Winter Semester 2009/10
Prof. Dr. Detmar Meurers
Holger Wunsch
Kilian Evang

ISCL, Universität Tübingen
Computational Approaches
to Functional Elements

Lab Session November 25, 2009: Language Models for predicting articles and prepositions in English

1 The SRI language modeling toolkit

SRILM is a toolkit for creating of and experimenting with statistical language models. It is installed at

`/afs/sfs/lehre/dm/ws-09-10-functional-elements/tools/SRI-LMTK/bin/i686`

To avoid having to type in this long path every time you start the tools, it is wise to update your path settings in `.profile` accordingly.

SRILM comes with a short manual that describes the most important components of the system:

`/afs/sfs/lehre/dm/ws-09-10-functional-elements/tools/SRI-LMTK/doc/paper.pdf`

You can find comprehensive information about the system on its homepage at

`http://www.speech.sri.com/projects/srilm/`

2 Two simple language models

To get a feeling of how SRILM works, create two simple language models, each on the basis of only one sentence.

1. Create a file `simple-1.txt` containing this sentence on one line:

   ```
   statistical language modeling is a science
   ```

   Do not include a period at the end of the sentence.

2. Create a language model by calling

   ```
   ngram-count -text simple-1.txt -lm simple-1.lm
   ```

   (A number of warnings will appear – why?)

3. Apply the language model to itself by calling

   ```
   ngram -lm simple-1.lm -ppl simple-1.txt -debug 3
   ```

   What does the output mean? Why are the results like this?

4. Create a file `simple-2.txt` containing this (a bit more complex) sentence:

   ```
   statistical language modeling is the science ( and often art ) of building
   models that estimate the prior probabilities of word strings
   ```

   (again, put the complete the sentence on one line)

5. Create the corresponding language model and apply it to itself.

   Discuss and interpret the results.

6. Experiment with the language model by modifying both training and test data. Try to predict the results before actually running the program, and then check whether the output matches your expectations.

3 Predicting articles and prepositions using a language model of English

In this project we will use a language model created on the basis of the BNC Sampler corpus to predict articles and prepositions. We will use the corpus data both for training and for testing. To be able to cover the whole corpus without testing on known data, we will employ ten-fold cross-validation, using in each fold 90% of the corpus as training data, and the remaining 10% as test data.

1. **Create a ten-fold partition of the BNC Sampler corpus.**

   In the directory

   `/afs/sfs/lehre/dm/ws-09-10-functional-elements/tools/10-fold`

   you will find `Preparer.class` (along with helper classes and source code). The program takes the BNC as its input and creates a ten-fold partition as described above. It must be called as follows:

   ```
   java Preparer bnc-samp.tt <output directory>
   ```

   The Preparer will write the ten partitions in ten separate subdirectories of `output directory`.

   Each partition consists of three files:

   - `train`, which contains the training data, one sentence per line[1].
   - `test-art`, the test data source file with articles masked out (one sentence per line).
   - `test-prep`, the test data source file with prepositions masked out.

   In the test data source files, the articles and prepositions are not yet removed, but rather marked as follows: `-*-MASKED-*-the`.

   Normalization of a and an: In both the training data and the test data, the indefinite articles *a* and *an* are merged to an artificial general indefinite article *a/an*, to reduce the complexity of the prediction task.

2. **Create a language model of the training data**

   For each fold, create a language model of the training data using the `ngram-count` tool as in task 2.2.

[1]Since sentence boundaries are not explicitly marked in `bnc-samp.tt`, we use a heuristic to start a new sentence after each punctation character occurring as a separate token labeled `YSTP, YEX`, or `YQUE`

3. **Write a program that predicts articles and prepositions using the language model**

   The program should perform the following tasks:

   - For each fold and each sentence in the test source data, create a set of sentences where each occurrence of a masked element is replaced with all elements to be tested. For articles, these elements are *the* and *a/an*. For prepositions, use the ten most frequent prepositions in the BNC Sampler.

     Example: From the sentence

     `these gains could be balanced out and even reversed in -*-MASKED-*-the remaining states .`

     the following set of test sentences should be created:

     (a) `these gains could be balanced out and even reversed in` ***`the`*** `remaining states .`

     (b) `these gains could be balanced out and even reversed in` ***`a/an`*** `remaining states .`
   - Run the `ngram` tool on this set of test sentences and determine the perplexity.
   - Select the sentence with the lowest perplexity as the one containing the article or preposition predicted by the language model.
   - Evaluate this choice by comparing the predicted sentence to the original sentence.

     Example: Assume the language model selected

     `these gains could be balanced out and even reversed in` ***`the`*** `remaining states .`

     as the sentence with the lowest perplexity. The correct solution can be retrieved from the original source sentence

     `these gains could be balanced out and even reversed in -*-MASKED-*-the remaining states .`

     by removing the `-*-MASKED-*-` string and comparing this to the output of the language model. In the example, both sentences are equal, thus, the prediction would be correct.

4. **Discuss the results.**