Hidden Markov Models for Spelling Recognition

Shieu-Hong Lin
Department of Mathematics and Computer Science
Biola University
shieu-hong.lin@biola.edu

1 Introduction

The mathematical framework of the Hidden Markov models (HMMs) along with the algorithms related to HMMs have played a very important role in the successful development of intelligent systems for a wide variety of AI tasks in speech recognition (Jelinek 1998) (Russell & Norvig 2003) and natural language processing (Charniak 1994) (Manning & Schütze 1999). In AI courses, automatic speech recognition is often cited as a challenging and yet well accomplished AI task by using HMMs and probabilistic reasoning (Russell & Norvig 2003). To illustrate the general mechanism of modern speech recognition systems, the mathematical definition of HMMs is provided along with description of the development of the underlying acoustic model as an HMM (Russell & Norvig 2003). This approach only presents a somewhat abstract theoretical framework of the use of HMMs, and the students do not have a chance to be involved in the process of modelling and problem solving using HMMs.

In this handout, we describe the use of Hidden Markov models for automatic spelling recognition. Through a process of programming and experiments along with these models, the students can cultivate the understanding of HMMs in a simple context of spelling recognition. Both automatic spelling recognition (Jurafsky & Martin 2000) and automatic speech recognition (Jelinek 1998) can be viewed as tasks of decoding corrupted messages received from a noisy-channel (Mackay 2003), and the HMMs used for spelling recognition in the project are structurally parallel to those used in speech recognition. This allows us to conveniently find analogy to explain the general mechanism of modern speech recognition systems based on the experiences with the spelling recognition systems the students are building.

2 Spelling-Recognition Tasks in a Chat-Room

In the chat-room environment, the participants under their pseudo names can communicate with one another by exchanging text entered from the keyboard. Because of cognitive and typographic mistakes, character strings appearing in the text inputs in the chat room may not be valid words in the vocabulary, and we have to guess what the actual intended words are. For example, when the string "*iis*" appears in the text input from a participant, we have to figure out the most likely words in the vocabulary such as "*is*" and "*its*" as the source of the corrupted string. When we are familiar with the patterns of corrupted words from a frequent participant, we may well recognize the original message from the corrupted text inputs. Even when the frequent participant p is under the disguise of a new pseudo name, we may be able to recognize this seemingly unknown person by analyzing the sample text inputs and its consistency with the spelling characteristics of the well known participant p.

To highlight the analogy to speech recognition, we intentionally use the term *spelling recognition* instead of spelling correction. We focus on spelling recognition for English and assume the text inputs are case-insensitive. In section 3, we describe very simple probabilistic models for characterizing how chat-room participants may (erroneously or not) enter words through the keyboard. Given (i) the probabilistic models for text inputs through the keyboard and (ii) a vocabulary set V of words in Σ^* with respect to the English alphabet Σ, we formulate the following spelling recognition tasks in the context of the chat-room environment.

Single-word spelling recognition: Given a character string x entered by a specific participant p, the task of single-word spelling recognition is to determine the top k most likely words in V that could be (erroneously) typed by the participant p as the character string x, where k is either one or a bigger integer.

Multi-word spelling recognition: Given a sequence of c character strings $\mathbf{X} = \langle x_1, x_2, \ldots x_c \rangle$ entered by a specific participant p, the task of multi-word spelling recognition is to determine the most likely sequence of c words that may end in the sequence of character strings $\mathbf{X}$ when entered by the participant p.

Participant identity recognition: Given a sequence f c character strings $\mathbf{X} = \langle x_1, x_2, \ldots x_c \rangle$ entered by an unidentified participant, the task of participant identity recognition is to determine, among a set of well known participants, the most

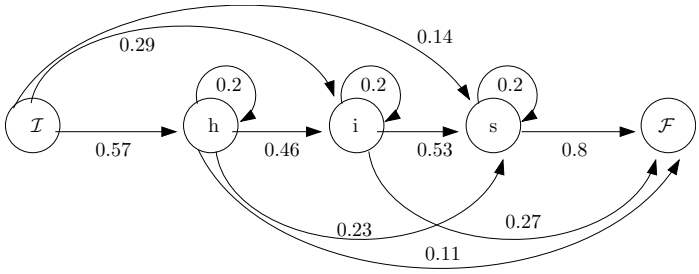

Figure 1: The spelling model regarding the word "*his*" with the parameters $deg_{sp} = 2$, $p_{repeat} = 0.2$, and $p_{moveOn} = 0.8$

likely participant that may generate the strings in **X**.

