CS563

Natural Language Processing

Assignment-I

2011MC04_2011MC17

Mathematics and Computing

Part-of-Speech (PoS) tagging using HMM

==========

Problem Statement:

Problem Statement: Part-of-Speech (PoS) tagging assigns grammatical categories to every token in a sentence. In this assignment, you have to develop a PoS tagger using Hidden Markov Model (HMM) and Recurrent Neural Network.

Dataset: WSJ (Wall Street Journal)

Number of PoS tags: 46

List of tags : 'MD', 'TO', 'WP', 'WP$', '.', 'PRP', 'PDT', '#', 'POS', 'VBN', '-RRB-', 'DT', '"', ':', 'EX', 'RP', 'RBR', '-NONE-', 'UH', 'VBZ', 'VBG', '$', 'RBS', 'JJR', 'IN', ',', 'VBD', 'LS', 'JJS', 'WRB', 'VBP', '-LRB-', 'NNP', 'NNS', 'PRP$', 'JJ', 'CC', 'FW', 'CD', 'VB', 'NN', 'NNPS', 'SYM', 'WDT', '``', 'RB'

==========

Dataset:

```
df.head()
```

	tokenized_sentences	tags
0	['Pierre', 'Vinken', ',', '61', 'years', 'old'...	['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', ',', 'M...
1	['Mr.', 'Vinken', 'is', 'chairman', 'of', 'Els...	['NNP', 'NNP', 'VBZ', 'NN', 'IN', 'NNP', 'NNP'...
2	['Rudolph', 'Agnew', ',', '55', 'years', 'old'...	['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', 'CC', '...
3	['A', 'form', 'of', 'asbestos', 'once', 'used'...	['DT', 'NN', 'IN', 'NN', 'RB', 'VBN', '-NONE-'...
4	['The', 'asbestos', 'fiber', ',', 'crocidolite...	['DT', 'NN', 'NN', ',', 'NN', ',', 'VBZ', 'RB'...

Preprocessing :

Creating two separate dataframes for tokenized_sentences and tags respectively

```
col1=df['tokenized_sentences']
col2=df['tags']
```

```
col1.head()
```

```
0    ['Pierre', 'Vinken', ',', '61', 'years', 'old'...
1    ['Mr.', 'Vinken', 'is', 'chairman', 'of', 'Els...
2    ['Rudolph', 'Agnew', ',', '55', 'years', 'old'...
3    ['A', 'form', 'of', 'asbestos', 'once', 'used'...
4    ['The', 'asbestos', 'fiber', ',', 'crocidolite...
Name: tokenized_sentences, dtype: object
```

```
col2.head()
```

```
0    ['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', ',', 'M...
1    ['NNP', 'NNP', 'VBZ', 'NN', 'IN', 'NNP', 'NNP'...
2    ['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', 'CC', '...
3    ['DT', 'NN', 'IN', 'NN', 'RB', 'VBN', '-NONE-'...
4    ['DT', 'NN', 'NN', ',', 'NN', ',', 'VBZ', 'RB'...
Name: tags, dtype: object
```

Creating (Word,TAG) pairs

```
list1=[]
list2=[]
list5=[]
for i in range(len(col1)):
    list4=[]
    list1=re.findall(r"'(.*?)'", col1[i], re.DOTALL)
    list2=re.findall(r"'(.*?)'", col2[i], re.DOTALL)
    for j in range(len(list1)):
        list3=[]
        list3.append(list1[j])
        list3.append(list2[j])
        list4.append(tuple(list3))
    list5.append(list4)
```

```
# Splitting into train and test
import random
random.seed(1)
train_set, test_set = train_test_split(list5,test_size=0.2)
print(len(train_set))
print(len(test_set))
```

```
2739
1175
```

Creating Vocabulary Set

```
# vocabulary
V = set(Word_Token)
print(len(V))
```

```
9221
```

Creating Tag Set to pick up distinct tags

```
# number of tags
T = set([pair[1] for pair in Tagged_words])
len(T)
```

```
45
```

Emission Probabilities P(w/t)

```
#Calculating P(w/t)
t = len(T)
v = len(V)
w_given_t = np.zeros((t, v))

#Calculating  Probability of a word given a tag: Emission Probability
def prob_of_word_given_tag(word, tag, train_bag = Tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)
    w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
    count_w_given_tag = len(w_given_tag_list)

    return (count_w_given_tag, count_tag)
```

