Machine Learning Project Report
Group 11

Chenyan 1003620
Pang Luying 1003631
Gladys Chua 1003585

Part 2

How to Run the code

click run all cells for **part2.ipynb**, and the dev.p2.out file will be generated in the following folder eg. ./EN(2)/dev.p2.out, ./SG(1)/dev.p2.out, ./CN/dev.p2.out.
To change the input files, they can be changed by changing the path of the file at the last cell before the 'testing' header and the very last cell in **part2.ipynb**.

Approach

Emission parameters, $e(x|y) = \frac{count\ (y \rightarrow x)}{count\ (y)}$

A series of dictionaries is used to store the observations and tags.
For training,

1. trainingDataArray: numpy array of the data in train
2. states dictionary: Its keys are the tags and the value is a list of the total count of the tag and a string with the positions of where the observations were found.
3. all_the_words dictionary: the key is the tag while its value is a list of the observations that appeared with the tag
4. numerator dictionary: the key is a string such as "word state", where word is the observation and state is the tag. the values is the total count of the word→state
    a. "#UNK# state" is added in as a key with its value as k
5. emissionParameters dictionary: the key is the same as the numerator dictionary, but the values is $\frac{count\ (y \rightarrow x)}{count\ (y)+k}$
    a. for keys with "#UNK state", the value is $\frac{k}{count\ (y)+k}$
6. getEmissionParameters(inputFile,k): this function takes in the input file which is the train file given to us and the k value which is the occurrence of an event when a word is not found in the training set for the validation set.
    a. wordInTraining.txt is generated for use in testing

For testing,

1. testForEmissionParameters(inputFile, train_words, b_uo): this function take in the dev.in file as inputFile and train_words is a .txt file that was generated from getEmissionParameter() which enables the test set to find out the unknown words and b_uo is the dictionary with the emission parameters from train.
    a. dev.p2.out file is generated

Result

For EN
#Entity in gold data: 13179
#Entity in prediction: 18650

#Correct Entity : 9542
Entity precision: 0.5116
Entity recall: 0.7240
Entity F: 0.5996

#Correct Sentiment : 8456
Sentiment precision: 0.4534
Sentiment recall: 0.6416
Sentiment F: 0.5313

For SG
#Entity in gold data: 4301
#Entity in prediction: 12682

#Correct Entity : 2427
Entity precision: 0.1914
Entity recall: 0.5643
Entity F: 0.2858

#Correct Sentiment : 1805
Sentiment precision: 0.1423
Sentiment recall: 0.4197
Sentiment F: 0.2126

For CN
#Entity in gold data: 700
#Entity in prediction: 4747

#Correct Entity : 378
Entity precision: 0.0796
Entity recall: 0.5400
Entity F: 0.1388

#Correct Sentiment : 186
Sentiment precision: 0.0392
Sentiment recall: 0.2657
Sentiment F: 0.0683

Part3

How to Run the code

The code is inside the jupyter file named part3.ipynb. You can run all code at once by simply press the run all button in jupyter notebook

Implementation Process

Step1.Load and Store data

We stored the input data in List[List[Str]] (1st List include all sentences in a file 2nd List include all words in one sentence,Str is one of the world in each sentence)(Remember to change to your local path by yourself)

Step2. Data Preprocessing

We get training data by get_data() method and get test data by get_tdata() method.For the data preprocessing, we convert string features into numerical features by bag of words.

Step3. Defined HMM class by using Viterbi algorithm

Function: __init__(self, tf)

- Read data and create vocab
- Create values for our start of sentence, end of sentence, and sentence padding special tokens.What the above states is that our stat of sentence token (literally 'SOS') will take index spot '1' in our token lookup table once we make it. Likewise, the end of sentence ('EOS') will take the index spot '2'.
- Transfer text to int and store in self.x and self.y. The word2index variable is a dictionary to hold word token to corresponding word index value

Function: train(self)

- Run transition_para(self) to get transition parameter matrix

  1.1 Rows: transition from current state : (v1, v2, ..., 'SOS')

  1.2 Cols: transition to next state :   (v1, v2, ..., 'EOS')

  1.3 Label tags transition from position 0 to len(yvalue)

  1.4 Initialing a transition matrix and Loop each yvalue in y list for the length if yvalue, count the number to calculate the Denominator of transition matrix.

