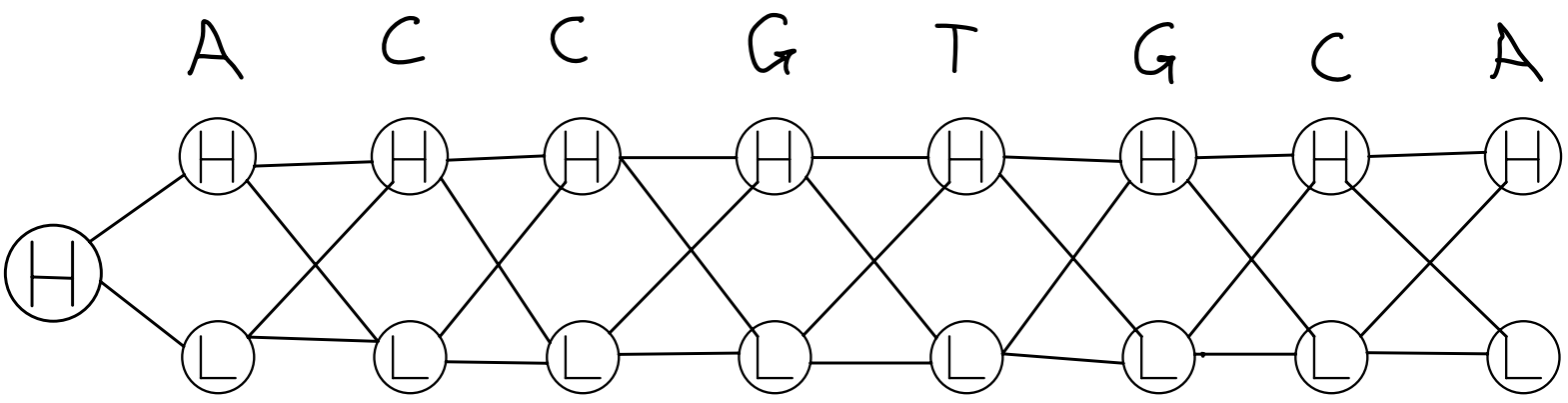
עיבוד שפה טבעית 67658

תרגיל ②

אורי רביד - 207766478

שיר רשקובסקי - 209144013

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.



כעת, נגדיר לכל את רצף ה-S: $S_i^* = s_1,\dots,s_i$ עבור $1 \le i \le 8$ $\pi(i,H)$ - ההסתברות עבור רצף ה-states

הסביר ביותר $\alpha^*_1,\dots,\alpha^*_i$ המתאים את הרצף S_i^* כך ש: $\alpha^*_i = H$ ו: $\pi(i,L)$ - ההסתברות עבור רצף

ה-states הסביר ביותר $\alpha^*_1,\dots,\alpha^*_i$ המתאים את הרצף S_i^* כך ש: $\alpha^*_i = L$. נוסף לכל מרצף s_i^* נשמור

את המצביעים μ_i^H ו-μ_i^L אל ה-state α^*_{i-1} ברצף ה-states הסביר ביותר כאשר α^*_i הוא H או L בהתאמה.

ראשית, נבחין כי לאור שה-state ההתחלתי $\alpha_0 = H$ מתקיים $\mu_1^H = \mu_1^L = H$ וכי:

$$\pi(1,H) = P(S_1 = A \mid H)\cdot P(H \mid H) = 0.2 \cdot (1-0.5) = 0.1$$

$$\pi(1,L) = P(S_1 = A \mid L)\cdot P(L \mid H) = 0.3 \cdot 0.5 = 0.15$$

כעת, נבחין כי לכל $2 \le i \le 8$ מתקיים לפי הרקורסיה כי:

$$\pi(i,\beta) = \max_{\alpha\in\{H,L\}} \{\pi(i-1,\alpha)\cdot P(S_i \mid \beta)\cdot P(\beta \mid \alpha)\} \quad \text{ו-} \quad \mu_i^\beta = \operatorname{argmax}_{\alpha\in\{H,L\}} \{\pi(i-1,\alpha)\cdot P(S_i \mid \beta)\cdot P(\beta \mid \alpha)\}$$

