HMM POS Tagger

Modelling HMM Steps (classes) and Data structures used in the project

1. **HmmPosTaggerExecutor**

This class is used the run the project and print the error rate.

1. **HMMParser**

The class takes input from training data set and generates HMM.

Data structures used:

/\*\* Stores tag count used in the training set \*/

**private** HashMap<String, Integer> tagCount;

/\*\*

\* The data structure is used to store tag(key) that contains multiple

\* words(key)

\* and its corresponding count(value) being in that tag.

\*/

**private** HashMap<String, HashMap<String, Integer>> tagToWordCount;

/\*\*

\* The data structure is used to store word(key) that contains correspond to

\* multiple tags(key) and its corresponding count(value) of being in that

\* tag.

\*/

**private** HashMap<String, HashMap<String, Integer>> wordToTagCount;

/\*\* The data structure is used to store tag-to-tag counts \*/

**private** HashMap<String, HashMap<String, Integer>> tagToTagCount;

To fill up the values in the data structure I have used the below code snippet:

**private** String parseLine(String line, String prevTag)

{

String[] posTaggerWord = line.split("/");

String word = posTaggerWord[0].toLowerCase();

String tag = posTaggerWord[1];

tagCount(tag);

tagToWordCount(tag, word);

wordToTagCount(word, tag);

tagtoTagCount(prevTag, tag);

**return** tag;

}

1. **HMMGenerator**

**private** HashMap<String, HashMap<String, Double>> transitionProbabilities;

**private** HashMap<String, HashMap<String, Double>> emissionProbabilities;

/\*\*

\* For each tag calculate the transition probability and emission

\* Probability

\* and stores them in transitionProbabilities and emissionProbabilities

\* **transitionProbabilities** is map (fromTagKey of map ( toTagKey,

\* Prob(fromTag - toTag))

\* **emissionProbabilities** is map (fromTagKey of map ( toWordKey,

\* Prob(fromTag - toTag))

\*/

The class generates an HMM Model for probabilistic analysis and calculates transition and emission probabilities.

1. **ViterbiExecutor**

The class is used to execute the Viterbi algorithm on the HMM Model using the transitionProbabilities and transitionProbabilities.

/\*\*

\* This is basically used as memoized dynamic programming to avoid

\* re-computation of the recursive functions.

\* ***It is stored as <Word~POS, Probability Value>***

\*/

**private** HashMap<String, Double> viterbiStore;

Now to handle known/unknown words I use the following strategy:

If given word not present in the training HMM model

**Unknown Word:** - unknownWord

HashMap<String, integer> tagsForWord = getTagsForWord(prevWord);

For each tag in tagsForWord do:

HashMap<String, integer> **emissionTags** = Get transition probability for ( tag );

For each emissionTag do:

Get the maximum probability of going to this emissionTag.

And Set the **mostProbableTag** = emissiontag;

End Loop

End Loop

**However, there could be a scenario where the mostProbableTag is not found when the prevWord is also an unknown word.**

**Since that word had been processed and stored in the Viterbi store.**

We handle this by running a loop on Viterbi store and get the tag whose word startsWith the prevWord, the key in Viterbi is stored as ***Word~POS***. Split the key to get POS. Find the highest probability for the tag in the Viterbi store and get the **maxProbableTag**.

**We use the Viterbi store to store the previous word and tags and its corresponding highest probability.**

**known Word:** - knownWord, prevWord

HashMap<String, Integer> tagsForWord = getTagsForWord( knownWord )

For each tag in tagsForWord do:

findMaximum in the Viterbi store ***Word~POS*** that where the

*findMaximum* (knownWord, tagForWord, prevWord)

If ( **prevWordTag**.startsWith( **prevWord** + "~" )

String prevTag = prevWordTag.split("~")[1];

**MULTIPLY** ( viterbiStore.get(**prevWordTag**) **\*** probabilityForTagGivenPrevTag(

prevTag, tagForWord));

**Once we get the maximum probability for the all the previous state that could lead us to the tag we calculate**

**MULTIPLY** *findMaximum* (knownWord, tagForWord, prevWord) **\***

probabilityForWordGivenTag(tagForWord, observationWord);

**BOTH unknown and known RETURN** mostProbableTag;

1. **Error Rate**

We keep a counter for total words in test set and the probability returned from the Viterbi algorithm

Thus,

Error rate is found to be = 0.46 = 46 %