Sequence Labeling

Lab Session*

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 4) implement the Viterbi algorithm.

1 Introduction

Before continuing with this lab sheet, please download a copy of the lab template (lab-seq.py) for this lab from the course moodle. This template contains code that you should use as a starting point when attempting the exercises for the first part of this lab. First, 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, and implement the Viterbi decoding algorithm.

2 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 `lab-seq.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: `http://universaldependencies.org/u/pos/`. 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]
```

*Based on Goldwater and Thompson, 2016.

Exercice 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. Use the list of tags to compute a frequency distribution over the tags, using 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?

3 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 labeled dataset. In Section 2 of this lab we learned how to access the labeled 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?

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

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']`?

What are 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 $P(NN|VBD)$ and $P(NN|DT)$

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?

5 Implementing the Viterbi Algorithm

We will start by simulating the Viterbi algorithm on a toy example to make sure you understand how it works.

Consider a simple HMM POS tagger with only five tags (plus the beginning and end of sentence markers, `<s>` and `</s>`). The transition probabilities for this HMM are given by the table on the left below, where cell $[i, j]$ is the probability of transitioning from state i to j (i.e., $P(state_j|state_i)$). A subset of the output probabilities are given by the table on the right, where cell $[i, j]$ is the probability of state i outputting word j (i.e., $P(word_j|state_i)$). We assume there are other possible output words not shown in the table, and that the `<s>` and `</s>` states output `<s>` and `</s>` words respectively, with probability 1.

	CD	PRP	NN	VB	VBD	</s>
<s>	.5	.2	0	.3	0	0
CD	.2	0	.3	.2	.2	.1
PRP	.1	.1	0	.3	.4	.1
NN	.05	.15	.2	.25	.3	.05
VB	0	.2	.6	0	0	.2
VBD	0	.1	.6	0	0	.3

	one	cat	dog	bit	...
CD	.1	0	0	0	
PRP	.02	0	0	0	
NN	.05	.03	.04	.007	
VB	0	0	.03	0	
VBD	0	.0	0	.06	

a. Using the HMM probability matrices, compute $P(\mathbf{w}, \mathbf{t})$ (the joint probability of words and tags) for the sentence $\mathbf{w}$ = `<s> one dog bit </s>` with tags $\mathbf{t}$ = `<s> CD NN NN </s>`

b. Hand-simulate the Viterbi algorithm in order to compute highest probability tag sequence $\mathbf{t}'$ for the given sentence, and the joint probability $P(\mathbf{t}', \mathbf{w})$, without enumerating all possible tag sequences. That is, fill in the cells in the following table, where cell $[j, i]$ should contain the Viterbi value for state j at time i, and you should also use backpointers to keep track of the best path. The rows of the table are already labeled with the different states, and the columns are already labeled with the observations at each time step.
Hint: For this particular HMM, a lot of the cells will have zeros in them. Try to work out ahead of time which these are, so you only need to do the Viterbi computations for the other cells.

	<s>	one	dog	bit	</s>
<s>					
CD					
PRP					
NN					
VB					
VBD					
</s>					

c. As you've seen, Viterbi probabilities get very small very fast. In practice, the algorithm is normally implemented using log probabilities to avoid underflow. The value in each cell is now a negative log probability (or cost), and we end up computing $-\log P(\mathbf{w}, \mathbf{t})$. Work out what the equations need to be in this version of the algorithm. That is, what do we compute to get the value in cell $[j, i]$?

Exercise 5

In this exercise we will implement the Viterbi algorithm.

a. Implement the Viterbi algorithm. The `viterbi` function takes a sentence to be tagged, a list of all possible tags and the parameters of the HMM built in the previous exercises, and returns the tag sequence.

Uncomment the test code. Compare the obtained tag sequence with the manual annotation provided in the corpus.