50.007 Machine Learning 2023 Spring Project

Group members

- Alphonsus Tan (1005534)
- Cheh Kang Ming (1005174)
- John Lim (1004642)

Part 1 - Emission Parameters

The formula for emission parameters are given as follow:

$$e(x|y) = \begin{cases} \frac{Count(y \rightarrow x)}{Count(y) + k}, \text{If the word token x appears in the training set} \\ \frac{k}{Count(y) + k}, \text{If word token x is the special token } \texttt{\#UNK\#} \end{cases}$$

Plan

We decided to store the emission probabilities of each hidden state emitting the unique words in the training set in a form of a 2d numpy array with size (T x O), with T being the number of hidden states and O being the size of unique observed words in the training set.

With that in mind, a zero matrix of size (T x O) was first initialized, and two lists of unique hidden states and observed values are used to keep track of the indexing of the matrix, so that we know which row in the matrix belongs to which hidden state and which column of the matrix belongs to which observed value.

Reading of Data

When reading the training data, we first check if we are reading an empty line, which we will ignore. We then separated the line into observed words and its respective tag. This would leave us with a list of all observed values from the training set and a list of their respective tags in the

same order. We can then generate our lists of unique hidden states and observed values as mentioned above from the data we collected.

Emission Parameters

In order to get the total count of a hidden state emitting an observed value, we iterate through the observed values and their respective tags from the training set, and increment their count in their respective position in the emission matrix. For aid of understanding, emission_matrix[i][j] would store the total count of hidden_state[i] emitting observed_value[j] after the iteration.

The total count of each hidden state appearing in the training set is stored in a dictionary with key being the hidden state and value being their respective count.

We then update the emission matrix from above to store the emission probability instead of the emission count by applying the formula provided.

Laplace Smoothing

To take into account of words in the test data not appearing in the training data, we introduced a #UNK# token for each hidden state such that each hidden state would have a probability of emitting words that are not seen in the training data. The probability of each state emitting an unseen word is then calculated based on the formula given and appended to the emission matrix.

Testing

To test our emission parameters, each word in the test data will be tagged to the state that has the highest probability of emitting the word based on our emission matrix. The result will then be written into an output file which then can be compared with the actual correct tagging of each word and the f score for our emission parameters can be calculated.

Result

The following f score shows the accuracy of our emission parameters for the EN and FR training and testing set.

EN	FR
#Entity in gold data: 802 #Entity in prediction: 1148 #Correct Entity : 614 Entity precision: 0.5348	#Entity in gold data: 238 #Entity in prediction: 1114 #Correct Entity : 186 Entity precision: 0.1670

Entity recall: 0.7656 Entity F: 0.6297 #Correct Sentiment : 448 Sentiment precision: 0.3902 Sentiment recall: 0.5586 Sentiment F: 0.4595	Entity recall: 0.7815 Entity F: 0.2751 #Correct Sentiment : 79 Sentiment precision: 0.0709 Sentiment recall: 0.3319 Sentiment F: 0.1169

Part 2 - Transition Parameters and Viterbi

Transition Parameters

Transition parameters were based on the following equations:

$$a_{u,v} = \frac{count(u,v)}{count(U)}$$

This is used to build our transition probability matrix. The joint probability that gives us the predicted hidden state v for a given hidden state u, and observed state o.

$$P(j,v) = P(j-1,u) * a_{u,v} * b_v(j)$$

Prevent Underflow Issue

In order to prevent underflow, we use the log of the joint probability.

$$P(j,v) = P(j-1,u) + log(a_{u,v}) + log(b_v(j))$$

To avoid log errors when a transition probability that has a 0 value, we add a small constant to all the $count(u,v)$ values.

Handling START and STOP

When reading the training data, at the start and end of every sentence we add the START and STOP state to the array of hidden states. So when parsing this new array of hidden states we can track all the possible u and v state pairs. In our counting we ensure that START state cannot be v state and STOP state cannot be u state.

When transiting from START to y1, the probability is always 1. So the START column in the transition matrix is populated with value 1.

The Viterbi Algorithm

The Viterbi vertice matrix, (K * N+2) is populated iteratively from the n=1 to n=N+2. Each cell is the joint probability of the hidden kth state being predicted for nth in the sequence. This matrix is populated in the forward pass. The backtracking matrix, (K*K) where each cell is the most

likely previous u state given the current v state. It is populated during the forward state as we find v, we populate the previous u in the backtracking matrix. The u chosen is from the u,v pair with highest joint probability In the backward pass. Once we have completed the forward pass we have a populated vertice and backtracking matrix. We backtrack the sequence using the backtracking matrix, returning the sequence of hidden states with the highest probability. This forms our final prediction.

