# CSCI 544 Assignment 3

Mounika Mukkamalla

February 23, 2024

## 1

Initially, I imported necessary packages. Then upload train, test and dev data to google colab using $files.upload()$ function, since it's easier to work on colab.
Later, I read train, test and dev data into pandas dataframes and separated the data using tab separator.
I created a new dataframe for vocab, and printed initial number of rows in the traind data which is 912095
Then I grouped by word and summed up the count of words in the given training data and added a new column sum in vocab dataframe for it.
I removed rare words with count less than 3 and created a vocab file and printed out in vocab file with 'unk' word in the first row and sorted remaining and printed in the next rows.
Number of unk words after removing rare words are: 32537
Number of unique words in vocab file after removing rare words are: 16920
In the vocab file, threshold I used is 3 and replaced rare words with unk words.

## 2

Number of transition parameters are: 1416
Number of emission parameters are: 50286
I have calculated transition and emission probabilities on train_data and outputted in HMM.json file with pair of states as key for transisiont and pair of state and observation as key for emission.
In emission probability, I used all the words without replacing with unk words since this gave a better prediction percentage.

## 3

I have implemented greedy algorithm here.
For dev data, I got an accuracy of 93.35802319227734 using my accuracy implementation(same as given in eval) and using eval.py(93.36)
I have used smoothing here if transition/emission probabilities for any pair doesn't exist, instead of completely disregarding that pos tag, added a small value of probability to add that pos tag into consideration.
Added 0.01 for transition and 0000001 for emission
Along with dev, greedy algorithm is implemented on train data too to check if the model is over-fitting. Acccuracy I got is around 96 and dev is 93.35. Gap doesn;t seems to be huge, implying better generalisation. Hence our model can perform well on unseen data too. If test data lies in this similar distribution, we need to get accuracies around this range.

## 4

I have implemented viterbi algorithm here.
Fir dev data, I got an accuracy of 92.82982211159007 using my accuracy implementation(same as

given in eval) and also using eval.py(92.83).
I used smoothing for transition and emission probabilities in case a pair of state transition of observation emission doesnot exist.
I tweaked these values accordingly for the dev data.
I have calculated train data accuracy, and I got 94.15817431298275 accuracy.
Train and dev data accuracies do not vary much and they are reasonably high enough, indicating that our model is able to generalise well on the unseen data. If test data is from the same distribution then since it's unseen data as dev data, our accuracies should of this range.

```
import pandas as pd
import numpy as np
import re

from google.colab import files
uploads = files.upload()

<IPython.core.display.HTML object>

Saving dev to dev (1)
Saving test to test (1)
Saving train to train (1)

train_data = pd.read_csv("train", sep='\t', header=None,
engine='python')
test_data = pd.read_csv("test", sep="\t", header=None,
engine="python")
dev_data = pd.read_csv('dev', sep='\t', header=None, engine='python')

vocab = pd.DataFrame(train_data[1])
print(vocab.head())

        1
0  Pierre
1  Vinken
2       ,
3      61
4   years

# Number of rows in the given training data
print("Number of rows in the given training data: ")
print(len(vocab))
print("\n")

912095

#Adding another column just to make it easier to sum up
vocab[2] = 1

# Summing up all the rows based on words index at 1

vocab['sum'] = vocab.groupby([1])[2].transform('sum')

print(vocab.shape)

(912095, 3)

#Replacing all the words indexed at 1 whose count is less than 3 with
'unk'

vocab.loc[vocab['sum'] < 3, 1] = 'unk'
```

```
# summing up again since we replaced a few words with 'unk'

vocab['sum'] = vocab.groupby([1])[2].transform('sum')

# dropping duplicate words

vocab = vocab.drop_duplicates([1, 2])

# Number of vocab words after preprocessing
print("Final vocab size:")
print(vocab.shape)
print("\n")
```

```
(16920, 3)
```

```
# Getting unknown row first and sorting the remaining words based on
word count
unk_row = vocab[vocab[1]=='unk']
remaining_vocab = vocab[vocab[1]!='unk'].sort_values(by='sum',
ascending=False)

# Writing the vocab to vocab.txt file

vocab_output_file = 'vocab.txt'
with open(vocab_output_file, 'w') as file:
  file.write(str(unk_row[1].values[0]) + "\t" + "0" + "\t" +
str(unk_row['sum'].values[0]) + "\n")
  j = 1
  for index, row in remaining_vocab.iterrows():
    file.write(str(row[1]) + "\t" + str(j) + "\t" + str(row['sum']) +
'\n')
    j = j+1

# vcoab.txt is saved on google colab, this command helps to download
to local system
from google.colab import files
files.download(vocab_output_file)
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

#Model Learning

