Library used in this program

```
import nltk
nltk.download('brown')
nltk.download('universal_tagset')
import numpy as np
import pandas as pd
from nltk.corpus import brown
from tqdm import tqdm
from sklearn.model_selection import train_test_split, KFold
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
```

DataSet

```
# splitting the corpus into two for now
train_corpus, test_corpus = train_test_split(brown.tagged_sents(tagset='universal'), test_size=0.2)
```

```
class storage:
  #creating dataset for storing the computed data.
  def __init__(self):
    self.id = 0
    self.index2tag = dict()
    self.value2id = dict()
    self.values = set()

  def get_length():
    return self.id

  def insert(self, value):
    self.index2tag[self.id] = value
    self.value2id[value] = self.id
    self.values.add(value)
    self.id += 1

  def retrive(self, key, method ='id'):
    if method == 'id':
      return self.index2tag[key]
    elif key in self.values:
      return self.value2id[key]
    else:
      return None

# val variable for hadling some cases
alpha = 0.000001
```

HMM Model

```
def get_word(train_corpus):
  words = storage()

  for sent in tqdm(train_corpus):
    for word,tag in sent:
      if words.retrive(word.lower(), 'temp') == None:
        words.insert(word.lower())
  return words

def get_tag():
  tag_list = set([tag for words,tag in brown.tagged_words(tagset='universal')])
  tags = storage()

  for tag in tqdm(tag_list):
    if tags.retrive(tag, 'temp') == None:
      tags.insert(tag)
  return tags
```

```
words = get_word(train_corpus)
tags = get_tag()
```

```
100%|██████████| 45872/45872 [00:00<00:00, 50519.96it/s]
100%|██████████| 12/12 [00:00<00:00, 15787.84it/s]
```

```
def get_hmm_matrix(train_corpus, words, tags, alpha):
  transmission_matrix = np.zeros([tags.id,tags.id])
  emission_matrix = np.zeros([tags.id,words.id])
  tags_prob = np.zeros([tags.id])

  for sent in tqdm(train_corpus):
    for index in range(len(sent)):
      word = sent[index][0]
      tag = sent[index][1]

      word_index = words.retrive(word.lower(), 'temp')
      tag_index = tags.retrive(tag,'temp')

      tags_prob[tag_index] +=1
      emission_matrix[tag_index,word_index] +=1
      if index != len(sent) - 1:
        next_tag = tags.retrive(sent[index + 1][1], 'temp')
        transmission_matrix[tag_index,next_tag] +=1

  transmission_matrix = np.divide((transmission_matrix+ alpha), (np.reshape(tags_prob,[-1,1])+(alpha*12)))
  emission_matrix = np.divide((emission_matrix+alpha), (np.reshape(tags_prob,[-1,1])+ alpha*12))
  tags_prob = np.divide(tags_prob, np.sum(tags_prob))


  transmission_matrix[transmission_matrix == 0] = alpha
  emission_matrix[emission_matrix == 0] =alpha
  tags_prob[tags_prob == 0] = alpha

  return transmission_matrix,emission_matrix,tags_prob
```

```
transmission_matrix,emission_matrix,tags_prob = get_hmm_matrix(train_corpus,words,tags,alpha)
```

```
     100%|██████████| 45872/45872 [00:03<00:00, 13794.08it/s]
```

