

1D Group Project Report 50.007 Machine Learning SUTD 2021 ISTD

Group 4

Ma Yuchen	1004519
Chung Wah Kit	1004103
James Raphael Tiovalen	1004555

Table of Contents

able of Contents	. 2
arts 1-3	. 3
art 4	. 8
eneral Comments	11

Parts 1-3

For parts 1-3, we simply followed the project instructions document closely as our approach. We utilized the NumPy library to aid with all of our calculations so that our matrix manipulations are much more efficient as compared to manually forming and manipulating such matrices. Using NumPy, we form matrices similar to the ones specified in the HMM classes, slides, and notes. This is applicable and relevant for parts 1-3.

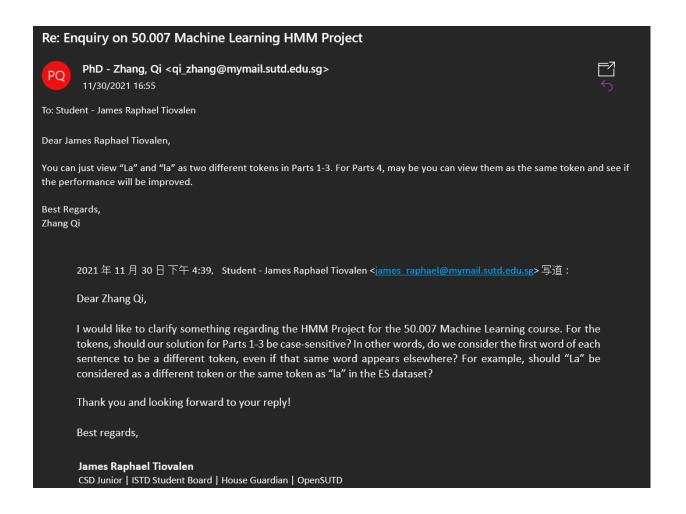
To outline our general approach, we first read the input files by simply splitting each line by the rightmost whitespace character to obtain the word-tag pairs. Then, we pre-process these words and their corresponding tags to form the NumPy matrices that we desire. Then, we execute the algorithm for each part according to the equations specified in the project instructions document and train our HMM model. After obtaining the relevant matrices as the parameters of our model, we read the words from the relevant test dataset files, predict their tags, and write them onto the corresponding output files. These output files can then be evaluated using the provided evaluation script so as to obtain the precision, recall, and F-scores of said models.

For part 1, the main functions are the `calculate_emission_parameters` and the `qet_label_from_token` functions.

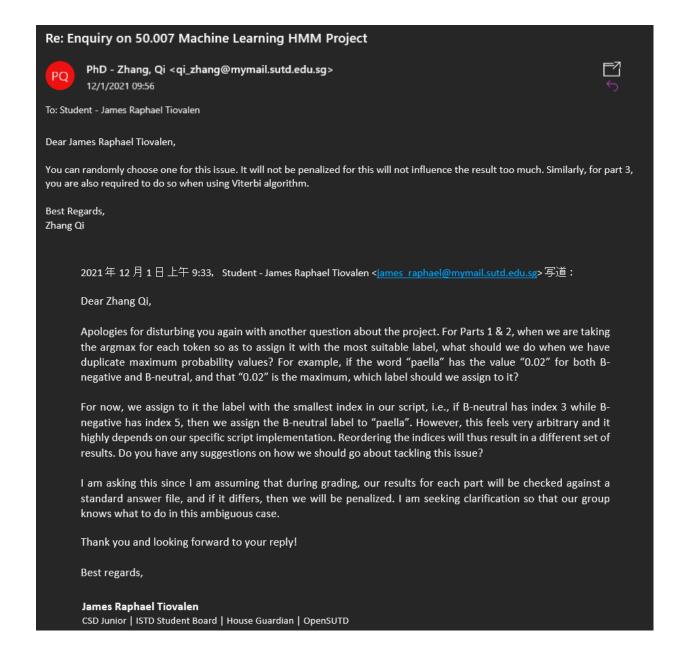
For part 2, the main functions are the `calculate_transition_parameters` and the `viterbi_log` functions.

For part 3, the main function is the `viterbi_best_five_log` function.

Our approach for different letter cases would be to treat them differently, as instructed in this email correspondence with the Teaching Assistant, Zhang Qi:



One special case that we took care of would be when there are multiple tags with the same maximum value of e(x|y), and thus are all possible valid candidates as the argmax tag. After our correspondence, attached here:



We decided to randomly select between those valid tags, using the sum of our student IDs as the input seed to NumPy to ensure reproducible behavior. This randomizer might cause the F-scores to be slightly lower as compared to without any randomizer, since the randomizer uniformly chooses between all valid candidates, instead of weighing them based on how often they appear in the training data set, which is in contrast to the non-uniform distribution of the tags (for example: the label "O" appears most often).

In the case whereby we encounter numerical underflow issues in parts 2 and 3, we utilize the log-likelihood and maximize that instead of the normal likelihood instead (and disable any warnings regarding np.log(0) since they will possess the minimum value possible of -np.inf anyway).

For part 3, we simply modified the Viterbi algorithm in part 2 to store the top 5 scores and paths for each node (defined by the position and tag) instead of the best score for each node. Hence, when we reach the STOP node, we have the 5 best paths, and thus, this allows us to return the 5^{th} -best output tag sequence.

