q1

November 23, 2019

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from collections import defaultdict
        from sklearn.model_selection import train_test_split
```

```
In [2]: def isNumeric(x):
            return x.isnumeric()
```

```
In [3]: def getAllCapsInd(data):
            for i in range(len(data)):
                d = data[i]

                if allCaps(d):
                    return i
            return 0
```

```
In [4]: def getQ(v, w, u):
             return (trigram_counts[' '.join([w, u, v])] + lam) / (bigram_counts[' '.join([w, 
```

```
In [5]: def getE(x, v):
            return (emissions[' '.join([x.lower(), v])] + lam) / (tag_counts[v] + lam * vocab_
```

```
In [6]: file = open('Brown_train.txt')

        X = []
        y = []

        for line in file:

            line = line.rstrip()
            temp_line = line.split(' ')
            temp_sent_db = ['*', '*']
            temp_tag_db = ['*', '*']

            for word in temp_line:
                temp_word = word.split('/')
                temp_sent_db.append("".join(temp_word[0:-1]).lower())
                temp_tag_db.append(temp_word[-1])

            X.append(temp_sent_db)
            y.append(temp_tag_db)

In [7]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

In [8]: emissions = defaultdict(int)
        tag_counts = defaultdict(int)
        trigram_counts = defaultdict(int)
        bigram_counts = defaultdict(int)
        s_tags = []


        for i in range(len(X_train)):
            temp_sent_db = X_train[i]
            temp_tag_db = y_train[i]

            for i in range(len(temp_sent_db)):
                emissions[' '.join([temp_sent_db[i], temp_tag_db[i]])]+=1
                tag_counts[temp_tag_db[i]]+=1
                s_tags.append(temp_tag_db[i])


            for i, t in enumerate(temp_tag_db):
                if i+2 < len(temp_tag_db):
                    trigram_counts[' '.join([temp_tag_db[i], temp_tag_db[i+1], temp_tag_db[i+2
                if i+1 < len(temp_tag_db):
                    bigram_counts[' '.join([temp_tag_db[i], temp_tag_db[i+1]])] += 1

In [9]: tags = set(s_tags)
        vocab_size = len(tags)
        lam = 0.25

In [10]: tags

Out[10]: {'*',
          '.',
          'ADJ',
          'ADP',
          'ADV',
          'CONJ',
          'DET',
          'NOUN',
          'NUM',
          'PRON',
          'PRT',
          'VERB',
          'X'}
```

```
In [11]: def applyViterbi(sentence):
             pi = defaultdict(lambda: defaultdict(int))
             ws = defaultdict(lambda: defaultdict(lambda: ""))
             pi[0]["* *"] = 1

             n = len(sentence)
             max_pair = [None, 0]

             for k in range(1, n+1):

                 for u in tags:

                     for v in tags:
                         max_w = None
                         temp = 0

                         for w in tags:
                             temp = pi[k-1][w + ' ' + u] * getQ(v, w, u) * getE(sentence[k-1],

                             if pi[k][u + " " + v] < temp or max_w is None:
                                 pi[k][u + " " + v] = temp
                                 max_w = w

                         ws[k][u + " " + v] = max_w

                 if k == n:
                     qValue = getQ(".", u, v)
                     if max_pair[0] is None or max_pair[1] < pi[n][u + " " + v] * qValue:
                         max_pair = [[u, v], pi[n][u + " " + v] * qValue]


             answer = [*max_pair[0]]
             answer.reverse()
             for k in range(n-1, 0, -1):
                 t = ws[k][answer[-1] + " " + answer[-2]]
                 answer.append(t)

             answer.reverse()
             return answer
```

```
In [12]: l = applyViterbi(X[0])
```

```
In [13]: l, y[0]
```

```
Out[13]: (['*',
           '*',
           'X',
           '*',
           'ADP',
           'DET',
           'NOUN',
           'NOUN',
           'NOUN',
           'VERB',
           'ADJ',
           'CONJ',
           'ADJ',
           'NOUN',
           'PRT',
           'VERB',
           'PRON',
           'PRON'],
          ['*',
           '*',
           'ADP',
           'DET',
           'NOUN',
           'NOUN',
           'NOUN',
           'VERB',
           'ADJ',
           'CONJ',
           'ADJ',
           'NOUN',
           'PRT',
           'VERB',
           'DET',
           'NOUN',
           '.'])
```

```
In [14]: tagwise_correct_wrong_counts = {}
         for t in tags:
             tagwise_correct_wrong_counts[t] = [0, 0]
```

```
In [15]: len(X_test)
```

```
Out[15]: 5499
```

```
In [16]: y_pred = []
         z = 0
         for s in X_test:
             y_pred.append(applyViterbi(s))
             if z % 100 == 0:
                 print(z)
             z += 1
```

```
0
100
```

```
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
2400
2500
2600
2700
2800
2900
3000
3100
3200
3300
3400
3500
3600
3700
3800
3900
4000
4100
4200
4300
4400
4500
4600
4700
4800
4900
```

```
5000
5100
5200
5300
5400
```

```
In [17]: for i in range(len(y_pred)):
             p = y_pred[i]
             r = y_test[i]

             for j in range(len(r)):
                 tagwise_correct_wrong_counts[r[j]][1] += 1
                 if r[j] == p[j]:
                     tagwise_correct_wrong_counts[r[j]][0] += 1

         for k in tagwise_correct_wrong_counts:
             if tagwise_correct_wrong_counts[k][1] == 0:
                 tagwise_correct_wrong_counts[k] = 'NA'
             else:
                 tagwise_correct_wrong_counts[k] = tagwise_correct_wrong_counts[k][0] / tagwise

         tagwise_correct_wrong_counts
```

```
Out[17]: {'ADP': 0.06747862924578259,
          'PRT': 0.041392285983066796,
          '.': 0.05596833553225471,
          'ADV': 0.04526449736411561,
          'ADJ': 0.03814562160657107,
          'VERB': 0.12719176175897579,
          'DET': 0.020172454710861484,
          'NOUN': 0.10238429172510519,
          'NUM': 0.08603667136812412,
          'CONJ': 0.014632799558255107,
          '*': 1.0,
          'X': 0.27722772277227725,
          'PRON': 0.07051857168490938}
```