עיבוד שפות טבעיות

תרגיל בית 2

Salam Qais,327876116

Abbas Ismail 214742025

שלב 1)

def calc_grams(self):

```python
def calc_grams(self):
    try:
        for _, row in self.corpus_data.iterrows():
            prot_type = row['protocol_type']
            sentence = row['sentence_text']

            if self.type == prot_type:
                tokens = sentence.strip().split(" ")
                tokens=["<s1>"]+["<s2>"]+tokens
                for idx in range(len(tokens)):
                    self.Unigrams[tokens[idx]] += 1
                    if idx < len(tokens) - 1:
                        self.Bigrams[(tokens[idx], tokens[idx + 1])] += 1
                    if idx < len(tokens) - 2:
                        self.Trigrams[(tokens[idx], tokens[idx + 1], tokens[idx + 2])] += 1


        self.dicinoary_size = len(self.Unigrams)
        self.total_words = sum(self.Unigrams.values())
    except Exception as e:
```

פונקציה זו פשוט מוסיפה שני טוקני דמי בהתחלה של כל משפט

ואז מחשבת את מספר ההופעות של כל Unigram,Bigram,Trigram שנמצא ב Courps

שלנו,וספורת את מספר מילים השונים ואת מספר המילים.

def calculate_prob_of_sentence:

```python
def calculate_prob_of_sentence(self, sentence, smoothing="Linear"):
    tokens = sentence.strip().split()
    total_log_prob = 0.0

    for idx in range(2, len(tokens)):
        curr_trigram = (tokens[idx - 2], tokens[idx - 1], tokens[idx])
        curr_bigram = (tokens[idx - 2], tokens[idx - 1])
        curr_unigram = tokens[idx - 2]

        num_of_curr_trigram = self.Trigrams.get(curr_trigram, 0)
        num_of_curr_bigram = self.Bigrams.get(curr_bigram, 0)
        num_of_curr_unigram = self.Unigrams.get(curr_unigram, 0)
        V = self.dicinoary_size
        trigram_prob = (num_of_curr_trigram + 1) / (num_of_curr_bigram + V)
        bigram_prob = (num_of_curr_bigram + 1) / (num_of_curr_unigram + V)
        unigram_prob = (self.Unigrams[tokens[idx]] + 1) / (self.total_words + V)
```

עובר על כל Unigram,Bigram,Tigram במשפט

-מחשבת מספר ההופעות של Unigram,Bigram,Tigram הנוכחי מתוך מה שהגדרנו בפונקציה קודמת.

-מחשבת הסתברות של כל אחד מהם על ידי החלקת Lablace ומכפילה כל אחד בדילתא שמתאים

```
if smoothing == "Linear":
    tmp_prob = self.lambda1 * trigram_prob + self.lambda2 * bigram_prob + self.lambda3 * unigram_prob
elif smoothing == "Laplace":
    tmp_prob = trigram_prob
```

ואז בודקת מה ה smothing הנתון ומחשבת ה tmp_prop בהתאם.

```
        if tmp_prob > 0:
            total_log_prob += math.log(tmp_prob)
        else:
            total_log_prob += float('-inf')

    return total_log_prob
```

ואז היא מוסיפה את log ההסתברות של ה unigram,trigram,bigram הנוכחים

(בפועל אנחנו היינו צריכים כל פעם להכפיל את tmp_prop של חלק זה ממשפט בtmp_prop אחרי שהאידקס של הלולאה גדל באחד וככה עד סוף הלולאה

,אבל עשינו את זה כי log של מכפלה הוא סכום של log ים של כל אחד).

def generate_next_token:

```python
def generate_next_token(self, sentence):
  try:
    max_probality = float('-inf')
    tmp_tkn = None
    sentnce_tokens = sentence.strip().split(' ')
    if("<s1>" not in sentnce_tokens):
        sentnce_tokens=["<s1>"]+["<s2>"]+sentnce_tokens
    if len(sentnce_tokens) > 2:
        sentnce_tokens = sentnce_tokens[-2:]
        sentence = sentnce_tokens[0] + ' ' + sentnce_tokens[1]
    for token in (self.Unigrams.keys()):
      if (("<s1>" not in token) and ("<s2>" not in token)):
        tmp_sentence = sentence + " " + token
        prob = self.calculate_prob_of_sentence(tmp_sentence, smoothing: "Linear")
        if prob > max_probality:
            max_probality = prob
            tmp_tkn = token
    next_token = tmp_tkn
    return next_token
  except Exception as e:
```

