Creating Word-Level Language Models for Handwriting Recognition

John F. Pitrelli and Amit Roy
IBM Research, P.O. Box 218, Yorktown Heights, NY 10598 U.S.A.
pitrelli@us.ibm.com

Abstract

For large-vocabulary handwriting-recognition applications, such as note-taking, word-level language modeling is of key importance, to constrain the recognizer's search and to contribute to the scoring of hypothesized texts. We discuss the creation of a word-unigram language model, which associates probabilities with individual words. Typically, such models are derived from a large, diverse text corpus. We describe a three-stage algorithm for determining a word unigram from such a corpus. First is tokenization, the segmenting of a corpus into words. Second, we select for the model a subset of the set of distinct words found during tokenization. Complexities of these stages are discussed. Finally, we create recognizer-specific data structures for the word set and unigram. Applying our method to a 600-million-word corpus, we generate a 50,000-word model which eliminates 45% of word-recognition errors made by a baseline system employing only a character-level language model.

1. Introduction

The IBM on-line handwriting-recognition system is a hidden-Markov-model-based (HMM) system which seeks to label any handwriting signal with the highest-probability word sequence by employing a statistical character-shape model, which represents *how* people write, and a language model, which represents *what* people are likely to write. The language model typically consists of word or character N-grams. In view of the boundless variety of handwriting styles, language modeling is critical given that the character-shape model can never be complete. We consider general-text applications such as note-taking, for which we have previously found that word-level language modeling will be of primary importance [6]. We use a **word unigram** to associate words with probabilities to contribute to the recognizer's scores when it hypothesizes text labels for handwriting. The unigram's **word set** constrains the recognizer's search. For such large-vocabulary applications, language models are derived from a large, diverse text corpus chosen to represent general usage of the language.

We describe a three-stage algorithm by which we derive a unigram from such a corpus. First is **tokenization**, in which we segment a corpus into words. Note that this is not simply a matter of segmenting at spaces and carriage returns in the text. For example, a period is typically adjacent to preceding text, but is not part of the final word in a sentence unless that word is an abbreviation. Tokenization produces a **source** unigram – a table associating each distinct word with its frequency of occurrence.

The second stage is **selection**, in which we choose a subset of the source to become the language model for recognition. The number of words selected is regulated by limits of recognizer size and by diminishing benefit as rare words add little coverage but weaken the constraint provided by the language model. Selection, however, should not consist of simply choosing the N most-frequently-occurring words from the source. Other issues arise, such as avoiding counter-intuitive inconsistencies in the treatment of words with a systematic relationship. For example, if `thirty-fourth` is included in the word set, but `thirty-third` is not and so cannot be recognized, users would likely become displeased with the unintuitive behavior of the system.

The third stage is to create the data structures for the word set and unigram. The formats of these structures are specific to our recognizer's implementation, and so are of least general interest and not discussed further. The key is to separate this stage from the other stages, which are largely generalizable across recognizers.

The first two stages are described in detail below, in the context of American English. Interspersed are results of applying this algorithm to generate a language model from a 600-million-word text corpus gathered from news, office correspondence, medical and legal documents, etc. Although this corpus's representativeness of the note-taking task is questionable, in view of uncertainty about the applicability of *any* corpus our priority was to obtain enough text to train a large model. Finally, a simple experiment quantifies the contribution of the language model to recognition accuracy.

0-7695-1263-1/01/$10.00 © 2001 IEEE

2. Stage 1: Tokenization

This stage converts a plain-text corpus with various formats into a single, shared format which systematically delimits the text passages which we will treat as "words" for purposes of language modeling. The primary challenge of this stage is in fact to determine what should constitute a word. For handwriting recognition, generally we want to treat as a single word any sequence of non-white-space characters occurring between white-space characters. However, we want several exceptions for which recognition search can manage alternatives to requiring inter-word spacing between occurrences of members of its word set. Such exceptions include parentheses, quotation marks, end-of-sentence periods, commas, etc. – punctuation which is adjacent to a word but occurs largely independently of that word. Thus, we do not want a word-level language model to have separate entries for a word like `it` with and without a following comma or period, with and without quotation marks, etc., which would greatly inflate the size of the language model, arbitrarily and inappropriately because the punctuation is effectively independent of the word `it`.

