01.112 Machine Learning

Yeoh Chuen Wen 1001775 | Christabella Adelina 1001620 | Aditya Manikashetti 1001819

Design Project

Part 2: Simple Sentiment Analysis

I. Approach

The simple sentiment analysis produces the most probable tag sequence for a given token sequence by choosing the highest emission probability of a tag for each given token. Unrecognised words in the input file (a word that appears less than k=3 times, the learning threshold) will use the emission probability for the general token #UNK#.

For every input token, our algorithm checks if the token is found in our emission probability table. If it is found in the emission probability table, the tag which produces the highest emission probability for this token will be recorded. If the token is not found in the table, the tag which produces the highest emission probability for the unknown token #UNK# is used instead. The predicted tag for each token is compiled into a list resulting in a predicted tag sequence for a given token sequence.

$$y* = \arg\max_{y} e(x \mid y)$$

The simple method takes the input file tokens and the emission parameters and outputs as a predicted tag sequence (for a given token sequence).

II. Result

	EN	FR	CN	SG
Entity in gold data	226	223	362	1382
Entity in prediction	1201	1149	3318	6542
Correct entity	165	182	183	780
Entity precision	0.1374	0.1584	0.0552	0.1192
Entity recall	0.7301	0.8161	0.5055	0.5644
Entity F	0.2313	0.2653	0.0995	0.1969
Correct sentiment	71	68	57	311
Sentiment precision	0.0591	0.0592	0.0172	0.0475
Sentiment recall	0.3142	0.3049	0.1575	0.2250
Sentiment F	0.0995	0.0991	0.0310	0.0785

Part 3: Viterbi Algorithm

I. Approach

The Viterbi algorithm produces the most probable tag sequence for a given token sequence. This is done by choosing the tag sequence (calculation of the transition parameters) with the highest probability of producing the given token sequence (calculation of the emission parameters). Similarly, unrecognised words in the input file (words that appear less than k=3 times, the learning threshold) will use the emission probability for the general token #UNK#.

$$r(y_1, \ldots, y_k) = \prod_{i=1}^{k} a_{y_{i-1}, y_i} \prod_{i=1}^{k} b_{y_i}(x_i)$$

$$p(x_1, \ldots, x_n, y_0, y_1, \ldots, y_n) = r(y_1, \ldots, y_n) \bullet a_{y_n, y_{n+1}} = r(y_1, \ldots, y_n) \bullet a_{y_n, \mathrm{STOP}}$$

$$\pi(k, v) = \max_{(y_1, \ldots, y_n) \epsilon\, S(k, v)} r(y_1, \ldots, y_k)$$

Where S(k,v) is the set of tag sequences y1,…, yk such that yk =v.

$$\pi(0,\mathrm{START}) = 1$$

$$\pi(1, v) = \max_{u \epsilon T} \{\pi(0,\mathrm{START}) \bullet a_{\mathrm{START},\, u} \bullet b_v(x_k)\}$$

The algorithm is initialized whereby for the first input token, we calculate the product of the transition probability from the 'START' tag to a possible tag and the emission probability of the same tag to the input token. This probability is calculated for every possible tag recursively:

$$\pi(\mathrm{k}, v) = \max_{u \epsilon T} \left\{ \pi(k-1, u) \bullet a_{u,v} \bullet b_v(x_k) \right\}$$

The next token in the sequence calculates the product of the transition and emission probabilities for each possible tag along with the total probability for the previous tag. This is repeated until the last word of sequence, where the calculation involves only the probability for the previous tag and the transition probability between the previous tag and the 'STOP' tag.

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

$$y_n^* = \arg\max_{v} \{\pi(n, v) \bullet\ a_{v,\mathrm{STOP}}\}$$

$$y_{n-1}^* = \arg\max_{u} \{\pi(n-1, u) \bullet\ a_{u, y_n^*}\}$$

$$y_1{}^*, \ldots, y_n{}^* = \arg\max_{y_1, \ldots, y_n} p(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

The Viterbi method takes the input file tokens, the transition parameters and the emission parameters and outputs a predicted tag sequence (for a given token sequence).

II. Result

	EN	FR	CN	SG
Entity in gold data	226	223	362	1382
Entity in prediction	162	166	158	717
Correct entity	104	112	63	382
Entity precision	0.6420	0.6747	0.3987	0.5328
Entity recall	0.4602	0.5022	0.1740	0.2764
Entity F	0.5361	0.5758	0.2423	0.3640
Correct sentiment	64	72	46	242
Sentiment precision	0.3951	0.4337	0.2911	0.3375
Sentiment recall	0.2832	0.3229	0.1271	0.1751
Sentiment F	0.3299	0.3702	0.1769	0.2306

Part 4: Max-Marginal Decoding Algorithm

I. Approach

The max-marginal decoding algorithm calculates the forward and backward scores for a given token at a given position in the sequence. The alpha or the forward scores, $\alpha_u(j)$ are calculated by taking the sum of scores of all paths from START to the state <j, u>. The beta values or the backward scores, $\beta_u(j)$ are calculated by taking the sum of scores of all paths from state <j, u> to the final state STOP. For a given token sequence, the forward and backward probabilities are calculated recursively for all possible path sequences.

