Name: Dipali Chandrakant Telavane
USC ID: 1487027774

HW2 Report

1. External libraries used
   Following external libraries were used in the homework:
   pandas, numpy, json

   Command to install these libraries:
   !pip install pandas
   !pip install numpy
   !pip install json

2. Python version
   - Python 3.11.1

3. Task 1: Vocabulary Creation
   - First, loaded the training dataset in the current location and imported it into dataframe.
   - Created a word_vocab dictionary to save all the words and their frequencies.
   - Set the frequency threshold as 4 and remove all the words with a frequency less than or equal to the threshold.
   - Created vocab.txt file using file operations.
   - Added total count of words having a frequency less than the threshold as <unk> words with their frequencies.

Q. What is the selected threshold for unknown words replacement?

A. **4**. All words having frequency less than or equal to 4 are replaced as unknown.

Q. What is the total size of your vocabulary?

A. **43193**

Q. what is the total occurrences of the special token '< unk >' after replacement?

A. **50296**

4. Task 2: Model Learning
   - First, replaced the pos_tags of words in training data having a frequency less than the threshold to '<unk>'
   - Created a list of unique pos_tags and as POSTag_list and their count as POSTag_count for future reference.
   - Traversed the train data sequentially to find transition probabilities of every possible combination in the dataset. Created dictionary trans_prob of (s, s') as key and probability as value.
   - Traversed the train data sequentially to find emission probabilities of every possible combination in the dataset. Created dictionary emi_prob of (s, x) as key and probability as value.
   - Dumped the dictionaries trans_prob and emi_prob in hmm.json file using json library.

- Converted both dictionaries to the matrix to account for combinations that did not occur in the dataset and their probability as 0.

Q. How many transition and emission parameters are in your HMM?

A. Total transition parameters: **2025**

Total emission parameters: **525960**

Total non-zero transition parameters: **1378**

Total non-zero emission parameters: **17116**

5. Task 3: Greedy Decoding with HMM
   - First, loaded the dev dataset in the current location and imported it into dataframe.
   - Created POSTag_prob list to calculate initial probabilities of every pos tag.
   - In the greedy algorithm, first checked if a word exists in the known word list. If not, then handled it as unknown.
   - Wrote separate logic to calculate possible probabilities of the first word in a sentence and other words separately.
   - Used the given formula in pdf to calculate possible probabilities.
   - Used argmax function to find the index of max probability.
   - Saved the tags in tags_greedy list with a tuple of word and its tag.
   - Accuracy of greedy model: **92.28644283892903**
   - Run the code on test data and created greedy.out file to store words and their predicted tags.

   Q. What is the accuracy on the dev data?

   A. **92.28644283892903**

6. Task 4: Viterbi Decoding with HMM
   - In the Viterbi algorithm, first separated the dev data in sentences and used the for loop of list of sentences to access every sentence and its word.
   - Created lists probs, current_tags and previous_tags to store respective data in every stage of algorithm.
   - For every word, checked if a word exists in the known word list. If not, then handled it as unknown.
   - Wrote separate logic to calculate possible probabilities of the first word in a sentence and other words separately.
   - In case of first word of sentences, used the given formula in pdf to calculate possible probabilities and then used argmax function to find the index of max probability.
   - Saved the data in probs, current_tags and previous_tags for future reference.
   - For other words in sentence, found the possible probabilities of every tag and stored then in dictionary te_products with key as tag from previous stage and list of possible probabilities with every tag as value.

- Converted dictionary te_products to two separate lists to identify max probabilities and its indices in every combination of tags.
- Added max of probabilities and indices in probs, current_tags dictionaries and stored previous tags used to obtain these values in previous_tags dictionary.
- If full stop is obtained, marked the sentence as finished.
- At the end of sentence, traversed previous_tags and current_tags in reverse manner to track the tags of every word in sentence and stored them in sentence_tags list.
- Reversed the sentence_tags list and appended it to global list tags_viterbi to store tag of every word in dev data.
- Accuracy of Viterbi model: **93.53712585756784**
- Run the code on test data and created viterbi.out file to store words and their predicted tags.

Q. What is the accuracy on the dev data?

A. **93.53712585756784**

7. Observations
   - Viterbi model gives better accuracy than Greedy model.

8. References

- https://www.kdnuggets.com/2019/11/create-vocabulary-nlp-tasks-python.html
- https://medium.com/data-science-in-your-pocket/pos-tagging-using-hidden-markov-models-hmm-viterbi-algorithm-in-nlp-mathematics-explained-d43ca89347c4
- https://realpython.com/python-append/#:~:text=Python%20provides%20a%20method%20called,Learning%20how%20to%20use%20.
- https://stackoverflow.com/questions/4111412/how-do-i-get-a-list-of-indices-of-non-zero-elements-in-a-list
- https://stackoverflow.com/questions/5914627/prepend-line-to-beginning-of-a-file
- https://stackoverflow.com/questions/903853/how-do-you-extract-a-column-from-a-multi-dimensional-array

