.4	Basic statistics of the LOB, Brown and Wellington corpora.	51
5.5	Basic statistics of the selected query keywords for CS and IAMDB. Data from the validation CS set is averaged across the 10 partitions.	53
5.6	Example showing the difference between AP and MAP. .	57
5.7	Baseline AP and MAP for CS.	60
5.8	Baseline AP and MAP for IAMDB.	60
5.9	Best AP and MAP achieved using the three proposed line-level smoothing methods on the CS database.	64
5.10	Best AP and MAP achieved using the three proposed line-level smoothing methods on the IAM database.	64
5.11	Best AP and MAP achieved using the two proposed frame-level smoothing methods on the CS database.	65
5.12	Best AP and MAP achieved using the two proposed frame-level smoothing methods on the IAM database.	65
5.13	Best AP and MAP achieved with the Back-off combination heuristic for different smoothing methods, CS database. .	68
5.14	Best AP and MAP achieved with the Back-off combination heuristic for different smoothing methods, IAM database.	69
5.15	Summary table showing the test AP and MAP on the CS and IAM databases for all the presented methods and the original baselines.	72
5.16	Summary table showing the best AP and MAP achieved for each smoothing level and the relative improvement respect the baseline WG method.	73

5.17 Average number of frames and resulting number of frames after compression, for the test partition of the CS and IAM databases.	75
---	----