Transition Probabilities P(t2/t1)

```
#Calculating the Probability of a tag given a tag: P(t2/t1) i.e. Transition Probability

def t2_given_t1(t2, t1, train_bag = Tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])      #Counting number of occurences of t1
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2: #Counting number of times t2 follows t1
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)
```

```
len(Tagged_words)
```

```
69580
```

Viterbi Algorithm

```
# Viterbi_Algorithm Function !
def Viterbi_Algorithm(words, train_bag = Tagged_words):
    state = []
    T = list(set([pair[1] for pair in train_bag]))

    for key, word in enumerate(words):
        #initializing a list of probability column for a given observation
        p = []
        for tag in T:
            if key == 0:
                transition_probability = tags_df.loc['.', tag]      # P(tag|start) = P(tag|'.')
            else:
                transition_probability = tags_df.loc[state[-1], tag]

            #Calculating emission and state probabilities
            emission_probability = prob_of_word_given_tag(words[key], tag)[0]/prob_of_word_given_tag(words[key], tag)[1]
            state_probability = emission_probability * transition_probability
            p.append(state_probability)

        pmax = max(p)
        # Finding the state for which probability is maximum
        state_max = T[p.index(pmax)]
        state.append(state_max)
    return list(zip(words, state))
```

Model Evaluation

3-fold cross validation

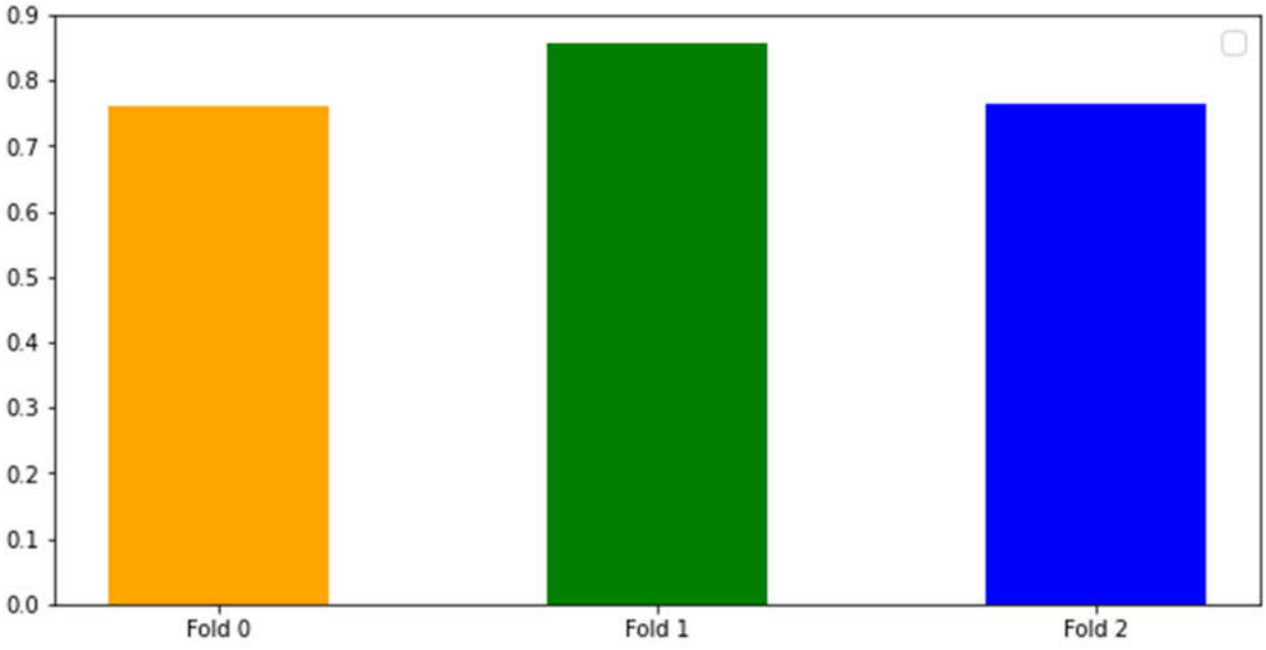
```
num_sents = len(list5[0:3912])
k = 3
foldsize = int(num_sents/k)
foldsize
```

```
1304

Fold 0
from sent 0 to 1304
accuracy = 0.7613636363636364

Fold 1
from sent 1304 to 2608
accuracy = 0.8571428571428571

Fold 2
from sent 2608 to 3912
accuracy = 0.7655172413793103
```