While literature on tokenization does exist [1] [2] [3], most such work pursues purposes other than handwriting-recognition language modeling. Specifically, studies have focused on web search applications, detection of specific text types such as named entities, and natural-language understanding (NLU). Such different purposes imply the need for different criteria for tokenization. For example, NLU applications often require splitting morphemes, *e. g.* splitting `won't` into tokens representing `will` and `not` [2], whereas for our purposes `won't` is effectively a single word. These differing needs limit the applicability of past work to the current task.

Some tokenization issues are illustrated by texts which superficially appear similar but are in fact best treated differently. For example, in `... the abbr. listed here, ...` we treat `abbr.` as a single word, while in `... the item. Listed here, ...` the period should not be considered part of the word `item`; rather, it indicates the end of a sentence. Similar issues exist with commas, which may be part of a word such as in `... in August 19,998 people ...` but more frequently occur between words, such as in `... on August 19, 1998, people ...`

Other tokenization issues concern hyphens. Often, two independent words are joined by a hyphen to form a compound adjective such as `job-hunting`; we consider this to be three separate lexical elements, as we do not expect to extract significant statistics on such sequences. However, some such sequences should be considered to be single tokens, such as `Haagen-Dazs` or `wishy-washy`, where the separate words do not maintain their identities, or have no identity at all. We refer to these as **hyphenated pairs**. In other cases, one of the "words" is dependent on the hyphen while the other is not; thus tokenization must be sensitive to **prefixes** such as `multi-` and **suffixes** such as `-hearted`.

Statistics of the corpus help to make some of these distinctions. Continuing our example, the corpus shows that `abbr` is nearly always followed by a period while `item` is not. For this reason, tokenization is implemented as a multi-pass procedure, in which early passes identify special character sequences such as abbreviations, and the final pass produces the actual tokenized corpus. Finally, we compute a table of frequencies of occurrence of the distinct words in the tokenized corpus. The sections below detail each step.

2.1. Text normalization

For various reasons, some corpora have inconsistent representations for text. If a corpus was produced for a text-formatting program, macros may appear in place of characters which have special meaning to that program, and other mark-ups may be interspersed for formatting. To recover plain text, some corpora require individualized substitution and deletion steps.

2.2. Determination of special words with hyphens and periods

2.2.1 Prefix selection

The purpose of prefix selection is to determine when a hyphen is part of the word which precedes it. For example, `now-nationalized` is a compound adjective in which the hyphen links two stand-alone words, whereas in `multi-national`, `multi-` is a prefix; `multi` alone is not a word. We chose the following rule to decide whether word `A` is to be treated as a prefix, that is, attached to a following hyphen: If `A-` is much more frequent than `A` without the hyphen ($>$ 20 times as frequent), `A-` occurs frequently enough to be observed reliably (relative frequency of `A-` $> 10^{-7}$), and `A` without a following hyphen is rare enough that it is not likely to be a legitimate stand-alone word (frequency of `A` without following hyphen $< 10^{-6}$), then consider `A-` a prefix. This rule choice was empirical, to select only "words" which our knowledge of the language confirms as truly existing only as prefixes and not as stand-alone words. With this rule, we obtain a list of 56 prefixes from our corpus; a sampling is `agri-`, `anti-`, `demi-`, `ethno-`, `multi-`, `pre-`, `semi-`, and `socio-`.

2.2.2 Suffix selection

Similarly, suffix selection determines when a hyphen is part of the following word. For example, in `stand-by`, the hyphen ties together two stand-alone words, whereas in

stand-offs, -offs is a suffix; offs alone is not a valid word. Our suffix selection method is analogous to prefix selection; if -A is much more frequent than A without the hyphen (> 20 times), -A occurs frequently enough to be observed reliably (frequency of -A > 10^{-7}), and A without a preceding hyphen is rare enough that it is not likely to be a legitimate stand-alone word (frequency of A without preceding hyphen < 10^{-6}), then -A is a suffix. With this rule, we obtain 150 suffixes; some examples are -ager, -esque, -footer, -mindedly, and -hearted.

