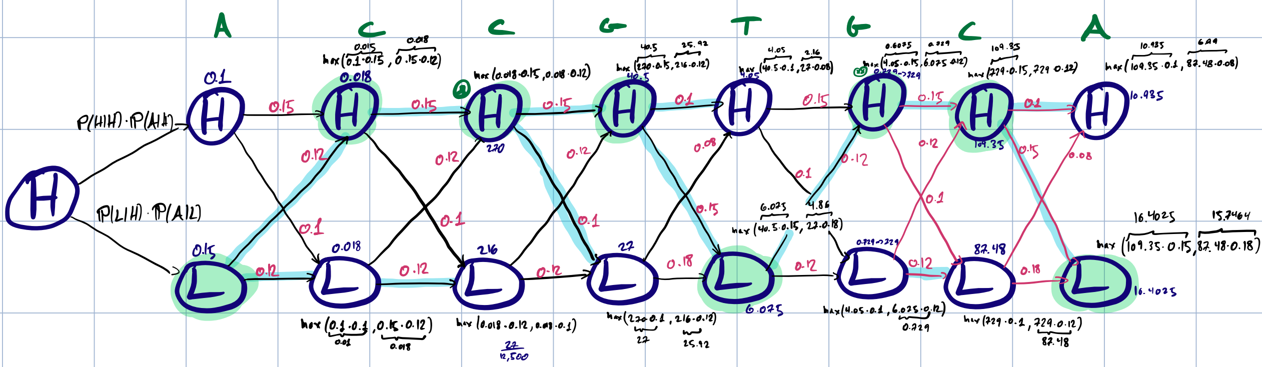
תרגיל 2 - NLP

נג'ב פרחאת 208965251

1. (10 pts) Consider this (toy) biological setup:
A cell can be in one of two states - H, for high GC-content, and L for low GC. On each time step the cell produces one nucleotide, A,C,T or G, and might also change its state. The probability of changing from state H to L is 0.5, and from state L to H is 0.4.
In state H the probabilities for producing nucleotides are 0.2 for A, 0.3 for C, 0.3 for G and 0.2 for T.
In L the probabilities are 0.3 for A, 0.2 for C, 0.2 for G and 0.3 for T.
Consider the nucleotide sequence $S = ACCGTGCA$. Use the Viterbi algorithm to find the best state-sequence and calculate the probability of S given this state-sequence. Assume the previous state before S was H.

① נתאר קודם את המצבים ואת המעברים:



P(H|H) : 0.5
P(L|H) : 0.5
P(L|L) : 0.6
P(H|L) 0.4

P(A|H)=0.2
P(C|H) =0.3
P(G|H)=0.3
P(T|H)=0.2

P(A|L)=0.3
P(C|L) =0.2
P(G|L)=0.2
P(T|L)=0.3

– הערה, במעבר ②, הכפלתי את ההסתברות ב 100,000 כדי לא לעבוד עם מספרים קטנים.

לכן ** הכפלתי ב 1,000.

לכן נקבל שהרצף הוא:

L-H-H-H-L-H-H-L

2. (10 pts) In class we saw the trigram HMM model and the corresponding Viterbi algorithm. We will now make two main changes. First, we will consider a four-gram tagger, where p takes the form:

$$p(x_1 \cdots x_n, y_1 \cdots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-3}, y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i) \quad (1)$$

We assume in this definition that $y_0 = y_{-1} = y_{-2} = *$, where $*$ is the START symbol, $y_{n+1} = STOP$, and $y_i \in \mathcal{K}$ for $i = 1 \cdots n$, where $\mathcal{K}$ is the set of possible tags in the HMM.
Second, we consider a version of the Viterbi algorithm that takes as input **an integer** n (and not a sentence $x_1 \cdots x_n$ as we saw in class) and finds

$$\max_{y_1 \cdots y_{n+1}, x_1 \cdots x_n} p(x_1 \cdots x_n, y_1 \cdots y_{n+1})$$

for a four-gram tagger, as defined in Equation 1. $x_1 \cdots x_n$ may range over the values of some fixed vocabulary $\mathcal{V}$. Complete the following pseudo-code of this version of the Viterbi algorithm for this model . The pseudo-code must be efficient.

