FNLP: Lab Session 3

Hidden Markov Models - Construction and Use

Aim

The aims of this lab session are to 1) familiarize the students with the POS tagged corpora and tag sets available in NLTK 2) introduce the HMM tagger available in NLTK, how to train, tag and evaluate with this tagger 3) build the transition and emission models needed to train a HMM tagger. Successful completion of this lab is important as the second assignment for FNLP builds on some of the concepts and methods that are introduced here. By the end of this lab session, you should be able to:

- Access corpora tagged with part-of-speech information.
- Train an HMM Tagger, use it to tag sentences and evaluate it on a test set.
- Compute the transition and emission probabilities.

1 Running NLTK and Python Help

1.1 Running NLTK

NLTK is a Python module, and therefore must be run from within Python. To get started on DICE, type the following in a terminal window:

```
$: python
>>> import nltk
```

1.2 Python Help

Python contains an inbuilt help module that runs in an interactive mode. To run the interactive help, type:

```
>>> help()
```

To exit, press CTRL-d.

If you know the name of the module that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>)
```

To exit, type "q" (for "quit").

If you know the name of the module **and** the method that you want to get help on, type:

```
>>> import <module_name>
>>> help(<module_name>.<method_name>)
```

To exit, type "q" (for "quit").

2 Introduction

Before continuing with this lab sheet, please download a copy of the lab template (**lab3.py**) for this lab from the FNLP course website. This template contains code that you should use as a starting point when attempting the exercises for this lab.

In this lab we will look at how to train and use the Hidden Markov Model (HMM) tagger provided by NLTK.

```
>>> help(nltk.tag.hmm.HiddenMarkovModelTagger)
>>> help(nltk.tag.hmm.HiddenMarkovModelTagger.train)
```

The task for which we will train the HMM tagger will be part-of-speech (POS) tagging. We will also take a closer look at the components of the HMM model.

3 Corpora tagged with part-of-speech information

NLTK provides corpora annotated with part-of-speech (POS) information and some tools to access this information. The Penn Treebank tagset is commonly used for annotating English sentences. We can inspect this tagset in the following way:

```
>>> nltk.help.upenn_tagset()
```

Loading `lab3.py` will show you this, as well as some tagged sentences from the corpus we will be working with.

The Brown corpus provided with NLTK is also tagged with POS information, although the tagset is slightly different than the Penn Treebank tagset. Information about the Brown corpus tagset can be found here:
http://www.scs.leeds.ac.uk/ccalas/tagsets/brown.html

We can retrieve the tagged sentences in the Brown corpus by calling the *tagged_sents*() function and looking at an annotated sentence:

```
>>> tagged_sentences = brown.tagged_sents(categories= 'news')
>>> print tagged_sentences[29]
```

Sometimes it is useful to use a coarser label set in order to avoid data sparsity or to allow a mapping between the POS labels for different languages. The Universal tagset was designed to be applicable for all languages:
https://code.google.com/p/universalpostags/.
There are mappings between the POS tagset of several languages and the Universal tagset. We can access the Universal tags for the Brown corpus sentences by changing the tagset argument:

```
>>> tagged_sentences_universal = \
                    brown.tagged_sents(categories= 'news', tagset='universal')
>>> print tagged_sentences_universal[29]
```

Exercise 1:

In this exercise we will compute a Frequency Distribution over tags that appear in the Brown corpus. The template of the function that you have to implement takes two parameters: one is the category of the text and the other is the tagset name. You are given the code to retrieve the list of (word, tag) tuples from the brown corpus corresponding to the given category and tagset.

a. Convert the list of word+tag pairs to a list of tags

b. Using the list of tags to compute a frequency distribution over the tags, useing `FreqDist()`

c. Compute the total number of tags in the Frequency Distribution

d. Retrieve the top 10 most frequent tags

Uncomment the test code. What do you observe by comparing the number of tags and most frequent tags across different genres? What happens when you change the tagset?

4 Training and Evaluating an HMM Tagger

NLTK provides a module for training a Hidden Markov Model for sequence tagging.