2.2.3 Hyphenated-pair selection

Here the goal is to identify word-hyphen-word sequences whose words' appearances are so strongly correlated that they should be treated together as single entries, such as Haagen-Dazs and wishy-washy. Our rule must decide if A-B is to be treated as a single hyphenated-pair entry, vs. any of four other possibilities: prefix A- and stand-alone B, stand-alone A and suffix -B, prefix A- and suffix -B, or stand-alone A, hyphen and B. The rule is: If strong majorities (70%) of appearances of both A- and -B occur as A-B, or a very strong majority (90%) of appearances of either A- or -B occur as A-B, then if A-B occurs frequently enough to be observed reliably (frequency of A-B > 10^{-6}) it is considered to be a hyphenated pair. With this rule, we obtain 133 hyphenated pairs; examples include self-esteem, per-capita, avant-garde, fiber-optic, front-runner, and north-south.

2.2.4 Abbreviation selection

The purpose of abbreviation selection is to determine when a period is a part of the word which precedes it, typically in an abbreviation, like Mr., vs. when it is not, such as at the end of a sentence. Our rule is very similar to prefix selection, differing primarily because periods not in abbreviations are common at ends of sentences, while hyphens are rarer outside of prefixes, suffixes, and hyphenated pairs. The rule is: If A. is much more frequent than A without the period (> 20 times), A. occurs frequently enough to be observed reliably (frequency of A. > 10^{-6}), and A without a following period is rare enough that it is not likely to be a legitimate stand-alone word (frequency of A without following period < 5×10^{-6}), then A. is an abbreviation. With this rule, we obtain 110 abbreviations; examples include Sgt., pp., Jr., Dec., vols., Corp., and Ariz. We also obtain a spurious result, Overruled., presumably due to this word appearing frequently as a one-word sentence in legal text, and rarely elsewhere.

2.3. Final tokenization

Following creation of the special lists needed to handle periods and hyphens, we are ready for final text processing. We treat the procedure as one of **adjusting the spaces** in the corpus so that they delineate what we want to model as words. We divide space adjustment into two passes: character-level and word-level.

2.3.1 Character-level space adjustment

Here, we apply all space-adjustment rules which are definable in terms of a small, fixed-character-length context. These rules may be applied in parallel with the creation of special word lists above.

- Identify some periods which should definitely remain adjacent to the preceding characters. If in doubt, such as for a period which may indicate either an abbreviation or the end of a sentence, put a space before it to defer to word-level space adjustment. The rule is to ensure a space before a period unless any of the following is true: the period is preceded (without a space) by an upper-case letter (U.S.), it is followed by a digit or a letter (3.14, a.k.a.), it is preceded by a lower-case letter that in turn is preceded by a space (v.), or it is followed by an optional space followed in turn by a lower-case letter or sentence-medial punctuation (ampersand, comma, slash, colon, or semicolon).
- Ensure spaces on each side of commas and hyphens, deferring treatment of cases in which they are part of larger "words", such as 300,000,000 and Haagen-Dazs, to word-level space adjustment.
- Ensure spaces on either side of other punctuation marks, making them separate "words", except allow # and $ to precede digits, % to follow digits, and apostrophes to follow letters, as in two dinners' food, to precede digits following a space, as in '99, and to remain surrounded by letters, as in don't.

2.3.2 Word-level space adjustment

The purpose of this step is to determine whether the now-isolated commas, hyphens, and periods are parts of words or independent of the words next to them, based on evaluating their context at the word level. Rules are:

- If the word following a comma consists of three digits, optionally followed by a decimal point and optionally other characters which are all digits, and/or ending in %, and it is preceded by up to three digits optionally preceded by # or $, then eliminate spaces around comma. This distinguishes commas which are part of long numbers from those which are not part of a

"word", making `19,998` into a single word while `19`, comma, and `1998` remain separate in the example in Section 2.

- If the word preceding a period is on the abbreviation list, such as `Mr.`, remove the space between the word and the period.
- If the pair of words surrounding a hyphen is on the hyphenated-pairs list, *e. g.* `per-capita`, remove both spaces from around the hyphen.
- Otherwise, if the word preceding the hyphen is on the prefix list, remove the space before the hyphen, *e. g.* `pre- school`. If the word following the hyphen is on the suffix list, remove the space after the hyphen, *e. g.* `teen -ager`. If both, duplicate the hyphen, attaching one to each word with a space in between, making the first word a prefix and the second a suffix, *e. g.* `multi- -masted`. If neither, as for `job-hunting`, leave the spaces around the hyphen, treating `job`, hyphen, and `hunting` as three separate words.