לכן, נקבל כי:

$$\pi(2,H) = \max\{\pi(1,H)\cdot P(S_2=C \mid H)\cdot P(H \mid H),\ \pi(1,L)\cdot P(S_2=C \mid H)\cdot P(H \mid L)\} =$$

$$= \max\{0.1\cdot 0.3\cdot 0.5,\ 0.15\cdot 0.3\cdot 0.4\} = \max\{0.015,\ 0.018\} = 0.018 \Rightarrow \mu_2^H = L$$

$$\pi(2,L) = \max\{\pi(1,H)\cdot P(S_2=C \mid L)\cdot P(L \mid H),\ \pi(1,L)\cdot P(S_2=C \mid L)\cdot P(L \mid L)\} =$$

$$= \max\{0.1\cdot 0.2\cdot 0.5,\ 0.15\cdot 0.2\cdot 0.6\} = \max\{0.01,\ 0.018\} = 0.018 \Rightarrow \mu_2^L = L$$

$$\pi(3,H) = \max\{\pi(2,H)\cdot P(S_3=C \mid H)\cdot P(H \mid H),\ \pi(2,L)\cdot P(S_3=C \mid H)\cdot P(H \mid L)\} =$$

$$= \max\{0.018\cdot 0.3\cdot 0.5,\ 0.018\cdot 0.3\cdot 0.4\} = \max\{0.0027,\ 0.00216\} = 0.0027 \Rightarrow \mu_3^H = H$$

$$\pi(3,L) = \max\{\pi(2,H)\cdot P(S_3=C \mid L)\cdot P(L \mid H),\ \pi(2,L)\cdot P(S_3=C \mid L)\cdot P(L \mid L)\} =$$

$$= \max\{0.018\cdot 0.2\cdot 0.5,\ 0.018\cdot 0.2\cdot 0.6\} = \max\{0.0018,\ 0.00216\} = 0.00216 \Rightarrow \mu_3^L = L$$

$$\pi(4,H) = \max\{\pi(3,H)\cdot P(S_4=G \mid H)\cdot P(H \mid H),\ \pi(3,L)\cdot P(S_4=G \mid H)\cdot P(H \mid L)\} =$$

$$= \max\{0.0027\cdot 0.3\cdot 0.5,\ 0.00216\cdot 0.3\cdot 0.4\} = \max\{0.000405,\ 0.0002592\} = 0.000405 \Rightarrow \mu_4^H = H$$

$$\pi(4,L) = \max\{\pi(3,H)\cdot P(S_4=G \mid L)\cdot P(L \mid H),\ \pi(3,L)\cdot P(S_4=G \mid L)\cdot P(L \mid L)\} =$$

$$= \max\{0.0027\cdot 0.2\cdot 0.5,\ 0.00216\cdot 0.2\cdot 0.6\} = \max\{0.00027,\ 0.0002592\} = 0.00027 \Rightarrow \mu_4^L = H$$

$$\pi(5,H) = \max\{\pi(4,H)\cdot P(s_5=T\mid H)\cdot P(H\mid H),\ \pi(4,L)\cdot P(s_5=T\mid H)\cdot P(H\mid L)\} =$$

$$= \max\{0.0000h05, 0.0000216\} = 0.0000h05 \Rightarrow \mu_5^H = H$$

$$\pi(5,L) = \max\{\pi(4,H)\cdot P(s_5=T\mid L)\cdot P(L\mid H),\ \pi(4,L)\cdot P(s_5=T\mid L)\cdot P(L\mid L)\} =$$

$$= \max\{0.0000h075, 0.0000h486\} = 0.0000h075 \Rightarrow \mu_5^L = H$$

$$\pi(6,H) = \max\{\pi(5,H)\cdot P(s_6=G\mid H)\cdot P(H\mid H),\ \pi(5,L)\cdot P(s_6=G\mid H)\cdot P(H\mid L)\} =$$