-פו פשוט אם קבלתי משפט אם אין לו s1 (זה אומר שאין לו גם s2)אני מוסיף אותם

- אם קבלתי משפט שהוא מאורך יותר מ2 ,אז הייתי משאיר רק שתי מילים אחרונות אחרת משאיר כל המילים.(אין אחרת כי לפחות יהיה s1,s2)

ואז אנחנו נעבור על כל מילה שהיא ב unigram מה שאומר שהיא נמצא ב voc שלנו

-אני שולח את הקבוצה שלי עם מילה הנוכחית הזו ל calculate_prop_of_sentnce

**אז אחרי שעובר על כל האפציות ב voc אני מחזיר את ה token שנותן הסתברות מקסמאלית.

השתמשתי במקדמים

$$\lambda_3 = 0.0000001, \qquad \lambda_2 = 0.2999999, \qquad \lambda_1 = 0.7$$

$$\begin{aligned}\hat{P}(w_n \mid w_{n-2}w_{n-1}) &= \lambda_1 P(w_n \mid w_{n-2}w_{n-1}) \\ &+ \lambda_2 P(w_n \mid w_{n-1}) \\ &+ \lambda_3 P(w_n)\end{aligned}$$

$\lambda_1 = 0.7$

בחרתי את הערך הזה כי אנחנו בונים מודל מבוסס Trigrams ואז בטח אנחנו רוצים שההשפעה החזקה תהיה מצד ה trigrams

$\lambda_2 = 0.2999999$

בחרתי את הערך הזה כי אני רוצת לתת קצת השפעה מה bigrams

כי שמתי לב שיש הרבה זוגות של מילים שהם באים ביחד ואז זה נותן להם קצת חשיבות אבל גם לא יותר מה trigrams כי בסוף זה מודל מבוסס Trigrams

$\lambda_3 = 0.000001$

זה נראה קצת מוזר כי בחרתי ערך מאוד קטן אבל זה הכי טוב

כי יש לנו מילים שהם מופיעים הרבה פעמים לדוגמה:

אלה 3 המילים שמופעים הכי פעמים ב plenary

```
,(16851 ,'את') ,(56771 ,'.') ,(85966 ,',')]
```

ואלה לגבי committee

```
,(11728 ,'את') ,(34760 ,',') ,(36527 ,'.')]
```

במקרה כזה גם אם בוחרים $\lambda_3 = 0.1$

זה לא מספיק קטן כי ההסתברות של למשל "," היא מאוד גדולה יחסית למילים אחרים זה גורר שרוב הפעמים המודל שלנו ינבה "," אבל אין משמעות לזה לכן נתתי λ_3 מאוד קטן, חוץ מזה שקצת חד משמעית לנבא מילה לפי מופעים שלה בלי התייחסות למקום שלה.

לגבי אתגרים שנובעים מהוספת s1, s2:

קודם כל אני הוספתי את תוקני הדמה בשלב ה preprocessing

1)ואז הבעיה הייתה אם אני מקבל משפט שאין פו s1,s2 למשל בשלב 3 אין להם הם לא עברו בבריברוסיסנג אז פשוא הייתי מוסיף אותם בהתחלת המשפט בפונקציה generate_next_token

```
def generate_next_token(self, sentence):
    try:
        max_probality = float('-inf')
        tmp_tkn = None
        sentnce_tokens = sentence.strip().split(' ')
        if("<s1>" not in sentnce_tokens):
            sentnce_tokens=["<s1>"]+["<s2>"]+sentnce_tokens
```

אם s1 לא נמצא ברור שגם s2

2)בעייה שניה היא שלא רציתי לנבא s1,s2 בשימוש ב genrate_next_token

הפתרון היה פשוט שלא אקח מילים שהם (s1 or s2)

אז ככה אני אף פעם לא אנבא אותם כי אני לא בודק אותם בכלל

```
for token in (self.Unigrams.keys()):
    if (("<s1>" not in token) and ("<s2>" not in token)):
        tmp_sentence = sentence + " " + token
        prob = self.calculate_prob_of_sentence(tmp_sentence, smoothing: "Linear")
        if prob > max_probality:
```

אז אני עכשו רוצה להסביר למה הוספתי אותם ב preprocessing ולא במקום אחק ובמה זה עוזר להוסיף אותם :