2.4. Creating word-frequency table

Finally, we create a table of word frequencies to serve as the source language model for subsequent stages. Having finished space adjustment, a "word" is now simply the text that occurs between white-space. On our corpus, tokenization yields a source unigram consisting of 1.2 million distinct words, 830,000 of which are purely alphabetic.

3. Stage 2: Selection

This algorithm extracts from the source a word-unigram table of a chosen size, N entries, to serve as the language model for recognition. While a simple approach would be to pick the N most-frequently-occurring words, several issues inherently complicate performing this task well for particular combinations of source corpus and application:

1. Applications may require some words to be included regardless of their occurrence in the corpus, for example, application-specific terms, phone numbers, or postal codes.
2. Some words in the corpus, such as mark-up tags and obscenities, may need to be excluded from the word set even if they are in the top N.
3. Some word subsets should be included or excluded in their entirety. For example, if `waterfall`, `thirty-fourth`, `9am` and `XXVII` were in the top N and so could be recognized, but `waterfalls`, `thirty-third`, `10am` and `XXVI` were not, users would likely become confused or displeased with the unintuitive behavior of the system.
4. Some words need to be changed due to corpus anomalies, such as words included in all-capitals.

Because of these complications, selection is considerably more complex than simply picking the source's top N. It is also not fully automatable – manual intervention is called for at several points.

The inputs to selection are a set of required words (see items 1 and 3 above), which must be output into the word set, a set of excluded words (see 2 and 3 above), which must not appear in the output, the target number N of words desired in the word set, and the source unigram. Optionally, **augmentation** parameter pairs E and M may be provided; once the top N source words form an initial unigram, they are then augmented with any more words in the top M (that is, in the next $M - N$ after the top N) whose first E characters match those of a word which is already in the word set. Successive pairs of E and M may be used, but each successive E must get larger, and each successive M must get larger and be greater than N. We augment for three reasons: (1) it helps prevent user-displeasing patterns like some endings with a word being included, and others not, like `waterfall` being recognized but `waterfalls` being unknown, (2) for a character-tree-structured word set, these words enlarge the structure less than other words would, due to the E shared characters, and (3) words added this way add less confusability to recognition than words chosen without such text correlation to previously chosen words, because they branch off the existing network of word candidates further into the tree.

Output of the selection stage is a table associating probabilities with N words, or more if augmenting. We obtained a 50,000-word unigram by setting $N = 35,000$ and using one augmentation pair, $E = 4$ and $M = 60,000$.

3.1. Algorithm

1. Correct any anomalous all-capitals entries: some parts of the corpus have in all-capitals some words which should be lower-case or mixed-case. First, extract from the source any words which may need some re-casing. Currently, we take any word with more than one capital in it, excluding a few particular patterns like `McXxxx`. Manually remove from this list any word whose casing is already acceptable, like `USA`. Of the remaining words, any which has a lower-case or first-letter-only-capitalized version elsewhere in the source is automatically down-cased to match the existing version. Remaining words must be re-cased manually.
2. Remove excluded words from source; also find in source all re-casings and superstrings of excluded words; manually choose which of those to remove.
3. Start the output unigram with the R required words. For each word, if it is in the source, move its entry to

the output. If not, create an entry using 20000th-most-common-word's occurrence count, presuming that a required word will not be very rare.

4. Form top-N unigram: Sort the remaining source entries by frequency of occurrence; move the top $N - R$ entries from source to output.
5. Augment the output unigram as described above.
6. Convert occurrence counts to probabilities by dividing each count by the sum of all counts in the unigram.

4. Recognition Experiment

Experiments are performed using the IBM on-line handwriting recognizer in the IBM Ink Manager™ software; most algorithms are described in detail in previous papers [5] [6] [7]. Data are collected as a stream of (x, y) points indexed in time, re-sampled to be equi-distant. Features based on distances and angles are computed at each point. Windows of temporally-adjacent points are assembled around window centers, which are typically local extrema in x and y. Feature vectors of the points within a window are spliced together to form window feature vectors. These are projected onto a lower-dimensional space; the resulting vectors are called frames.

