01.112 Machine Learning

Design Project

# **Part 2: Simple Sentiment Analysis**

## 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.

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

## 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**

## 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#.

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

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:

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.

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).

## 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**

## 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, 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, 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:

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).

## 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**

## Approach

### 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

### 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

## 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 |

# **Calculating Emission Parameter**

Each emission parameter for and , is the probability of emitting word *x* given a tag *y:* 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.

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 for and , is the probability of transitioning from state *i* to state *j:* for any i. The transition parameters are thus calculated by counting the number of times a tag transitions to tag , this result is divided by the number of times tag appears in the input file.

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).