אם נתנו ל "" לgenrate_next_token

אז היא לא תבחר מילה אקראית

הפונרציה תוסיף שני s ים בהתחלה

אז עכשיו יש לנו (<s1>,<s2>)

ואז עכשיו המשפט שלנו בגודל 2 ,אנחנו נעבור על כל מילה אפשרית ב voc למשל word1

ואז מקבלים(<s>,<s>,word1)

אז הקבוצות הנוכחיות הן:

unigram=<s>,bigram=<s>,<s> ,trigram = <s>,<s>,word1

ואז מחשבים את ההסברות של משפט זה שנניח במקרה החלקה לינארית הוא יהיה

$$\frac{num\_of\_curr\_unigram+1}{voc\_size+num\_of\_words}+\frac{num\_of\_curr\_bigram+1}{voc\_size+num\_of\_curr\_unigram}+\frac{num\_of\_curr\_trigram+1}{voc\_size+num\_of\_curr\_bigram}=$$

$$\frac{1}{voc\_size+num\_of\_words}+\frac{1}{voc\_size}+\frac{num\_of\_curr\_trigram+1}{voc\_size+num\_of\_curr\_bigram}$$

אז אני אסביר מה זה מהווה , קודם כל את אלה הם קבועים עבור כל מילה כי הם לא תלויים ב word נוכחי

$$\frac{1}{voc\_size+num\_of\_words}+\frac{1}{voc\_size}$$

את זה תלוי בכמה פעמים הופיע ה trigram (<s>,<s>,word1)

$$\frac{num\_of\_curr\_trigram + 1}{voc\_size + num\_of\_curr\_bigram}$$

אז זה מה שנותן ההסתברות באמת כי השאר קבועים לכל מילה

ואז על ידי הוספת את שני הטוקנים האלה אנחנו יכולים לתת הסתברות קטנה למילים שהם אף פעם לא יהיו בתחילת משפטים כי אני מוסיף את טוקני הדמה בהתחלה אם הם אף פעם לא היו בהתחלה אז כשספרתי את מספר ה trigrams הן לא נמצאות אף פעם

לעומת זאת אם יש מילה שיש הרבה משפטים שהם מתחילים במילה הזאת אז יהיה לה הרבה מופעים <s>,<s>,word1 במה שספרתי בהתחלה ולכן יהיה לה יותר הסתברות לקחת אותה.

אז זאת הסיבה שאני החלטי להוסיף אותם בזמן ה preprocessing

ולהוסיף אותם למשפטים שאין להם אותם ב genrate_next_token כדי שאם רצו לנבא מילה ראשונה במשפט אז יהיה קצת הגיון

שלב 2)

אני אסביר את הפונקציה

```
get_k_n_collocations
```

```
n_grams = Counter()
for _, row in corpus_data.iterrows():
    sentence = row['sentence_text']
    tokens = sentence.strip().split(" ")
    for i in range(len(tokens) - n + 1):
        sent_n = tuple(tokens[i:i + n])
        n_grams[sent_n] += 1
```

קודם כל אני סופר את כל הקולוקציות מאורך n

```
if type == "frequency":
    k_common_kolk = n_grams.most_common(k)
```

אם רוצים לפי תדירות סיימנו זה קל מאוד .

```python
elif type == "tfidf":
    num_of_sentences_with_term = Counter()
    num_of_sentences = 0
    total_tfidf = {}
    term_counts = Counter()

    for _, row in corpus_data.iterrows():
        num_of_sentences += 1
        sentence = row['sentence_text']
        tokens = sentence.strip().split(" ")
        seen_terms = set()

        for i in range(len(tokens) - n + 1):
            sent_n = tuple(tokens[i:i + n])
            seen_terms.add(sent_n)

        for term in seen_terms:
            num_of_sentences_with_term[term] += 1
```

אחרת

-מחשבים מספר המשפטים ב corpus נתון

אחרי זה בודק בלולאה מה ה terms שהופיעו

ואז בלולאה אחרת עבור כל term מוספים אחד לתוך counter

לא הוספתי ל counter מיידית כי אני רוצה לספור פעמים ש term הופיע במשפטים שונים ולא אותו משפט

```
for _, row in corpus_data.iterrows():
    sentence = row['sentence_text']
    tokens = sentence.strip().split(" ")
    tf_sent = Counter()

    for i in range(len(tokens) - n + 1):
        sent_n = tuple(tokens[i:i + n])
        tf_sent[sent_n] += 1

    for term, tf in tf_sent.items():
        tf_last = (tf / (len(tokens) - n + 1))
        idf = math.log(num_of_sentences / (num_of_sentences_with_term[term] + 1))
        tfidf = tf_last * idf
        total_tfidf[term] = total_tfidf.get(term, 0) + tfidf
        term_counts[term] += 1
```

