Report

1. Viterbi implementation:

For given emission scores, transitional scores, start and end scores, viterbi algo is to use dynamic programming to store previous state values.

Start and end arrays are scores pertaining to the start and values in matrix representation like explained in class.

Dimensions of dp array is Output sequences x states.

For every state, I'm storing optimal values.

I'll be appending start scores to initial dp array and end scores after calculating all intermediate node.

To get the best path from a node to another node, I'll dp arrray and intermediate values and pick the optimal one.

2. Features added:

Basic features:

- BIAS
- SENT_BEGIN / SENT_END
- WORD itself and WORD.lower()
- IS_ALNUM
- IS_NUMERIC
- IS_DIGIT
- IS_UPPER
- IS_LOWER

My features:

- IS_TWEETLANGUAGE

  Purpose: This is basically for grouping Usernames, Hashtags, URLs, Emojis, Punctuations.

  At first, I was checking if each token falls under one of this. For example, below were some features that I was assigning to given below sentence.

  Movie rocks #Avengers :D www.avengers.com !!!!!!!!!!!!!!!!!!!!

  IS_HASHTAG IS_EMOJI IS_URL IS_PUNCTUATION

  In this case, I was getting some accuracy. I realized since for the task parts of speech tagging, it doesn't matter if token is emoji/hashtag/username/url/punctuation. They are fall under same category. So, I used a common feature labelled as 'IS_TWEETLANGUAGE' and is assigned to a token if it falls under one of the above. This gave me better accuracy than earlier case.

- IS_CAMELCASE
  Purpose: This feature is to group tokens like ‘speedRanch’, ‘roadBlocks’ etc..
  If few tokens are given as camel case, they might be due to concatenation of 2 words. For such cases, I’m adding this feature. This did not decrease accuracy although, it did not improve by much either which is expected considering the sparseness in train data.

- CLUSTER_<cluster_ID>
  Done as part of pre-processing in preprocess_corpus().
  Purpose: This is done to group tokens based on clusters. Instead of implementing brown clustering, I took twitter dataset from CMU’s research on twitter. These clusters are constructed by improvising brown clustering and their results looked appealing as confidence for their parameters is relatively higher than native methods. I’m comparing my tokens against this data set and adding appropriate clusterID as a feature.
  There were suggestions in piazza to add counts from clustering methods as features. I feel its wrong as we’ll up creating lots of unnecessary features and will lead to less accuracy and execution time. Reason being, counts will be different for each token and we’ll end up creating sparse feature set if we considering the amount of load we are add for all tokens just by adding counts.
  I’m doing this in preprocess_corpus() and using results in token2features().
  DataSet file: 6mpaths.txt (included in zip)
  Tries that didn’t work as expected:
  - I tried doing spell correct for all words in input sequences - worst decision (the amount of correct words in tweets vs number of expressions/notWords tokens are more in twitter data which makes this not useful)
  - I tried converting repetitive social media tokens with expressions like yaaaaaaaaaaaay -> yay, lollllll -> lol. I was doing this by removing duplicates and comparing against nltk corpus (this to avoid books -> boks case, code should stop after it converts to books). This didn’t do much good for me for the very same reason mentioned in above point.

- NOUN/ADJ/VERB/ADV_<suffix>
  I took common suffixes for each type of word that might contribute for its POS like below.
  NOUN = set(['acy', 'cy', 'ade', 'ad', 'age', 'al', 'ial', 'an', 'ian', 'ate', 'dom', 'en', 'et', 'ette', 'let', 'hood', 'ice', 'ic', 'tic', 'ics', 'ine', 'in', 'ing', 'ism', 'ist', 'ive', 'ment', 'ness', 'ship', 'th', 'tude', 'ure', 'ance', 'ence', 'ancy', 'ency', 'ant', 'ent', 'ary', 'ery', 'ry', 'ory', 'er', 'or', 'ar', 'eer', 'ier', 'ee', 'ess', 'ion', 'ity'])
  ADJ = set(['able', 'ible', 'al', 'ial', 'an', 'ian', 'ant', 'ent', 'ary', 'ory', 'ar', 'ate', 'ful', 'ic', 'ical', 'istic', 'istical', 'ish', 'ive', 'less', 'ly', 'eous', 'ious', 'uous', 'ing', 'ed', 'y', 'en', 'er', 'or', 'ern', 'ese', 'id', 'ile', 'ine', 'ist', 'ite', 'like', 'some', 'th', 'eth', 'ward'])
  VERB = set(['ate', 'en', 'ify', 'efy', 'ish', 'ize', 'ise', 'er'])