Package imports

In [1]:

```
import pandas as pd
import numpy as np
import json
```

Read Data

In [2]:

```
df_train = pd.read_table("data/train", header=None, names = ['index', 'word', 'POS_tag'])
```

Task 1: Vocabulary Creation

In [3]:

```
word_vocab = {}

for i in df_train.index:
    word = df_train['word'][i]
    if word not in word_vocab:
        word_vocab[word] = 1
    else:
        word_vocab[word] +=1

word_vocab = dict(sorted(word_vocab.items(), key = lambda item: item[1], reverse = True))

threshold = 4
unknown = 0
i = 1

f = open("vocab.txt", "w")

for key,val in word_vocab.items():
    if val > threshold:
        text = key + "\\t" + str(i) + "\\t" + str(val) + "\n"
        f.write(text)
        i += 1
    else:
        unknown = unknown + val

f.close()

f = open("vocab.txt", "r+")

content = f.read()
f.seek(0, 0)
line = "<unk>\\t0\\t" + str(unknown)
f.write(line.rstrip('\r\n') + '\n' + content)

f.close()
```

In [4]:

```
print(len(word_vocab))
```

```
43193
```

Task 2: Model Learning

In [5]:

```
df_train['word'] = df_train['word'].apply(lambda x: '<unk>' if int(word_vocab.get(x, "0")) <= threshold else x)

POSTag_list = list(df_train['POS_tag'].unique())
word_list = list(df_train['word'].unique())

POSTag_count = {}

for item in POSTag_list:
    POSTag_count[item] = 0

for i in df_train.index:
    POSTag_count[df_train['POS_tag'][i]] += 1
```

In [6]:

```
trans_sum = {}
trans_prob = {}

for i in range(0, len(df_train)-1):
    if (df_train['POS_tag'][i], df_train['POS_tag'][i+1]) in trans_sum:
        trans_sum[(df_train['POS_tag'][i], df_train['POS_tag'][i+1])] += 1
    else:
        trans_sum[(df_train['POS_tag'][i], df_train['POS_tag'][i+1])] = 1

for key in trans_sum:
    trans_prob[key] = trans_sum[key] / POSTag_count[key[0]]
```

In [7]:

```
emi_sum = {}
emi_prob = {}

for i in range(0, len(df_train)):
    if (df_train['word'][i], df_train['POS_tag'][i]) in emi_sum:
        emi_sum[(df_train['word'][i], df_train['POS_tag'][i])] += 1
    else:
        emi_sum[(df_train['word'][i], df_train['POS_tag'][i])] = 1

for key in emi_sum:
    emi_prob[(key[1], key[0])] = emi_sum[key] / POSTag_count[key[1]]
```

In [8]:

```
stringified_trans_prob = {str(key): value for key, value in trans_prob.items()}
stringified_emi_prob = {str(key): value for key, value in emi_prob.items()}

with open("hmm.json", "w") as outfile:
    json.dump(stringified_trans_prob, outfile)
    json.dump(stringified_emi_prob, outfile)
```

In [9]:

```
m = n = len(POSTag_list)

trans_prob_matrix = [[0 for j in range(n)] for i in range(m)]

for i in range(m):
    for j in range(n):
        trans_prob_matrix[i][j] = trans_prob.get((POSTag_list[i], POSTag_list[j]), 0)

print("Total non-zero transition parameters: ", len(trans_prob))
print("Total transition parameters: ", m*n)

n = len(word_list)

emi_prob_matrix = [[0 for j in range(n)] for i in range(m)]

for i in range(m):
    for j in range(n):
        emi_prob_matrix[i][j] = emi_prob.get((POSTag_list[i], word_list[j]), 0)


print("Total non-zero emission parameters: ", len(emi_prob))
print("Total emission parameters: ", m*n)
```

```
Total non-zero transition parameters:  1378
Total transition parameters:  2025
Total non-zero emission parameters:  17116
Total emission parameters:  525960
```

Task 3: Greedy Decoding with HMM

In [10]:

```
POSTag_prob = [0] * len(POSTag_count)

i = 0
for key,val in POSTag_count.items():
    POSTag_prob[i] = val / len(df_train)
    i = i + 1
```

In [11]:

```
df_dev = pd.read_table("data/dev", header=None, names = ['index', 'word', 'POS_tag'])
```

In [12]:

```
def greedy_algo(df):

    tags_greedy = []
    first_word = True
    k = 0

    for i in df.index:
        te_product = []
        word = df['word'][i]
        if word in word_list:
            ind = word_list.index(word)
        else:
            ind = word_list.index('<unk>')

        if first_word:
            for j in range(len(POSTag_list)):
                te_product.append(POSTag_prob[j] * emi_prob_matrix[j][ind])
            first_word = False
        else:
            for j in range(len(POSTag_list)):
                te_product.append(trans_prob_matrix[prev_ind][j] * emi_prob_matrix[j][ind])
            if word == ".":
                first_word = True

        prev_ind = np.argmax(te_product)
        tags_greedy.append([word, POSTag_list[prev_ind]])

    return tags_greedy


dev_tags_greedy = greedy_algo(df_dev)
```