3 Modelling the Input Process of Words

To be able to conduct automatic spelling recognition, we need to devise probabilistic models of how participants input individual words through the keyboard. As the first step of the project, we define a very simple parameterized *spelling model* and a very simple parameterized *keyboard model* in the following for this purpose, from which we can construct HMMs used in probabilistic reasoning for spelling recognition.

3.1 The Spelling Model

The main purpose of the spelling model is to model the possibilities of spelling mistakes such as repeating or skipping characters when a participant enters a word as a sequence of characters.

Probabilistic state-transition diagrams: For each distinct participant p and a given word w, the spelling model employs a probabilistic state-transition diagram $\mathcal{SP}_p^w$ defined in the following to describe the possible transitions of cognitive states and the associated probabilities when w is entered as a sequence of characters by p. See the example in Figure 1 and the definitions of the parameters involved later in Section 3.1.

States and the order of states in the diagrams: For each distinct participant p and a given word w, the states in $\mathcal{SP}_p^w$ are ordered and represented as nodes listed from left to right in the diagram. The initial state $\mathcal{I}$ is the first state, which models p's cognitive state of acknowledging the beginning of inputting w. Then one by one for each character c in the word w, there is a corresponding *character state* in $\mathcal{SP}_p^w$, which models p's cognitive state of commitment to typing that particular character c. In the end, the final state $\mathcal{F}$ models p's cognitive state of acknowledging the end of inputting w. This gives us $n+2$ distinct states in total for a word w of n characters.

State transitions and spelling mistakes: Possible state transitions are represented as arcs in the diagram as shown in Figure 1. From the initial state $\mathcal{I}$, it is possible to transition into to any character state. From each character state, it is possible to transition into either the same state again, or to any later character state (i.e. character states to its right), or to the final state. Each time the participant p goes through the process of inputting the word w, a series of cognitive state transitions occurs, starting from the initial state $\mathcal{I}$ and ending in the final state $\mathcal{F}$. If the word w is spelled out perfectly without mistake, it should start from the initial state, go through the character states one by one in the order, and end in the final state. However, spelling mistakes may occur and end in repetition and/or skipping of some character states in the series of state transition.

Transition probabilities: In general, the probability of a state transition from a state s to a state t can be any real number in the range of $[0, 1]$ and is associated with the arc from s to t in the diagram. The only constraint is that the sum of transition probabilities from a state s must be one for every state s other than the final state.

Parameters for modelling cognitive mistakes: To ease the initial programming tasks for the students, we allow the use of three parameters deg_{sp}, p_{repeat}, and p_{moveOn} to further simplify the spelling model.

- p_{repeat}: In every character state, p_{repeat} is the probability of transition into the same state again.

- p_{moveOn}: In every character state, p_{moveOn} is the sum of probabilities of transition into any later character state or to the final state (i.e. the states to its right). Obviously, the sum of p_{moveOn} and p_{repeat} must be one.

- deg_{sp}: the parameter deg_{sp} is the rate of exponential degrading of probability with respect to the number of character skipped. The probability of skipping d (for $d > 0$) characters is proportional to deg_{sp}^{-d}. In other words, the probability of skipping d (for $d > 0$) characters is simply some constant times deg_{sp}^{-d}. This setting ensures that the probability of skipping d_1 characters is $deg_{sp}^{-(d_1-d_2)}$ times the probability of skipping d_2 characters.

Calculating the transition probabilities automatically: Note that (i) the sum of transition probabilities from each specific state (other than the final state) into new states must equal p_{moveOn} and (ii) the relative scales of transition probabilities from each specific state into new states are completely determined by deg_{sp}^{-d}, depending on the skipping distance d. This allows us to easily write code to automatically calculate the probabilities of all possible transitions in the diagram $\mathcal{SP}_p^w$ based on the parameters deg_{sp} and p_{moveOn}.