Double-click (or enter) to edit

```
# Viterbi algorithm

def get_pos(sent_list, transmission_matrix, emission_matrix, tags_prob, words, tags, alpha = 0.000001):
  if len(sent_list) == 0:
    return []

  seq_score_matrix = np.zeros([tags.id, len(sent_list)])
  back_pointer = np.zeros([tags.id, len(sent_list)])
  # First step in viterbi Intialization
  word_id = words.retrive(sent_list[0].lower(), 'temp')
  for i in range(tags.id):
    if word_id == None:
      seq_score_matrix[i,0] = tags_prob[i] * alpha
    else:
      seq_score_matrix[i,0] = tags_prob[i] * emission_matrix[i,word_id]
    back_pointer[i,0] = 0

# Second step is Iteration
for p in range(len(sent_list)):
  if p!= 0:
    for i in range(tags.id):
      word_id = words.retrive(sent_list[p].lower(), 'temp')
      transmission_vector = np.multiply(seq_score_matrix[:, p-1], transmission_matrix[:, i])
      tag_max_arg = np.argmax(transmission_vector)
      # print(len(transmission_vector
      back_pointer[i,p] = tag_max_arg
      # print(back_pointer)

      if word_id == None:
        seq_score_matrix[i,p] = transmission_vector[tag_max_arg] * alpha
      else:
        seq_score_matrix[i,p] = transmission_vector[tag_max_arg] * emission_matrix[i,word_id]

# Third Step is Sequence Identification
tag_index = np.zeros([len(sent_list)])
tag_index[-1] = np.argmax(seq_score_matrix[:, len(sent_list)-1])
# print(back_pointer)
# print(tag_index)
for i in reversed(range(len(sent_list)-1)):
  tag_index[i] = back_pointer[int(tag_index[i+1]), int(i+1)]
# print(tag_index)
return [tags.retrive(index, 'id') for index in tag_index]
```

```
import re
sent = 'the man saw bank near river bank'
sent = re.findall( r'\w+|[^\s\w]+', sent)
```

```
print(get_pos(sent, transmission_matrix,emission_matrix,tags_prob, words, tags))
```

```
    ['DET', 'NOUN', 'VERB', 'NOUN', 'ADP', 'NOUN', 'NOUN']
```

```
def prediction(test_corpus, transmission_matrix, emission_matrix, tags_prob, words, tags, alpha):
  confusion_matrix = np.zeros([tags.id,tags.id], dtype=np.int32)

  for test_sent in test_corpus:
    test_tag = [item[1] for item in test_sent]
    test_token = [item[0] for item in test_sent]

    predicted_tag = get_pos(test_token,transmission_matrix, emission_matrix, tags_prob, words, tags, alpha)
    for (predicted, test) in zip(predicted_tag, test_tag):
      confusion_matrix[tags.retrive(predicted,'temp'), tags.retrive(test,'temp')] += 1

  return confusion_matrix
```

Cross Validation

```
def cross_validation():
  tags = get_tag()
  confusion_matrix = np.zeros([tags.id,tags.id], dtype=np.int32)
  dataset = np.array(brown.tagged_sents(tagset='universal'))
  kfold = KFold(n_splits=5,shuffle=True)
  kfold.get_n_splits(dataset)

  for train, test in kfold.split(dataset):
    train_courpus = dataset[train]
    test_courpus = dataset[test]
    print("Train Data Size: ",len(train))
    print("Test Data Size: ",len(test))
    words = get_word(train_corpus)
    transmission_matrix,emission_matrix,tags_prob = get_hmm_matrix(train_corpus,words,tags,alpha)
    confusion_matrix += prediction(test_corpus, transmission_matrix, emission_matrix, tags_prob, words, tags, alpha)

  return confusion_matrix
```

```
confusion_matrix = cross_validation()
```

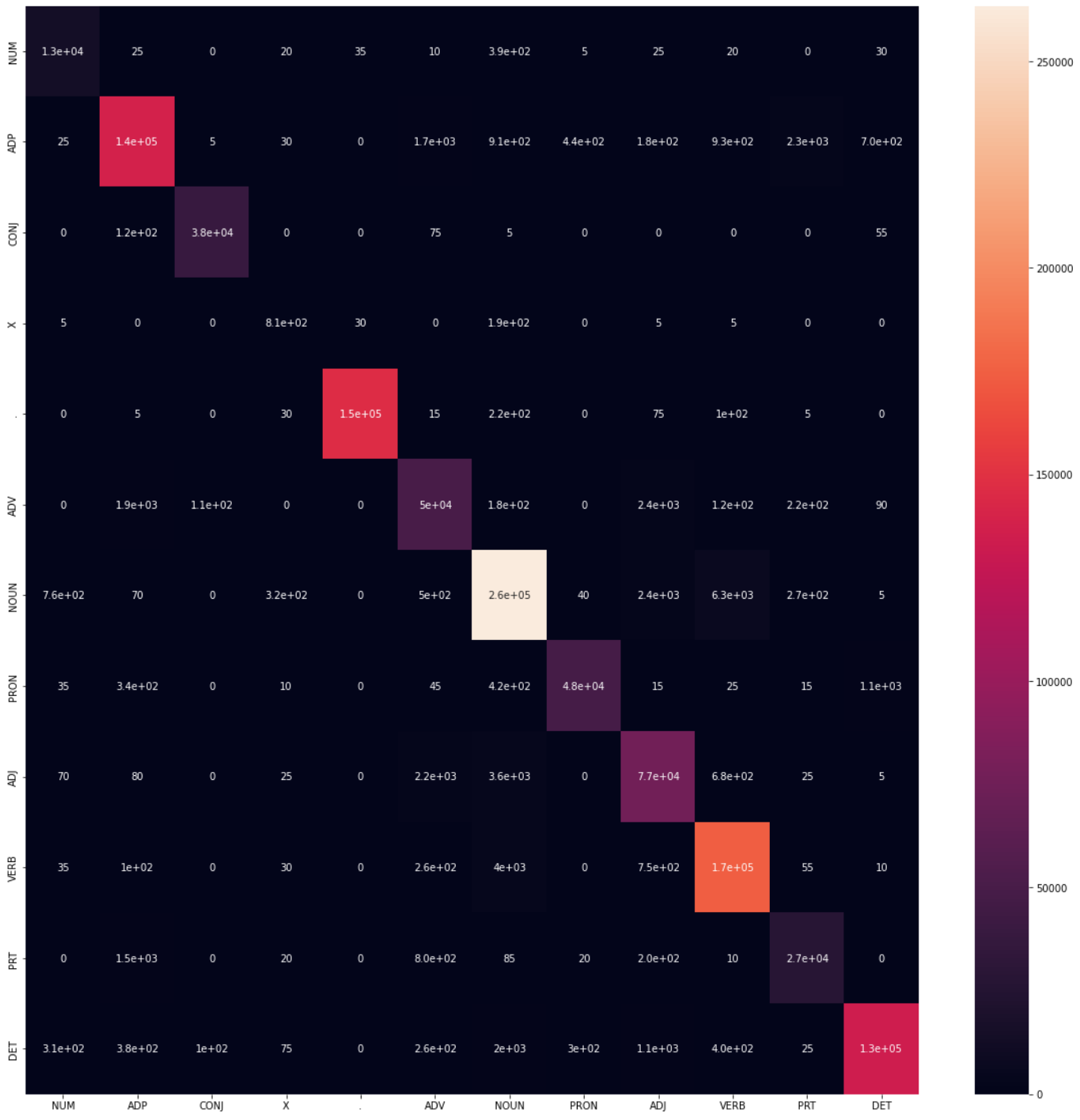
```
    100%|██████████| 12/12 [00:00<00:00, 84307.62it/s]
    /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: VisibleDeprecationWarning: Creating an ndarray from ra
      after removing the cwd from sys.path.
    Train Data Size:  45872
    Test Data Size:  11468
    100%|██████████| 45872/45872 [00:00<00:00, 52212.35it/s]
    100%|██████████| 45872/45872 [00:03<00:00, 13956.03it/s]
    Train Data Size:  45872
    Test Data Size:  11468
    100%|██████████| 45872/45872 [00:00<00:00, 52418.30it/s]
    100%|██████████| 45872/45872 [00:03<00:00, 14100.38it/s]
    Train Data Size:  45872
    Test Data Size:  11468
    100%|██████████| 45872/45872 [00:00<00:00, 52221.21it/s]
    100%|██████████| 45872/45872 [00:03<00:00, 13901.27it/s]
    Train Data Size:  45872
    Test Data Size:  11468
    100%|██████████| 45872/45872 [00:00<00:00, 51917.43it/s]
    100%|██████████| 45872/45872 [00:03<00:00, 13971.56it/s]
    Train Data Size:  45872
    Test Data Size:  11468
    100%|██████████| 45872/45872 [00:00<00:00, 52393.04it/s]
    100%|██████████| 45872/45872 [00:03<00:00, 13939.98it/s]
```