Input: An integer n, parameters $q(w|t, u, v)$ and $e(x|s)$.
Definitions: Define $\mathcal{K}$ to be the set of possible tags. Define $\mathcal{K}_{-2} = \mathcal{K}_{-1} = \mathcal{K}_0 = \{*\}$, and $\mathcal{K}_k = \mathcal{K}$ for $k = 1 \cdots n$. Define $\mathcal{V}$ to be the set of possible words.
Initialization: $\cdots$
Algorithm: $\cdots$
Return: $\cdots$

2 ראשית נתאר תיאור High-level של האלגוריתם.

האלגוריתם דומה כמו של Viterbi רגיל, רק שהפעם ל 3 קודקודים

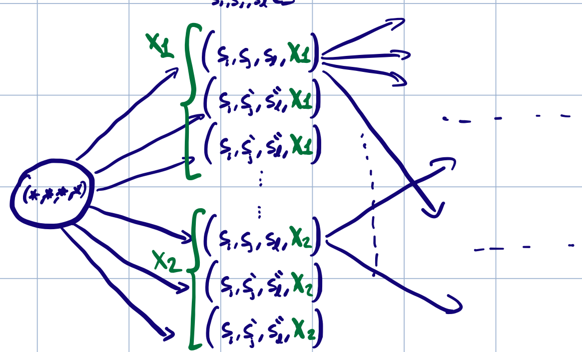
נחזיק שלושה labels שלמעלה מתארים את שלושת ה-label-ים

הקודמים, בהתאם ל 4-gram.

כמו כן לכל קודקוד נחזיק את x המקסימלי, כאשר $x \in V$

!נחזיר את הטבלה האחרונה.

כאשר, אם $V=\{x_1,x_2\}$, SL בגודל 4 יראה ראשון כך:



כעת, נתאר את הפסאודו קוד:

:Init

$$\text{Set } \pi(0,\underbrace{*,*,*}_{\text{מייצג את ה labels הקודמים}},*)=1$$

מייצג כל x אפשרי (כי צב הנוכחי הם 0).

:Algorithm

For $k=1,\ldots\ldots n$:

For $u_2\in S_{k-2}, u_1\in S_{k-1}, u\in S_k, x\in V$:

$$\pi(k,u_2,u_1,u,x)=\max_{\substack{w\in S_{k-3}\\ x_1\in V}}\left(\pi(k-1,w,u_2,u_1,x_1)\cdot q(u\mid w,u_2,u_1)\cdot e(x\mid u)\right)$$

Return:

$$\text{Return}\max_{\substack{u_2\in S_{n-2},\,u_1\in S_{n-1}\\ u\in S_n,\,x\in V}}\left(\pi(n,u_2,u_1,u,x)\cdot q(STOP\mid u_2,u_1,u)\right)$$

חלק מעשי

(b) **Implementation of the most likely tag baseline**

i. Using the training set, compute for each word the tag that maximizes $p(tag|word)$, based on the maximum likelihood estimation. Assume that the most likely tag of all the unknown words is "NN". (Unknown words are words that appear in the test set but not in the training set.)

ii. Using the test set, compute the error rate (i.e., $1-accuracy$) for known words and for unknown words, as well as the total error rate.

```
Error rate for known words is: 0.0704399684933048
Error rate for unknown words is: 0.743455497382199
total Eroor rate is: 0.14731386424798165
```

b

(c) **Implementation of a bigram HMM tagger**

i. Training phase: Compute the transition and emission probabilities of a bigram HMM tagger directly on the training set using maximum likelihood estimation.

ii. Implement the Viterbi algorithm corresponding to the bigram HMM model. (Choose an arbitrary tag for unknown words.)

iii. Run the algorithm from c)ii) on the test set. Compute the error rates and compare to the results from b)ii).

c

```
Error rate for known words is: 0.16057162146956228
Error rate for unknown words is: 0.7382198952879582
total Eroor rate is: 0.22655237715538723
```

(iii) נראה שיש ירידה בביצועים של המודל לעומת המודל בסעיף הקודם.
זה הגיוני, היות שהמידע ב Bigram-Hmm-model יכול להיות נמוך, כי ה emission
מחושב רק על $P(x|y)$, ולכן תלוי במילים הקודמות.

