ASSIGNMENT 3: POS TAGGING AND OCR

Abhilash Kuhikar(akuhikar)
Darshan Shinde(dshinde)
Dhruuv Agarwal(dagarwa)

1) POS TAGGING

Data structures:

- pos_init = {}
  Dictionary of Initial Probabilities of all the 12 POS.
  Key - POS name, value - corresponding initial probability
- transition = {}
  Dictionary of transition Probabilities from all the 12 POS to all the 12 POS.
  Key - 'from' POS name, value - dictionary of all the 'to' POS names with their corresponding transition probability
- emission = {}
  Dictionary of emission probabilities for all the observed words in the training text File.
  Key - observed word, value - dictionary storing all the probabilities of the 12 POS tags
- pos = {}
  Dictionary of total count of all the POS tags in the training text file. (bc.train)
- mapPOS = {}
  Key - POS name, value - index
- inv_mapPOS = {}
  Key -index, value - POS name
- transitionMatrix = np.zeros((12,12))
  Np array of the transition dictionary
- complex_trans = {}

Models:-

1. simplified:
   Naive bayes aproach where each word pos tag depends only on current element and no dependency on past elements
   so we use P(W|S)*P(S) and take the max value to simulate

2. MCMC:
   Here we use complex model than viterbi and use past two tasg to determins the tag for ccurrent word.
   This we do using sampling not deterministically.
   We use the cooncept of gibbs samplin where we sample one variable by keeping others constant.

Here in our case, in each iteration we take one word by other.
for each word we toggle the pos tag to the 12 values to get possible joint probabilities .
with this we add to get marginal probability of rest and use it to get conditional probability.

Finally we sample based on this conditional of the remaining elements/tags to get a new sample.

In mcmc we have a burn period which allows time for samples to approximate true distribution.
Thus after that we collect samples and find common tag for each word in sentence. This lis tof common tags is our final output.

Our mcmc approach is faster and better than other because :
1. initial smaple isnt random, and is derived from emission probability. This helps in reaching staionary distribution faster.
2. we calculate the joint probability of the markov field around the current changed element not the whole, this speedends the process.
3. for ghost words we dont have emission proabbilty so we do a bit of word processing to get some cases of words like words with ed at end are more likely to be verb .

3. Refer report file for Viterbi implementation

Viterbi for POS:

The function “hmm_viterbi” in pos_solver.py takes care of the viterbi implementation for the POS tagging. (Question 1)

In Viterbi algorithm, we create the table of dimensions (#posTags, #no of words in a sentence).
In this table the first column entries are the emission probabilities of those words multiplied by their respective initial probabilities of observing those POS tags.
P(POS|Word) = P(Word|POS)*P(POS)/constant
We ignore the constant as we only care for the maximum of these values.
For the rest of the columns, we consider a transition from each of the previous POS tags and their respective values.

We populate the entries of this table using the below formula:

$$v_j(t+1) = e_j(O_{t+1}) \max_{1 \leq i \leq N} v_i(t) p_{ij}$$

Here, (t+1) denotes the current column we are populating in V and (t) is the previous column entries in V.

While populating these entries in V, I am keeping a track of the best values found for a V[i,j] from the previous values from column V[:,j-i].

Once done with populating all the values for V, I am checking for the POS corresponding to the max value in last column of V. Let's say I found the max value at i = 4 index.
Now I will look for the 4th index in the last column of my backtrack matrix which will give me the index corresponding to the previous max value.
I will continue this till the first column of backtrack and hence will find the best sequence.

2) OCR

We tried to solve this problem using Hidden Markov Model(HMM).
Here we have some train data (training image). Using which we are training our model. We are separating every character of text image and trying to identify it using three different models.
1. Simplified model
2. HMM model and
3. More complicated model.

In this complete problem, every character from the test data is the hidden state and observed pixel matrix for that hidden state is the observed state.
In simplified model, every hidden state depends on only the observed state of that particular hidden state.
In HMM model, every hidden state depends on the observed state of this particular hidden state and the previous hidden state.
In more complex model, we are assuming that every hidden state depends on the respective observed state and the two previous hidden states.

In this particular problem, we have total 72 hidden states.

Initial probability (P(S)): Number of occurrences of character (as first letter of every word) divided by total number of occurrences of first letters of all words from the train document. For every other character which will not occur as first letter of any word, small value (1e-100) is considered as initial probability.

Transition probability (P(A|B)): Number of occurrences of transition from state B to state A in complete document divided by total transitions from state B. And for all transitions which are not available in training document, small value (1e-100) is considered as transition matrix.

Storing this transition probabilities in 2D numpy array for further use. e.g., transition Matrix[A][B] is P(B|A).

To solve this problem of hidden markov model, we need three different types of probabilities.