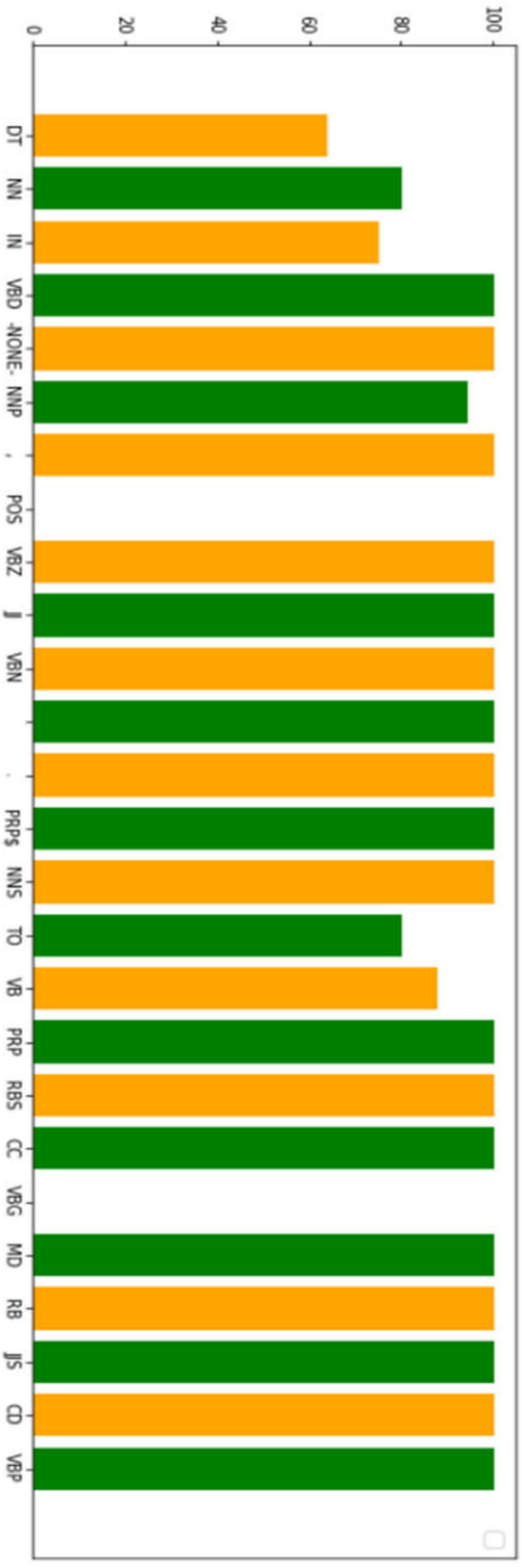
Maximum Accuracy

```
print(max(fold_accurracies))
F=fold_accurracies.index(max(fold_accurracies))
print("Fold ",F)
```

```
0.8571428571428571
Fold  1
```



Classwise Accuracy




```
{'DT': 63.63636363636363,
 'NN': 80.0,
 'IN': 75.0,
 'VBD': 100.0,
 '-NONE-': 100.0,
 'NNP': 94.11764705882352,
 ',': 100.0,
 'POS': 0.0,
 'VBZ': 100.0,
 'JJ': 100.0,
 'VBN': 100.0,
 '': 100.0,
 '.': 100.0,
 'PRP$': 100.0,
 'NNS': 100.0,
 'TO': 80.0,
 'VB': 87.5,
 'PRP': 100.0,
 'RBS': 100.0,
 'CC': 100.0,
 'VBG': 0.0,
 'MD': 100.0,
 'RB': 100.0,
 'JJS': 100.0,
 'CD': 100.0,
 'VBP': 100.0}
```

Observations

- Tags ‘POS’and ’VBG’ shows 0% accuracy ,indicating that they mostly incorrectly tagged a word.
- ‘DT’,’IN’,’NN’have accuracy ranging between 63.3% - 80% indicating there were certain instances where they incorrectly tagged a word.
- The other tags have performed well with good accuracy.
- The accuracy of the best fold of this HMM model was found to be 85.71% which will be improved using the RNN model.