עם זאת, עבור ה unknown words, יש שיפור בביצועים.

(d) **Using Add-one smoothing**

i. Training phase: Compute the emission probabilities of a bigram HMM tagger directly on the training set using (Laplace) Add-one smoothing.

ii. Using the new probabilities, run the algorithm from c)ii) on the test set. Compute the error rates and compare to the results from b)ii) and c)iii).

(e) Using pseudo-words

(d)

```
With Smoothing:
Error rate for known words is: 0.16293462360751665
Error rate for unknown words is: 0.7347294938917975
Total Eroor rate is: 0.22824678560749523
```

נשים לב שיש שיפור משמעותי רק ב unknown words, בהשוואה
למודל Bigram-Hmm ללא החלקה.

כמו כן, מתקיים שעדיין ב Bigram-Hmm הבסיסי התוצאות פחות טובות; ולכן
נקבל שה Total error נמוך יותר מהמודל ב d.
(למרות שעבור ה unknown words אכן יש שיפור).

(e) **Using pseudo-words**

i. Design a set of pseudo-words for unknown words in the test set and low-frequency words in the training set.

ii. Using the pseudo-words as well as maximum likelihood estimation (as in c)i)), run the Viterbi algorithm on the test set. Compute the error rates and compare to the results from b)ii), c)iii) and d)ii).

iii. Using the pseudo-words as well as Add-One smoothing (as in d)i)), run the Viterbi algorithm on the test set. Compute the error rates and compare to the results from b)ii), c)iii), d)ii) and e)ii). For the results obtained using both pseudo-words and Add-One smoothing, build a confusion matrix and investigate the most frequent errors. A confusion matrix is an $|\mathcal{K}|$ over $|\mathcal{K}|$ matrix, where the (i, j) entry corresponds to the number of tokens which have a true tag i and a predicted tag j.

e

```
With Pseudo Words:
Error rate for known words is: 0.17321986426801683
Error rate for unknown words is: 0.6573333333333333
Total Eroor rate is: 0.20940895046347052
```

נשים לב שכאן כבר אפשר לראות שיפור יותר משמעותי ב error של ה unknown words בהשוואה למודל הקודם.

כמו כן, בהשוואה למודל ב c ו-d, המודל הנוכחי מצליח להשיג את התוצאות הטובות ביותר.

מבחינת ה Total Error, המודל ב d הוא עדיין הכי טוב, אבל מבחינת ה Error unknown words, המודל הנוכחי הוא הכי טוב.