Formula to get highest probability for a give path

$$Forward\ pass\ \ \pi(u_{i+1}, v)\ =\ max_{u\in\tau}\ (\ \pi(u_i, v) + log(a_{u,v})\ \ + log(b_u(o))\ )$$

$$Backtracking\ y_i\ =\ argmax_{\ u\in\tau}\ (\ \pi(i,u) + log(a_{u,y_{i+1}})\ )$$

$$Backtracking\ last\ y_n\ =\ argmax_{u\in\tau}(\pi(n,u)\ *\ a_{u,STOP})$$

Tokens and Sentences

In the training, i.e the construction of the transition and emission probability matrices. We do not need to consider each sentence by itself, but the whole training dataset, as our base of $count(u)$ should be the whole dataset. But for the prediction dataset, we need to consider each sentence by itself, to account for the transition probability of START and STOP.

Result

EN	FR
#Entity in gold data: 802 #Entity in prediction: 821 #Correct Entity : 537 Entity precision: 0.6541 Entity recall: 0.6696 Entity F: 0.6617 #Correct Sentiment : 470 Sentiment precision: 0.5725 Sentiment recall: 0.5860 Sentiment F: 0.5792	#Entity in gold data: 238 #Entity in prediction: 445 #Correct Entity : 136 Entity precision: 0.3056 Entity recall: 0.5714 Entity F: 0.3982 #Correct Sentiment : 76 Sentiment precision: 0.1708 Sentiment recall: 0.3193 Sentiment F: 0.2225

Part 3 - Second-Order HMM

Transition Parameters

Since we are now considering second order transitions, we need to modify the transition matrix to account for all the possible triplets of states that exist. To begin, we use the the first order

transition matrix computed by $\frac{count(u,v)}{count(u)}$. The new transition parameters are then computed by using $\frac{count(t,u,v)}{count(u,v)}$. Additionally, since we are now considering up to two states prior, we need to create a new state before start, in order for the transition from START to the first token to be computed. This is done by appending PRESTART before start. This results in the following typical transition for each tweet: $PRESTART \rightarrow START \rightarrow y_0 \rightarrow \ldots \rightarrow y_n \rightarrow STOP$.

We did not use log probability as we did not encounter any underflow issue and the log values returned the same result for EN.

Emission Parameters

No change to the computation of emission parameters

Decoding with Viterbi

For the Viterbi algorithm, instead of considering u to be a single state, each u (row) in the Viterbi algorithm matrix represents a unique tuple from the set of states during training. To begin, the initialization is $\pi(j, (PRESTART, START)) = 1$. For each j from 1 to $N - 1$, where N is the total number of observations, we compute $\pi(j, (u, v)) = max_{(t,u)\in T}\{\pi(j, (t, u)) \times a_{t,u,v} \times b_v(x_j)\}$. The final step is $\pi(N, (u, STOP)) = max_{u\in T}\{\pi(n, (t, u)) \times a_{t,u,STOP}\}$. To facilitate backtracking, when we compute $\pi(j, (u, v))$ at each stage, we store the value of the grandparent t in a backtracking matrix. To backtrack, we simply determine which state it ends in using argmax of $\pi(N, (u, STOP))$ and from there trace the parent of u all the way back to the beginning.

Review of Results

We notice that the EN F scores are lower compared to the first-order HMM, we suspect it could be due to overfitting. FR has 7 hidden states compared with EN 18 hidden states. This means that the EN model has a higher complexity than the FR model. A higher complexity could mean that EN model would experience more overfitting than FR, hence when we use second order HMM to further increase the model complexity.

So the EN model may start to experience overfitting before the FR model causing its decrease in the F score. While the FR model is not experiencing overfitting due to the increase in complexity so its F score increases. However the FR model experiences more inaccuracy due to underfitting than EN's inaccuracy due to overfitting. So FR F score is lower than the EN model. A possible solution is to increase the variety and size of the training data, to limit the issue of overfitting in higher order Hidden Markov Models.

EN #Entity in gold data: 802 #Entity in prediction: 697	FR #Entity in gold data: 238 #Entity in prediction: 292

#Correct Entity : 450 Entity precision: 0.6456 Entity recall: 0.5611 Entity F: 0.6004 #Correct Sentiment : 391 Sentiment precision: 0.5610 Sentiment recall: 0.4875 Sentiment F: 0.5217	#Correct Entity : 133 Entity precision: 0.4555 Entity recall: 0.5588 Entity F: 0.5019 #Correct Sentiment : 76 Sentiment precision: 0.2603 Sentiment recall: 0.3193 Sentiment F: 0.2868

Part 4 - Experiment

Plan

Naives Bayes computationally less complex. It also requires less data to generate repeatable predictions. Given our smaller dataset, we were worried that the HMM model may be overfitting due to it being sensitive to outliers. In addition Naive Bayes is known to be more robust towards outliers.

Naive Bayes Model

The formula of naive bayes model is given below:

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}} \log P(c) + \sum_{i \in positions} \log P(w_i|c)$$

Where P(c) = the probability of Classifier=c appearing in the training data, $P(w_i|c) =$ the probability of word=word(i) given Classifier=c, similar to emission parameter given in Hidden Markov Model, $C_{NB} =$ predicted classifier.