ואז עוברים על כל שורה

וסופר את מספר ההופעות של כל קולקציה

ואז מחשב את ה tf והidf

ומכפיל אותם

```
avg_tfidf = Counter()
for term, tfidf in total_tfidf.items():
    avg_tfidf[term] = tfidf / term_counts[term]

k_common_kolk =avg_tfidf.most_common(k)
```

ואז פו עושה ממוצע לכל tfidfכפי שדרשתם על פני משפטים שמכילים ה term

ואז לוקחים ה k קולקציוצ מקסמאלית.

```
tmp_kolk = []
for tmp, _ in k_common_kolk:
    tmp_kolk.append(tmp)
k_common_kolk = tmp_kolk

return k_common_kolk
```

פו אני מתעלם מהמספרים (תדירות ,tfidf)

ומחזיר רק הקולקציות מקסמאליות ממוינות

והשתמשתי ב

```
def print_res(self):
```

כדי להדפיס שלב 2 לא אתייחס אליה כי זה מיותר פשוט השתמשתי בפונקציה קודמת והייתי מדפיס

ולגבי הסתברות של משפט פשוט השתמשתי בcalc_prop_of_sentnce שעשיתי בשלב 1

שלב 3)

הגדרתי לשלב זה מחלקה בשם test שיש לה שני אוביקטים מהמחלקה

Trigram_lm

-אחד ל plearny

-אחד לcommitte

```
class Test:
    def __init__(self, corpus_data):
        self.plenary_Trigram = Trigram_LM(corpus_data, protocol_type: "plenary")
        self.committee_Trigram = Trigram_LM(corpus_data, protocol_type: "committee")
```

Def complete_sentnce:

```
    def complete_sentences(self, type):
        generated_sentences = []
        genera_tokens = []
        real=[]
        with open('masked_sentences.txt', 'r', encoding='utf-8') as file:
            for line in file:
                real.append(line)
                last_sent = ""
                new_tokens_for_sent = ""
                parts = line.strip().split('[*]')
                for idx in range(len(parts)):
                    if idx < len(parts) - 1:
                        if type == 'plenary':
                            token = self.plenary_Trigram.generate_next_token(parts[idx])
                            new_tokens_for_sent = new_tokens_for_sent + token + ','
                            last_sent += parts[idx] + token
                            parts[idx] = parts[idx] + token
                        elif type == 'committee':
                            token = self.committee_Trigram.generate_next_token(parts[idx])
                            new_tokens_for_sent = new_tokens_for_sent + token + ','
                            last_sent += parts[idx] + token
                            parts[idx] = parts[idx] + token
                    else:
```

```
                    parts[idx] = parts[idx] + token
                else:
                    last_sent += parts[idx]
                    new_tokens_for_sent = new_tokens_for_sent[:-1]

                    genera_tokens.append(new_tokens_for_sent)

            generated_sentences.append(last_sent)

    return generated_sentences, genera_tokens,real
```

את האמת שפו אין הרבה מה להסביר פשוט חלקתי את המשפט לחלקים לפי הכובים

ואז התחלתי לעבור על החלקים ובודק מה הסוג של המשפט or comiittee plenary

ואז בהתאם אני משתמש באופיקט של המחלרה lm_trigram

אם plenary משתמש ב plenary אופיקט

אחרת משתמש ב committee

ואז על פעם הייתי מנבה מה ה tokens אחרי part נוכחי ושומר אותו

וככה עד הסוף ואז אני מחזיר את ה tokens שהוספתי וגם את המשפט החדש

וגם את המפשט המקורי.