$$= \max\{0.00000h075, 0.00000729\} = 0.00000729 \Rightarrow \mu_6^H = L$$

$$\pi(6,L) = \max\{\pi(5,H)\cdot P(s_6=G\mid L)\cdot P(L\mid H),\ \pi(5,L)\cdot P(s_6=G\mid L)\cdot P(L\mid L)\} =$$

$$= \max\{0.0000h05, 0.00000729\} = 0.00000729 \Rightarrow \mu_6^L = L$$

$$\pi(7,H) = \max\{\pi(6,H)\cdot P(s_7=C\mid H)\cdot P(H\mid H),\ \pi(6,L)\cdot P(s_7=C\mid H)\cdot P(H\mid L)\} =$$

$$= \max\{0.00000h0955, 0.0000008748\} = 0.00000h0955 \Rightarrow \mu_7^H = H$$

$$\pi(7,L) = \max\{\pi(6,H)\cdot P(s_7=C\mid L)\cdot P(L\mid H),\ \pi(6,L)\cdot P(s_7=C\mid L)\cdot P(L\mid L)\} =$$

$$= \max\{0.0000007729, 0.0000008748\} = 0.0000008748 \Rightarrow \mu_7^L = L$$

$$\pi(8,H) = \max\{\pi(7,H)\cdot P(s_8=A\mid H)\cdot P(H\mid H),\ \pi(7,L)\cdot P(s_8=A\mid H)\cdot P(H\mid L)\} =$$

$$= \max\{0.0000000h0935, 0.0000000h0798\} = 0.0000000h0935 \Rightarrow \mu_8^H = H$$

$$\pi(8,L) = \max\{\pi(7,H)\cdot P(s_8=A\mid L)\cdot P(L\mid H),\ \pi(7,L)\cdot P(s_8=A\mid L)\cdot P(L\mid L)\} =$$

$$= \max\{0.0000000h1625, 0.0000000h1574\} = 0.0000000h1625 \Rightarrow \mu_8^L = H$$

כאן, נשים לב שכדי להגיע ל- states הסביר ביותר $x_1, \ldots, x_8$ עבור S, נראה כי עלינו להתחיל מ $\pi(8,H)$ ו- $\pi(8,L)$ ולחשב כי:

$$x_8^* = \arg\max_{x\in\{H,L\}} \{\pi(i,x)\} = L$$

ולאחר מכן, נבדוק כי $x_{i-1}^* = \mu_i^{x_i^*}$, ולכן נקבל כי states הסביר ביותר עבור S הינו:

$$L, H, H, H, L, H, H, L$$

נחשב כי ההסתברות $S \cdot S$ שבו נקבל states זה הינה:

$$P(A\mid L)\cdot P(C\mid H)\cdot P(C\mid H)\cdot P(G\mid H)\cdot P(T\mid L)\cdot P(G\mid H)\cdot P(C\mid H)\cdot P(A\mid L) = 0.3^8 = 0.000065900195$$

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:

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

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: $\cdots$

נגדיר לכל $1 \leq i \leq n$ ולכל $a \in K_i$, $b \in K_{i-1}$ ו- $c \in K_{i-2}$ את $\pi(i,c,b,a)$ שהוא ה"ג את ההסתברות המקסימלית לפני סדרה כך ש $y_i = a$, $y_{i-1} = b$ ו- $y_{i-2} = c$. נגדיר נוסחה רקורסיבית ל- π ע"י:

$$i=1 \quad \pi(1,c,b,a) = \max_{x \in V} \{ q(a|*,*,*) \cdot e(x|a) \}$$

$$i \geq 2 \quad \pi(i,c,b,a) = \max_{\substack{d \in K_{i-3} \\ x \in V}} \{ \pi(i-1,d,c,b) \cdot q(a|d,c,b) \cdot e(x|a) \}$$

ונשמור לכל רביעיה $\alpha = (i,c,b,a)$ מערכים μ^α ו- σ^α אשר מכילים את $d \in K_{i-3}$ ו- $x \in V$ המתאימים אשר מקסמו את המשוואה הנ"ל.

