11791 Homework1 Report

Yuchen Tian Andrew ID:yuchent

October 17, 2012

1 Method

1.1 Hidden Markov Model

Hidden Markov Model is a model that is widely used in natural language processing.

The intuition of applying HMM in NLP, is inspired by the following question: Given a word w_i in a sequence, how can we assign it a tag? We want to estimate probability $P(t|w)$.

Following Bayes Rule, we have:

$$P(t|w) = \frac{P(w|t)P(t)}{P(w)}$$

But the above model is clearly flawed, in the way that it doesn't take the position of the word into account. If we consider that every sentence is a Bayes net, that the tag of a word in a sentence is decide not only by the word but also conditioned on the the tag of its previous word, then we can generalize the above equation into:

$$P(t_i|w_i) \propto P(w_i|t_i)P(t_i|t_{i-1})$$

The above equation can be interpreted into Markov Model and because part of the states cannot be observed, it is called Hidden Markov Model.

1.2 Training HMM

If we have sufficient training data, in this case, many manually tagged sentences, then we can estimate $P(t_i|t_{i-1})$ and $P(w_i|t_i)$ simply using MLE estimation, by counting their occurrences, but if we don't have enough data, then we should using EM algorithm to train the model.

1.3 Using HMM in tagging

By applying Viterbi algorithm we can compute a sequence of tags that maximize $\prod_i^n P(w_i|t_i)P(t_i|t_{i-1})$, where n is the length of this sentence.

1.4 How is it related to our homework

There is a widely used dataset called GENETAG, which is a collection of sentences that is tagged as follows:

On _TAG the_TAG other_TAG hand_TAG factor_GENE1 IX_GENE1 activity_TAG is_TAG decreased_TAG in_TAG coumarin_TAG treatment_TAG with_TAG factor_GENE2 IX_GENE2 antigen_TAG remaining _TAG normal_TAG

Using this dataset as training examples, and using HMM, when given a sentence, we can predicted the most likely tag sequence, like **TAG-TAG-GENE...**, then the text that is tagged as GENE is output.

2 Tools

2.1 LingPipe

LingPipe is java toolkit for NLP. We used it because it provide very convenient API to train HMM Model. Besides, it provide a specific parser called GeneTagParser to handle corpus like GENETAG, which greatly simplies our work of data cleaning.

Following the example code [1]

```
public static void main(String[] args) throws IOException{

  TokenizerFactory factory
  = IndoEuropeanTokenizerFactory.INSTANCE;

  HmmCharLmEstimator hmmEstimator
  = new HmmCharLmEstimator(MAX_N_GRAM,
        NUM_CHARS, LM_INTERPOLATION);

  CharLmHmmChunker chunkerEstimator
  = new CharLmHmmChunker(factory,hmmEstimator);

  com.aliasi.corpus.parsers.GeneTagParser parser
  = new com.aliasi.corpus.parsers.GeneTagParser();

  parser.setHandler(chunkerEstimator);
  parser.parse(corpusFile);

}
```

What is worth noting here is that the handler, **chunkerEstimator** is composed of two components: **HmmCharLmEstimator** and **TokenizerFactory**, and the former is used to train the model, the later is used to generate tokenized output.

[1] http://alias-i.com/lingpipe/demos/tutorial/ne/src/TrainGeneTag.java

3 Pipeline Architecture

3.1 Overview

One of the most useful feature of UIMA is that it is highly pipeline-oriented. The workflow of our gene-tagging program can be shown in the following figure:

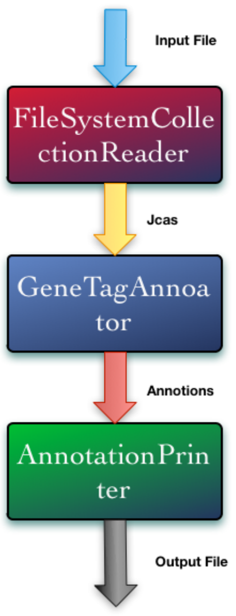

Figure 1: Pipeline of Genetagging Task

We borrow code from the UIMA example, and modified it according to requirement of the task, like FileCollectionReader and AnnotationPrinter. Here, the GeneTagAnnotator plays a critical role and contains the logic behind this task here, so we will focus it.

3.2 Type System

Here we define a our annotation type, GeneTag, which has the following feature:

Feature	Type	Descripition
Location	String	This feature specifies the line number in which a gene tag appears
Nbegin	Integer	This feature specifies the begin offset of a tag within a sentence
Nend	Integer	This feature specifies the end offset of a tag within a sentence

The reasoning behind this design is related to format of our output:

P01985924A0554|188 200|pol II delta 4/7

the output ask us to record the line number and the offset within a sentence, but the default **begin** and **end** attribute of Annotation is associated with document, so we design a separate feature to hold the offset within a sentence.

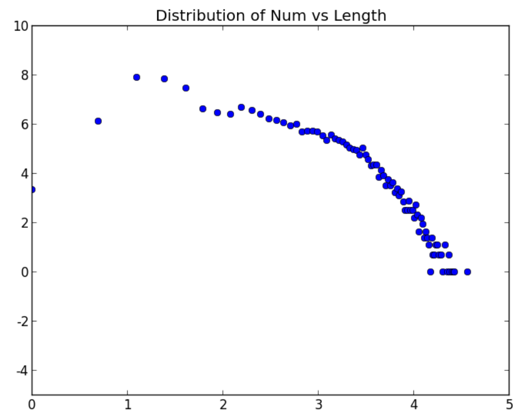

Figure 2: Distribution of Gene Pharse

3.3 Gene Annotator

The algorithm we used for Gene Annotator is as follows:

```
1  initialization;
2  import models;;
3  document_position = 0;;
4  for each sentence in the document do
5      tagging the sentence;
6      get line number;
7      for each tag do
8          if tag == Gene then
9              calculate within sentence offset;
10             calculate within document offset;
11             create annotation combining sentence and document offsets
               and line number;
12             addToIndexes;
13         end
14     end
15     update document_position
16 end
```

Algorithm 1: GeneAnnotator

What is worth note here is that we keep the record of the current position of the document, so every time we only add sentence offsets to this value, then we get the offsets within document. Also worth noting here is that we import the train model from a **File**, we do this for two reasons: First, training model takes very long time, we don't need to do it every time once we trained it. Second the GENETAG dataset is not available at that time, we use the trained model file that LingPipe provided as a demo on their website [2]

4 Simple Optimization

By observing the sample output, we find a very interesting distribution, shown in Figure 2:

This plot is the sentence length over the count distribution put in log-log scale. This clearly reminisce the power-law distribution. From the plot, we can find when $x = 3$, means when the gene length goes over to 20, the count decrease rapidly.

[2]http://alias-i.com/lingpipe/demos/models/ne-en-bio-genetag.HmmChunker

So we can come up with a very simple trick here: if the GENE length we detected is longer, it is more likely to be misclassified, because most genes sequences have very small length. So we simply discard those long sequences. Here we set the threshold to 30.

5 Performance Evaluation

We test the performance of our program using three popular metrics:

- Precision: the precision against sample output is 0.724.
- Recall:the recall here is 0.768.
- F-score:F-score is 0.74

As we mention before that the short gene sequence consists most of the data, so we test our data on its ability to detect short gene sequences. In our case we set the threshold to 30.

- Precision: the precision against sample output is 0.724.
- Recall:the recall here is 0.876.
- F-score:F-score is 0.793

The recall has greatly improved, which means it has better coverage of short genes. The result also tells us that the HMM can model very short sequences well, but when it comes to longer sequence, the algorithm becomes less accurate.