In [13]:

```
similarity = 0
for i in df_dev.index:
    if df_dev['POS_tag'][i] == dev_tags_greedy[i][1]:
        similarity += 1

accuracy = similarity / len(df_dev) * 100

print("Accuracy of Greedy model:", accuracy)
```

```
Accuracy of Greedy model: 92.28644283892903
```

In [14]:

```
df_test = pd.read_table("data/test", header=None, names = ['index', 'word', 'POS_tag'])

test_tags_greedy = greedy_algo(df_test)
```

In [15]:

```
f = open("greedy.out", "w")

index = 1
for i in df_test.index:

    if i != 0:
        if df_test['index'][i] == 1:
            index = 1
            text = " \n"
            f.write(text)
        else :
            index += 1

    text = str(index) + " \t " + str(test_tags_greedy[i][0]) + "\t" + str(test_tags_greedy[i][1]) + "\n"
    f.write(text)

f.close()
```

Task 4: Viterbi Decoding withHMM

In [16]:

```
def viterbi_algo(df):

    probs = []
    current_tags = []
    previous_tags = []

    prev_sent_len = 0
    tags_viterbi = []

    sentences = []
    sentence = []

    for i in df.index:
        if(df['word'][i] == "."):
            sentence.append(".")
            sentences.append(sentence)
            sentence = []
        else:
            sentence.append(df['word'][i])

    for i in range(len(sentences)):

        sentence_tags = []
        first_word = True
        te_products = {}

        for j in range(len(sentences[i])):

            te_product = []
            te_products = {}

            max_prob = 0
            max_ind = 0

            word = sentences[i][j]

            if word in word_list:
                ind = word_list.index(word)
            else:
                ind = word_list.index('<unk>')

            if first_word:

                prob = [0] * len(POSTag_list)
                current_tag = [0] * len(POSTag_list)
                previous_tag = [0] * len(POSTag_list)

                for k in range(len(POSTag_list)):
                    te_product.append(POSTag_prob[k] * emi_prob_matrix[k][ind])

                max_prob = np.max(te_product)
                max_ind = np.argmax(te_product)

                prob[max_ind] = max_prob
                current_tag[max_ind] = POSTag_list[max_ind]

                first_word = False

            else:

                for l in range(len(prob)):
                    te_product = []
                    if(prob[l] > 0): #selecting all previous probability with values greater than 0 i.e. only changed tags

                        for k in range(len(POSTag_list)):
                            te_product.append(trans_prob_matrix[l][k] * emi_prob_matrix[k][ind] * prob[l])

                        te_products[l] = te_product

                prob = [0] * len(POSTag_list)
                current_tag = [0] * len(POSTag_list)
                previous_tag = [0] * len(POSTag_list)

                list_of_keys = list(te_products.keys())
                list_of_values = list(te_products.values())

                max_values = []
                max_indices = []

                for column_index in range(len(POSTag_list)):
                    if(len(list_of_values) > 0):
                        max_prob = max([row[column_index] for row in list_of_values])
                        max_ind = np.argmax([row[column_index] for row in list_of_values])

                    if(max_prob > 0):
                        prob[column_index] = max_prob
                        current_tag[column_index] = POSTag_list[column_index]
                        previous_tag[column_index] = POSTag_list[list_of_keys[max_ind]]

                if word == ".":
                    first_word = True
                    current_tag[POSTag_list.index(".")] = "." #in case probability of one word converges to 0

            probs.append(prob)
            current_tags.append(current_tag)
            previous_tags.append(previous_tag)


        sentence_len = len(sentences[i])
        sentence_tags.append(".")

        for index in range(sentence_len - 1, 0, -1):

            if index == sentence_len - 1:
                prev_index = current_tags[prev_sent_len + sentence_len - 1].index(".")
            else:
                prev_index = current_tags[prev_sent_len + index].index(prev_tag)

            prev_tag = previous_tags[prev_sent_len + index][prev_index]

            sentence_tags.append(prev_tag)

        prev_sent_len += sentence_len

        sentence_tags.reverse()
        tags_viterbi.extend(sentence_tags)

    return tags_viterbi


dev_tags_viterbi = viterbi_algo(df_dev)
```

In [17]:

```
similarity = 0
for i in df_dev.index:
    if df_dev['POS_tag'][i] == dev_tags_viterbi[i]:
        similarity += 1

accuracy = similarity /len(df_dev) * 100

print("Accuracy of Viterbi model:", accuracy)
```

```
Accuracy of Viterbi model: 93.53712585756784
```

In [18]:

```
test_tags_viterbi = viterbi_algo(df_test)
```

In [19]:

```
f = open("viterbi.out", "w")

index = 1
for i in df_test.index:

    if i != 0:
        if df_test['index'][i] == 1:
            index = 1
            text = " \n"
            f.write(text)
        else :
            index += 1

    text = str(index) + " \t " + str(df_test['word'][i]) + " \t " + str(test_tags_viterbi[i]) + "\n"
    f.write(text)

f.close()
```

In [ ]: