



01.112 Machine Learning, Fall 2018
Design Project Report

By

Vishal Ramanathan

Kasper Halme

Shobhit Narayanan

Contents

Part 1	3
How to run the code	3
Part 2	4
Part 3	4
Part 4	5
Part 5	5
Results	6
EN	6
Part 2	6
Part 3	6
Part 4	6
Part 5	6
FR	7
Part 2	7
Part 3	7
Part 4	7
Part 5	7
CN	8
Part 2	8
Part 3	8
SG	8
Part 2	8
Part 3	8

Part 1

The first and most important step towards building a supervised machine learning system is to get annotated data. In order to do so, we were tasked to tag about 500 tweets based on named entities and sentiment related to each one based on the tweet.

How to run the code

Code_ML.ipynb has our group's code. We collaborated and wrote our code using Google Colab. In order to run the code, follow these steps:

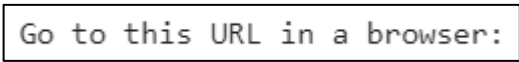

Step 0: Create a folder in Google Drive


Step 1: Upload Code_ML.ipynb and other relevant folders (EN, FR, CN, SG) with all the files (train, test.in, dev.in, etc.) in it to the folder created before in Google Drive.

Step 2: Double click to open the notebook. Google Colab should launch.

Step 3: On the right-hand side, click  to connect to a hosted runtime.

Step 4:  First run this chunk to allow the notebook to connect to your Google Drive to access input files

Step 5: Click the URL next to  and select your Google account and then click 

Step 6: Copy the entire code displayed as in and paste into the dialog box below 

Step 7: Specify the path to the folder that contains the code and folders (EN, FR, SG, CN) in the variable as follows: `file_path = "/content/drive/My Drive/<Folder Name created in Step 0 that contains everything >/<The data set folder E.g. EN, FR, CN, SG>/"`

Step 8: Run the following chunks that has all the code for the different parts

Part 2

Emission parameter estimates are done in this part. The probabilities are calculated by taking count of the words that appear, given a tag. Then divided by the total number of times a tag appears.

In our approach, we get the tag counts using `value_counts()` and saving it in a dictionary (`tagc_dict`). Then count the words for a given tag and store it in a dictionary of dataframes (`wordc_dictdf`) where the key of the dictionary is the tag and the values are a dataframe that has column as the tag same as the dictionary key and then words as the index. The counts are then stored and retrieved using the tag and word. To calculate the emission parameter, we divide the counts in `wordc_dictdf` by `tagc_dict` and store it in `emm_dictdf`.

We then use smoothing to find the tag in cases when we encounter a word that is not in the training set. Using a function, we retrieve the tag that has the maximum emission parameter given a word and write it to file after appending it to the input word.

Part 3

In this part we calculated transition parameters and implemented Viterbi algorithm to compute optimal set of tags for a given sentence.

To store the calculated transition parameters, we created a dataframe where the columns contained all 'u' tags (all tags including Start) and the index was all the 'v' tags (all tags including Stop). We first obtained the counts of the tag transitions ($u \rightarrow v$) then divided by the total count of each source tag 'u'.

Our implementation of Viterbi algorithm first took in the input file and parsed it to obtain tweets individually to run Viterbi algorithm on each on separately. We calculated the optimal tag going from base case, recursive case ending at the final $v \rightarrow \text{Stop}$ case and appended the output to the list of input words in the tweet and wrote to file.

Part 4

Part 3 had a first order assumption when calculating transition probabilities. We now try to incorporate second order assumption which means, we look at the tag 2 states before current one while calculating transition probabilities. We then run second order Viterbi algorithm to obtain the optimal set of tags for each tweet.

Part 5

To improve performance, we tried to model a perceptron based on Part 3 implementation to update the emission and transition probabilities to improve our results.

We read in our predicted results from Part 3 and compare it with the actual result from dev.out. After comparing the outputs, we update the emission probabilities based on the comparison if the actual and predicted match or not. We also look the actual and predicted state transitions and update transition probabilities as well. We only focus on first order assumption over second order assumption to keep our model more generalized and reduce chance of overfitting the parameters of emission and transition probabilities.

