Handed out: 05-03-2024
Due: 08-04-2024 (11:59 pm)

Instructions

This project requires you to work on some tasks related to HMM: (a) implementation of a POS tagger, (b) making naive predictions (Question 2), (c) implementing the Viterbi algorithm (Question 3), (d) listing possible improvements, through their application to Twitter tweets (Question 4).

- This project is meant to be done in a group of three students.
- All work submitted must include the names of its group members to the appropriate Canvas folder.
- All Python3 code must be submitted as `.py` files.
- Do not submit just the pseudo-code or code that does not run.

Make sure to answer all the questions, and submit your code, generated files, and answers to the appropriate Canvas folder.

Question 1. Group Formation (5 points)

This project is to be done in a group of three students. Question 1 expects you to submit your group name and its members to the appropriate Canvas folder. You can search for group mates in the piazza. The search for teammates feature in Piazza is described in the following link: `https://support.piazza.com/support/solutions/articles/48001158117-search-for-teammates`. Hope you find it useful. Read on to find out more about the project context and the other questions.

POS tagging of Tweets

Twitter (`www.twitter.com`) is a popular microblogging site that contains a large amount of user-generated posts, each called a tweet. By analyzing the sentiment embedded in tweets, we gain valuable insights into commercial interests such as the popularity of a brand or the general belief in the viability of a stock[1]. Many companies earn revenue from providing such tweet sentiment analysis. In such sentiment analysis, a typical upstream task is the part-of-speech (POS) tagging of tweets, which generates POS tags that are essential for other downstream natural language processing tasks.

This question requires you to build a POS tagging system using the hidden Markov model (HMM). The files which you will need for this task are given below. Please note that these are real-world data and may contain rude words (for the constitutionally fragile: you are hereby forewarned). Of course, we don't expect you to read and tag each one of these manually.

(i) `twitter_train.txt`

(ii) `twitter_dev_no_tag.txt`

(iii) `twitter_dev_ans.txt`

(iv) `twitter_tags.txt`

(v) `hmm.py`

`twitter_train.txt` is a labelled training set. Each non-empty line of the labeled data contains one token and its associated POS tag separated by a tab. An empty line separates sentences (tweets). Below is an example with two tweets (there is an empty line after "?? ,").

```
@USER_123d5421 @
I O
like V
IPhones N
```

[1]https://ieeexplore.ieee.org/document/8455771

```
@USER_2346f3f51 @
wat O
wearin V
4 P
the D
party N
??  ,
```

Both `twitter_dev_no_tag.txt` and `twitter_dev_ans.txt` have the same format as `twitter_train.txt`, except that the former does not contain tags and the latter does not contain tokens.

`twitter_dev_ans.txt` contains the tags of the corresponding tokens in `twitter_dev_no_tag.txt`. Both files are used to evaluate your code. (We will evaluate your code with more data from the same domain.) `twitter_train*.txt` and `twitter_dev*.txt` contain 1101 and 99 tweets respectively.

`twitter_tags.txt` contains the full set of 25 possible tags. `hmm.py` contains skeleton code, and you have to write your code in this file.

Recall that the joint likelihood of the observed data $x_1, x_2, ..., x_n$ and their associated tags $y_1, y_2, ..., y_n$ is given by an HMM as:

$$P(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n) = \prod_{t=0}^{n} a_{y_t, y_{y+1}} \prod_{t=1}^{n} b_{y_t}(x_t)$$

where y_0 and y_{n+1} are the START (*) and STOP states respectively, $a_{i,j} = P(y_t = j | y_{t-1} = i)$ is the transition probability, and $b_j(w) = P(x_t = w | y_t = j)$ is the output probability.

Question 2. Naive Approaches
Question 2.1 Naive Approach 1 (5 points)

(a) Write a function to estimate the output probabilities from the training data `twitter_train.txt` using maximum likelihood estimation (MLE), i.e.,

$$P(x = w|y = j) = b_j(w) = \frac{count(y = j \rightarrow x = w)}{count(y = j)}$$

where the numerator is the number of times $\overset{\text{output}}{\text{token}}$ w is associated with tag $\overset{\text{state}}{j}$ in the training data, and the denominator is the number of times tag j appears. Save these output probabilities into a file named `naive_output_probs.txt`.

A problem that you may encounter is that the training data `twitter_train.txt` does not contain some tokens (words) that appear in the test data. You can handle this "unseen token" problem by smoothing the output probability as

$$b_j(w) = \frac{count(y = j \rightarrow x = w) + \delta}{count(y = j) + \delta \times (num\_words + 1)}$$

where num_words is the number of unique words in the training data, δ is a real number, and the $+1$ in the denominator accounts for unseen tokens (all unseen tokens "collapse" into a single token).
Typical values of δ to try are: $0.01, 0.1, 1, 10$. Pick one that works best for you. (This smoothing applies to the other questions, too.)

