Université Paris Saclay

Common Trunk - course 4

Inference and learning algorithms on big scale

Typos correction using Hidden Markov Model

Authors:
Issa Hammoud
Salim Tabarani

Supervisor:
Prof. Alexandre Allauzen

January 6, 2019

Contents

1 Hidden Markov Model(HMM)

We use hidden Markov model with discrete time and state space. This model is chosen for the sake of hypothesis simplicity. Otherwise, the problem of typos correction will be difficult. Thus, we assume that Markov hypothesis hold and are summarized as follow:

- The observation on time t, X_t, depends only on the state on time t, Y_t.
- Y_t depends only on $Y_{t-1:t-n}$ where n define the order of the model.
- $P(Y_t/Y_{t-1:t-n})$ and $P(X_t/Y_t)$ are the same for all t.

1.1 HMM for typos correction

The problem we want to solve is to get the most probable sequence of letters in order to correct misspelled words. Typos in general are not random, but are related to the physical position on the keyboard.

Data for this problem was generated as follows: starting with a text document, in this case, the Unabomber's Manifesto, all numbers and punctuation were converted to white space and all letters converted to lower case. The remaining text is a sequence only over the lower case letters and the space character, represented in the data files by an underscore character. Next, typos were artificially added to the data as follows: with 90% probability, the correct letter is transcribed, but with 10% probability, a randomly chosen neighbor (on an ordinary physical keyboard) of the letter is transcribed instead. Space characters are always transcribed correctly. In a harder variant of the problem, the rate of errors is increased to 20%.
Then, a misspelled word in the data looks like this:

[('t', 't'), ('h', 'h'), ('w', 'e'), ('k', 'm')]

The first element in each tuple is the typed letter and the second one is the true letter, which is equivalent to say that the first one is the observation and the second one is the state, hence $A_X = A_Y = \{a, ..., z\}$ where A_X and A_Y are the sets of realizable values for X and Y respectively.

In a more formal way, the observations, X, are the typed letters, and the states, Y, are the true letters, and we aim to get the maximizing sequence of hidden states, Y^*, given a sequence of observations, X, of length n.

$$Y^* = \underset{Y}{\operatorname{argmax}}\ P(Y/X); Y = Y_1^n = Y_1, Y_2, ..., Y_n \ and \ X = X_1^n = X_1, X_2, ..., X_n$$

Moreover, we know that maximizing over Y given X is the same as maximizing the joint probability on Y and X, then we can deduce:

$P(Y, X) = P(X_n/X_1^{n-1}, Y) \times P(Y_n/X_1^{n-1}, Y_1^{n-1}) \times P(Y_1^{n-1}, X_1^{n-1})$

$P(Y_1, X_1) = P(Y_1) \times P(X_1/Y_1)$

It is clearly noticeable the model complexity, hence the interest of Markov model.

1.2 First order HMM

As we have seen, it is difficult to solve the problem without assuming Markov hypothesis to simplify the terms dependency. So after applying them with first order assumption, the expression will be:

$$P(Y, X) = P(Y_1) \times P(X_1/Y_1) \times \prod_{t=2}^{n} P(Y_t/Y_{t-1}) \times P(X_t/Y_t)$$

What is remaining is to estimate the HMM parameters, which are:

- $P(Y_1)$: the initial state distribution, which is estimated by the probability for a word to begin with a given letter, *e.g.* $P(Y_1 = a) = P(a\ word\ begin\ with\ letter\ "a")$
- $P(X_t/Y_t)$: the emission distribution, which is estimated by the probability of having a typed letter given the true letter, *e.g.* $P(X_t = a/Y_t = b) = P(the\ typed\ letter\ is\ "a"\ knowing\ that\ the\ true\ letter\ is\ "b")$.
- $P(Y_t/Y_{t-1})$: the transition probability, which is estimated by the probability to go from true letter to the next one, *e.g.* $P(Y_t = a/Y_{t-1} = b) = P(to\ have\ letter\ "a"\ followed\ by\ letter\ "b")$.

These distributions are estimated on the training data.

1.3 Second order HMM

The only difference with the first order is the states dependency order. Thus, Y_t will depend on Y_{t-1} and Y_{t-2} and the formula will be:

$$P(Y, X) = P(Y_1) \times P(X_1/Y_1) \times P(Y_2/Y_1) \times P(X_2/Y_2) \times \prod_{t=3}^{n} P(Y_t/Y_{t-1}, Y_{t-2}) \times P(X_t/Y_t)$$

In order to find Y^* with maximum $P(Y, X)$, we introduce a combined state sequence $Z = (Z_1, Z_2, ..., Z_n)$, where $Z_1 = Y_1$ and $Z_t = Y_{t-1}Y_t$, for $2 \leq t \leq n$.