iii

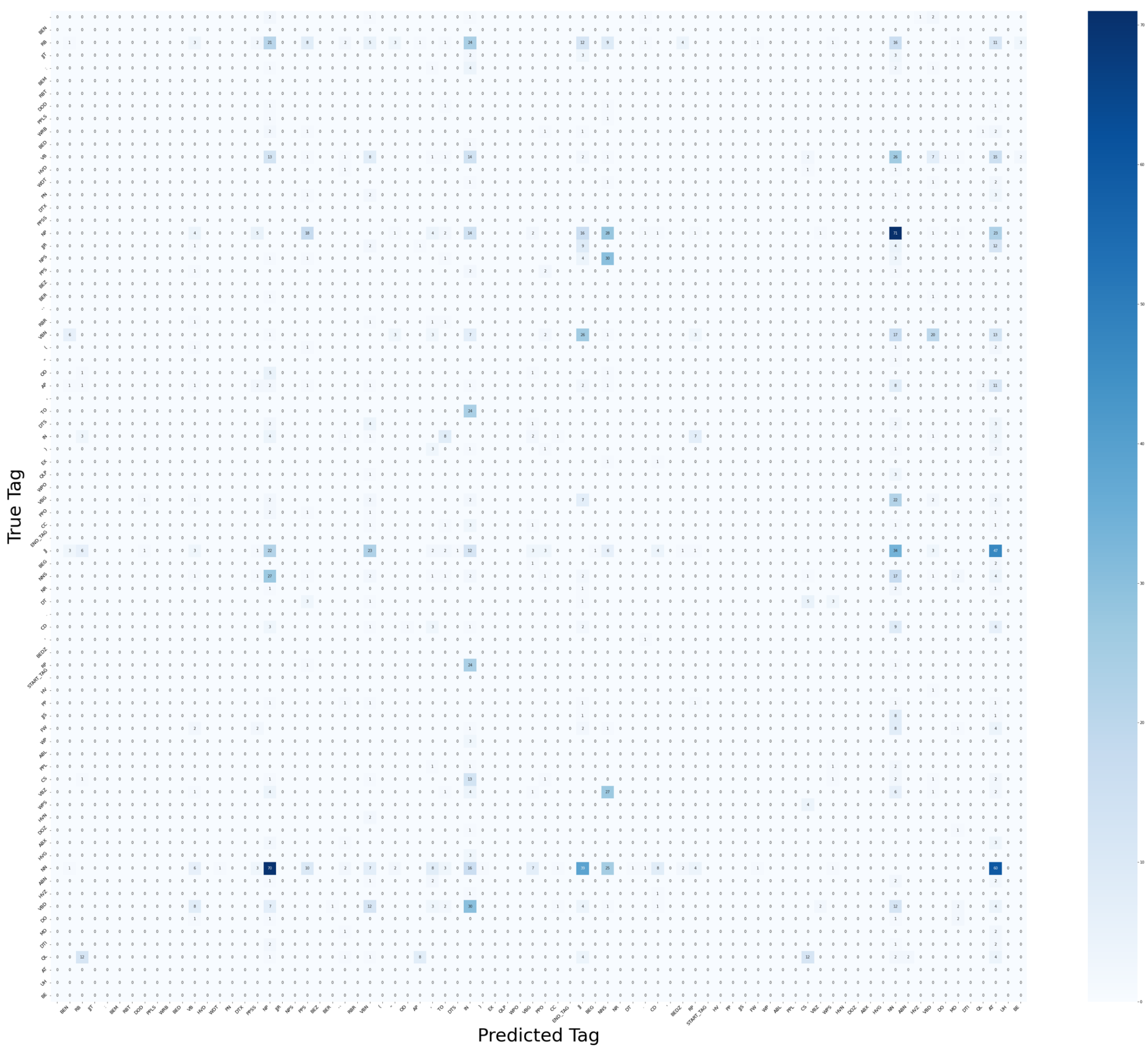
```
With Pseudo Words and Add One smoothing:
Error rate for known words is: 0.1361628783798341
Error rate for unknown words is: 0.6253333333333333
Total Eroor rate is: 0.1727299910296023
```

ניתן לראות שהמודל הנוכחי מצליח לשפר את ה Error של ה unknown words.

מבחינת ה Total Error, המודל ב a היה אמין יותר אך,
אם נשים לב שמבין המודלים שיצרנו, המודל השני מספק את התוצאות
הכי טובות על ה unknown words.

ניתוח ב confusion matrix:

*קשה קצת לראות, צירפתי את התמונה גם בקובץ נפרד בתיקיית zip.



ניתן לראות שהמודל שלנו טועה בעיקר כאשר ה tag האמיתי הוא "JJ", ובמקום מנחש "NN".

ניתן לראות שעוד טעות נפוצה היא כאשר ה tag האמיתי הוא "NN", ובמקום מנחש "AT".