Results

EN

Part 2

```
D:\Term 6\01.112 - ML\Project\EN>python evalResult.py dev.out dev.p2.out

#Entity in gold data: 802
#Entity in prediction: 1044

#Correct Entity : 577
Entity precision: 0.5527
Entity recall: 0.7195
Entity F: 0.6251

#Correct Entity Type : 448
Entity Type precision: 0.4291
Entity Type recall: 0.5586
Entity Type F: 0.4854
```

Part 3

```
D:\Term 6\01.112 - ML\Project\EN>python evalResult.py dev.out dev.p3.out

#Entity in gold data: 802
#Entity in prediction: 623

#Correct Entity : 407
Entity precision: 0.6533
Entity recall: 0.5075
Entity F: 0.5712

#Correct Entity Type : 338
Entity Type precision: 0.5425
Entity Type recall: 0.4214
Entity Type F: 0.4744
```

Part 4

```
D:\Term 6\01.112 - ML\Project\EN>python evalResult.py dev.out dev.p4.out

#Entity in gold data: 802
#Entity in prediction: 602

#Correct Entity : 393
Entity precision: 0.6528
Entity recall: 0.4900
Entity F: 0.5598

#Correct Entity Type : 323
Entity Type precision: 0.5365
Entity Type recall: 0.4027
Entity Type F: 0.4601
```

Part 5

```
D:\Term 6\01.112 - ML\Project\EN>python evalResult.py dev.out dev.p5.out

#Entity in gold data: 802
#Entity in prediction: 598

#Correct Entity : 416
Entity precision: 0.6957
Entity recall: 0.5187
Entity F: 0.5943

#Correct Entity Type : 339
Entity Type precision: 0.5669
Entity Type recall: 0.4227
Entity Type F: 0.4843
```

FR

Part 2

```
D:\Term 6\01.112 - ML\Project\FR>python evalResult.py dev.out dev.p2.out

#Entity in gold data: 238
#Entity in prediction: 1114

#Correct Entity : 186
Entity precision: 0.1670
Entity recall: 0.7815
Entity F: 0.2751

#Correct Entity Type : 79
Entity Type precision: 0.0709
Entity Type recall: 0.3319
Entity Type F: 0.1169
```

Part 3

```
D:\Term 6\01.112 - ML\Project\FR>python evalResult.py dev.out dev.p3.out

#Entity in gold data: 238
#Entity in prediction: 57

#Correct Entity : 35
Entity precision: 0.6140
Entity recall: 0.1471
Entity F: 0.2373

#Correct Entity Type : 22
Entity Type precision: 0.3860
Entity Type recall: 0.0924
Entity Type F: 0.1492
```

Part 4

```
D:\Term 6\01.112 - ML\Project\FR>python evalResult.py dev.out dev.p4.out

#Entity in gold data: 238
#Entity in prediction: 42

#Correct Entity : 23
Entity precision: 0.5476
Entity recall: 0.0966
Entity F: 0.1643

#Correct Entity Type : 14
Entity Type precision: 0.3333
Entity Type recall: 0.0588
Entity Type F: 0.1000
```

Part 5

```
D:\Term 6\01.112 - ML\Project\FR>python evalResult.py dev.out dev.p5.out

#Entity in gold data: 238
#Entity in prediction: 44

#Correct Entity : 32
Entity precision: 0.7273
Entity recall: 0.1345
Entity F: 0.2270

#Correct Entity Type : 22
Entity Type precision: 0.5000
Entity Type recall: 0.0924
Entity Type F: 0.1560
```

CN

Part 2

```
D:\Term 6\01.112 - ML\Project\CN>python evalResult.py dev.out dev.p2.out

#Entity in gold data: 1081
#Entity in prediction: 5161

#Correct Entity : 602
Entity precision: 0.1166
Entity recall: 0.5569
Entity F: 0.1929

#Correct Entity Type : 354
Entity Type precision: 0.0686
Entity Type recall: 0.3275
Entity Type F: 0.1134
```

Part 3

```
D:\Term 6\01.112 - ML\Project\CN>python evalResult.py dev.out dev.p3.out

#Entity in gold data: 1081
#Entity in prediction: 478

#Correct Entity : 198
Entity precision: 0.4142
Entity recall: 0.1832
Entity F: 0.2540

#Correct Entity Type : 149
Entity Type precision: 0.3117
Entity Type recall: 0.1378
Entity Type F: 0.1911
```

SG

Part 2

```
D:\Term 6\01.112 - ML\Project\SG>python evalResult.py dev.out dev.p2.out

#Entity in gold data: 4092
#Entity in prediction: 8775

#Correct Entity : 1902
Entity precision: 0.2168
Entity recall: 0.4648
Entity F: 0.2956

#Correct Entity Type : 1128
Entity Type precision: 0.1285
Entity Type recall: 0.2757
Entity Type F: 0.1753
```

Part 3

```
D:\Term 6\01.112 - ML\Project\SG>python evalResult.py dev.out dev.p3.out

#Entity in gold data: 4092
#Entity in prediction: 1926

#Correct Entity : 711
Entity precision: 0.3692
Entity recall: 0.1738
Entity F: 0.2363

#Correct Entity Type : 409
Entity Type precision: 0.2124
Entity Type recall: 0.1000
Entity Type F: 0.1359
```