```
>>> help(nltk.tag.hmm.HiddenMarkovModelTagger)
```

We can train the HMM for POS tagging given a labelled dataset. In Section 1 of this lab we learned how to access the labelled sentences of the Brown corpus. We will use this dataset to study the effect of the size of the training corpus on the accuracy of the tagger.

Exercise 2:

In this exercise we will train a HMM tagger on a training set and evaluate it on a test set. The template of the function that you have to implement takes two parameters: a sentence to be tagged and the size of the training corpus in number of sentences. You are given the code that creates the training and test datasets from the tagged sentences in the Brown corpus.

a. Train a Hidden Markov Model tagger on the training dataset. Refer to `help(nltk.tag.hmm.HiddenMarkovModelTagger.train)` if necessary.

b. Use the trained model to tag the sentence

c. Use the trained model to evaluate the tagger on the test dataset

Uncomment the test code. Look at the tagged sentence and the accuracy of the tagger. How does the size of the training set affect the accuracy?

5 Computing the Transition and Emission Probabilities

In the previous exercise we learned how to train and evaluate an HMM tagger. We have used the HMM tagger as a black box and have seen how the training data affects the accuracy of the tagger. In order to get a better understanding of the HMM we will look at the two components of this model:

- The transition model
- The emission model

The *transition model* estimates $P(tag_{i+1}|tag_i)$, the probability of a POS tag at position $i+1$ given the previous tag (at position i). The *emission model* estimates $P(word|tag)$, the probability of the observed word given a tag.

Given the above definitions, we will need to learn a Conditional Probability Distribution for each of the models.

```
>>> help(nltk.probability.ConditionalProbDist)
```

Exercise 3:

In this exercise we will estimate the emission model. In order to compute the Conditional Probability Distribution of $P(word|tag)$ we first have to compute the Conditional Frequency Distribution of a word given a tag.

```
>>> help(nltk.probability.ConditionalFreqDist)
>>> help(nltk.probability.ConditionalProbDist)
```

The constructor of the ConditionalFreqDist class takes as input a list of tuples, each tuple consisting of a condition and an observation. For the emission model, the conditions are tags and the observations are the words. The template of the function that you have to implement takes as argument the list of tagged words from the Brown corpus.

a. Build the dataset to be passed to the `ConditionalFreqDist()` constructor. Words should be lowercased. Each item of data should be a tuple of *tag* (a condition) and *word* (an observation).

b. Compute the Conditional Frequency Distribution of words given tags.

c. Return the top 10 most frequent words given the tag **NN**.

d. Compute the Conditional Probability Distribution for the above Conditional Frequency Distribution. Use the `MLEProbDist` estimator when calling the `ConditionalProbDist` constructor.

e. Compute the probabilities

$P(year|NN)$
$P(year|DT)$

Uncomment the test code. Look at the estimated probabilities. Why is $P(year|DT) = 0$? What is `emission_FD['NN']['year']`? Contrast that with `emission_FD['DT']['year']`?

Whatare the problems with having zero probabilities and what can be done to avoid this?

Exercise 4:

In this exercise we will estimate the transition model. In order to compute the Conditional Probability Distribution of $P(tag_{i+1}|tag_i)$ we first have to compute the Conditional Frequency Distribution of a tag at position $i+1$ given the previous tag.

```
>>> help(nltk.probability.ConditionalFreqDist)
>>> help(nltk.probability.ConditionalProbDist)
```

The constructor of the `ConditionalFreqDist` class takes as input a list of tuples, each tuple consisting of a condition and an observation. For the transition model, the conditions are tags at position i and the observations are tags at position $i+1$. The template of the function that you have to implement takes as argument the list of tagged sentences from the Brown corpus.

a. Build the dataset to be passed to the `ConditionalFreqDist()` constructor. Each item in your data should be a pair of condition and observation: (tag_i, tag_{i+1})

b. Compute the Conditional Frequency Distribution of a tag at position $i+1$ given the previous tag.

c. Compute the Conditional Probability Distribution for the above Conditional Frequency Distribution. Use the `MLEProbDist` estimator when calling the `ConditionalProbDist` constructor.

d. Compute the probabilities

$P(NN|VBD)$
$P(NN|DT)$

Uncomment the test code. Are the results what you would expect? The sequence **DT NN** seems very probable. How will this affect the tagging of real, longer, sequences?

6 Going further

Modify your code for exercise 3 to use a different estimator, to introduce some smoothing, and compare the results with the original.

In exercise 4 we didn't do anything about the boundaries. Modify your code for exercise 4 to use `<s>` at the beginning of every sentence and `</s>` at the end. Explore the resulting conditional probabilities. What is the most likely tag at the beginning of a sentence? At the end?