For part 1, the output precision, recall, and F scores for both the ES and RU datasets are:

ES

Entity in gold data: 255
Entity in prediction: 1734

of Correct Entity: 205 Entity precision: 0.1182 Entity recall: 0.8039 Entity F: 0.2061

of Correct Sentiment: 113 Sentiment precision: 0.0652 Sentiment recall: 0.4431

Sentiment F: 0.1136

RU

Entity in gold data: 461
Entity in prediction: 2089

of Correct Entity: 335
Entity precision: 0.1604
Entity recall: 0.7267
Entity F: 0.2627

of Correct Sentiment: 136
Sentiment precision: 0.0651
Sentiment recall: 0.2950

Sentiment F: 0.1067

For part 2, the output precision, recall, and F scores for both the ES and RU datasets are:

ES

Entity in gold data: 255
Entity in prediction: 686

of Correct Entity: 131 Entity precision: 0.1910 Entity recall: 0.5137 Entity F: 0.2784

of Correct Sentiment: 104
Sentiment precision: 0.1516
Sentiment recall: 0.4078

Sentiment F: 0.2210

RU

Entity in gold data: 461
Entity in prediction: 575

of Correct Entity: 223
Entity precision: 0.3878
Entity recall: 0.4837
Entity F: 0.4305

of Correct Sentiment: 145 Sentiment precision: 0.2522 Sentiment recall: 0.3145

Sentiment F: 0.2799

For part 3, the output precision, recall, and F scores for both the ES and RU datasets are:

ES

Entity in gold data: 255 # Entity in prediction: 557

of Correct Entity: 129
Entity precision: 0.2316
Entity recall: 0.5059
Entity F: 0.3177

of Correct Sentiment: 103 Sentiment precision: 0.1849 Sentiment recall: 0.4039

Sentiment F: 0.2537

RU

Entity in gold data: 461
Entity in prediction: 535

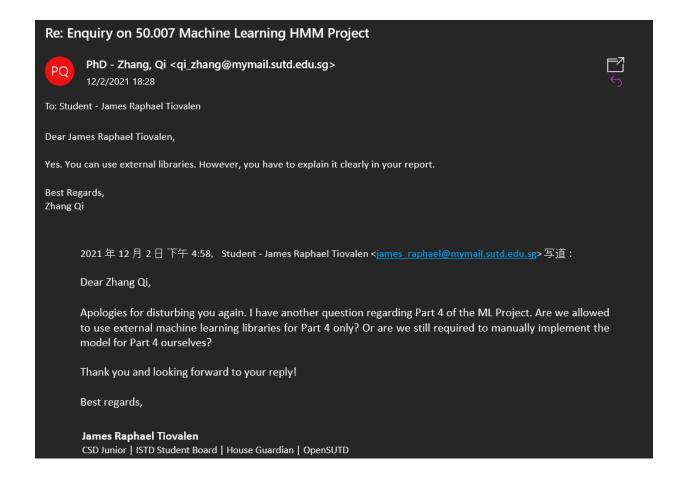
of Correct Entity: 220 Entity precision: 0.4112 Entity recall: 0.4772 Entity F: 0.4418

of Correct Sentiment: 144 Sentiment precision: 0.2692 Sentiment recall: 0.3124

Sentiment F: 0.2892

Part 4

For part 4, after reading into the literature, and since we were allowed to use external machine learning libraries for this part:



We decided to implement a unidirectional LSTM model using PyTorch to capture sequential information. Since the specific details of the LSTM are quite tedious to implement, we opted to use PyTorch to simplify the interface that we are dealing with and not get bogged down by the small nitty-gritty details, while still capturing the same essential idea of the model that we would like to implement. We also decided to convert all training words into lowercase as it ever-so-slightly improves our results.

The overall structure of our LSTM model is as such:

```
LSTMTagger(
  (word_embeddings): Embedding(X, 16)
  (lstm): LSTM(16, 16, num_layers=4)
  (hidden2tag): Linear(in_features=16, out_features=7, bias=True)
),
```

where X = 4805 for the ES dataset and X = 7480 for the RU dataset.

For part 4, the <u>best</u> output precision, recall, and F scores for both the ES and RU datasets that we have obtained so far are:

ES RU Number of Epochs: 26 Number of Epochs: 45 # Entity in gold data: 255 # Entity in gold data: 461 # Entity in prediction: 206 # Entity in prediction: 329 # of Correct Entity: 135 # of Correct Entity: 208 Entity precision: 0.6553 Entity precision: 0.6322 Entity recall: 0.5294 Entity recall: 0.4512 Entity F: 0.5857 Entity F: 0.5266 # of Correct Sentiment: 106 # of Correct Sentiment: 153 Sentiment precision: 0.4650 Sentiment precision: 0.5146 Sentiment recall: 0.4157 Sentiment recall: 0.3319 Sentiment F: 0.4599 Sentiment F: 0.3873

Due to the stochastic nature of the LSTM model, the exact model, and hence, the resulting predictions will differ slightly with different runs.

Future improvements could potentially be to improve the unidirectional LSTM by changing it into a bidirectional one, even though it would require more time and resources to train due to the increase in number of parameters. This trade-off in amount of training time and resources in exchange for better F-scores of the model is prevalent

in the machine learning world, as models get larger and larger. Other potentially promising models include a BERT-BiLSTM-CRF model, a structured perceptron model, <u>Facebook's RoBERTa</u>, <u>OpenAI's GPT-3</u>, or even the <u>latest Google's GLaM model</u>.

General Comments

To inspect, execute, or modify any parts of our code, simply open up the Jupyter Notebook files `parts_1_to_3.ipynb` and `part_4.ipynb`. Each cell in the 2 notebooks is interactively executable as per the Jupyter Notebook standard.

Our GitHub repository is available here: https://github.com/jamestiotio/humu_humu