ADV = set(['ly', 'ward', 'wards', 'ways', 'wise', 'ely', 'ilely', 'lily', 'ply', 'bly', 'tly', 'dly', 'ably', 'ibly', 'ally', 'ially', 'antly', 'ently', 'arily','arly', 'orily', 'ately', 'fully', 'icly', 'ically', 'isticly', 'istically', 'ingly', 'edly', 'ishly', 'ively', 'lessly', 'ously', 'eously', 'iously', 'uously', 'ily'])

I'll check if my tokens are ending with any of above and add relevant suffixes accordingly like VERB_ify. Reason for prefixing the feature with it probable POS is, I want to bind that feature to that particular POS as there are cases with suffixes like 'en' which can be ADJ or VERB. With this binding, I want to leave it to patterns in train data set based on which appropriate weights will be assigned for such cases,

3. My features vs basic features:

I explained in above section regarding why I introduced above features and their impact On why they are behaving in that way. For the sake of redundancy, I'm not repeating them here. Below is the tabular form of accuracies after adding each feature in MEMM and CRF.

	MEMM / LR				**CRF**			
Feature	**TokenWise Accuracy**	**Token Wise Macro**	**Token Wise Micro**	**Sentence accuracy**	**Token Wise Accuracy**	**Token Wise Macro**	**Token Wise Micro**	**Sentence accuracy**
All basic features	84.38	83.33	84.38	8.92	84.29	83.21	84.29	11.6
plus IS_CAMELCASE	84.48	83.41	84.48	9.82	84.81	84.6	84.81	11.6
plus IS_TWEETLANGUAGE	84.91	83.77	84.91	12.5	84.62	84.29	84.62	12.5
plus CLUSTER_<cluster_ID>	86.61	85.7	86.61	14.28	86.56	85.79	86.56	18.75
plus NOUN/ADJ/VERB/ADV_<suffix>	87.27	86.02	87.27	16.96	88.22	86.79	88.22	19.64

Row1: for all basic features (ABF)
Row2: ABF + IS_CAMELCASE
Row3: ABF + IS_CAMELCASE + IS_TWEETLANGUAGE
Row4: ABF + IS_CAMELCASE + IS_TWEETLANGUAGE + NOUN/ADJ/VERB/ADV_<suffix>

I'm reporting the +ve cases where accuracy was increasing unlike the cases I mentioned in '**Tries that didn't work as expected**' in above section.

4. MEMM vs CRF Observations:

- Table from above section clearly gives a picture that CRF performs better than MEMM under appropriate features (if we don't overloading them with unnecessary stuff) and good training data..
- The better the train data and with right kind of feature set, CRF results out weighs MEMM's
- Weird thing I observed is, my CRF accuracy drops from 88.22 to 88.07 and LR/MEMM accuracy increases from 87.27 to 87.32. This is when I treat feature 'IS_CAMELCASE' as 'IS_TWEETLANGUAGE' feature.
  Tokens like 'smackMyHead' fall under this case and wrt parts of speech, they are no different from ones that I included in IS_TWEETLANGUAGE. Yet, it still decreases in accuracy. In my final .pred files, I left both of them un-clubbed.
- Workload/execution time to create model file is less in MEMM when compared to CRF as the latter spends more time on calculating conditional probability.
- CRF is a discriminative model while MEMM is a state classification model.

References:

http://web.mit.edu/neboat/www/6.046-fa09/rec6.pdf
http://www.cs.cmu.edu/~ark/TweetNLP/smaller_clusters.html
http://usefulenglish.ru/writing/list-of-nouns-with-suffixes