50.007 MACHINE LEARNING
PROJECT REPORT

You Hongzhou
Yang Wei
Yuan Wolong

December 7, 2016

1. Part 1

Omitted.

2. Part 2

(a) We first use maximum likelihood estimation to estimate the emission parameters from the training set.

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

In our implementation, we use a map $count\_xy$ and an array $count\_y$ to store $Count(y \to x)$ and $Count(y)$ respectively. In the $train$ function, we initialize $count\_xy$ and $count\_y$ from training set. And for each natural language word x and tag y, we get the corresponding emission parameter as

$$e(x|y) = \frac{count\_xy[(y, x)]}{count\_y[y]}$$

The function in our final code is as follows:

```
double emission_MLE(wstring x, int y) {
        return (double)count_xy[y_x(y, x)] / count_y[y];
}
```

(b) According to the idea issued in this problem, we implement the $emission\_fixed$ function to improve the computing of emission parameters. For each natural language word x and tag y,

If x did not appear in the training set:

$$e(x|y) = \frac{1}{count\_y[y] + 1}$$

If x appeared in the training set:

$$e(x|y) = \frac{count\_xy[(y, x)]}{count\_y[y] + 1}$$

The function in our final code is as follows, where appear is a set to store the words that have appeared:

```
double emission_fixed(wstring x, int y) {
        if (appear.find(x) == appear.end()) {
                return 1.0 / (count_y[y] + 1);
        }
        return (double)count_xy[y_x(y, x)] / (count_y[y] + 1);
}
```

(c) In the train phase of this simple sentiment analysis system, we initialize *count_xy* and *count_y* from training set. In the test phase, we read words from *dev.in* files and make predications word by word. For each word x, we calculate emission parameters for all 7 tags. Then we choose the tag with highest emission parameter as predicated tag y and write it to *dev.p2.out* file.

For each dataset, we evaluate our system using the *evalResult.py* script. We compare our outputs with the gold-standard outputs and get the precision, recall and F scores of such a baseline system for each dataset as below.

EN	precision	recall	F score
Entity	0.1350	0.5423	0.2162
Sentiment	0.0417	0.1677	0.0668

Table 1: Results of Part 2 in EN dataset

CN	precision	recall	F score
Entity	0.0925	0.5337	0.1577
Sentiment	0.0401	0.2310	0.0683

Table 2: Results of Part 2 in CN dataset

ES	precision	recall	F score
Entity	0.1408	0.5784	0.2264
Sentiment	0.0451	0.1855	0.0726

Table 3: Results of Part 2 in ES dataset

SG	precision	recall	F score
Entity	0.1759	0.5315	0.2643
Sentiment	0.0537	0.1624	0.0807

Table 4: Results of Part 2 in SG dataset

3. Part 3

(a) In this part, we use maximum likelihood estimation to estimate the transition parameters from the training set.

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

In our implementation, we use a two-dimensional array *count_yy* to store the transition parameters. We also use two arrays *count_start_y* and *count_y_stop* to store $q(y_1|\text{START})$ and $q(\text{STOP}|y_n)$ respectivily. The variable *count_start* represents the number of sentences. For two tags (including START and STOP) u and v,
If u is START:

$$q(v|u) = \frac{count\_start\_y[v]}{count\_start}$$

If v is STOP:

$$q(v|u) = \frac{count\_y\_stop[u]}{count\_y[u]}$$

Else:

$$q(v|u) = \frac{count\_yy[u][v]}{count\_y[u]}$$

The function in our final code is as follows:

```
double transition_MLE(int u, int v) { // -1: START; -2: STOP
        if (u == -1) {
                return (double)count_start_y[v] / count_start;
        }
        if (v == -2) {
                return (double)count_y_stop[u] / count_y[u];
        }
        return (double)count_yy[u][v] / count_y[u];
}
```

(b) In our implementation of Viterbi algorithm, we use two arrays pu and pv to store scores of all tags for previous word and current word. We also use a vector fa to store the index of tag with highest score for each word in a sentence. For each dataset, we learn the emission parameters and transition parameters with *train*. Then in the test phase, for each sentence, we get predicted tags using the Viterbi algorithm we implement. We run this model on the *dev.in* files and write outputs to *dev.p*3.*out* files.

In case to encounter potential numerical underflow issue, we take log value to all scores, and in the same time modify all multiplication to addition when calculating the scores.

Then we evaluate our system using the *evalResult.py* script. We compare our outputs with the gold-standard outputs and get the precision, recall and F scores of such a baseline system for each dataset as below.

EN	precision	recall	F score
Entity	0.2310	0.3535	0.2794
Sentiment	0.1066	0.1631	0.1290

Table 5: Results of Part 3 in EN dataset

CN	precision	recall	F score
Entity	0.2873	0.4139	0.3392
Sentiment	0.1707	0.2460	0.2016

Table 6: Results of Part 3 in CN dataset

ES	precision	recall	F score
Entity	0.2548	0.3876	0.3075
Sentiment	0.1319	0.2006	0.1591

Table 7: Results of Part 3 in ES dataset

SG	precision	recall	F score
Entity	0.3081	0.2934	0.3006
Sentiment	0.1000	0.0952	0.0975

Table 8: Results of Part 3 in SG dataset

4. Part 4

We modify the Viterbi algorithm to find the top-k best output sequences. Instead of using two one-dimensional arrays, here we use two two-dimensional arrays pu and pv. Besides, fa is also 2-dimensional so that we can store top-k best tags.
For dataset EN and ES, we evaluate our system using the *evalResult.py* script. We compare our outputs with the gold-standard outputs and get the precision, recall and F scores of such a baseline system for each dataset as below.

EN	precision	recall	F score
Entity	0.1985	0.3958	0.2644
Sentiment	0.0841	0.1677	0.1120

Table 9: Results of Part 4 in EN dataset

ES	precision	recall	F score
Entity	0.2149	0.4238	0.2852
Sentiment	0.0918	0.1810	0.1218

Table 10: Results of Part 4 in ES dataset

5. Part 5

What we improve in the HMM model is to consider not just the highest-scored prediction but the top-k results (where we fix k to 10 in this case). We add the i-th result a weight $\frac{1.1-0.1i}{0.5+0.5i}$, and sum up all weight of all prediction for each word and then give the highest-weighted prediction as the result.
For dataset EN and ES, we evaluate our system using the *evalResult.py* script. We compare our outputs with the gold-standard outputs and get the precision, recall and F scores of such a baseline system for each dataset as below.

EN	precision	recall	F score
Entity	0.2310	0.3580	0.2808
Sentiment	0.1072	0.1662	0.1303

Table 11: Results of Part 5 in EN dataset

ES	precision	recall	F score
Entity	0.2563	0.3922	0.3100
Sentiment	0.1326	0.2029	0.1604

Table 12: Results of Part 5 in ES dataset