נחשב את $\pi(i,a,b,c)$ לכל $1 \leq i \leq n$ בסדר עולה, לכל אפשרויות לבחור של a,b,c

כעת, נמצא את הערכים של y_n, y_{n-1} ו- y_{n-2} ע"י:

$$y_n, y_{n-1}, y_{n-2} = \arg\max_{\substack{a \in K_n \\ b \in K_{n-1} \\ c \in K_{n-2}}} \{ \pi(n,c,b,a) \cdot q(STOP|c,b,a) \}$$

כעת, נמצא את שאר הערכים בעזרת μ^α כי נחשב את $y_1, \ldots, y_{n-3}$ כאשר:

$$y_i = \mu^{(i+3, y_{i+1}, y_{i+2}, y_{i+3})}$$

ובנוסף נשמור את $x_1, \ldots, x_n$ כך ש: $x_i = \sigma^{(i, y_{i-2}, y_{i-1}, y_i)}$

לבסוף, כאשר מצאנו את $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ נחזיר את הערך של $\max_{y_1 \ldots y_n, x_1 \ldots x_n} P(y_1, \ldots, y_n, x_1, \ldots, x_n)$ כדלהלן:

$$q(y_1, \ldots, y_n, x_1, \ldots, x_n) = \prod_{i=1}^{n+1} q(y_i | y_{i-3}, y_{i-2}, y_{i-1}) \cdot \prod_{i=1}^{n} e(x_i|y_i) \quad (y_{n+1} = STOP, \; y_0 = y_{-1} = y_{-2} = *)$$

(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.

נראה כי אחוזי השגיאה שקיבלנו על שיערוך התגים באמצעות שימוש ב: Maximum Likelihood Estimation הינם:
מודל זה תמיד חוזה את התג של מילה מסוימת להיות התג שהופיע
יחד עם המילה הכי הרבה יחסית לשאר התגים. עבור מילים לא מוכרות,
שלא הופיעו בסט האימון, תמיד חזינו את התג להיות "NN". מכיוון שתג זה
יחסית נפוץ ומילים רבות נוטות להופיע איתו, לפעמים הפרדיקציה מצליחה ולכן
השגיאה היא בערך ב-3/4 מהפעמים. אמנם מכיוון שהמודל לא מתחשב
כלל בקונטקסט, בסביבה בה המילה מופיעה, וניתן תמיד את אותו החיזוי לתג של מילה, ובשפה מילים
עשויות להיות חלק Part of Speech שונים בהקשרים שונים, השגיאות המתקבלות הן גבוהות.
* "NN" מייצג בדרך כלל שם עצם, יחיד או מסה. לדוגמה שם עצם המייצג אדם, מקום, דבר או רעיון.

Known Words Error Rate: 0.07044

Unkown Words Error Rate: 0.74346

Total Error Rate: 0.14731

(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).

נראה כי אחוזי השגיאה שקיבלנו על שיערוך התגים ע"י הרצת מודל Bigram Hidden Markov Model:

Known Words Error Rate: 0.13795

Unkown Words Error Rate: 0.74346

Total Error Rate: 0.20712

נבחין כי השגיאה על המילים הלא ידועות שווה לשגיאה מן הסעיף הקודם. זה נובע מכך ששני המודלים מתייחסים באותה צורה למילים
לא ידועות (בחרנו "NN" כתג הארבטררי למילים לא ידועות והגדרנו $e(x|y)=0$ לכל $y\neq$"NN" ו-$e(x|y)=1$ עבור $y=$"NN"
ולכן בעצם שאם המסלול האופטימלי ל-NN מתפספס אז כן בכל זאת נבחר במסלול זה).

