Second-Order HMM for typos correction

Xiaoxiao Chen, Dong Fei, Mihaela Sorostinean

December 9, 2016

Abstract

The purpose of this project was to create a model which is able to correct wrong typed words. This model was built through both a first and second order HMM. We report on the challenges we faced while building the model and on the results we obtained. Finally, we provide some ideas on how this model can be extended.

1 Introduction

As explained before, the goal of our project was to design a model which is able to detect typing mistakes. To do this, a data set for training and testing was provided. The data set contains 30.558 examples, where each example contains a variable number of letter pairs, the correct letter and the typed letter. Typos where added to this data in a artificial manner and there are two options of the data set. The first one contains 10% errors, while the second one is a more difficult problem and has 20% errors.

2 First Order HMM

2.1 Model

For implementing this algorithm, we need two steps to construct the model and predict the potential states from the input observation:

- Collecting the necessary statistical information to train our first order hidden Markov model.
- With the input observation, the model uses the first-order Viterbi algorithm to predict the states and compare to the correct letter. The performance is evaluated by the results and the right letter.

There are five elements needed to define an HMM:

1 N - the number of distinct states in the model. For the Unabomber's Manifesto, N is the number of letters in Alphabet.

2 M - the number of distinct observations.For the Unabomber's Manifesto, M is also the number of letters in Alphabet.

3 $A = (a_{ij})$ - the state transition probability distribution.

4 $B = (b_{ij})$ - the observation/state probability distribution..

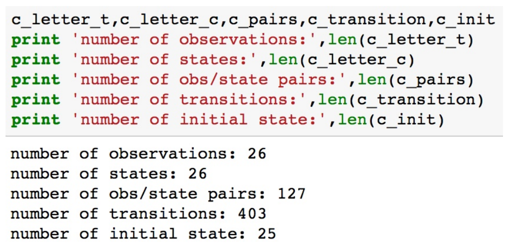
5 $\Pi = \pi_i$ - the initial state distribution.

With all the statistical information, we calculate the different distribution listed above.
As for prediction, given an observation sentence V, calculate the sequence U of states that maximizes $\hat{P}(V|U)$. The Viterbi algorithm is a common method for calculating the most likely state sequence when using an HMM. The details are presented below:

- Initialization(t=0):
  $\delta_0 = \pi_i b_i(o_0), 1 \leq i \leq N, \psi_1(i) = 0$
- Time Recursion:
  For $1 \leq t \leq T, 1 \leq j \leq N$
  $\delta_t(j) = \max_{1 \leq i \leq N}[\delta_{t-1}(i)a_{ij}]b_j(O_t)$
  $\psi_t(j) = \arg\max_{1 \leq i \leq N}[\delta_{t-1}(i)a_{ij}]$
- Termination:
  $P_{max} = \max_{1 \leq i \leq N}[\delta_T(i)]$
  $i_T = \arg\max_{1 \leq i \leq N}[\delta_T(i)]$
- State sequence backtracking:
  For $t = T-1, T-2, \ldots, 1 : i_t = \psi_{t+1}(i_{t+1})$

2.2 Implementation

First, we need to get the statistical information, which is showed in Figure 1. Then we calculate the distribution in the algorithm. Here special attention should be paid on the "Observation estimation". Since sometimes we cannot expect that the train set includes all the possible observations, a smoothing factor can be added to B. We do not have to care about this problem in our data set.



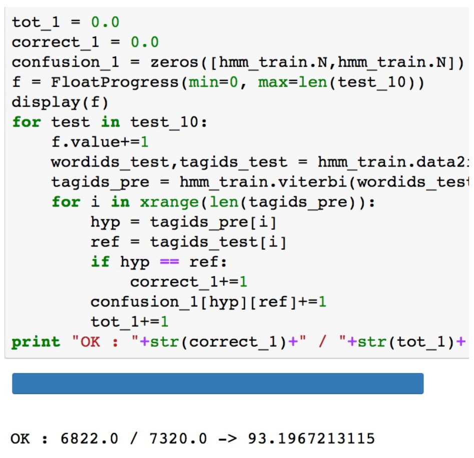
```
c_letter_t,c_letter_c,c_pairs,c_transition,c_init
print 'number of observations:',len(c_letter_t)
print 'number of states:',len(c_letter_c)
print 'number of obs/state pairs:',len(c_pairs)
print 'number of transitions:',len(c_transition)
print 'number of initial state:',len(c_init)
```

```
number of observations: 26
number of states: 26
number of obs/state pairs: 127
number of transitions: 403
number of initial state: 25
```

