HMMnotebook

January 28, 2019

0.0.1 NLP Homework 1 - Simple HMM POS Tagger

Author: Jose Figueroa

Used the spanish data set to train the algorithm and test it inthe test data set.

Import libraries and define constants:

```
In [1]: from conllu import parse
        from io import open

        def AMOUNT_OF_TAGS ():
            return 18
```

Parameters to access training data:

```
In [2]: def path_name():
            return '/Users/jose.figueroa/Desktop/NLP-HW1/'
        def train_file_path():
            return path_name() + 'udtb/es-ud-train.conllu'
        def dev_file_path():
            return path_name() + 'udtb/es-ud-dev.conllu'
        def test_file_path():
            return path_name() + 'udtb/es-ud-test.conllu'
```

Access and parse training data for access:

```
In [3]: train_path = train_file_path()
        train_file = open(train_path,"r", encoding="utf-8")
        train_data = train_file.read()
        contents = parse(train_data)
```

Create structures to calculate hold probabilities:

```
In [4]: tag_prob_of_word = {}
        total_count_of_word = {}
        tag_prob_given_tag = {}
        total_count_of_tag = {}
```

Calculate emision and transition probabilities.

```
In [5]: for tokenList in contents:
            for i in range(0, len(tokenList)):
                word = tokenList[i]['form'].lower()
                tag = tokenList[i]['upostag']
                #Check if the word has been tagged prior
                if (word in tag_prob_of_word):
                    if (tag in tag_prob_of_word[word]):
                        tag_prob_of_word[word][tag] += 1
                    else:
                        tag_prob_of_word[word][tag] = 1
                else:
                    tag_prob_of_word[word] = {tag:1}

                #Count the amount of times the word is seen
                if (word in total_count_of_word):
                    total_count_of_word[word] += 1
                else:
                    total_count_of_word[word] = 1

                #Check for start tag
                if (i==0):
                    if ('start' in tag_prob_given_tag):
                        #check if start has been mapped to the given tag
                        if (tokenList[i]['upostag'] in tag_prob_given_tag['start']):
                            tag_prob_given_tag['start'][tag] += 1
                        else:
                            tag_prob_given_tag['start'][tag] = 1
                    else:
                        tag_prob_given_tag['start'] = {tag : 1}
                    if ('start' in total_count_of_tag):
                        total_count_of_tag['start'] += 1
                    else:
                        total_count_of_tag['start'] = 1


                else:
                    #Check the amount of times a tag occurs after a previous tag
                    prevTag = tokenList[i-1]['upostag']
                    if (prevTag in tag_prob_given_tag):
                        if (tag in tag_prob_given_tag[prevTag]):
                            tag_prob_given_tag[prevTag][tag] += 1
                        else:
                            tag_prob_given_tag[prevTag][tag] = 1
                    else:
                        tag_prob_given_tag[prevTag] = {tag : 1}
                    #Count the amount of times a tag is seen
```

```
                if (tag in total_count_of_tag):
                    total_count_of_tag[tag] += 1
                else:
                    total_count_of_tag[tag] = 1

        #Calculate and return Transition and Emition probabilities.
        #Emissions
        for word in tag_prob_of_word:
            for tag in tag_prob_of_word[word]:
                tag_prob_of_word[word][tag] /= total_count_of_word[word]
        #Transitions
        for prev in tag_prob_given_tag:
            for tag in tag_prob_given_tag[prev]:
                tag_prob_given_tag[prev][tag] /= total_count_of_tag[prev]
```

Example emission and transition probabilities:

```
In [6]: print(tag_prob_of_word['medina'], '\n')
        print(tag_prob_given_tag['ADP'])
```

```
{'PROPN': 0.9166666666666666, 'NOUN': 0.08333333333333333}

{'DET': 0.49805739440729685, 'VERB': 0.038035050274299244, 'NUM': 0.0681329309166097, '_': 0.0
```

Write probabilities to file:

```
In [7]: pn = path_name()
        with open(pn + 'emissionProb.txt', 'x') as f:
            for word in tag_prob_of_word:
                print ('{}  |  {}'.format(word, tag_prob_of_word[word]), file = f)

        with open(pn + 'transitionProb.txt', 'x') as f:
            for prev in tag_prob_given_tag:
                print('{}  |  {}'.format(prev, tag_prob_given_tag[prev]), file = f)
```

Create a list of tags for numeric representation for dynamic programing:

```
In [8]: tagList = []
        for key in total_count_of_tag:
            if (key == 'start'):
                tagList.insert(0, key)
            else:
                tagList.append(key)
```

Define tagSentence function: Input: 1. sentence in the form of a token list, where each word is it's own dictionary. 2. The length of the sentence.

Output: A list of (word, tag) tuples.

The algorithm uses two matricies. The first calculate probabilities in each iteration given prior probabilities, transition and emission probabilities. The second tracks the back path of the largest probability.

```
In [9]: def tagSentence (sentence, length):
            mat = [[0 for i in range(AMOUNT_OF_TAGS())] for i in range(length+1)]
            mapTags = [[[] for i in range(AMOUNT_OF_TAGS())] for i in range(length+1)]
            mat[0][0] = 1
            mapTags[0][0] = [-1,-1]
            for i in range(len(mat)-1):
                for j in range(len(mat[0])):
                    if (mat[i][j] == 0):
                        continue
                    for k in range(1, len(mat[0])):
                        word = sentence[i]['form'].lower()
                        prevTag = tagList[j]
                        currTag = tagList[k]
                        wordProbability = 0
                        tagProbability = 0
                        if word not in tag_prob_of_word and currTag == 'NOUN':
                            wordProbability = 1
                        elif word not in tag_prob_of_word:
                            wordProbability = 0
                        elif word in tag_prob_of_word and currTag not in tag_prob_of_word[word]
                            wordProbability = 0
                        else:
                            wordProbability = tag_prob_of_word[word][currTag]
                        try:
                            tagProbability = tag_prob_given_tag[prevTag][currTag]
                        except:
                            tagProbability = 0
                        probability = mat[i][j] * wordProbability * tagProbability
                        if probability >=  mat[i+1][k]:
                            mat[i+1][k] = probability
                            mapTags[i+1][k] = [i, j]
            wordIndex = length
            tagIndex = 0
            taggedSentence = []
            maxx = 0
            for i in range(AMOUNT_OF_TAGS()):
                if maxx<mat[wordIndex][i]:
                    maxx = mat[wordIndex][i]
                    tagIndex = i
            while wordIndex > 0 and tagIndex >0:
                nextIndex = mapTags[wordIndex][tagIndex][0]
                couple = (sentence[nextIndex]['form'], tagList[tagIndex])
                tagIndex = mapTags[wordIndex][tagIndex][1]
                wordIndex = nextIndex
```

```
            taggedSentence.insert(0, couple)
        return taggedSentence
```

Open and parse test file:

```
In [10]: test_path = test_file_path()
         test_file = open(test_path,"r", encoding="utf-8")
         test_file_contents = test_file.read()
         test_file_parse = parse(test_file_contents)
```

Tag new sentences and tally results, return accuracy:

```
In [11]: correctTag = 0
         TotalTag = 0

         for i in range(len(test_file_parse)):
             taggedSentence = tagSentence(test_file_parse[i], len(test_file_parse[i]))
             for j in range(len(test_file_parse[i])):
                 if taggedSentence[j][1] == test_file_parse[i][j]['upostag']:
                     correctTag += 1
                 TotalTag +=1
         print((correctTag/TotalTag)*100)
```

```
90.37254422434174
```