בנוסף ניתן לראות כי השגיאה על המילים הידועות עלתה ביחס לסעיף הקודם. ניתן להסביר זאת ע"י ה-Sparsity
שקיים במודל Bigram, אשר כמות זוגות (תג, תג) האפשריים גבוהה יותר ממספר התגים האפשריים ולכן קיימת דלילות רבה
של מידע זמין לאימון המכוון את הפרמטרים של המודל, שמקשה לשערך נכון את ההסתברויות המעברים, ובכך משפיע על תוצאות החיזוי.
באופן דומה נוכל לומר על דלילות של (מילה, תג), מכיוון שישנן מילים שיכולות להופיע עם מספר רב של תגים ולכן בקורפוס סופי נתקלים
בדלילות. בזוגות שלא נצפו באימון מקבלים הסתברות אפס, מה שעלול להוביל לאיפוס ההסתברות לקבלת כל רצפי התגים עבור משפט מסוים
ולבחירה אקראית של תגים
לסיכום, נבחין כי השגיאה הכוללת של המודל עלתה ביחס לסעיף הקודם, בגלל השגיאה שעלתה על מילים ידועות.

(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).

נראה כי אחוזי השגיאה שקיבלנו על שיערוך תגיות ע"י הנחת מודל Bigram Hidden Markov Model עם שימוש ב- Add-One Smoothing:

Known Words Error Rate: 0.14369

Unkown Words Error Rate: 0.71291

Total Error Rate: 0.20871

נבחין כי בשונה מהסעיפים הקודמים השגיאה על המילים הלא ידועות ירדה. זאת מכיוון שה-add 1-smoothing על ה-emission Probabilities נתנה הסתברות גבוהה מאפס עבור זוגות מילים-תגיות שלא הופיעו באימון, וכך יכולנו לתת שערוך לטובת תגית המתאימה גם למשפט שמורכב ממילים שלא נצפו באימון ביחד עם תג מסוים. בשונה מהמצב שקיבלנו בסעיף הקודם כך: להתמודד עם מילה שלא הופיעה בסט האימון, המשערך את המסלול הבחור לעבור בתג "NN" האקראי שבחרנו למילים לא מוכרות. הפעם הבחירה של התג עבור המילה הלא מוכרת מושפעת מכמות הפעמים שהתג הופיע באימון ומהסתברויות המעבר התגיות (מכיוון ש- $q_{add-\delta}(x|y) = \frac{c(x|y)+\delta}{c(y)+\delta|V|}$. פה גם נתחשב במסלול האופטימלי עד כה ולא רק במסלול האופטימלי ל- NN). לכן היינו מצפים לראות שיפור.

* ככל שתג מופיע יותר פעמים באימון היינו מצפים בהסתברות גבוהה יותר שיופיע את המילה שהוא סופג, ולכן נרצה שגודל c(y) ישפיע

אולם, ניתן לראות כי השגיאה על מילים ידועות עלתה ביחס לסעיף הקודם. גם זאת ניתן להסביר ע"י החלקת ההסתברויות - פעולה זו שינתה את שיערוך ה- emission Probabilities על מילים ידועות, והקטינה אותם כדי לספק שיערוך הסתברותי למילים לא ידועות, ושינוי זה יכל להשפיע על יכולת המודל למצוא את התג הנכון עבור כל מילה.

לסיום, נבחין כי השגיאה הכוללת של המודל עלתה ביחס לסעיפים הקודמים, זאת ניתן להסביר ע"י כך

שכפי שדיברנו בשיעור, כדי להימנע מהסתברויות ששוות לאפס, נוכל להפעיל שיטות החלקה כדי לתת גם לאירועים נדירים (ביחס לסט אימון והתקווה גם במציאות) הסתברות גדולה מאפס. Add-one-smoothing היא שיטה כזאת, אולם, היא עדיין נותנת הסתברות קטנה לאירועים שלא נצפו בסט אימון ולכן ה- MLE לא אמין. כמו כן בהינתן תגית כלשהי אנחנו נותנים לכל הזוגות (מילה, תג) שלא נצפו יחד באימון הסתברות שווה. וזה כמובן לא שערוך טוב מספיק כי לא סביר שההתפלגות האמיתית של מילים שלא היו באימון בהינתן התג הזה לא אחידה.