Figure 1: Statistical information of the dataset

2.3 Result

We run our test on the test set **test_10**.The final result shows that the model reaches the target accuracy we set originally. The error rate is around 7%, which suggests that the first model Markov model improves the accuracy of 3% compared to the original data set.



```
tot_1 = 0.0
correct_1 = 0.0
confusion_1 = zeros([hmm_train.N,hmm_train.N])
f = FloatProgress(min=0, max=len(test_10))
display(f)
for test in test_10:
    f.value+=1
    wordids_test,tagids_test = hmm_train.data2:
    tagids_pre = hmm_train.viterbi(wordids_test
    for i in xrange(len(tagids_pre)):
        hyp = tagids_pre[i]
        ref = tagids_test[i]
        if hyp == ref:
            correct_1+=1
        confusion_1[hyp][ref]+=1
        tot_1+=1
print "OK : "+str(correct_1)+" / "+str(tot_1)+
```

```
OK : 6822.0 / 7320.0 -> 93.1967213115
```

Figure 2: The result of first order HMM

3 Second Order HMM

3.1 Model

Compared to the first order Markov model, we need to add two new variables in our new model. In our context the best strategy is to use trigram probability instead of older bigram probability when calculating state transition probability[5]. At the same time, the observation is generated together by the current state and the previous state.

The two new variables are listed below:

1 $A_2 = (a_{ijk})$ - the current state depends on the previous two states.$P(i_{t+1} = q_k | i_t = q_j, i_{t-1} = q_i)$

2 $B = (b_{ijk})$ - the current observation depends on the current state and the previous state.$P(o_t = v_k | i_t = q_j, i_{t-1} = q_i)$

For the second order viterbi algorithm, we can "delete" the initialization phrase. In other words, we calculate one more step to form our "initialization" matrix. Since each time we need two previous states to evaluate the next states and observation. After that the procedure changes as:

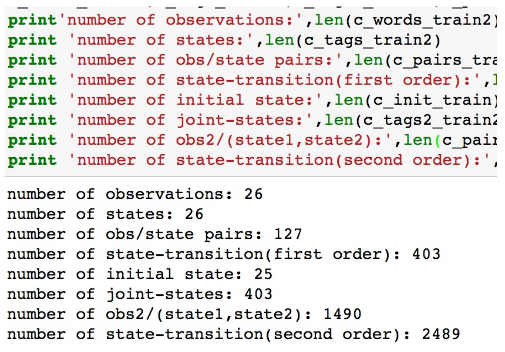
1 For $2 \le t \le T, 1 \le i, j, k \le N$
$\delta_t(j,k) = \max_{1 \le i \le N}[\delta_{t-1}(i,j) a_{ijk}] b_{jk}(O_t)$
$\psi_t(j) = \arg\max_{1 \le i \le N}[\delta_{t-1}(i) a_{ij}]$

2 $P_{max} = \max_{1 \le i,j \le N}[\delta_T(i)]$
$i_T = arg_j \max_{1 \le i,j \le N}[\delta_T(i)]$
$i_{T-1} = arg_i \max_{1 \le i,j \le N}[\delta_T(i)]$

3 For $t = T-2, T-3, \ldots, 1:$
$i_t = \psi_{t+1}(i_{t+1}, i_{t+2})$

3.2 Implementation

Since we need three more variables, new information are also added to the statistical information. The count of two-to-one transition and observation/state are collected to help us establish the second order transition and observation distributions.
In viterbi algorithm, the original matrix of first order HMM A and B are applied to calculate the probability sequence of the first two states. Then we can implement the second order viterbi algorithm normally. As for the word with one or two letters, we can implement the first order HMM to deal with them.



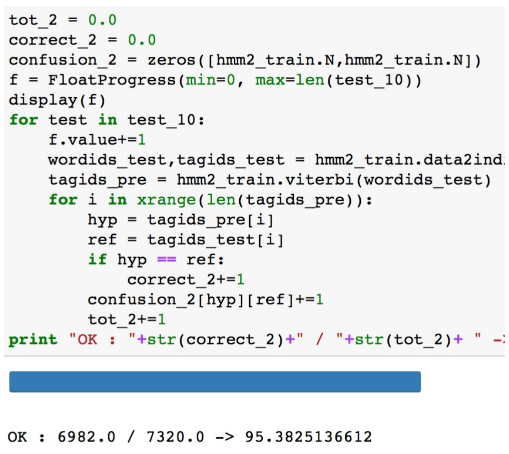
```
print'number of observations:',len(c_words_train2)
print 'number of states:',len(c_tags_train2)
print 'number of obs/state pairs:',len(c_pairs_tra
print 'number of state-transition(first order):',l
print 'number of initial state:',len(c_init_train)
print 'number of joint-states:',len(c_tags2_train2
print 'number of obs2/(state1,state2):',len(c_pair
print 'number of state-transition(second order):',
```

```
number of observations: 26
number of states: 26
number of obs/state pairs: 127
number of state-transition(first order): 403
number of initial state: 25
number of joint-states: 403
number of obs2/(state1,state2): 1490
number of state-transition(second order): 2489
```

Figure 3: Statistical information of the dataset(second order)

