50.007 Machine Learning, Spring 2023
Design Project

Due 14 Apr 2023, 11:59 pm

Instructions

Please read the following instructions carefully before you start this project:

- This is a group project. You are allowed to form groups in any way you like, but each group must consist of either 2 or 3 people.
- You are strictly NOT allowed to use any external resources or machine learning packages. You will receive 0 for this project if you do so. You are required to implement all functions yourself. You may use NumPy or CuPy to vectorise your functions to increase computation speed but you may not make use of any other functions implemented in those libraries.
- Each group should submit code together with a report summarizing your work, and give clear instructions on how to run your code. Please also submit your system's outputs. Your outputs should be in the same column format as that of the training set.

Project Summary

Many companies today are interested in developing automated systems for analyzing sentiment information associated with social media data. Such sentiment information can be used for making important decisions such as making product recommendations, predicting social stance and forecasting financial market trends. This is especially important for many social media companies which depend on the performance of their algorithms to generate revenue in lieu of charging platform subscription fees.

Many problems within the field of NLP such as sentiment analysis are essentially structured prediction problems, among which sequence labeling is the simplest class of problems. The hidden Markov model (HMM) that we have learned in class is a simple structured prediction model. In this design project, we would like to design our sequence labelling model for informal texts using the hidden Markov model (HMM) that we have learned in class. We hope that your sequence labelling system for informal texts can serve as the very first step towards building a more complex, intelligent sentiment analysis system for social media text.

The files for this project are in the files `EN.zip` and `FR.zip`. For each dataset, we provide a labelled training set `train`, an unlabelled development set `dev.in`, and a labelled development set `dev.out`. The labelled data has the format of one token per line with token and tag separated by tab and a single empty line that separates sentences. The format (for the `EN` dataset, for example) can be something like the following:

```
Best O
Deal O
Chiang B-positive
mai I-positive
Tours I-positive
, O
The O
North O
of O
Thailand B-neutral
To O
Get O
special O
Promotion O
and O
free O
Transfer O
roundtrip O
. O
Contact O
: O
...  O
http://t.co/sSn1OBTZ O
```

where labels such as `B-positive`, `I-positive` are used to indicate Beginning and the Inside of the entities which are associated with a positive sentiment. `O` is used to indicate the Outside of any entity. Similarly for `B-negative`, `I-negative` and `B-neutral`, `I-neutral`, which are used to indicate entities which are associated with negative and neutral sentiment, respectively. Overall, our goal is to build a sequence labelling system from such training data and then use the system to predict tag sequences for new sentences. Specifically, we will be building two sentiment analysis systems for two different languages from scratch.

1 Part 1 (25 points)

Recall that the HMM discussed in class is defined as follows:

$$p(x_1, ..., x_n; y_0, y_1, ..., y_n, y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i|y_i) \quad (1)$$

where $y_0 = START$ and $y_{n+1} = STOP$. Here q are transition probabilities, and e are emission parameters. In this project, x's are the natural language words, and y's are the tags (such as `O`, `B-positive`).

- Write a function that estimates the emission parameters from the training set using MLE (maximum likelihood estimation):

$$e(x|y) = \frac{Count(y \to x)}{Count(y)} \tag{2}$$

  (5 points)

- One problem with estimating the emission parameters is that some words that appear in the test set do not appear in the training set. One simple idea to handle this issue is as follows. We introduce a special word token `#UNK#`, and make the following modifications to the computation of emission probabilities:

$$e(x|y) = \begin{cases} \frac{Count(y \to x)}{Count(y)+k}, \text{If the word token x appears in the training set} \\ \frac{k}{Count(y)+k}, \text{If word token x is the special token } \texttt{\#UNK\#} \end{cases} \tag{3}$$

  (This basically says we assume from any label y there is a certain chance of generating `#UNK#` as a rare event, and empirically we assume we have observed that there are k occurrences of such an event.)
  During the testing phase, if the word does not appear in the training set, we replace that word with `#UNK#`.
  Set k to 1, implement this fix into your function for computing the emission parameters.
  (10 points)

- Implement a simple sentiment analysis system that produces the tag

$$y^* = arg\ max_y\ e(x|y) \tag{4}$$

  for each word x in the sequence.

  For all the datasets `EN`, `FR`, learn these parameters with `train`, and evaluate your system on the development set `dev.in` for each of the dataset. Write your output to `dev.p1.out` for the two datasets respectively. Compare your outputs and the gold-standard outputs in `dev.out` and report the precision, recall and F scores of such a baseline system for each dataset.

