Gede Ria Ghosalya – 1001841
01.112 Machine Learning Project

Part II

The code simply reads the training file, and converts them into a list of <word, tag> tuple. From this data, the code generates *emission parameter* estimates by collection <word, tag> counts and <tag> counts, and then dividing the two numbers to get estimated probabilities.

The simple sentiment analysis simply choose, for each word, the tag with the highest probability in the *emission parameter*, regardless of sequence.

This code is in `simple.py`. Here are the results.

Country		Count	Correct Instance	Precision	Recall	F-Score
CN	Entity	1848 (362 Gold)	117	0.0633	0.3232	0.1059
	Sentiment		72	0.0390	0.1989	0.0652
EN	Entity	757 (226 Gold)	139	0.1836	0.6150	0.2828
	Sentiment		47	0.0621	0.2080	0.0956
SG	Entity	2970 (1382 Gold)	527	0.1774	0.3813	0.2422
	Sentiment		274	0.0923	0.1983	0.1259
FR	Entity	754 (223 Gold)	184	0.2440	0.8251	0.3767
	Sentiment		72	0.0955	0.3229	0.1474

Part III

The code implements the Hidden Markov Model to predict the tag sequence. First of all, it generates the *transition parameter* estimates by collecting, for each tweet, all tag transition <current_tag, next_tag> counts, as well as tag counts, and then dividing the two numbers to get estimated probabilities.

While predicting tag sequence, the code uses Viterbi Algorithm that is memoized so as to save time during the recursion. The viterbi function, if the *stage* inputted is 0, returns 1 if the tag is “START”, and 0 otherwise. When the *stage* is not 0, it will loop over all possible tags, get the viterbi value of the previous *stage* being of the looped tag, and get the transition & emission parameter before multiplying these numbers. If the <word, tag> emission pair or <tag, next_tag> pair is not found in the parameter, 0 is used instead.

This code is in `hmm.py`. Here are the results (which is overall better than the simple analysis).

Country		Count	Correct Instance	Precision	Recall	F-Score
CN	Entity	158 (362 Gold)	64	0.4051	0.1768	0.2462
	Sentiment		47	0.2975	0.1298	0.1808
EN	Entity	162 (226 Gold)	104	0.6420	0.4602	0.5361
	Sentiment		64	0.3951	0.2832	0.3299
SG	Entity	723 (1382 Gold)	386	0.5339	0.2793	0.3667
	Sentiment		244	0.3375	0.1766	0.2318
FR	Entity	166 (223 Gold)	112	0.6747	0.5022	0.5758
	Sentiment		72	0.4337	0.3229	0.3702

Part IV

The idea for max-marginal decoding algorithm is as follows. In a sequence, for a tag *u* to be at position *i*, the probability is as follows:

$$P(y_i=u|X)=\frac{P(x_1,x_2,\dots,x_{i-1},y_i=u,x_i,\dots,x_n;\emptyset)}{P(x_1,\dots,x_n;\emptyset)}$$

which can be split into:

$$P(y_i=u|X)=\frac{P(x_1,x_2,\dots,x_{i-1},y_i=u)*P(x_i,\dots,x_n|y_i=u;\emptyset)}{P(x_1,\dots,x_n;\emptyset)}$$

as with HMM, the goal is to find a tag *u* that has the highest probability to be in the position *i*. Keeping this in mind, the code uses 2 dynamically programmed recursive function to calculate the left-side and the right-side of the multiplication, referred to as *alpha* and *beta* function. These functions can be recursed because:

$$\begin{gathered}alpha(i)\\=P(x_1,x_2,\dots,x_{i-1},y_i=u)\\=P(x_1,x_2,\dots,x_{i-2},y_{i-1}=u)*P(x_{i-1}|y_{i-1})*P(y_i=u|y_{i-1})\\=alpha(i-1)*P(x_{i-1}|y_{i-1})*P(y_i=u|y_{i-1})\end{gathered}$$

same goes for beta. The code hence loops through the positions and return the most likely tag given the word.

This code is in `maxmarginal.py`. Here are the results, which is a little less accurate than HMMs.

Country		Count	Correct Instance	Precision	Recall	F-Score
EN	Entity	159 (226 Gold)	100	0.6289	0.4425	0.5195
	Sentiment		59	0.3711	0.2611	0.3065
FR	Entity	168 (223 Gold)	112	0.6667	0.5022	0.5729
	Sentiment		73	0.4345	0.3274	0.3734

Part V

The idea is simple. The current structure rigidly attach an entity symbol (O, B, I) to a sentiment (positive, negative, neutral). So the code actually separates them, such that each word has two unrelated tags: entity tag and sentiment tags. Simply put, the code separates the tags in the file, and treat them as separate tags to feed into the HMM predictor.

Once both entity and sentiment sequence is predicted, the next step is to combine them. We know that only B and I tags will have sentiment, and that those will always have a sentiment attached to them. The code will ignore the sentiment value if the entity tag is O, and if there is any B or I tag with sentiment `none`, the sentiment tag for that position is replaced with the most common sentiment in the tweet (`neutral` if there is no sentiment in the sequence).

This code is in `dual_hmm.py`. As can be seen, it predicted more entities, and overall performs slightly better (+5-8%) in terms of F-score. Precision seems to suffer a little; this is because detaching sentiment from tag result in more entity prediction made. Some entities that are not picked up by normal HMM will be picked up as entity with neutral sentiment in this model.

Country		Count	Correct Instance	Precision	Recall	F-Score
EN	Entity	207 (226 Gold)	130	0.6280	0.5752	0.6005
	Sentiment		75	0.3623	0.3319	0.3464
FR	Entity	212 (223 Gold)	143	0.6745	0.6413	0.6575
	Sentiment		78	0.3679	0.3498	0.3586