The optimal tag y at the position i given the token sequence, is found by computing the marginal probability term:

$$p(y_i = u \,|\, x_1, \ldots, x_n; \theta) = \frac{p(x_1, x_2, \ldots, x_{i-1}, y_i = u, x_i, \ldots, x_n;\ \theta)}{p(x_1, \ldots, x_n;\ \theta)} = \frac{p(x_1, x_2, \ldots, x_{i-1}, y_i = u;\ \theta)p(x_i, \ldots, x_n \,|\, y_i = u;\ \theta)}{p(x_1, \ldots, x_n;\ \theta)}$$

$$= \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)}$$

$$y_i^* = \arg\max_u p(y_i = u \,|\, x_1, \ldots, x_n;\ \theta) = \arg\max_u \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)} = \arg\max_u \alpha_u(i)\beta_u(i)$$

The maxMarginal method takes the input file tokens, the transition parameters and the emission parameters and outputs a predicted tag sequence (for a given token sequence). Unlike the previous algorithms, our maxMarginal method takes the input token sequences, sentence by sentence (instead of all the input file tokens).

II. Result

	EN	FR
Entity in gold data	226	223
Entity in prediction	303	296
Correct entity	134	171
Entity precision	0.4422	0.5777
Entity recall	0.5929	0.7668
Entity F	0.5066	0.6590
Correct sentiment	79	96
Sentiment precision	0.2607	0.3243
Sentiment recall	0.3496	0.4305
Sentiment F	0.2987	0.3699

Part 5: Challenge – More efficient handling of data

I. Approach

We wanted to try and improve the max marginal model by filtering and transforming the data so it would yield more accurate results. We do this by:

a. Non-Case Sensitive
As we believe that an entity and sentiment analysis is not case-sensitive, we decided to convert all the word to lowercase

b. Classification
We further improved the emission probabilities, by classifying the unknown values into unknown word classes:
1. Numeric: Unknown word containing at least one numeric character
2. Symbols: Unknown word containing at least one symbol ($, (,), !, .,-, _, ', etc.)
3. Others: Unknown word that does not fit into the other classes

II. Result

	EN	FR
Entity in gold data	226	223
Entity in prediction	314	303
Correct entity	140	175
Entity precision	0.4459	0.5576
Entity recall	0.6195	0.7848
Entity F	0.5185	0.6654
Correct sentiment	81	110
Sentiment precision	0.2580	0.3630
Sentiment recall	0.3584	0.4933
Sentiment F	0.3000	0.4183

Previous implementation

EN	FR
226	223
303	296
134	171
0.4422	0.5777
0.5929	0.7668
0.5066	0.6590
79	96
0.2607	0.3243
0.3496	0.4305
0.2987	0.3699

As can be seen, the model performs better on an average.

Calculating Emission Parameter

Each emission parameter $b_y(x) = p(X = x \mid Y = y)$ for $y = 1, \dots, \left|T\right| - 2$ and $x \in X$, is the probability of emitting word *x* given a tag *y:* $\sum_{x \in X} b_y(x) = 1$ for any y. The emission parameters are essentially calculated by counting the number of times a word *x* has a tag *y*, this result is then divided by the number of times tag *y* appears in the input file.

$$e\left(x \middle| y\right) \to \frac{Count(y \to x)}{Count(y)}$$

The calculateEmission method takes the input file tags, tokens (words) and a learning threshold (to determine tokens to be classified as unknown) and outputs the emission parameters.

Calculating Transition Parameter

Each transition parameter $a_{i,j} = p(y_{next} = j \mid y_{current} = i)$ for $i = 0, 1, \dots, \left|T\right| - 2$ and $j = 1, .., \left|T\right| - 1$, is the probability of transitioning from state *i* to state *j:* $\sum_{k=1}^{|T|-1} a_{i,k} = 1$ for any i. The transition parameters are thus calculated by counting the number of times a tag y_{i-1} transitions to tag y_i , this result is divided by the number of times tag y_{i-1} appears in the input file.

$$q\left(y_i \middle| y_{i-1}\right) \to \frac{Count(y_{i-1} \to y_i)}{Count(y_{i-1})}$$

The calculateTransition method takes the input file tags and outputs the transition parameters.

Miscellaneous

Parsing of Training Data

We separate the training data into two lists for training, one containing the input tokens and the other containing the input tags. This allows us to calculate the emission and transition parameters with greater ease.

Parsing of Input Data

The input data is stripped of white spaces and contains the input tokens (whose tag sequence is to be predicted).