Assignment 4: HMM Part-of-Speech Tagging

Warning

Making assignments requires us to strike a balance between being novel and practical. With that in mind, we know that solutions or near-solutions to this particular task may exist online. **Do not use outside solutions, as this would be plagiarism, which is an academic offence.** To earn marks on this assignment, you must develop your solutions. We will run software similarity checks on all submissions to ensure that everybody meets the University's academic integrity standards. So please, do your own work.

Overview

Natural Language Processing (NLP) is a subset of AI that focuses on the understanding and generation of written and spoken language. This involves a series of tasks from low-level speech recognition on audio signals to high-level semantic understanding and inferencing on the parsed sentences.

One task within this spectrum is Part-Of-Speech (POS) tagging. Every word and punctuation symbol is understood to have a syntactic role in its sentence, such as nouns (denoting people, places or things), verbs (denoting actions), adjectives (which describe nouns) and adverbs (which describe verbs), to name a few. Each word in a text is therefore associated with a part-of-speech tag (usually assigned by hand), where the total number of tags can depend on the organization tagging the text.

You can find a list of all the part-of-speech tags **HERE (https://q.utoronto.ca/courses/293717/pages/part-of-speech-tags-for-assignment-4)** .

While this task falls under the domain of NLP, having prior language experience doesn't offer any particular advantage. Ultimately, the main task is to create a hidden Markov model that can figure out a sequence of underlying states given a sequence of observations.

Your Tasks

Your task for this assignment is to create a hidden Markov model (HMM) for POS tagging.

1. Creating the initial, transition and observation probability tables in the HMM model by using the text-tag pairs in the training files.
2. Predicting POS tags for untagged text by performing inference with the HMM model created in step 1.

We will grade your solution by calculating the accuracy of your predictions compared to the ground-truth POS tags in the data set.

Running Your Program

You will submit a file named **tagger.py**, which contains your program that creates a hidden Markov model to predict the POS tags for given sentences.

Your program must use python3 and run on the teach.cs server (where we run our auto-testing scripts).

We will test your program using several training and test file combinations. For each combination, we will run the following command.

```
python3 tagger.py --trainingfiles <training files> --testfile <test file> --outputfile <output file>
```

Each command specifies one or more training files (separated by spaces), one test file, and one output file.

For the example below, we ran the program on two training files (data1.txt and data2.txt) and one test file (test3.txt). The program produced one output file (output123.txt).

python3 tagger.py --trainingfiles data1.txt data2.txt --testfile test3.txt --outputfile output123.txt

Training, Test and Output File Formats

Each **training file** contains some text and POS tags. Each line has one word/punctuation, a colon, and the POS tag. We consider the POS tags in the training files "correct" or "ground-truth." See the first five lines of **training1.txt** below.

```
Detective : NP0
Chief : NP0
Inspector : NP0
John : NP0
McLeish : NP0
```

Each **test file** contains the text only without the POS tags. See the first five lines of **test1.txt** below.

```
Detective
Chief
Inspector
```

```
John
McLeish
```

You can easily take any training file and produce a corresponding test file by removing all the POS tags.

The **output file** has the same format as the training files. However, the POS tags in the output file are predicted by your HMM model instead of the "ground-truth" POS tags from the training files.

Here are examples of a **training file (https://q.utoronto.ca/courses/293717/files/25531821?wrap=1)** **(https://q.utoronto.ca/courses/293717/files/25531821/download?download_frd=1)** and a **test file (https://q.utoronto.ca/courses/293717/files/25531820?wrap=1)** **(https://q.utoronto.ca/courses/293717/files/25531820/download?download_frd=1)** .

Submission and Marking Scheme

You should submit one file **tagger.py** on MarkUs.

We will determine your marks based on the predictive accuracy of your HMM model on the training and test files in our automated tests. We will set up 12 automated tests of varying difficulty on MarkUs. Your program must terminate within **5 minutes** to earn marks for each test case. We strongly recommend testing your program on the teach.cs server to ensure that it terminates within the time limit.