```
#Transition probabilities
from collections import defaultdict

prev_index = 0
prev_tag = '<S>'
end_tag = '<E>'
transition = defaultdict(int)
```

```
tag_count = defaultdict(int)
belief_states = {}

for i, row in train_data.iterrows():
  current_index = row[0]
  current_tag = row[2]

  tag_count[current_tag] = tag_count[current_tag] + 1
  if(i==0):
    tag_count[prev_tag] = tag_count[prev_tag] + 1
  if(current_index == 1 and i!=0):
    states = (prev_tag, end_tag)
    transition[states] = transition[states] + 1
    prev_tag = '<S>'
    tag_count[prev_tag] = tag_count[prev_tag] + 1

  #if current_index > prev_index:
  states = (prev_tag, current_tag)
  prev_tag = current_tag
  transition[states] = transition[states] + 1

for states, value in transition.items():
  transition[states] = value/tag_count[states[0]]

print(len(transition))
```

```
1416
```

```
#emission probabilities
emission = defaultdict(int)
tags = defaultdict(int)

for i, row in train_data.iterrows():
  tag = row[2]
  word = row[1]
  tags[tag] = tags[tag] + 1
  em = (tag, word)
  emission[em] = emission[em] + 1

for em, value in emission.items():
  emission[em] = value/tags[em[0]]

print(len(emission))
```

```
50286
```

```
# Writing emission and transition probabilities to hmm.json file

import json

transition_dict = {str(k): v for k, v in transition.items()}
```

```
emission_dict = {str(k): v for k, v in emission.items()}

model_dict = {
  'transition': transition_dict,
  'emission': emission_dict
}

with open('hmm.json', 'w') as f:
  json.dump(model_dict, f, indent=4)

files.download('hmm.json')
```

```
<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

#Greedy Decoding

```
def greedy(input_data):
  for i, row in input_data.iterrows():
    max_prob = -1
    current_word = row[1]
    current_index = row[0]
    if(current_index==1):
      prev_tag = '<S>'
    for tag in tags:
      prob = transition.get((prev_tag, tag), 0.01)*emission.get((tag,
current_word), 0.0000001)
      if(prob>max_prob):
        max_prob=prob
        max_tag = tag
    prev_tag = max_tag
    input_data.loc[i, 'predict'] = max_tag

def get_accuracy(input_data):
  total = 0
  positives = 0
  for i, row in input_data.iterrows():
    total = total + 1
    if(row[2]==row['predict']):
      positives = positives + 1

  accuracy = positives/total
  return accuracy

greedy(dev_data)

get_accuracy(dev_data)
```

```
0.9335802319227734
```

```
greedy_dev_output = 'greedy_dev.out'

with open(greedy_dev_output, 'w') as file:
  for index, row in dev_data.iterrows():
    if(row[0] == 1 and index != 0):
      file.write('\n')
    file.write(str(row[0]) + "\t" + row[1] + "\t" + row['predict'] +
'\n')

greedy(train_data)

get_accuracy(train_data)

0.9604185967470493

# Predict on test data
greedy(test_data)

# Get greedy output for test data and output into greedy.out file

greedy_test_output = 'greedy.out'

with open(greedy_test_output, 'w') as file:
  for index, row in test_data.iterrows():
    if(row[0] == 1 and index != 0):
      file.write('\n')
    file.write(str(row[0]) + "\t" + row[1] + "\t" + row['predict'] +
'\n')

# Downloads dev.out(for my test purpose) and greedy.out file

from google.colab import files
files.download(greedy_test_output)
files.download(greedy_dev_output)

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```

#Viterbi Algorithm