3.3 Result

We run our test on the test set **test_10**.The final result shows that the model reaches the target accuracy we set originally. The error rate is around 4%, which suggests that the second order model Markov model improves the accuracy of 3% compared to the first order Markov model.



```
tot_2 = 0.0
correct_2 = 0.0
confusion_2 = zeros([hmm2_train.N,hmm2_train.N])
f = FloatProgress(min=0, max=len(test_10))
display(f)
for test in test_10:
    f.value+=1
    wordids_test,tagids_test = hmm2_train.data2ind
    tagids_pre = hmm2_train.viterbi(wordids_test)
    for i in xrange(len(tagids_pre)):
        hyp = tagids_pre[i]
        ref = tagids_test[i]
        if hyp == ref:
            correct_2+=1
        confusion_2[hyp][ref]+=1
        tot_2+=1
print "OK : "+str(correct_2)+" / "+str(tot_2)+ " -
```

```
OK : 6982.0 / 7320.0 -> 95.3825136612
```

Figure 4: The result of second order HMM

4 Critics and Evolution

Even when the second order model has an improved accuracy, our HMM model is somehow limited, for instance, due to some rare abbreviations, or because of insertions of additional information inside the observation sequences. This is because in these cases, one observation must be consumed each time the HMM makes a transition. As a result, our models must be as flexible as possible by allowing skipping and inserting characters[1].

4.1 Insertion problem

To deal with the problem of randomly insertion, we might add a **noise model** to take into account additional text. In this case, this noise model must be augmented by a null transition between its first and last states to allow the model to skip it in the more likely situation where no such "noise" text appears.

Note that since this noise data is quite rare in practice, it becomes clear that considering a noise model for every preceding character will ultimately lead to poor estimation of the noise models. One solution to overcome this problem is to tie the states of all the noise models, which is the same as considering one noise model regardless of the preceding character.

4.2 Omission problem

Opposite to the last case, maybe we can add **null transitions** (transitions not consuming any observation) between states where an event is thought to be sometimes missing, to better allow omission. For instance, to model a missing character in a word, every character HMM model can be augmented by a null transition between the first and last states (assuming that observations are emitted along transitions). This mix between noise models and null transitions is the key to getting as flexible HMM models as possible in highly adverse conditions.

5 Unsupervised Training

Under the unsupervised case, assume that we are given unlabeled sequences $(\mathbf{x}_1^k)_1$,...,$(\mathbf{x}_1^k)_N$ as data. We suspect that there is a HMM responsible for generating these sequences, and try to estimate the model parameters $A(ij)$, $B(ij)$, Π from the data, where

- $A = (a_{ij})$ is the transition matrix, $a_{ij} = P(y_t = j|y_{t-1} = i)$
- $B = (b_{ij})$ is the observation matrix, $b_{ij} = P(x_t = j|y_t = i)$
- $\Pi = \pi_i$ is the initialization distribution, $\pi_i = P(y_1 = i)$

The training steps are as follows using the idea of *EM* algorithm[4]:

1. Initialize $A(ij)$, $B(ij)$, $\Pi(i)$ randomly.
2. Until convergence,
    - **Expectation** Step:
      For each sequence $(\mathbf{x}_1^k)$, compute $\alpha t(i)$, $\beta t(i)$, $\hat{p}(x)$ from the forward-backward algorithm, and get $\hat{P}(y_1|\mathbf{x})$, $\hat{P}(x_t, y_t|\mathbf{x})$, $\hat{P}(y_t, y_{t-1}|\mathbf{x})$ based on these values. Then compute the expected counts $\hat{C}(y_1)$, $\hat{C}(x_t, y_t)$, $\hat{C}(y_t, y_{t-1})$ by summing over all sequences
    - **Maximization** Step:
      Update the model parameters $\hat{P}(y_1)$, $\hat{P}(x_t|y_t)$, $\hat{P}(y_t|y_{t-1})$ with these counts.

A little limitation for expectation maximization idea is that, essentially the EM algorithm tends to converge to some local solution that is feasible for the observations and initialization distributions given, but may not be the actual parameters to the problem. (like K-means) We can try several times with different initialization distributions, thus take the best parameters estimated.

6 Conclusions

The main goal of our project was to design a correction model for words that are wrongly typed. To implement this model we used a first order and then a second order HMM. As it was to be expected, the second order HMM model improves the accuracy of our "correction algorithm" with about 3%, giving a total accuracy of 95.38%. We consider this to be a good result. Nevertheless, the model has some limitations, when it comes to omissions or insertions of characters, so we analyzed these problems and suggested possible ways in which our model can be improved to deal with them. Finally we discussed and proposed an approach on how unsupervised training can be applied for this task.

References

[1] Mohamed Cheriet, Nawwaf Kharma, Cheng-Lin Liu, and Ching Suen. *Character recognition systems: a guide for students and practitioners*. John Wiley & Sons, 2007.

[2] Todd K Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996.

[3] Adrian E Raftery. A model for high-order markov chains. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 528–539, 1985.

[4] Karl Stratos. Unsupervised learning 101: the em for the hmm. http://www.cs.columbia.edu/~stratos/research/em_hmm_formulation.pdf.

[5] Scott M Thede and Mary P Harper. A second-order hidden markov model for part-of-speech tagging. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 175–182. Association for Computational Linguistics, 1999.