For each test, your goal is to predict the "ground-truth" tags at least 90% of the time. Your mark for each test case is calculated below.

- 100% if your accuracy is at least 90%,
- your accuracy / 90 if your accuracy is less than 90%.

For example, if you achieve an 80% accuracy on a test, you will get 80/90 = 88.9% on that test case.

Starter Code

We have provided some code to help you parse the names of multiple training files, one test file, and one output file. You can find the starter code on MarkUs.

Provided Test Cases

We have provided three test cases to you on MarkUs. These test cases will be among the final 12 test cases. Your final mark depends on your program's predictive accuracy on all 12 test cases. Getting 100% on these three tests does not mean you are guaranteed 100% as your final mark.

Suggestions

Making the HMM Tagger

In the hidden Markov model, the hidden states represent the POS tags, and the evidence variables represent the words in the sentences.

You can assume that the sentences are independent. As a result, you can split each training/test file into individual sentences without worrying about losing any useful information in the training/prediction process.

During the training phase, you will need to learn three probability tables described below:

- The initial probabilities over the POS tags (how likely each POS tag appears at the beginning of a sentence)
- The transition probabilities from one POS tag to another.
- The observation probabilities from each POS tag to each observed word.

Learning these probabilities will require you to perform counting.

For instance, you need to do the following to determine the initial probability for the POS tag **NP0**.

(1) Count the number of times **NP0** is the tag for the first word in a sentence. Let's call this number X.
(2) Count the total number of sentences in the training files. Let's call this number Y.
The initial probability for **NP0** is equal to X / Y.

You can decide how to store these probability tables.

During the test phase, you will use these probability tables to determine the most likely tag for each word in a sentence. We recommend using the Viterbi algorithm for this task, but you can use any algorithm.

How do I split this text into multiple sentences?

Some training files may have complicated text, e.g. conversations between multiple people. In this case, finding a clean-cut way to split the text into multiple sentences may be challenging.

Remember that you are dealing with real data sets in this assignment. Sometimes, there are no correct answers regarding real data sets. Your goal is to create an HMM model that achieves the best predictive accuracy on the test set. We encourage you to experiment with different ways of splitting the text into sentences to find the best way to improve predictive accuracy.

How do I predict the POS tag for a word in the test file that never appeared in the training files?

Aha! This is a perfect example of a problem you might encounter in a real data set. There are at least two strategies to deal with this situation.

(1) Calculate the most likely POS tag for the current word by counting the frequency of all the POS tags in the training files. This ignores the context of the word, i.e. the other words in the sentence.

(2) Calculate the most likely POS tag for the current word given the word's position in the sentence.

(3) Calculate the most likely POS tag for the current word given the previous word in the sentence.

(4) Calculate the most likely POS tag for the current word given the previous sequence of words in the sentence (This is similar to the Viterbi algorithm).

What are ambiguity tags, and how should I handle them?

The main problem with POS tagging is ambiguity. In English, many common words have multiple meanings and, therefore, multiple POS. The job of a POS tagger is to resolve this ambiguity accurately based on the context of use. For example, "shot" can be a noun or a verb. It could be in the past tense or past participle when used as a verb.

An ambiguity tag is a part-of-speech tag presenting two alternatives for a correct tag, which has a label consisting of two tags linked by a hyphen (such as VVD-VVN for VVD 'past tense' or VVN 'past participle'). Such a tag indicates that the automatic tagger could not determine the correct tag. As a result, both possible tags are assigned to the word. Ambiguity tags represent a small percentage of tags (in the British National Corpus, about 5 percent) (cf. Baker 2006: 10; Garside 1997: 120, 148).

On the **page (https://q.utoronto.ca/courses/293717/pages/part-of-speech-tags-for-assignment-4)** listing all the POS tags, you will find 61 POS tags and 15 ambiguity tags. Note that an ambiguity tag may appear in either order in the training file. For example, the first ambiguity tag **AJ0-AV0** may appear as **AJ0-AV0** or **AV0-AJ0** in the training files. Your HMM model needs to account for 61 POS tags plus 30 possible ambiguity tags.