With this definition we have:

- $P(Z_1) = P(Y_1)$
- $P(Z_2/Z_1) = P(Y_2, Y_1/y_1) = P(Y_2/Y_1, Y_1) \times P(Y_1/Y_1) = P(Y_2/Y_1)$
- $P(Z_t/Z_{t-1}) = P(Y_t, Y_{t-1}/Y_{t-1}, Y_{t-2}) = P(Y_t/Y_{t-1}, Y_{t-1}, Y_{t-2}) \times P(Y_{t-1}/Y_{t-1}, Y_{t-2}) = P(Y_t/Y_{t-1}, Y_{t-2})$

Substituting these probabilities in the equation above will lead to:

$$P(Y, X) = P(Z_1) \times P(X_1/Y_1) \times \prod_{t=2}^{n} P(Z_t/Z_{t-1}) \times P(X_t/Y_t)$$

So now we have a similar form of the first order HMM. In addition, all the probability distributions are the same but for the transition probability.

As we have shown previously, $P(Z_t/Z_{t-1}) = P(Y_t/Y_{t-1}, Y_{t-2})$ so the transition probability for the second order HMM, can be estimated by the probability to be in a state Y_t given the two previous states Y_{t-1} and Y_{t-2}.[2]

2 Viterbi algorithm

After estimating the HMM parameters, now it is time to compute the most probable sequence, $Y*$ that maximize $P(Y/X)$.
In order to do so, we will use the Viterbi algorithm, which is a dynamic programming algorithm, to solve the problem. But before we should mention a useful property to be used later:
$\max_{a,b} f(a) \times g(a,b) = \max_{a} [f(a) \times \max_{b} g(a,b)]$ if $f(a) \geq 0$ and $g(a,b) \geq 0 \; \forall \; a,b$

2.1 Viterbi algorithm for first order HMM

The idea is to solve the problem to the sequence of length $t < n$ and then find a recursion to use the subsolution. Clearly we should know how to solve the problem for t = 1.
Let $\mu_t(Y_t) = \max_{Y_{1:t-1}} P(Y_{1:t}/X_{1:t}) = \max_{Y_{1:t-1}} P(X_t/Y_t) \times P(Y_t/Y_{t-1}) \times P(Y_{1:t-1}, X_{1:t-1}) =$
$\max_{Y_{t-1}} [P(X_t/Y_t) \times P(Y_t/Y_{t-1}) \times \max_{Y_{1:t-2}} P(Y_{1:t-1}/X_{1:t-1})$

$\mu_t(Y_t) = \max_{Y_{t-1}} [P(X_t/Y_t) \times P(Y_t/Y_{t-1}) \times \mu_{t-1}(Y_{t-1})]$ and $\mu_1(Y_1) = P(X_1/Y_1) \times P(Y_1)$

Finally, we maximize the $\mu_n(Y_n)$ over n. Then, the same thing is applied to get the maximum sequence but instead of using the max operator, we should use the argmax operator.
Now we will give some hints for the implementation:

The Viterbi function takes 2 arguments, the HMM with estimated parameters and an observation(a word).
μ_1 is the elementwise multiplication between the initial state probability and the emission probability for the first letter in the observation.
Then, for each remaining letters in the observation and for each state, each new element of μ is the maximum over the emission probability for the currents letter and state, times the transition probability for that state, times the latest μ. Finally, we maximize over the last μ.
Each time we keep the argmax, i.e. the state that gives the maximum. But, till now we don't know what is the most probable sequence, so we reconstruct the path as follows: The last argmax is the index of the last letter. Then, each time, the current letter is the letter saved in the matrix of argmax.

2.2 Extended Viterbi algorithm

This section is based on the paper [2] of Yang He, who described the extension of Viterbi algorithm for order 2. To find the maximum $P(X_t, Y_t)$ for each Z_t, we need only to (a) remember the maximum $P(X_{t-1}, Y_{t-1})$ for each Z_{t-1} (b) extend each of these probabilities to every Z_t, and (c) select the maximum $P(X_t, Y_t)$ for each Z_t. By increasing t by 1 until t = n and repeating (a) through (c), the maximum $P(X, Y)$ for each Y_n can finally be found. Then, among all of the $P(X, Y)$ for every Z_n, we choose the maximum one and backtrace the sequence leading to this maximum probability. The result is the optimal combined state sequence Z^*. If the first original state in each combined state is omitted, the result is the optimal sequence Y^*.
At the first step, we compute $P(Y_1, X_1) = P(Z_1) \times P(X_1/Y_1)$ which is a vector of size $k = |A_Y|$. Then, for $P(Y_2, X_2) = P(Y_1, X_1) \times P(Z_2/Z_1) \times P(X_2/Y_2)$ we have to loop over k^2 state, so the max of $P(Y_2, X_2) = P(Y_1, X_1) \times transition\_matrix\times$ observation_matrix[the second letter].
Note that $P(Y_1, X_1)$ and observation_matrix[the second letter] are vectors, so the first one is broadcasted horizontally and the second one vertically, and let call the resulting matrix as "d".
Then, each value in the matrix for the combined states d (equivalent of μ for the first order) is the max of a vector V times the emission matrix on the second state of the combined states and the current letter in the observation.
The vector V is the elementwise product between 2 vectors, the first one is the corresponding column of matrix d for the first state in the combined states, and the second one is the vector made by all the state combination but with fixing the second sate to the corresponding second state. Same thing is done for the path but with saving the argmax instead of the max.
For clearer explanation, it is recommended to return to the original paper, but just pay attention to the difference in notations.