Parts of Speech Tagging with RNN

Dataset

```
In [3]: df=pd.read_csv("WSJ_treebank_corpus.csv")
        df.head()
```

Out[3]:

	tokenized_sentences	tags
0	['Pierre', 'Vinken', ',', '61', 'years', 'old'...	['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', ',', 'M...
1	['Mr.', 'Vinken', 'is', 'chairman', 'of', 'Els...	['NNP', 'NNP', 'VBZ', 'NN', 'IN', 'NNP', 'NNP'...
2	['Rudolph', 'Agnew', ',', '55', 'years', 'old'...	['NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', 'CC', '...
3	['A', 'form', 'of', 'asbestos', 'once', 'used'...	['DT', 'NN', 'IN', 'NN', 'RB', 'VBN', '-NONE-'...
4	['The', 'asbestos', 'fiber', ',', 'crocidolite...	['DT', 'NN', 'NN', ',', 'NN', ',', 'VBZ', 'RB'...

Implementation Details:

1. Size of training data: 70%
2. Size of validation(development) data: 10%
3. Size of testing data: 20%
4. Embedding Used: Word2Vec
5. Word2Vec dataset: GoogleNews-vectors-negative300.bin.gz
6. Word2Vec limit: 20,000
7. Embedding Dimension: 300
8. Max - Sequence - Length (Input): 100
9. Total Output Tags: 46
10. Hyper - Parameters Used:
    6.1 Epochs: 15
    6.2 Batch-Size: 64
11. Number of units in RNN layer: 128

12. Activation function:
    12.1 ReLu for RNN
    12.2 Softmax for final output layer
13. Techniques used to avoid overfiting: Dropout(0.2)
14. Optimization: Adam

Summary of RNN Model:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 300)          3416400
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 100, 128)          54912
_________________________________________________________________
dropout (Dropout)            (None, 100, 128)          0
_________________________________________________________________
time_distributed (TimeDistri (None, 100, 47)           6063
=================================================================
Total params: 3,477,375
Trainable params: 3,477,375
Non-trainable params: 0
```

Techniques Used for Model Evaluation:

3-Fold Cross Validation

Results & Observations:

Fold	Training Accuracy	Testing Accuracy
1	99.7092	98.1200
2	99.6574	98.2068
3	99.7057	98.2503

1. Overall accuracy of training set: 99.6907%
2. Overall accuracy of testing set: 98.1924%
3. Best training accuracy: 99.7092
4. Best testing accuracy: 98.2503
5. Class Wise accuracy of each tag from the fold with best testing accuracy:

```
**********************************************************************************************
              class-wise accuracies of the different classes
**********************************************************************************************
Accuracy of tag - class 1 is 92.61693080788558 %
Accuracy of tag - class 2 is 98.15762538382803 %
Accuracy of tag - class 3 is 86.83368869936035 %
Accuracy of tag - class 4 is 99.45021380574221 %
Accuracy of tag - class 5 is 97.72554965883245 %
Accuracy of tag - class 6 is 91.61624891961971 %
Accuracy of tag - class 7 is 81.35593220338984 %
Accuracy of tag - class 8 is 100.0 %
Accuracy of tag - class 9 is 100.0 %
Accuracy of tag - class 10 is 92.45014245014245 %
Accuracy of tag - class 11 is 90.06309148264984 %
Accuracy of tag - class 12 is 83.1918505942275 %
Accuracy of tag - class 13 is 93.86973180076629 %
Accuracy of tag - class 14 is 99.52718676122932 %
Accuracy of tag - class 15 is 100.0 %
Accuracy of tag - class 16 is 86.41975308641975 %
Accuracy of tag - class 17 is 91.92399049881234 %
Accuracy of tag - class 18 is 99.70674486803519 %
Accuracy of tag - class 19 is 87.29641693811075 %
Accuracy of tag - class 20 is 86.4 %
Accuracy of tag - class 21 is 99.43181818181817 %
Accuracy of tag - class 22 is 100.0 %
Accuracy of tag - class 23 is 100.0 %
Accuracy of tag - class 24 is 100.0 %
Accuracy of tag - class 25 is 100.0 %
Accuracy of tag - class 26 is 98.125 %
Accuracy of tag - class 27 is 100.0 %
Accuracy of tag - class 28 is 71.26436781609196 %
Accuracy of tag - class 29 is 79.1044776119403 %
Accuracy of tag - class 30 is 35.714285714285715 %
Accuracy of tag - class 31 is 100.0 %
Accuracy of tag - class 32 is 58.69565217391305 %
Accuracy of tag - class 33 is 92.85714285714286 %
Accuracy of tag - class 34 is 100.0 %
Accuracy of tag - class 35 is 29.411764705882355 %
Accuracy of tag - class 36 is 100.0 %
Accuracy of tag - class 37 is 100.0 %
Accuracy of tag - class 38 is 100.0 %
Accuracy of tag - class 39 is 0.0 %
Accuracy of tag - class 40 is 0.0 %
Accuracy of tag - class 41 is 100.0 %
Accuracy of tag - class 42 is 100.0 %
Accuracy of tag - class 43 is 0.0 %
Accuracy of tag - class 44 is 0.0 %
Accuracy of tag - class 45 is 0.0 %
Accuracy of tag - class 46 is : NA
```

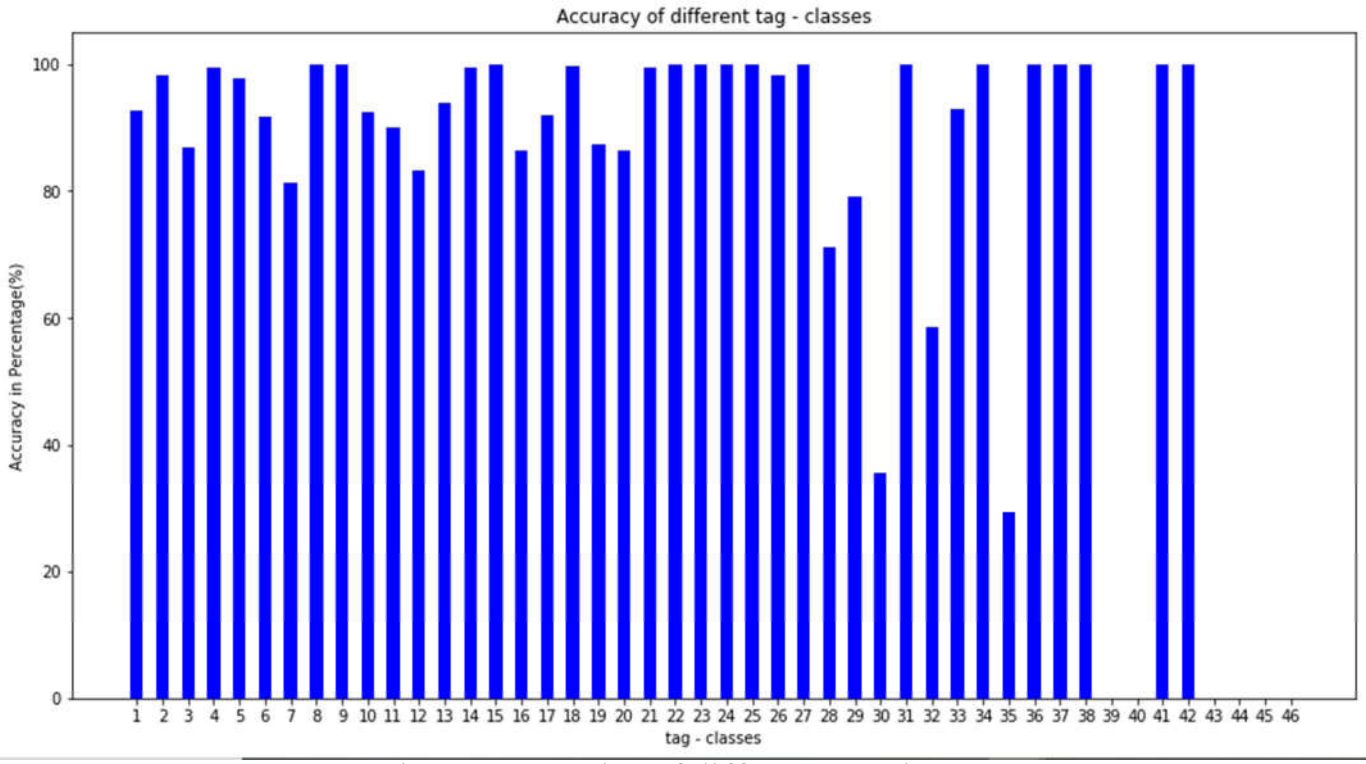


Fig 1: Accuracies of different tag classes

Observations:

1. RNN is performing much better than HMM model.
2. Tag Classes 8,9,15,22,23,24,25,31,34,36,37,38,41 and 42 have been predicted corrected with 100% accuracy.
3. Tag Classes 39,40,43,44 and 45 have been not correctly predicted even for a single example i.e., with 0% accuracy.
4. No single instance of Tag class 46 is present in the best fold test set.
5. RNN requires more data to perform better.
6. RNN can also be used for pre-training for some other model also where data is lacking.