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

Word Class:	detection		
Capitalized	first word is uppercase (and not all word)	Adverb	ends with "able" or "ible"
Acronym	all uppercase	Adjective	Starts with ly
Hyphenated	contains "-"		
Gerund	ends with "ing"		
Plural	ends with s (and not ss)		
Past	ends with "ed"		
Numeral	all chars are digits		
Currancy	contains "$"		
Percentage	contains "%"		
Alphenumeric	contains numbers and not belong to previous options		
Unknown	others		

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).

c- Pseudo-words:

Known Words Error Rate: 0.12580

Unkown Words Error Rate: 0.34904

Total Error Rate: 0.1513

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.

נראה כי אחוז השגיאות שקיבלנו על שיערוך התגים ע"י הרצת מודל Bigram Hidden Markov Model עם שימוש ב- Pseudo-Words ו- Add-One Smoothing:

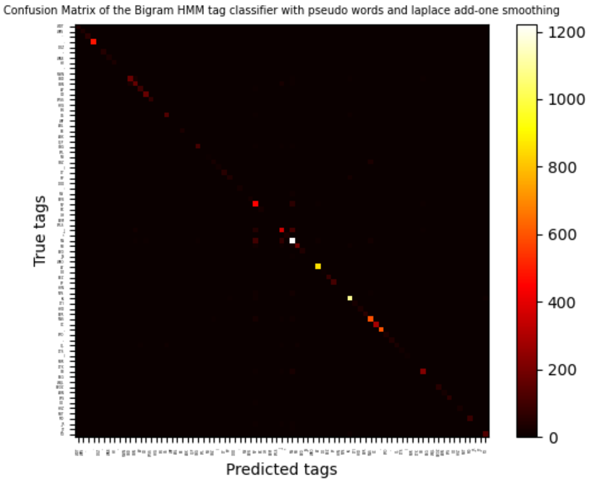
Known Words Error Rate: 0.11084

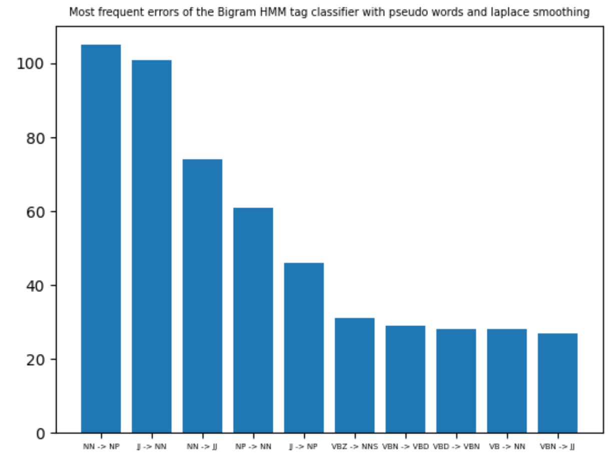
Unkown Words Error Rate: 0.33595

Total Error Rate: 0.133595

נראה כי התרומה של שימוש ב: Pseudo-Words ביחד Add-One smoothing הוריד את d סוג השגיאות, בניגוד לסעיף d, שם התרומה הייתה רק עבור מילים לא מוכרות. זאת מכיוון שבסעיף זה המודל נחשף בכמות גדולה בהרבה לרוב המילים, גם לאלו שהיו בסט מוכרות/בסט מבחן מכך שמספר רב של מילים עם תכונות דומות (תגים דומים) מופו לאותה מילה, ולכן ההסתברות מאוד נמוכה המודל יכול במבחן במילה (פסאודו מילה) שלא נתקל טוב מה התג שמתאים לה. בנוסף, בגלל יותר אם $C(x|y)$ קטן, אז ההסתברות של y יפלוט את x היא באמת קטן, ולכן ה- MLE יותר אמין. כמו כן ההחלקה באמת תורמת למניעת הסתברות 0 וקביעת סדרת תגים אקראית במקרים שהם באמת נדירים ולעומת קודם, שלא ידענו בהכרח לומר שבכל מקרה נדיר ולא נובע מדלילות הקורפוס).