1. Initial probability:

From the training text document, we are calculating initial probabilities for every state. We are scanning the whole documents and counting the occurrences of first character of every word. Then, to get the actual probability of every state, the number of occurrences of that state is divided by sum of occurrences of all states. If occurence of any state from the problem (as a first character of the word) is not available, then we are assuming the very less probability (1e-100) for this state.

2. Transition probability:
We are calculating transition probability from the training text document. For we every single line from the document, we are counting the transitions of any character to the next character. e.g., there are total 3 occurrences where there is X after A in the whole document. Then we are storing transition from A to X as 3. To get the actual transition probabilities for any state, we are dividing the number of transitions from given character to next character by sum of all transitions from given character. i.e. if there are 3 transitions from A to X and total 29 transitions from A, then we will get transition probability of going from A to X (P(X|A)) as 3/29. We are storing this probabilities in 2D numpy array for further use. Hence, we will get the 2D array of shape 72*72. For every missing value from the table, we are storing a small probability of 1e-100.

3. Emission probability:
The main challenge in this problem is to calculate emission probability. We have test image which we are separating into characters. For every character, we are converting pixel image into pixel matrix of 1 and 0. 1 for black pixels and 0 for blank pixels. TO get emission probability for every character from the test image, we are comparing it to every train character image pixel matrix. Every test image may have some noise like missing pixels or some extra unwanted pixels. We are assuming there is uniform noise throughout the image and considering it as 0.35. Now we are comparing the test character pixel matrix with the every training character pixel matrix to get matching pixels. Let say we will get m matching pixels. We assume that it will follow the binomial distribution and hence we will get the probability of being that test character is that particular label as (1-p)^m * p(N-m) where p is the noise (i.e. 0.35), m is the number of matching pixels and N is the total number of pixels in training character pixel matrix. This is how will get total 72 emission probabilities for every character from test image.
We have handled the special case of ' ' separately. In emission function, which compares the label matrix (training character pixel matrix) to test matrix (test character pixel matrix). If we have 0 black pixels i.e. the training character is ' ' and the test character pixel matrix has less than 6 black pixels then we are considering the test character is ' ' and returning the emission probability as 1e-60, otherwise of test character matrix has more than 6 black pixels the we are assuming it is something other than ' ' and returning emission probability as 1e-100.

Simplified model:

In simplified model, every hidden state depends on the observed state for that particular hidden state. To get the character sequence using this model, for every character in test image, we are taking the state which are giving the max value for emission probability * probability of being that character in training document. i.e. max(P(O|S)*P(S))

Viterbi for OCR:

The function “hmm_viterbi” in pos_solver.py takes care of the viterbi implementation for the OCR prediction. (Question 2)

In Viterbi algorithm, we create the table of dimensions (#labels i.e. 72 in our case, #no of characters in a image sentence).
In this table the first column entries are the emission probabilities of those character images multiplied by their respective initial probabilities of observing those characters in the text file.
P(label|test) = P(test|label)*P(label)/constant
We ignore the constant as we only care for the maximum of these values.
For the rest of the columns, we consider a transition from each of the previous POS tags and their respective values.

We populate the entries of this table using the below formula:

$$v_j(t+1) = e_j(O_{t+1}) \max_{1 \le i \le N} v_i(t) p_{ij}$$

Here, (t+1) denotes the current column we are populating in V and (t) is the previous column entries in V.

While populating these entries in V, I am keeping a track of the best values found for a V[i,j] from the previous values from column V[:,j-i].

Once done with populating all the values for V, I am checking for the POS corresponding to the max value in last column of V. Let’s say I found the max value at i = 4 index.
Now I will look for the 4th index in the last column of my backtrack matrix which will give me the index corresponding to the previous max value.
I will continue this till the first column of backtrack and hence will find the best sequence.

Emission2
In this different approach for finding the emission probabilities, we only considered the black pixels (assuming pixels with value 1) of the hidden label image. For example if I have to calculate the emission probability P(test = ‘1’|label = ’A’), I will check for all the pixels in test where the pixels in ‘A’ are 1. This is achieved by masking the test image with the mask of ‘A’ in trained label.
test*label
Where test and label are the numpy arrays with entries 0(white pixels) and 1(black pixels)

maxEmission
In this function I am finding the best matching label for the test image using the emission2 probabilities.

Here are the few things we have tried for OCR:

- We have tried the error function for coming up with the error value. Here we have considered the error as ratio of the number of mismatched pixels with the size of the matching image found by the maxemission function.
- However we tried this error value but could not achieve significant results and hence are sticking to the hardcoded error value of 0.35.
- With this value of p we found that the algorithm now is doing fine but isn't doing well for handling the spaces.
- We have tried using different values to handle spaces (line no. 65 in ocr.py) but now are sticking to 5 for the better results we observed at this value.