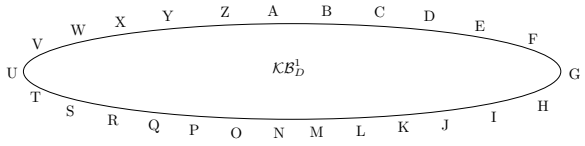

Figure 2: The one-dimensional keyboard model $\mathcal{KB}_D^1$

3.2 The Keyboard Model

The main purpose of the keyboard model is to model the probabilities of typographic mistakes of pressing a wrong key different from the intended character targeted by the current cognitive state.

Probabilistic parameters for typographic mistakes: To ease the initial programming tasks for the students, we use three probabilistic parameters p_{hit}, p_{miss}, and deg_{kb} to simplify the keyboard model.

- p_{hit}: the parameter p_{hit} is the probability that the exact key for the intended character is pressed to generate the character.
- p_{miss}: the parameter p_{miss} is the probability that a wrong key is pressed. Obviously, the sum of p_{hit} and p_{miss} must equal one.
- deg_{kb}: The parameter deg_{kb} is the rate of exponential degrading of probability with respect to the distance between the key of the intended character and the key wrongly pressed. The probability of pressing a wrong key a distance of d (for $d > 0$) away is proportional to deg_{kb}^{-d}. In other words, the probability of of pressing a wrong key a distance of d (for $d > 0$) away is simply some constant times deg_{sp}^{-d}. This setting ensures that the probability of hitting a wrong key that is in a distance of d_1 away from the intended key is $deg_{kb}^{-(d_1-d_2)}$ times the probability of hitting a wrong key that is in a distance of d_2 distance away from the intended key.

A simple distance metrics $\mathcal{KB}_D^1$ is considered in the following.

One-dimensional keyboard distance metric $\mathcal{KB}_D^1$: Imagine a special keyword in the shape of a circle as shown in Figure 2 with the keys for the twenty six English letters evenly arranged on the keyboard in the alphabetical order. For example, 'A' is immediately adjacent to 'Z' and 'B', 'B' is immediately adjacent to 'A' and 'C',. . ., and 'Z' is immediately adjacent to 'Y' and 'A'. The distance between two keys is simply one plus the minimum number of keys separating them. For example, the distance between 'A' and 'B' is one, the distance between 'A' and 'C' is two,. . ., the distance between 'A' and 'Y' is two, and the distance between 'A' and 'Z' is one,

Calculating the probabilities of typographic mistakes: Note that (i) for each specific character c the sum of probabilities of pressing wrong keys must equal p_{miss} and (ii) the relative scales of the probabilities of pressing these wrong keys are completely determined by deg_{kb}^{-d} where d is the distance between the wrong key and the intended key. This allows us to easily write code to automatically calculate the probabilities of pressing wrong keys given the intended key based on the parameters deg_{kb}, p_{miss} and the selected keyboard distance metric.

A simplified example: Consider the situation that $p_{miss} = 0.1$, $p_{hit} = 0.9$, and $deg_{kb} = 2$. If the one-dimensional keyboard only has 4 keys a, b, c, d (instead of the full 26 keys), the probabilities of typographic mistakes when trying to type a are

- $Pr(Char = b|State = a) = 0.04$,
- $Pr(Char = c|State = a) = 0.02$, and
- $Pr(Char = d|State = a) = 0.04$.

3.3 The Language Model and Further Extensions

For simplicity, our basic framework implicitly uses the null language model, which assumes that each word is equally likely to be used regardless of the context and therefore sentences of the same length are equally likely. It is possible to adopt a more sophisticated language model such as the bigram language model (Charniak 1994) (Manning & Schütze 1999). See section 4 about how this would allow us to take advantage of the context of the observed strings when conducting multi-word spelling recognition. Also see section 5.2 for further extensions of the spelling model and the keyboard model.

References

Charniak, E. 1994. *Statistical Language Learning*. MIT Press.

Jelinek, F. 1998. *Statistical Methods for Speech Recognition*. MIT Press.

Jurafsky, D., and Martin, J. H. 2000. *Speech and Lnaguage Processing*. Prentice Hall.

Mackay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge Universirt Press.

Manning, C. D., and Schütze, H. 1999. *Foundations Of Statistical Natural Language Processing*. MIT Press.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach,*. Prentice Hall, 2nd edition.