

01.112 Machine Learning, Fall 2018

Design Project Report

By

Vishal Ramanathan

Kasper Halme

Shobhit Narayanan

Contents

[Part 1 3](#_Toc532163072)

[How to run the code 3](#_Toc532163073)

[Part 2 4](#_Toc532163074)

[Part 3 4](#_Toc532163075)

[Part 4 5](#_Toc532163076)

[Part 5 5](#_Toc532163077)

[Results 6](#_Toc532163078)

[EN 6](#_Toc532163079)

[Part 2 6](#_Toc532163080)

[Part 3 6](#_Toc532163081)

[Part 4 6](#_Toc532163082)

[Part 5 6](#_Toc532163083)

[FR 7](#_Toc532163084)

[Part 2 7](#_Toc532163085)

[Part 3 7](#_Toc532163086)

[Part 4 7](#_Toc532163087)

[Part 5 7](#_Toc532163088)

[CN 8](#_Toc532163089)

[Part 2 8](#_Toc532163090)

[Part 3 8](#_Toc532163091)

[SG 8](#_Toc532163092)

[Part 2 8](#_Toc532163093)

[Part 3 8](#_Toc532163094)

# 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 Goole 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 -> 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 -> 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

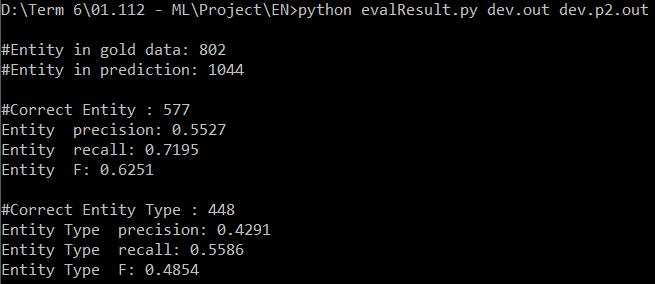
To improve performance, we tried to model a perceptron based on Part 3 implementation to update the emission and transition probabilities to improve our results.

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.