(b) Using these output probabilities, we can naively obtain the best tag j^* for a given token w with the following equation.

$$j^* = \underset{j}{\operatorname{argmax}}\, b_j(w) = \underset{j}{\operatorname{argmax}}\, P(x = w|y = j)$$

Write a function `naive_predict()` (see `hmm.py`) that uses the output probabilities in `naive_output_probs.txt` to predict the tags of the tweets in `twitter_dev_no_tag.txt` with this naive approach. Write your predictions into a file named `naive_predictions.txt` in the same format as `twitter_dev_ans.txt`.

(c) What is the accuracy of your predictions (number of correctly predicted tags / number of predictions)? (Use the `evaluate` function in `hmm.py`.)

(Remember to submit `naive_output_probs.txt` and `naive_predictions.txt`.)

Question 2.2 Naive Approach 2 (5 points)

A better approach is to estimate j^* using

$$j^* = \underset{j}{\operatorname{argmax}}\, P(y = j | x = w)$$

(a) How do you compute the right-hand side of this equation?

(b) Implement this approach in a method `naive_predict2()` (see `hmm.py`) that uses the output probabilities in `naive_output_probs.txt` to predict the tags of the tweets in `twitter_dev_no_tag.txt`. You can also make use of other information in `twitter_train.txt`. Write your predictions into a file named `naive_predictions2.txt` in the same format as `twitter_dev_ans.txt`.

(c) What is the accuracy of your predictions? (Use the `evaluate` function in `hmm.py`.)

(Remember to submit `naive_predictions2.txt`.)

Question 3. Viterbi Algorithm (20 points)

(a) Write a function to compute the output probabilities and transition probabilities from the training data `twitter_train.txt` using the MLE approach, saving the output probabilities to `output_probs.txt`, and the transition probabilities to `trans_probs.txt`. (You may reuse the function you have defined earlier to compute the output probabilities.) Recall that the transition probability is defined as

$$a_{i,j} = P(y_t = j | y_{t-1} = i) = \frac{count(y_{t-1} = i, y_t = j)}{count(y_{t-1} = i)}$$

where $count(y_{t-1} = i, y_t = j)$ is the number of times tag i transitions into tag j in the training data, and $count(y_{t-1} = i)$ is the number of times tag i appears in the training data. You may wish to "smooth" the transition probability in a similar manner as for the output probability.

(b) Write a function `viterbi_predict` that implements the Viterbi algorithm. This function uses the output probabilities and transition probabilities (stored in `output_probs.txt` and `trans_probs.txt` respectively) to predict the tags for tweets in `twitter_dev_no_tag.txt`, and writes the predictions to `viterbi_predictions.txt`. This function also accepts `twitter_tags.txt` so that it knows the full set of tags. Recall that the Viterbi algorithm computes the best tag sequence $y_1^*, y_2^*, ..., y_n^*$ for an observed token sequence $x_1^*, x_2^*, ..., x_n^*$ in this manner:

$$y_1^*, y_2^*, ..., y_n^* = \underset{y_1, y_2, ..., y_n}{\operatorname{argmax}} P(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$$

(c) What is the accuracy of your Viterbi algorithm on `twitter_dev_no_tag.txt`? (Use the `evaluate` function in `hmm.py`.)

(Remember to submit `output_probs.txt`, `trans_probs.txt` and `viterbi_predictions.txt`.)

Question 4. Improvements (10 points)

(a) How can you improve your POS tagger further? Describe your ideas to improve your previous POS tagger submission. You are not allowed to use any external resources or packages and must design your new POS tagger within the framework of HMMs. However, you are free to perform any automatic (i.e., not manual) preprocessing of the data.

Hints: Could your new POS tagger better handle unseen words? Could your new POS tagger better model transition probabilities? Could your new POS tagger take advantage of linguistic patterns present in tweets? Could the words be clustered into meaningful groups?

(b) Write a function `viterbi_predict2` that implements your improved system. This function takes the same input as `viterbi_predict`. If your changes affect the output probabilities and transition probabilities, write those new probabilities into `output_probs2.txt` and `trans_probs2.txt` respectively. If those probabilities are not affected, simply copy your previous files to `output_probs2.txt` and `trans_probs2.txt`. Write your predictions to `viterbi_predictions2.txt`

(c) What is the accuracy of your improved system? (Use the `evaluate` function in `hmm.py`.)

(Remember to submit `output_probs2.txt`, `trans_probs2.txt` and `viterbi_predictions2.txt`.)