  1.5 Use below function to calculate transition parameter

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

- Run emission_para(self) to get emission parameter matrix

Function: predict_top_k(self, inpfile, outfile, k=1)

- This is our decoding part, we use this function to label testing data
    1. Initialize the last pointer backward
    2. Initialize path as an array
    3. Reverse path to get right order of sequence
    4. Store the path in path
    5. Write to file dev.p3.out and save it in directory

Function: viterbi(self, x, k=1)

We exploit this structure in a dynamic programming algorithm.

- Initialize score, argmax as numpy array.
- Calculate Pi value through loop each tags for each words by following formula

$$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\}$$

- We set K=1 because we only need to get max Pi value for each word.
- Finally, we calculate the last transition from yn to STOP by the following function

$$\max_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_0 = \mathrm{START}, y_1, \ldots, y_n, y_{n+1} = \mathrm{STOP}) = \max_{v \in \mathcal{T}} \{\pi(n, v) \cdot a_{v,\mathrm{STOP}}\}$$

Testing Code :

1. Run HMM model for training set
2. Conduct hmm.train() method
3. Use predict_top_k(dev_x_EN, ENpart4_out, k=1) to get output label file.
4. Run evalResult.py with our output file with standard file to get precision, recall and F1-Score

Example code :

```
#HMM MODEL -----Test EN
hmm = HMM(Train_EN)
hmm.train()
ENpart4_out = './EN(2)/dev.p7.out'
hmm.predict_top_k(dev_x_EN, ENpart4_out, k=1)
```

Results for SG

```
#HMM MODEL -----Test SG
hmm = HMM(Train_SG)
hmm.train()
hmm.predict_top_k(dev_x_SG, SGpart3_out, k=1)
```

```
!python3 evalResult.py SGdev.out SGdev.p3.out
```

```
#Entity in gold data: 4301
#Entity in prediction: 3125

#Correct Entity : 1901
Entity  precision: 0.6083
Entity  recall: 0.4420
Entity  F: 0.5120

#Correct Sentiment : 1686
Sentiment  precision: 0.5395
Sentiment  recall: 0.3920
Sentiment  F: 0.4541
```

Results for EN

```
#HMM MODEL -----Test EN
hmm = HMM(Train_EN)
hmm.train()
hmm.predict_top_k(dev_x_EN, ENpart3_out, k=1)
```

```
!python3 evalResult.py Endev.out ENdev.p3.out
```

```
#Entity in gold data: 13179
#Entity in prediction: 12733

#Correct Entity : 10800
Entity  precision: 0.8482
Entity  recall: 0.8195
Entity  F: 0.8336

#Correct Sentiment : 10387
Sentiment  precision: 0.8158
Sentiment  recall: 0.7881
Sentiment  F: 0.8017
```

Results for CN

```
#HMM MODEL -----Test CN
hmm = HMM(Train_CN)
hmm.train()
hmm.predict_top_k(dev_x_CN, CNpart3_out, k=1)
```

```
!python3 evalResult.py CNdev.out CNdev.p3.out
```

```
#Entity in gold data: 700
#Entity in prediction: 311

#Correct Entity : 106
Entity  precision: 0.3408
Entity  recall: 0.1514
Entity  F: 0.2097

#Correct Sentiment : 71
Sentiment  precision: 0.2283
Sentiment  recall: 0.1014
Sentiment  F: 0.1405
```

Part4

Run the code

The code is inside the jupyter file named part4.ipynb. You can run all code at once by simply press the run all button in jupyter notebook

Aproach

Functions:

- load_data(path): input a file path, output a list of tags and a list of observations. if there are no tags in the file, it will return an empty list.
- tag_lib(tags): input a list of tags, output a list of all possible types of tags in the file.
- obs_lib(obs):input a list of observations, output all types of words in the file
- map(lib): input a list, output a dictionary where keys are the element in the list and values are the corresponding indexes in the list
- label(data,dictionary): input a list and a dictionary, output a list of of a list indexes of each observations
- get_emission_para(obslabel,taglabel,obslib,taglib): input observation label list, tag label list, observation library, tag library, output a matrix of emission parameters, where columns are words including #UNK#, rows are tags
- get_transition_para(obslabel,taglabel,obslib,taglib): input observation label list, tag label list, observation library, tag library, output a matrix of transition parameters, where columns are tags, rows are tags, including START and STOP tags.
- log(x,inf_replace=-100): input matrix, output matrix with all the logarithm values of all the elements and replace the negative infinity to -100.
- topk(observation,transition_matrix,emission_matrix,k,obs_dict): function to get a matrix of top k value where each entry stores top k value from start to the entry. Input one observation, transition parameters, emission parameters, observation dictionary and the k value. NOTE: code is algorithm based on viterbi algorithm but for our convenience, we change the multiplication to summation with the input of logarithm value of transition and emission parameters.
    - pseudo code:
        1. initialize pi to be a numpy array with rows=number of tags, columns= number of words in the observation
        2. for each word in the observation:

```
            if it is the first word:
                if the word is not  #UNK#, pi[:,0]=transition of START to all
    tags+emission of all tags to the word
                if the word is #UNK#, pi[:,0]=transition of START to all tags+ emission
    of all tags to #UNK#
            if it is no the first word:
                for each tag in tags:
                    if the word is not  #UNK#: temp_pi=pi[:,previous
    column]+transition of all tags to the tag+ emission of the tag to the word
                    if the word is #UNK#, ]=temp_pi=pi[:,previous
    column]+transition of all tags to the tag+ emission of the tag to the word
                    assign top k_th value to pi[tag,word]
        return pi
```

- find_path(observation,pi,emission_matrix,transtion_matrix,k,tag_lib,tag_dict): input observation, matrix pi, emission parameter, transition parameter, tags, tag dictionary, and value k, output list of tags.

Complete steps:

1. load train data using load_data()

2. get all possible words using obs_lib(), get all possible tags using tag_lib(), get tag dictionary and word dictionary using map(), and get observation label and tag label using label()
3. get transition parameter, get emission parameter by calling get_transiton_para() and get_emission_para()
4. transform emission parameters and transition parameters using log()
5. load dev.in using load_data() to get test_observations
6. initialize list pilst to store all pi of all observations, initialize list value to store top k value of each observation
7. for each observation in observations list:
    a. get pi of each observation by calling topk()
    b. add pi to pilst and top k value to value
8. initialize pathlst to store paths of all observations
9. for each observation in test_observations, for each pi in pilst:
    a. find path by calling find_path()
    b. reverse path to get right order of sequence
    c. store the path in pathlst
10. write to file dev.p4.out and save it in EN directory

Results

For EN:

#Entity in gold data: 13179
#Entity in prediction: 26059

#Correct Entity : 7271
Entity precision: 0.2790
Entity recall: 0.5517
Entity F: 0.3706

#Correct Sentiment : 193
Sentiment precision: 0.0074
Sentiment recall: 0.0146
Sentiment F: 0.0098

Part 5

Run the code

The code is inside the jupyter file named part5.ipynb. You can run all code at once by simply press the run all button in jupyter notebook

Approach

We used k-means clustering as we have did in HW1 together with count vectorisation to vectorise each word. We will use some functions from part 4

- use the same load_data() function to get tags and observations
- get observation library, tag library by calling obs_lib() and tag_lib(), get tag dictionary, observation dictionary using map()
- loop through all observation in observations, count the number of each word appear in the observations and add to the right entry of words_matrix accordingly
- modify the words matrix by reducing the dimensionality of matrix
- input the modified matrix together with the tags into run_kmeans() function, and save the list of assignment of each word
- for each cluster, find the majority tags of the words and label the cluster with the tag by calculating the centroids of the cluster
- predict tag of each word by calculating the distance of each input word vector to the centroids, label the word with the tag that has smallest distance among all tags.