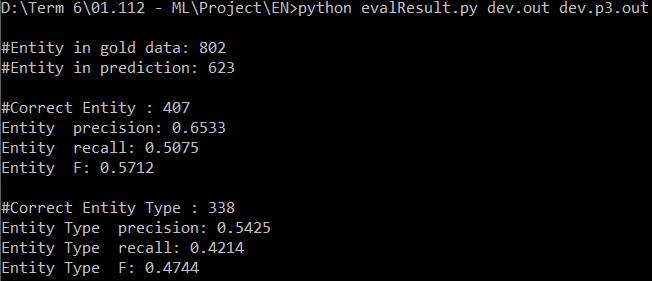
# Results

## EN

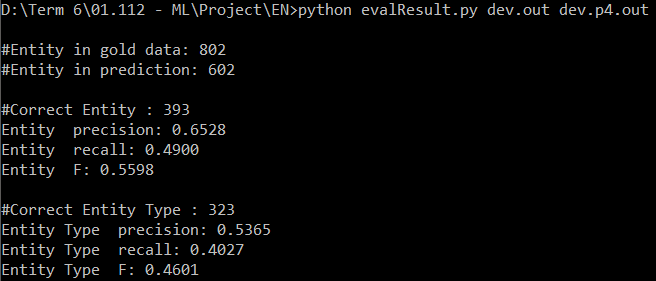
### Part 2



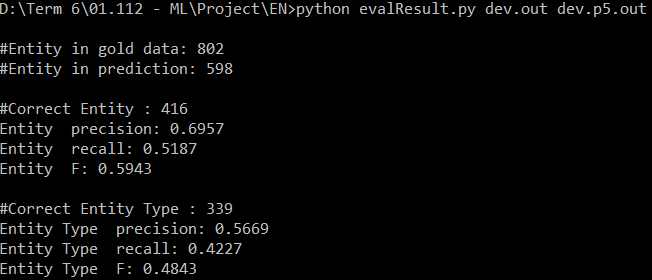
### Part 3



### Part 4

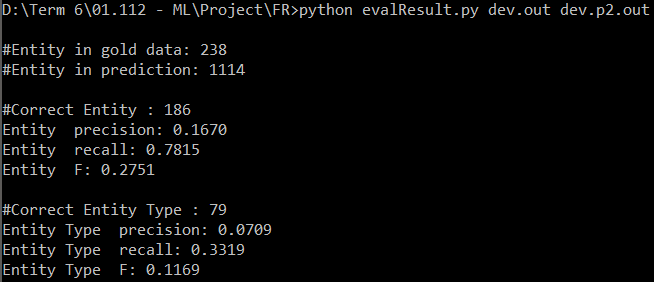


### Part 5

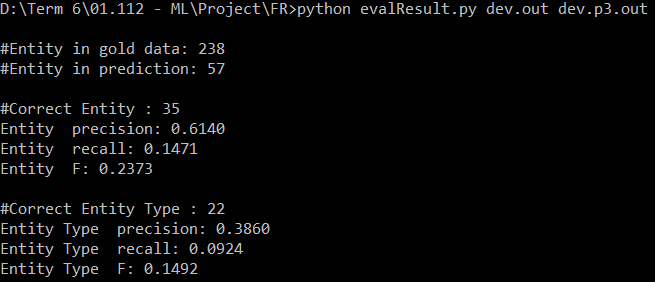


## FR

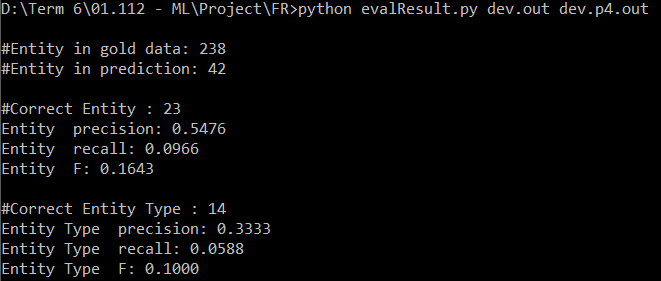
### Part 2



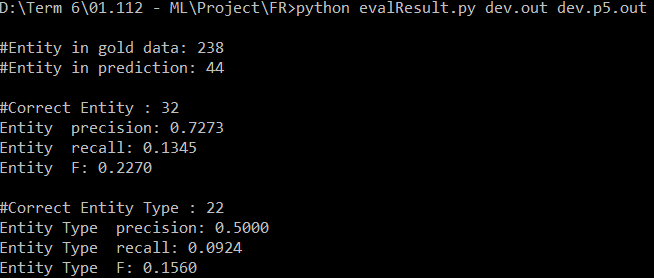
### Part 3



### Part 4

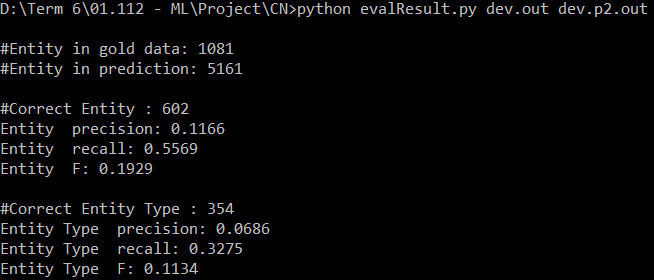


### Part 5

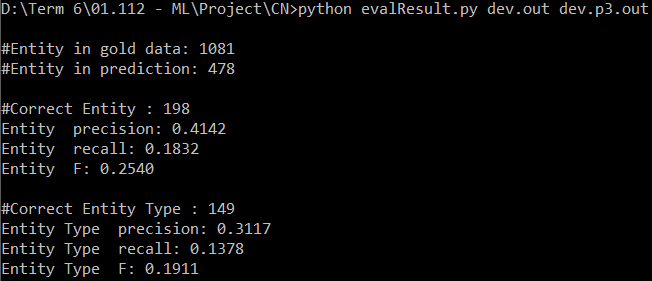


## CN

### Part 2

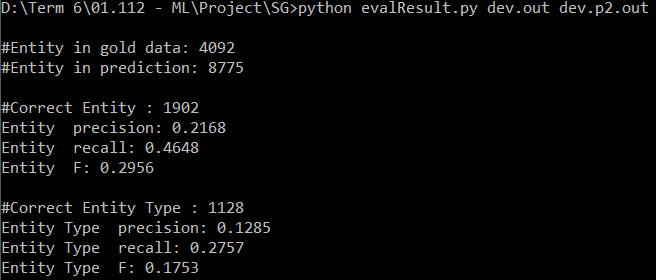


### Part 3



## SG

### Part 2



### Part 3