Naive Bayes Model predicts the hidden state based on

- The probability of hidden state being the frequency of the hidden state
    - We use pdf of gaussian distribution to get the p value of hidden state

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

- The emission probability of each state given the observed value

Prevent Underflow Issue

In calculating the joint probability we want to avoid the underflow issue, so we use the log value of the component probability. For words that were not observed in the training dataset, we add a small constant to all the state frequency values so we are able to safely apply log function.

$$p(c|word) = log(\frac{count(c \rightarrow word)}{count(c)}) + log(\frac{count(c)}{\#\ of\ c})$$

The highest probability gives us the predicted state. It does not consider the sequence of observed values in the prediction.

Data Preprocessing

We previously did not include any preprocessing on the input in the previous parts. As part of our NB model we try to change certain words into a single class. And to lowercase all tokens.

Result

Different types of preprocessing

EN	FR
None ● Entity 0.6263 ● Sentiment 0.5244 Lowercase only ● Entity 0.6322 ● Sentiment 0.5333 Tag links and usernames only ● Entity 0.6491 ● Sentiment 0.5404 Tag Punctuation only ● Entity 0.6230 ● Sentiment 0.5192 Tag stopwords only ● Entity 0.5871 ● Sentiment 0.3363	None ● Entity 0.3703 ● Sentiment 0.2321 Lowercase only ● Entity 0.3891 ● Sentiment 0.2476 Tag links and usernames only ● Entity 0.3703 ● Sentiment 0.2321 Tag Punctuation only ● Entity 0.3703 ● Sentiment 0.2321 Tag stopwords only ● Entity 0.3717 ● Sentiment 0.2330

Based on our results the final preprocessing choice is **lowercase** and **tag links and usernames** This final result was done with gaussian distribution.

EN	FR
#Entity in gold data: 802 #Entity in prediction: 955	#Entity in gold data: 238 #Entity in prediction: 497

#Correct Entity : 570 Entity precision: 0.5969 Entity recall: 0.7107 Entity F: 0.6488 #Correct Sentiment : 481 Sentiment precision: 0.5037 Sentiment recall: 0.5998 Sentiment F: 0.5475	#Correct Entity : 143 Entity precision: 0.2877 Entity recall: 0.6008 Entity F: 0.3891 #Correct Sentiment : 91 Sentiment precision: 0.1831 Sentiment recall: 0.3824 Sentiment F: 0.2476

Part 4 - Design Challenge

Unfortunately our experiment with Naive Bayes did not yield any better results. Our best results still came from part 2. So we applied the same preprocessing we had done in our experiments namely the lowercase of the results and the tagging of links and usernames to #LNK# tag. This preprocessing improved the results by a few percentage points, giving us our highest F scores.

EN	FR
#Entity in gold data: 802 #Entity in prediction: 807 #Correct Entity : 547 Entity precision: 0.6778 Entity recall: 0.6820 Entity F: 0.6799 #Correct Sentiment : 486 Sentiment precision: 0.6022 Sentiment recall: 0.6060 Sentiment F: 0.6041	#Entity in gold data: 238 #Entity in prediction: 424 #Correct Entity : 133 Entity precision: 0.3137 Entity recall: 0.5588 Entity F: 0.4018 #Correct Sentiment : 82 Sentiment precision: 0.1934 Sentiment recall: 0.3445 Sentiment F: 0.2477

Key Takeaways

Why FR has a lower score

In Naive Bayes the hidden state is determined by the frequency of the hidden state and the emission probability. This means that when the majority of the states are, for example “O”, the model would skew towards it, regardless of the lower emission. This happens when the emission probability << hidden state probability. In “dev.p4.out” and “train” for FR, the state “O” is

assigned to most of the tokens. So many tokens were incorrectly predicted to be in the “O” state.

Preprocessing helps

In conclusion the preprocessing is an important step that can reduce noise of the dataset, and thus improving the final F-score of the trained model. Noisy data can cause a model to overfit. When we preprocess the data we are helping the model to make better generalizations for predictions.

Concluding Remarks

Comparing the F score obtained using Emission Parameters, HMM and Naive Bayes Model, we observed that Emission Parameters < Naive Bayes Model < HMM which is in line with what we expected as Emission Parameters only take into account the probability of the hidden state emitting word, while Naive Bayes also factors in the probability of the hidden state appearing in the training dataset. HMM uses both Emission Parameters and Transition Parameters which factors in the sequence of words, which would explain the higher accuracy in prediction. Our F score for Second Order HMM being lower could be due to overfitting.

The Naive Bayes Model, despite being much more simple than the Hidden Markov Model, was able to match HMM’s accuracy for the EN dataset. Although for the FR dataset, overall both models had lower scores than the EN dataset, and lower scores than HMMs.