Natural Language Processing - Exercise 2

Due: 30.11.2022 23:55

Please submit a single zip file. The zip file should contain your code files, a README txt file and a single pdf file for the answers to the questions

1. (10 pts) Consider this (toy) biological setup:
   A cell can be in one of two states - H, for high GC-content, and L for low GC. On each time step the cell produces one nucleotide, A,C,T or G, and might also change its state. The probability of changing from state H to L is 0.5, and from state L to H is 0.4.
   In state H the probabilities for producing nucleotides are 0.2 for A, 0.3 for C, 0.3 for G and 0.2 for T. In L the probabilities are 0.3 for A, 0.2 for C, 0.2 for G and 0.3 for T.
   Consider the nucleotide sequence $S = ACCGTGCA$. Use the Viterbi algorithm to find the best state-sequence and calculate the probability of S given this state-sequence. Assume the previous state before S was H.

2. (10 pts) In class we saw the trigram HMM model and the corresponding Viterbi algorithm. We will now make two main changes. First, we will consider a four-gram tagger, where p takes the form:

$$p(x_1 \cdots x_n, y_1 \cdots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-3}, y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i) \tag{1}$$

   We assume in this definition that $y_0 = y_{-1} = y_{-2} = *$, where $*$ is the START symbol, $y_{n+1} = STOP$, and $y_i \in \mathcal{K}$ for $i = 1 \cdots n$, where $\mathcal{K}$ is the set of possible tags in the HMM.
   Second, we consider a version of the Viterbi algorithm that takes as input **an integer** n (and not a sentence $x_1 \cdots x_n$ as we saw in class) and finds

$$\max_{y_1 \cdots y_{n+1}, x_1 \cdots x_n} p(x_1 \cdots x_n, y_1 \cdots y_{n+1})$$

   for a four-gram tagger, as defined in Equation 1. $x_1 \cdots x_n$ may range over the values of some fixed vocabulary $\mathcal{V}$. Complete the following pseudo-code of this version of the Viterbi algorithm for this model . The pseudo-code must be efficient.

   Input: An integer n, parameters $q(w|t, u, v)$ and $e(x|s)$.
   Definitions: Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-2} = \mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \cdots n$. Define $\mathcal{V}$ to be the set of possible words.
   Initialization: $\cdots$
   Algorithm: $\cdots$
   Return: $\cdots$

3. (80 pts) In this programming exercise with Python, we will implement several versions of an HMM POS tagger.

Note: For all the below sections, unknown words are words that appear in the test set but did not appear in the training set.

(a) Use the NLTK toolkit for importing the Brown corpus. This corpus contains text from 500 sources, and the sources have been categorized by genre. Here we will use a portion of the corpus: the "news" category. Load the tagged sentences for this portion of the corpus. Then, divide the obtained corpus into training set and test set such that the test set is formed by the last 10% of the sentences.

(b) **Implementation of the most likely tag baseline**

   i. Using the training set, compute for each word the tag that maximizes $p(tag|word)$, based on the maximum likelihood estimation. Assume that the most likely tag of all the unknown words is "NN". (Unknown words are words that appear in the test set but not in the training set.)

   ii. Using the test set, compute the error rate (i.e., $1-accuracy$) for known words and for unknown words, as well as the total error rate.

(c) **Implementation of a bigram HMM tagger**

   i. Training phase: Compute the transition and emission probabilities of a bigram HMM tagger directly on the training set using maximum likelihood estimation.

   ii. Implement the Viterbi algorithm corresponding to the bigram HMM model. (Choose an arbitrary tag for unknown words.)

   iii. Run the algorithm from c)ii) on the test set. Compute the error rates and compare to the results from b)ii).

(d) **Using Add-one smoothing**

   i. Training phase: Compute the emission probabilities of a bigram HMM tagger directly on the training set using (Laplace) Add-one smoothing.

   ii. Using the new probabilities, run the algorithm from c)ii) on the test set. Compute the error rates and compare to the results from b)ii) and c)iii).

(e) **Using pseudo-words**

   i. Design a set of pseudo-words for unknown words in the test set and low-frequency words in the training set.

   ii. Using the pseudo-words as well as maximum likelihood estimation (as in c)i)), run the Viterbi algorithm on the test set. Compute the error rates and compare to the results from b)ii), c)iii) and d)ii).

   iii. Using the pseudo-words as well as Add-One smoothing (as in d)i)), run the Viterbi algorithm on the test set. Compute the error rates and compare to the results from b)ii), c)iii), d)ii) and e)ii). For the results obtained using both pseudo-words and Add-One smoothing, build a confusion matrix and investigate the most frequent errors. A confusion matrix is an $|\mathcal{K}|$ over $|\mathcal{K}|$ matrix, where the (i, j) entry corresponds to the number of tokens which have a true tag i and a predicted tag j.

Note: NLTK is a Python package for NLP. Please see here `http://www.nltk.org/` for downloading and documentation. You can download the Brown corpus using *nltk.download('brown')* and traverse its sentences using the *tagged_sents()* method. Please refer to the NLTK documentation for further information. Please use NLTK just for loading and reading the corpus.

A given word w may be annotated with a complex tag t, containing the symbols '+' and/or '-'. When encountering such a complex tag t, consider only the prefix of t that comes before the first occurrence of '+' or '-' in t as the POS tag of w.