Each character in our HMM system is represented by a set of four allograph models, each consisting of varying-length sequences of states. Mixture-Gaussian models, trained using an EM algorithm, represent the distribution of frame vectors for each state. Beam search, governed by character set or word set, begins with a forward pass using fast-match allograph models. Hypothesized words are then re-scored using a character 4-gram or word unigram, and then again using detailed-match models.

Our full character set consists of 93 characters: 26 lower- and 26 upper-case letters, 10 digits, and 31 punctuation marks and other symbols. We use writer-independent unconstrained-style character models trained on a total of approximately 165,000 words plus 330,000 discrete characters provided by 450 writers. Our baseline system employs no constraints when hypothesizing sequences of these 93 characters, just the character 4-gram. Our experimental system uses a 50,000-word unigram derived using the algorithms described in this paper, with the same 93-character set. Testing is done using sentences drawn from a variety of texts, written by writers not supplying training data. On this test set, totaling 1158 words, the experimental system eliminates 45% of the word errors made by the baseline system, with no rejection permitted.

5. Conclusion

A variety of issues influence the processing of a text corpus to produce word-level language models suited for handwriting recognition. During tokenization, it is necessary to address the discrepancy between what appears to be a word in running text and what is best considered to be a word for recognition-language-modeling purposes. This requires a variety of steps, such as detaching selected punctuation from adjacent words. When selecting words for inclusion in the model, it is necessary to allow for some words which must be included, some which must be excluded, and other groups of words which must be either included or excluded together, regardless of their frequencies of occurrence in the corpus. By taking appropriate care in tokenization and when selecting words, a model may be designed to increase recognition accuracy while allowing for reasonably intuitive system behavior regarding which words can and cannot be recognized.

6. Acknowledgments

We gratefully acknowledge Gene Ratzlaff and Jayashree Subrahmonia for feedback on these algorithms and this paper; Michael Picheny, Karthik Visweswariah, Mike Monkowski, Julian Chen, and Srinivasa Rao for assistance with text corpora, and the anonymous reviewers for suggestions which improved the paper.

References

[1] Grefenstette, G., and P. Tapanainen, "What is a Word, What is a Sentence? Problems of Tokenization", *Proc. of the 3rd Int'l. Conf. on Computational Lexicography (COMPLEX '94)*, Budapest, Hungary, 1994, pp. 79-87.

[2] Grefenstette, G., "Tokenization" in H. van Halteren, ed., *Syntactic Wordclass Tagging* (Dordrecht: Kluwer Academic Publishers, 1999), pp. 117-133.

[3] Grover, C., C. Matheson, A. Mikheev, and M. Moens, "LT TTT - A Flexible Tokenisation Tool", *Proc. of the Second Int'l. Conf. on Language Resources and Evaluation (LREC 2000)*, Athens, Greece, May 31 - June 2, 2000.

[4] Guyon, I., L. Schomaker, R. Plamondon, M. Liberman, and S. Janet, "UNIPEN Project of On-Line Data Exchange and Recognizer Benchmarks", *Proc. of the 12th Int'l. Conf. on Pattern Recognition (ICPR '94)*, Jerusalem, October, 1994, pp. 29-33.

[5] Nathan, K. S., H. S. M. Beigi, J. Subrahmonia, G. J. Clary, and H. Maruyama, "Real-Time On-Line Unconstrained Handwriting Recognition using Statistical Methods", *Proc. of the Int'l. Conf. on Acoustics, Speech and Signal Processing (ICASSP '95)*, Detroit, Michigan, U. S. A., May 8-12, 1995, v. 4, pp. 2619-2622.

[6] Pitrelli, J. F., and E. H. Ratzlaff, "Quantifying the Contribution of Language Modeling to Writer-Independent On-Line Handwriting Recognition", *Proc. of the Seventh Int'l. Workshop on Frontiers in Handwriting Recognition (IWFHR-7)*, Amsterdam, The Netherlands, September 11-13, 2000, pp. 383-392.

[7] Subrahmonia, J., K. Nathan, and M. Perrone, "Writer Dependent Recognition of On-Line Unconstrained Handwriting", *Proc. of ICASSP '96*, Atlanta, Georgia, U. S. A., May 7-11, 1996, v. 6, pp. 3478-3481.