Generate Hidden Markov Model

NAME

generate_hmm - generate hmm model file for a training file in tsv format

SYNOPSIS

generate_hmm [training_file] [output_file]

DESCRIPTION

generate_hmm reads in a tsv training file that contains words and their corresponding part-of-speech tags. It creates a HMM from it by computing the maximum likelihood estimation for all occuring bigrams and writes the properties of the trained HMM into a new file.
The sentences in the training file have to be marked with additional <BOS> and <EOS> tags (see section **Data Format** for information about the structure of training files). The generated model can later be used with a test file to compute the forward probability of sentences or the most probable hidden state path (see **compute_forward_prob** and **compute_viterbi** respectively).
There are three different smoothing options when creating the HMM object. If no additional smoothing parameter is given, the transitions and initial probabilities will be computed via standard add-one/Laplace smoothing. This is a very simple smoothing method and it is not the best option because it attributes too much probability mass to unseen events - there are better smoothing algorithms such as Good-Turing and Witten-Bell. As an alternative and second option, the desired smoothing summand (which is one in Laplace smoothing) can be specified manually and given as a HMM parameter. This is the best implemented option. As a third possibility, smoothing can be set to false to create an unsmoothed model. Although the forward and viterbi algorithm will work with these models as well, it is not recommended to use them because they will not yield very good results.

EXAMPLE

"bin/generate_hmm data/unicorn_train.tsv models/unicorn.hmm" uses an example training file and saves the generated model in the /models folder.

Compute Forward Probability

NAME

compute_forward_prob - compute forward probability for a text file

SYNOPSIS

compute_forward_prob [hmm] [textfile]

DESCRIPTION

compute_forward_prob computes the forward probability for a specified text file by loading a generated Hidden Markov Model and calling the forward algorithm for every single sentence in the text file. The programm prints the computed probabilities to the terminal, but the function get_forward_probs() also returns a double vector of forward probabilities with one entry for every sentence (in case the results are needed for further computations).

The loaded model can be generated by **generate_hmm** from a tsv training file (see section **Data Format** for information about the structure of training files). The model can either be unsmoothed, add-one smoothed or summand-smoothed (see **generate_hmm** for further explanations about smoothing options).

If a single zero probability occurs in the forward calculation, the resulting probability of the complete sentence will be zero. In add-one smoothed or summand- smoothed models, this could only happen if a word that was not seen in the training data occured in the test data. To ensure that no zero probability is used in such a case, the defined constant UNSEENWORDPROB is used for calculation instead. However, in computations with unsmoothed models there are even more possible zero probabilities because even for seen words, there can be no transition. To avoid zero probabilities in such a case, the constant UNSEENTRANSITIONPROB is used for calculation instead. Thus, it is possible to get a probability for every sentence with unsmoothed models as well, but it is recommended to use the smoothed models for better results.

It is possible to activate **debugging** by setting the debugging parameter to true when creating the HMM object (is set to false by default).

EXAMPLE

"bin/compute_forward_prob models/unicorn_summand_smoothed.hmm data/unicorn_test.txt" uses a summand-smoothed model generated from the file 'unicorn_train.tsv' and computes the forward probabilitiy of an example test sentence.

Compute Best Path (Viterbi)

NAME

compute_viterbi - compute best path (sequence of part-of-speech tags) for a text file

SYNOPSIS

compute_viterbi [hmm] [textfile]

DESCRIPTION

compute_viterbi computes the best path of hidden states (part-of-speech tags) for a specified text file by loading a generated Hidden Markov Model and calling the viterbi algorithm for a sequence of observations in the text file. The programm prints each word in the observation and its corresponding part-of-speech tag to the terminal, but the function get_most_likely_seq() also returns a string of the most probable part-of-speech tag sequence (in case the results are needed for further computations).

The loaded model can be generated by **generate_hmm** from a tsv training file (see section **Data Format** for information about the structure of training files). The model can either be unsmoothed, add-one smoothed or summand-smoothed (see **generate_hmm** for further explanations about smoothing options).