overall accuracy

```
#overall accuracy
total_examples = np.sum(confusion_matrix)
correct_predictions = np.trace(confusion_matrix)
print('The overall accuracy of the hmm model is:', correct_predictions * 100 / total_examples)
```

```
The overall accuracy of the hmm model is: 96.0176570693066
```

```
# plotting the heat map
plt.figure(figsize = (20, 20))
tag_list = [tags.retrive(i, 'id') for i in range(tags.id)]
confusion_figure = sns.heatmap(confusion_matrix, annot=True, xticklabels=tag_list, yticklabels=tag_list)
```




```
per_pos_dict = {'tag': [], 'precision': [], 'recall': [], 'f1-score': []}
for tag_id in range(tags.id):
  per_pos_dict['precision'].append(confusion_matrix[tag_id, tag_id] / np.sum(confusion_matrix[tag_id, :]))
  per_pos_dict['recall'].append(confusion_matrix[tag_id, tag_id] / np.sum(confusion_matrix[:, tag_id]))
  per_pos_dict['tag'].append(tags.retrive(tag_id, 'id'))
  per_pos_dict['f1-score'].append(2 * per_pos_dict['precision'][tag_id] * per_pos_dict['recall'][tag_id] / (per_pos_dict['r
per_pos_df = pd.DataFrame(per_pos_dict)
per_pos_df.to_csv('hmm_per_pos_accuracy.csv')
```

```
print(per_pos_df)
```

```
      tag  precision    recall  f1-score
0    NUM   0.959971  0.915474  0.937195
1    ADP   0.950139  0.968298  0.959132
2   CONJ   0.993482  0.994389  0.993935
3      X   0.775120  0.589091  0.669421
4      .   0.996934  0.999556  0.998243
5    ADV   0.908718  0.893963  0.901280
6   NOUN   0.960876  0.956444  0.958655
7   PRON   0.959453  0.983518  0.971336
8    ADJ   0.920172  0.914870  0.917513
9   VERB   0.970720  0.952881  0.961718
10   PRT   0.909153  0.900604  0.904858
11   DET   0.964596  0.985229  0.974803
```

```
Overall_precision = np.sum(per_pos_df['precision'])/12
Overall_recall = np.sum(per_pos_df['recall'])/12
```

```
f1_score = 2 * Overall_precision * Overall_recall / ( Overall_precision + Overall_recall)
f_half_score = 1.25 * Overall_precision * Overall_recall / ( (0.25*Overall_precision) + Overall_recall)
f2_score = 5 * Overall_precision * Overall_recall / ( (4*Overall_precision) + Overall_recall)
```

Fbeta = ((1 + beta^2) * Precision * Recall) / (beta^2 * Precision + Recall)

- F0.5-Measure (beta=0.5): More weight on precision, less weight on recall.
- F2-Measure (beta=2.0): Less weight on precision, more weight on recall
- F1-Measure (beta=1.0): Balance the weight on precision and recall.

```
print("Precision: ",Overall_precision)
print("Recall: ",Overall_recall)
print("F1_score: ",f1_score)
print("F0.5_score: ",f_half_score)
print("F2_score: ",f2_score)
```

```
Precision:  0.9391110574859298
Recall:  0.9211929771128807
F1_score:  0.9300657255970051
F0.5_score:  0.935471893707086
F2_score:  0.9247216838236312
```