The precision score is defined as follows:

$$Precision = \frac{\text{Total number of correctly predicted entities}}{\text{Total number of predicted entities}} \tag{5}$$

The recall score is defined as follows:

$$Recall = \frac{\text{Total number of correctly predicted entities}}{\text{Total number of gold entities}} \tag{6}$$

where a gold entity is a true entity that is annotated in the reference output file, and a predicted entity is regarded as correct if and only if it matches exactly the gold entity (i.e., both their boundaries and sentiment are exactly the same).

Finally the F score is defined as follows:

$$F = \frac{2}{1/Precision + 1/Recall} \tag{7}$$

Note: in some cases, you might have an output sequence that consists of a transition from `O` *to* `I-negative` *(rather than* `B-negative`*). For example,* "`O I-negative I-negative O`"*. In this case, the second and third words should be regarded as one entity with negative sentiment. You can use the evaluation script shared with you to calculate such scores. However it is strongly encouraged that you understand how the scores are calculated.*
(10 points)

2 Part 2 (25 points)

- Write a function that estimates the transition parameters from the training set using MLE (maximum likelihood estimation):

  $$q(y_i|y_{i-1}) = \frac{Count(y_{i-1}, y_i)}{Count(y_{i-1})} \tag{8}$$

  Please make sure the following special cases are also considered: $q(STOP|y_n)$ and $q(y_1|START)$.
  (10 points)

- Use the estimated transition and emission parameters, implement the Viterbi algorithm to compute the following (for a sentence with n words):

  $$y_1^*, ..., y_n^* = arg\ max_{y_1,...,y_n}\ p(x_1, ..., x_n, y_0, y_1, ..., y_n, y_{n+1}) \tag{9}$$

  For all datasets, learn the model parameters with `train`. Run the Viterbi algorithm on the development set `dev.in` using the learned models, write your output to `dev.p2.out` for the two datasets respectively. Report the precision, recall and F scores of all systems.
  Note: in case you encounter potential numerical underflow issue, think of a way to address such an issue in your implementation.
  (15 points)

3 Part 3 (25 points)

- The HMM discussed in class makes a simple first-order assumption, where the next state only depends on the previous state in the generative process. However, it is possible to extend the model discussed in class to have second-order dependencies. In other words, the HMM can be parameterised in the following way:

$$p(x_1, ..., x_n, y_{-1}, y_0, y_1, y_2, ..., y_n, y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i|y_i) \quad (10)$$

  where we define $y_{-1} = y_0 = START$ and $y_{n+1} = STOP$.
  In other words, the transition probabilities are changed from $q(y_i|y_{i-1})$ to $q(y_i|y_{i-2}, y_{i-1})$ now. Describe the Viterbi algorithm used for decoding such a second-order HMM model and implement it. In other words, describe and implement the dynamic programming algorithm that computes the following efficiently for such an HMM:

$$y_1^*, y_2^*, ..., y_n^* = arg\ max_{y_1,y_2,...,y_n} p(x_1, x_2, ..., x_n, y_{-1}, y_0, y_1, y_2, ..., y_n, y_{n+1}) \quad (11)$$

  For all datasets, learn the model parameters with `train`. Run your new decoding algorithm on the development set `dev.in`. Write the outputs to `dev.p3.out` for each dataset respectively. Report the precision, recall and F scores for the outputs for both datasets.
  Hint: How to train your model under the new assumptions? How to learn the new transition parameters? Any changes to the emission parameters? What will be the time complexity of the new Viterbi algorithm?
  (25 points)

4 Part 4 – Design Challenge (25 points)

- Now, based on the training and development set, think of a better design for developing an improved sentiment analysis system for tweets using any model you like. Please explain clearly the model/method that you used for designing the new system. We will check your code and may call you for an interview if we have questions about your code. As with previous parts, learn the model parameters with `train` and run your system on the development set `dev.in` for both datasets. Write your outputs to `dev.p4.out`. Report the precision, recall and F scores of your new systems for these two languages.
  (10 points)

- We will evaluate your system's performance on two held out test sets `EN/test.in` and `FR/test.in`. The test sets will only be released on 48 hours before the deadline. Use your new system to generate the outputs. Write your outputs to `EN/test.p4.outt` and `FR/test.p4.out`. The system that achieves the overall highest F score on the test sets will be announced as the winner. *(15 points)*

Items To Be Submitted

Upload to eDimension a single ZIP file containing the following: (Please make sure you have only one submission from each team only.)

- A report detailing the approaches and results (PDF file)
- Source code (.py files) with README (instructions on how to run the code)
- Output files
  - EN/
    - dev.p1.out
    - dev.p2.out
    - dev.p3.out
    - dev.p4.out
    - test.p4.out
  - FR/
    - dev.p1.out
    - dev.p2.out
    - dev.p3.out
    - dev.p4.out
    - test.p4.out