2.3 Results comparison

The time complexity of the problem using a brute force method is about k^n, with k is the number of states and n the sequence length. This complexity is reduced with the first order HMM to $n \times k^2$, which is much more faster. However, with the second order HMM, the computation required is approximately k times as much as for the first order [1], which is $n \times k^3$.

The baseline in the table 1 is the accuracy without doing anything. We can remark

Table 1: accuracy

baseline	89.82%	80.59%
first order	92.12%	88.95%
second order	94.18%	88.95%

Table 2: made and corrected mistakes with 10% and 20%

	made	corrected
first order	79	247
second order	129	448

	made	corrected
first order	363	1080
second order	524	1919

the improvement of the second order with respect to first order.

3 Insertion and deletion

In the above work, we treated the problem of substituted characters only, but in real life there are many other problems like insertion or deletion letters.

3.1 Insertion

The problem of noisy insertion of characters can be solved using the same process for substitution errors with just adding a special state to A_Y, for example a '#' to represent the state when an insertion appears in the data.
However, we should adjust the data to have such mistakes, else $P(Y_t/Y_t - 1)$ will be always 0 for Y_t or Y_{t-1} = '#', and the same for $P(Y_1)$.

3.2 Deletion

The problem of deletion is quite complicated and we didn't find a way to use HMM model to solve it.

4 Unsupervised training

In case we don't have a dictionary for training, we cannot estimate HMM parameters as we have done previously. Then, we should use Baum-Welch algorithm which is a particular case of the Expectation-Maximization(EM) algorithm to find the unknown parameters.
It deserves to mention that Baum-Welch algorithm finds a local minimum not a global one, and can be summarized in 4 steps:

- Initialization.
- Forward procedure.
- Backward procedure.
- Parameters update.

4.1 Initialization

The first thing to do is to initialize the parameters. It can be done randomly or using prior information about the parameters if it is available. In the later case this can speed up the algorithm and also steer it toward the desired local maximum.
As an example, we can initialize $P(Y_1)$ by the estimated probability of each letter (there are tables for that like in [2]) and 0 for non letters (like for # in case of insertion).

4.2 Forward procedure

The goal of the forward algorithm is to compute the joint probability $p(Y_t, X_{1:t})$. This computation directly would require marginalizing over all possible state sequences $Y_{1:t-1}$, the number of which grows exponentially with t. Instead, the forward algorithm takes advantage of the conditional independence rules of the hidden Markov model (HMM) to perform the calculation recursively.
Let $\alpha(i,t) = P(Y_t = i, X_{1:t})$, with $i \in A_Y$. Then, it is easy to show that:

$$\alpha(i,t) = P(X_t/Y_t = i) \times \sum_{j \in A_Y} P(Y_t = j) \times \alpha(j, t-1)$$

$$\alpha(i,1) = P(Y_t = i) \times P(X_1/Y_1 = i)$$

4.3 Backward procedure

In a similar way, we define $\beta(k,t) = P(X_{t+1,n}/Y_t = k)$. Thus:

$$\beta(k,t) = \sum_{j \in A_Y} \beta(j,t+1) \times P(X_t/Y_{t+1} = j) \times P(Y_{t+1} = j/Y_t = k)$$

$$\beta(k,n) = 1$$

4.4 Parameters update

Once we have calculated α and β, we will introduce another quantities for the sake of expression simplicity:

- $a_{ij} = P(Y_t = j/Y_t - 1 = i)$.
- $b_j(X_i) = P(X_t = X_i/Y_t = j)$.
- $\gamma_i(t) = P(X_t = i/Y, \theta) = \dfrac{\alpha_i(t) \times \beta_i(t)}{\sum_{j=1}^{k} \alpha_j(t) \times \beta_j(t)}$.
- $\xi_{ij}(t) = \dfrac{\alpha_i(t) \times a_{ij} \times \beta_j(t+1) \times b_j(X_{t+1})}{\sum_{i=1}^{k} \sum_{j=1}^{k} \alpha_i(t) \times a_{ij} \times \beta_j(t+1) \times b_j(X_{t+1})}$

Now, the affectation:

$$P^*(Y_1) = \gamma_i(1)$$

$$a_{ij}^* = \frac{\sum_{t=1}^{n-1} \xi_{ij}(t)}{\sum_{t=1}^{n-1} \gamma_i(t)}$$

$$b_i^*(v_k) = \frac{\sum_{t=1}^{n} 1_{X_t = v_k} \times \gamma_i(t)}{\sum_{t=1}^{n} \gamma_i(t)} \text{ where } 1_{X_t = v_k} = \begin{cases} 1 & if\ X_t = v_k \\ 0 & else \end{cases}$$

We reiterate until convergence.[1]

References

[1] Baum–welch algorithm. https://en.wikipedia.org/wiki/BaumWelch_algorithm.

[2] Yang He. Extended viterbi algorithm for second order hidden markov process. 9th International Conference on Pattern Recognition, 1998.