Final project

NLP

Due by 14.08.2014, 23:55

For the final project, you are to build a fully-automatic translation pipeline, involving some existing off-the-shelf components along with your own algorithms. The system must translate Biblical Hebrew sentences into English.

The Biblical Hebrew input is transcribed in English letters. This corpus is already tokenized. For your convenience, we include another file with the Hebrew original, only not tokenized. You may only use it as a reference; the translation is made on the tokenized transcript.

All project files can be downloaded from here:
http://www.cs.tau.ac.il/~lenadank/nlp/nlp-project/

<u>Building instructions:</u>
The system should be phrase-based; it uses two log-linearly combined feature functions (language and translation). The system should be implemented as a set of scripts, with each one implementing a single component. There are two main phases: (1) training, in which the system builds its necessary models for translation; and (2) testing, in which the system applies the models to a new input sentence.

<u>Training</u>:
The input data for the training phrase consists of:

1) Bilingual parallel texts, aligned on the sentence level. In this experiment we will work with the Bible. There are two files: bible.heb and bible.eng. In both files, each line represents a single sentence. The sentences are aligned by line number.
2) Additional monolingual English text for building the language model.

The relevant components for the training phase:

1. **Cleaning**:
   You are required to develop a cleaning script that removes tokenized sentences of more than 60 tokens. This is necessary, as usually the word-alignment algorithm doesn't perform well on longer sentences. You should allow the user to control this limit of tokens using an input parameter.

2. **Word alignment**:
   Using Giza++ (https://code.google.com/p/giza-pp/)
   You will have to run Giza++ twice, once from Hebrew to English and once from English to Hebrew. The word-alignment results are to be reported in the <input name>.A3 file.

3. **Phrase alignment and phrase table**:
   You are required to develop a phrase-aligning tool, based on the technique

described in class. The full original paper, "The Alignment Template Approach to Statistical Machine Translation ", is can be found in the project folder. You can also use this presentation to learn the idea from: http://www.statmt.org/book/slides/05-phrase-based-models.pdf (slides: 8-15). After you extract all the aligned phrases from all the sentences, you should calculate a translation score for each phrase pair, which is simply modeled by the maximum likelihood (as was shown in class). The results can be stored in a file, in which each line contains a phrase pair (Hebrew=>English) with its corresponding translation score (log probability).

4. **Language model**:
   You may use your language model tool from assignment #1. You should run the tokenizer on the monolingual English text first and then run the language-model builder. The monolingual English corpus can be found in the project folder.

The output from the training phrase is therefore a phrase table and a language model.

Testing:

The input data for the testing phase is:

1) File of Hebrew sentences to be translated by the system. Each sentence should be provided in a single line.
2) Multiple files, each containing a reference English translation of every input Hebrew sentence. These files are used for evaluation, as explained below.

The relevant components for the testing phase are:

1. **Lattice generator**:
   You are required to build a tool for finding all the translated phrases from the phrase table, for an input tokenized Hebrew sentence. The input for this tool is (1) the tokenized Hebrew sentence; and (2) the phrase table. The lattice should be stored in a file, formatted as follows:

```
1-1:
<English translation>  <Translation score>
<English translation>  <Translation score>
…
1-2:
<English translation>  <Translation score>
<English translation>  <Translation score>
…

2-2:
…
```

The spans *n-n* refer to the tokens of the input Hebrew sentence. For example, the span 1-1 refers to the Hebrew token #1, the span 1-2 refers to the Hebrew phrase starting at token #1 and ending at token #2. Then, after each span, a list of all English translations and their corresponding translation score are provided.

2. **Stack decoder**:
   You are required to develop a stack decoder, as presented in class (see: http://www.statmt.org/book/slides/06-decoding.pdf)
   The input for this tool is: (1) the lattice file, as generated by the lattice generator; and (2) the language model. The output is the best translation, with its score, printed on the screen. You may want to generated some log files that monitor the decoder steps for convenience reasons.
   Every hypothesis *e* should be evaluated by the log-linear model:

$$\boxed{score(e) = \lambda_{\phi} \sum_{i=1}^{n} log\phi(\bar{f}_i|\bar{e}_i) + \lambda_{LM} logLM(e)}$$

   As you can see in this project we are only using two models:
   ϕ - translation score.
   LM – language model score.

   Comments:
   a. In ideal settings, the weights of the models λ_{ϕ},λ_{LM}, are tuned automatically based on a development set. In our project we will set these weights manually, as explained below.
   b. You may want to add a distortion (reordering) model as well. However, this is not mandatory.
   c. The decoder should follow the histogram pruning approach, that is, setting a limit *n* (*n* is a variable), on the number of hypotheses exist in every stack. When this number is exceeded, the decoder keeps the *n* hypotheses with the highest score. The number *n* should be configurable by an input parameter.

3. **Evaluation**

You are provided with the following data:

   a. bilingual and monolingual texts
   b. a development set containing Hebrew sentences aligned with two reference English translations

A week before submission, we will publish a test set, which we will used to evaluate your results.
In order to evaluate the translation results, you can use an existing Bleu calculator script (as will be published on the course website) in order to calculate a Bleu score.

4. **Experiments**

In your final project report, you are requested to report on experimental results using different weight settings for the language and translation models. Make sure to include at least the settings below.

Your report should include the following sections:

a. Results - Experimental results using different weight settings for the language and translation models. Make sure to include the following settings:
    1. $\lambda_{\phi} = 0.5, \lambda_{LM} = 0.5$
    2. $\lambda_{\phi} = 0.9, \lambda_{LM} = 0.1$
    3. $\lambda_{\phi} = 0.1, \lambda_{LM} = 0.9$

b. Analysis – in this section you should discuss the results of your system, analyze several interesting examples (error analysis) and offer ideas to improve the system, based on your observation.

c. Resources – any resource that you used and wasn't mentioned in the project description.

d. Additional methods and algorithms – any algorithm/method you've implemented that wasn't mentioned in the project description.

5. **Submission instruction:**

You must submit your project both in Moodle and on the nova server.

Moodle submission: you should submit a zipped file containing:

1. All the programs you develop for the project, including all sources and external packages.
   We should be able to compile (if necessary) and run each program on Nova command line using a single command.
2. Execution instructions.txt file – this file should contain the execution commands for each program you submit. A template is provided with the project.
3. Report.pdf – this report discussed in section 6.
4. partners.txt – same file as in previous assignments. A template is provided with the project.

Submission on Nova:

You should create a folder on the Nova server. This folder will contain all your code and relevant scripts to run your programs. The full path to this folder should be written in "execution instructions.txt" and you should grant it a+r permission so we'll be able to read its content.