* תשובה: $q_{add-\delta}(x|y) = \frac{c(x|y)+\delta}{c(y)+\delta|V|}$





ניתן לשים לב שהריבוע המשמעותי הוא על האלכסון, משמע שהמודל חוזה בצורה טובה - לרוב actual tag= predicted tag.

בנוסף ניתן לנתח את הטעויות הנפוצות ולראות שהמודל טועה בעיקר במקרים באמת מבלבלים, שנובעים מכך שהשפה רב משמעית.

- למשל, NN->NP, NP->NN, הן השגיאות השכיחות ביותר. שני תגים אלו מתייחסים בדיוק כלל לשמות עצם. הבלבול בין שני התגים האלו יכול לנבוע מכך שהאופי הבייגרמי של ה-HMM לא מספק הקשר מספק כדי להבדיל בין שמות עצם כלליים לשמות עצם נכונים במקרים מסוימים. בנוסף, מיפוי של מילים לפסאודו מילים, כמו למשל Capitalized, אינו תורמת לזיהוי עצמיות של שמות עצם.

- או הבלבול בין JJ->IN, IN->JJ כלל שמבלבל בין מילות יחס לשם תואר. ניתן לייחס את הבלבול למילים דו משמעותיות - מילים מסוימות יכולות לשמש הן כשמות תואר והן כמילות יחס, בהתאם לזה הקשר. לדוגמה, המילה "לפני": בנוסף, הכללת מילים בתדירות נמוכה כפי שעשינו עם הפסאודו-מילים עשויה לטשטש את ההבחנה בין שתי הקטגוריות הללו.

```
################## MLE Most Likely Tag Model POS Tagger Results ##################
Error rate for known words: 0.07044
Error rate for unknown words: 0.74346
Total error rate: 0.14731
################## Bigram HMM Tagger POS Tagger Results ##################
Error rate for known words: 0.13795
Error rate for unknown words: 0.74346
Total error rate: 0.20712
################## Bigram HMM Tagger with Add-One Smoothing POS Tagger Results ##################
Error rate for known words: 0.14369
Error rate for unknown words: 0.71291
Total error rate: 0.20871
################## Bigram HMM Tagger with Pseudo Words POS Tagger Results ##################
Error rate for known words: 0.12580
Error rate for unknown words: 0.34904
Total error rate: 0.15130
################## Bigram HMM Tagger with Pseudo Words and Add-One Smoothing POS Tagger Results ##################
Error rate for known words: 0.11084
Error rate for unknown words: 0.33595
Total error rate: 0.13655
```

Natural Language Processing - Exercise 2

Due: 11.2.2024 23:59

Please submit a single zip file. The zip file should contain your code files, a README txt file and a single pdf file for the answers to the questions

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.

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:

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

   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: $\cdots$

3. (80 pts) In this programming exercise with Python, we will implement several versions of an HMM POS tagger.

Note: For all the below sections, unknown words are words that appear in the test set but did not appear in the training set.

(a) Use the NLTK toolkit for importing the Brown corpus. This corpus contains text from 500 sources, and the sources have been categorized by genre. Here we will use a portion of the corpus: the "news" category. Load the tagged sentences for this portion of the corpus. Then, divide the obtained corpus into training set and test set such that the test set is formed by the last 10% of the sentences.

(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.

(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).

(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**

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.

Note: NLTK is a Python package for NLP. Please see here `http://www.nltk.org/` for downloading and documentation. You can download the Brown corpus using *nltk.download('brown')* and traverse its sentences using the *tagged_sents()* method. Please refer to the NLTK documentation for further information. Please use NLTK just for loading and reading the corpus.

A given word w may be annotated with a complex tag t, containing the symbols '+' and/or '-'. When encountering such a complex tag t, consider only the prefix of t that comes before the first occurrence of '+' or '-' in t as the POS tag of w.