If a single zero probability occurs in the viterbi calculation, it will compromise the validity of all following predicted tags. In such a case, all temporary maxima from this point onwards will be zero, and the program will predict the pos-tag corresponding to the last maximum for all remaining words in the text file. In add-one smoothed or summand-smoothed models, this could only happen if a word that was not seen in the training data occured in the test data. To ensure that no zero probability is used in such a case, the defined constant UNSEENWORDPROB is used for calculation instead. However, in computations with unsmoothed models there are even more possible zero probabilities because even for seen words, there can be no transition. To avoid zero probabilities in such a case, the constant UNSEENTRANSITIONPROB is used for calculation instead. Thus, it is possible to get comparatively valid tags for unsmoothed models as well, but it is recommended to use the smoothed models for more accurate tag predictions.

It is possible to activate **debugging** by setting the debugging parameter to true when creating the HMM object (is set to false by default).

EXAMPLE

"bin/compute_viterbi models/unicorn_summand_smoothed.hmm data/unicorn_test.txt" uses a summand-smoothed model generated from the file 'unicorn_train.tsv' and computes the most probable hidden state sequence for an example test sentence.

Data Format

TRAINING FILES

The **training file** has to contain **one word** and its corresponding **pos tag per line**, seperated by **tabulators** or **spaces**. Each beginning and end of a sentence in the data must be marked with an additional <BOS> <BOS> and <EOS> <EOS> pair also seperated by spaces or tabulators.
The training files are located in /data.

HMM FILES

(Only relevant if a hand-written model shall be read in)
The **model files** have to match the following syntax and order:

1. **Metadata section:** List a number sign, the number of states, the number of observations and another number sign, each of them seperated by a tab, in the following formatting: "# states observations #", e.g.: "# 9 12 #".

2. List the **transitions** including **transition probabilities**, each of them seperated by a tab, in the following formatting: "state state probability", e.g.: "0 0 0.3" and use a line of tildes (~~~) to end the section.

3. List the **states** and their **part-of-speech tag representations** seperated by a tab in the following formatting: "state tag", e.g.: "1 NN" and use a line of tildes (~~~) to end the section.

4. List the **observed bigrams** and their **observation probabilities**, each of them seperated by a tab, in the following formatting: "observation observation probability", e.g.: "the otter 0.2" and use a line of tildes (~~~) to end the section.

5. List the **initial probability** for a state seperated by a tab in the following formatting: "state probability", e.g.: "1 0.9".

Give only ONE transition, observation, initial probability or state-string representation per line. The tilde cannot occur as observation because it is used as separator. In addition, make sure that all states are natural numbers.
Generated model files are located in /models.

TEST FILES

The **test file** has to contain **one word per line.** Each beginning and end of a sentence in the data must be marked with an additional <BOS> and <EOS> tag.
The test files are located in /data.

Enclosed Data Files

TRAINING FILES

complete_wsj_train.tsv – sections 02-21 from the Wall Street Journal (1,029,693 lines)

small_wsj_train.tsv – about 5 percent of the above-named data (50,446 lines)

unicorn_train.tsv – small hand-written demo file (13 lines)

TEST FILES

complete_wsj_test.tsv – section 23 from the Wall Street Journal (61,517 lines)

small_wsj_test.tsv – a few sentences from the above-named data (113 lines)

unicorn_test .tsv – small hand-written demo file (7 lines)

HMMS

complete_wsj_addone_smoothed.hmm - add-one smoothed model generated from training file complete_wsj_train.tsv

complete_wsj_summand_smoothed.hmm – summand-smoothed model with specified summand 0.75 generated from training file complete_wsj_train.tsv

complete_wsj_unsmoothed.hmm – unsmoothed model generated from training file complete_wsj_train.tsv

small_wsj_addone_smoothed.hmm - add-one smoothed model generated from training file small_wsj_train.tsv

small_wsj_summand_smoothed.hmm - summand-smoothed model with specified summand 0.75 generated from training file small_wsj_train.tsv

small_wsj_unsmoothed.hmm - unsmoothed model generated from training file small_wsj_train.tsv

unicorn_addone_smoothed.hmm - add-one smoothed model generated from training file unicorn_train.tsv

unicorn_summand_smoothed.hmm - summand-smoothed model with specified summand 0.75 generated from training file unicorn_train.tsv

unicorn_unsmoothed.hmm - unsmoothed model generated from training file unicorn_train.tsv

The training and test files are located in /data and the HMMs are located in /models.

Sources

[1] Daniel Jurafsky, James H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition,* Pearson Education, Upper Saddle River, NJ

[2] **Parse CSV File With Boost Tokenizer In C++**
http://mybyteofcode.blogspot.de/2010/02/parse-csv-file-with-boost-tokenizer-in.html