```
def viterbi(input_data):
  prev_dp = defaultdict(float)
  current_dp = defaultdict(float)
  start_tag = '<S>'
  end_tag = '<E>'
  predictions  = []
  current_sequence = [[] for _ in range(len(tags))]
```

```
  prev_sequence = [[] for _ in range(len(tags))]
  prev_value = 0
  current_value = 0
  cvalues = []
  values = []

  for i, row in input_data.iterrows():
    current_index = row[0]
    current_word = row[1]
    if(current_index == 1):
      if(i!=0):
        max_prob = -1
        j = 0
        for tag in tags:
          prob = current_dp[tag]*transition.get((tag, end_tag), 0.01)
          if(prob>max_prob):
            max_index = j
            max_prob = prob
          j = j+1
        predictions.extend(current_sequence[max_index])
        predictions.append("<E>")

      j = 0
      for tag in tags:
        current_dp[tag] = transition.get((start_tag, tag),
0.01)*emission.get((tag, current_word), 0.0001)
        current_sequence[j] = [tag]
        j = j+1
    else:
      prev_dp = defaultdict(float, current_dp)
      for i in range(len(current_sequence)):
        prev_sequence[i] = current_sequence[i][:]
      j = 0
      for c_tag in tags:
        max_prob = -1
        k = 0
        for p_tag in tags:
          prob = prev_dp[p_tag]*transition.get((p_tag, c_tag), 0.01)
          if(prob>max_prob):
            max_index = k
            max_prob = prob
          k = k+1
        current_dp[c_tag] = max_prob*emission.get((c_tag,
current_word), 0.00001)
        current_sequence[j] = prev_sequence[max_index][:]
        current_sequence[j].append(c_tag)
        j = j+1


  max_prob = -1
```

```
  j = 0
  for tag in tags:
    prob = current_dp[tag]
    if(prob>max_prob):
      max_index = j
      max_prob = prob
    j = j+1
  predictions.extend(current_sequence[max_index])

  return predictions

def get_viterbi_accuracy(input_data, predictions):
  total = 0
  positives = 0
  j = 0
  for i, row in input_data.iterrows():
    total = total + 1
    if (predictions[j] == '<E>'):
      j = j+1
    #print(row[2] + " " + predictions[j])
    if(row[2]==predictions[j]):
      positives = positives + 1
    j = j+1

  print(j)

  accuracy = positives/total
  print(accuracy)

# Run the algorithm on dev data

dev_predictions = viterbi(dev_data)

# Calculate dev accuracies
get_viterbi_accuracy(dev_data, dev_predictions)
```

```
137294
0.9282982211159007
```

```
# predict on train

train_predictions = viterbi(train_data)

# Calculate train accuracy

get_viterbi_accuracy(train_data, train_predictions)
```

```
950312
0.9415817431298275
```

```
# Save dev predictions to viterbi_dev.out
```

```
viterbi_dev_output = 'viterbi_dev.out'

with open(viterbi_dev_output, 'w') as file:
  j = 0
  for index, row in dev_data.iterrows():
    if(dev_predictions[j] == '<E>'):
      file.write('\n')
      j = j+1
    file.write(str(row[0]) + "\t" + row[1] + "\t" + dev_predictions[j]
+ '\n')
    j = j+1

#Download dev predictions

files.download(viterbi_dev_output)

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>

#Predict on test data

test_predictions = viterbi(test_data)

#Save test predictions to viterbi.txt

viterbi_test_output = 'viterbi.out'

with open(viterbi_test_output, 'w') as file:
  j = 0
  for index, row in dev_data.iterrows():
    if(dev_predictions[j] == '<E>'):
      file.write('\n')
      j = j+1
    file.write(str(row[0]) + "\t" + row[1] + "\t" + dev_predictions[j]
+ '\n')
    j = j+1

# Download test prediction to local system

files.download(viterbi_test_output)

<IPython.core.display.Javascript object>

<IPython.core.display.Javascript object>
```