Applied Textmining

Second Assignment

Please update the code from the GIT repository at *https://github.com/dimalabs/applied-textmining* by pulling the latest version of the project. to accomplish that simply type the following line into *GIT Bash*:

```
git pull
```

1 Language Model

Have a look at *de.tuberlin.dima.textmining.assignment2.LanguageModel*. This is the interface you will have to implement for your language model. At the beginning the *train* method will be called by the evaluation harness. It receives the training text corpus as a parameter. All counting and parameter estimation has to be performed in this function. The actuall inference will be done in *sentenceProbability()* which takes a sentence (collection of words) as inputs and has to return the probability of this sentence according to the estimated language model. In order to implement this function it makes sense to introduce a *getWordProbability()* method that computes the probability of a word given its history (the preceding words relevant for the probability computation - in the the case of bi-grams for example only the last word). In *de.tuberlin.dima.textmining.assignment2.UnigramModel* we already implemented a plain and simple empirical unigram model as a showcase.

Run our testcase de.tuberlin.dima.textmining.RunAssignment2 to test the UnigramModel and watch the output.

1.1 Implement a BiGram or higher model (n-gram with $n >= 2$)

Implement the methods in *de.tuberlin.dima.textmining.assignment2.NGramModel* for a bigram or higher. Note that some of the functions from the unigram model can be re-used - others not. For the details of counting n-grams please refer to the slides and remember the definition of conditional probabilities. This paper by Chen and Goodman also gives a decent introduction to n-gram language models: `http://www.cs.berkeley.edu/~klein/cs294-5/chen_goodman.pdf`. Please also note that you will have to introduce additional start tokens for bigrams and higher ngrams. for example take the example sentence `I was here.`. For bigrams this should become `<s>,I,was,here,</s>`. For trigrams `<s>,<s>,I,was,here,</s>`, and so on.

1.2 Implement at least one smoothing or interpolation technique

Implement at least one smoothing or interpolation technique to deal with unseen n-grams in the evaluation corpus. Ambitious students might also want to make use of held-out data to tune some parameters *;)*. For this task, please implement the methods in *de.tuberlin.dima.textmining.assignment2.SmoothNGramModel*. For a nice review of smoothing techniques you can also refer to this paper: `http://www.cs.berkeley.edu/~klein/cs294-5/chen_goodman.pdf`

Evaluation

As you can see we only provided you with some toy data to test your model. Unfortunately we have to do this due to copyright issues. Your models will be evaluated on a real corpus however.

Training

Several data objects are loaded by the harness. First, it loads about 1M words of WSJ text (from the Penn treebank). These sentences have been "speechified", for example translating "$" to "dollars", and tokenized for you. The WSJ data is split into training data (80%), validation (held-out) data (10%), and test data (10%).

In addition to the WSJ text, the harness loads a set of speech recognition problems (from the HUB data set). Each HUB problem consists of a set of candidate transcriptions of a given spoken sentence. For this assignment, the candidate list always includes the correct transcription and never includes words unseen in the WSJ training data. Each candidate transcription is accompanied by a pre-computed acoustic score, which represents the degree to which an acoustic model matched that transcription. These lists are stored in SpeechNBestList objects.

Once all the WSJ data and HUB lists are loaded, a language model is built from the WSJ training sentences (the validation sentences are ignored entirely by the provided baseline language model, but may be used by your implementations for tuning). Then, several tests are run using the resulting language model.

Evaluation

Each language model is tested in two ways. First, the harness calculates the perplexity of the WSJ test sentences. In the WSJ test data, there will be unknown words. Your language models should treat all entirely unseen words as if they were a single UNK token. This means that, for example, a good unigram model will actually assign a larger probability to each unknown word than to a known but rare word - this is because the aggregate probability of the UNK event is large, even though each specific unknown word itself may be rare. To emphasize, your model's WSJ perplexity score will not strictly speaking be

the perplexity of the extact test sentences, but the UNKed test sentences (a lower number).

Deadline

Please upload your solution as a patch in the ISIS system until 30th of October 2011. We will accept late submissions since this assignment was posted late as well. Since we want to show the results of your models in class however, we kindly urge you to submit by the deadline.