לגבי הדפסה עשיתי אותה ב

```
def print_res(self):
```

זה קצת מיותר להסביר עליהה פשוט הייתי משתמש ב complete_sentnce

```
plenary_complete, plenary_tokens,real = self.complete_sentences("plenary")
committee_complete, committee_tokens,real = self.complete_sentences("committee")
```

ואז מתחיל להדפיס

שלב 4)

1)

כן היה הבדל משמעותי ,בשלב 3 רוב ה [*] ,כל אחד ממודלי השפה החליף אותה במשהו שונה,רק מקרים מאוד פחותים הם כן היו נותנים אותה החלפה ,הסיבה של הדבר הזה היא ש בכל אחד ממודלי השפה אמנו אותו על data שונה כל אחד יש לו משפטים אחרים יש לו הסתבריות אחרות , יכול להיות שאחד מופיעה פו מילה הרבה פעמים אבל השני לא ,כל אחד מהם מדבר על משהו אחר, אז זה הגיוני לקבל הסתבריות שונות ואז החלפות שונות.

בשלב 2 גם במציאת הקולקציות הכי נפוצות לא קיבלנו אותן תשובות אפילו לא קרובות אחד לשנייה זה נובע משיש data שונה לכל סוג .

2)

אנחנו יכולים להבין קצת על מדובר בצואה כללית ,אבל על הנושאים עצמם לא,

עבור plenary יש קולקציות מאורך 2 המופיעה הכי הרבה היא

```
[('חבר', 'הכנסת')]
```

אז מזה אתה יכול לדעת רק שזה data של כנסת

אבל על המקצוע עצמו זה קצת קשה.

כי יש הרבה קולקציות הן רק מילה וסמין פסוק שזה דבר הגיוני כי יש מלא פסוקים אבל פסוקים כאלה לא יוזם בהבנת המקציועים.

וכמובן כל פעם שאנחנו לוקחים קולקציה יותר ארוכה היא תתן לנו יותר מידע אבל עדיין לא יוזם להבנת המקצוע במקרה של frequency

עבור Committee זה יותר גרוע אפילו עבור קולקציות מאורך 2

הכי נפוצות הן

```
[(('את', 'זה'), 2073), ((',', 'אני'), 1626), ((',', 'אבל'), 1425), (('אני', 'לא'), 1213), ((',')
```

```
[((',', 'זה'), 1143), (('זה', 'לא'), 1102), ((',', 'לא'), 915), (('אני', 'רוצה'), 867), ((',', 'אנחנו'), 738), (('זה', '.'), 702)]
```

כל הקולקציות מורכבות ממילים נפוצות שלא נותנות שום משמעות

או פסוקים שהם גם אינם נותנים מידע .

אם הולכים לקולקציות יותר ארוכות (מגודל 4)יש את זאת

```
Frequency:
Committee corpus:
[(33 ,('?' ,'ההסתייגות' ,'בעד' ,('מי'))
```

שהופיע 33 פעמים, ואנחנו יכולים להבין למשל שהם היה מצבעים על הסתייגות.

אבל עדיין זה לא נותן לנו הרבה מידע על מה מדובר ויש מספר מועט של קולקציות כאלה .

לגבי האם זה היה תאום לצפיות שלי

עבור plenary ו committee

כן זה היה תאום

עבור Plenary

הרבה קולקציות שמכילים מילים כמו

1)כנסת,יושבי ראש

שהן תלויות לכנסת

2)כינויים(:אני ,אנחנו),סימני פיסוק

שזה כן הייתה צפיה שלנו כי בסופו של דבר אלה מילים נפוצות במסמכי plenary

עבור committee

1)קריאה,יושב ראש

שהן תלויות לכנסת

2) כינויים(:אני ,אנחנו),סימני פיסוק

שזה כן הייתה צפיה שלנו כי בסופו של דבר אלה מילים נפוצות

3)עבור tf-idf

עבור Committee

כן יכולים להבין קצת על מה מדובר

יש לנו קולקציה למשל :

```
[('הבחירה הדמוקרטית בישראל .', 9.945301055349539),
```

שהיא כן נותנת מידע אנחנו יכולים להבין שהם היו מדברים על דמוקרטיה

עבור Plenary

כן יכולים להבין קצת על מה מדובר

יש לנו

```
('קוראים לזה ציונות .', 10.434600979998839)
```

יכולים לדעת שהם דברו על ציונות

יש גם למשל את אלה

```
Plenary corpus:
[('בברית-המועצות - 80% .', 10.434600979998839), ('בצרפת - 35% ;', 10.434600979998839), ('באיטליה - 60% ;', 10.434600979998839),
```

מפו נדע שהם דברו על מדינות אחרות וכל מדינה יש לה אחוז במשהו מסוים

לגבי צפיות שלי

לא היה לי צפיה במילים האלה כי קשה לצפות אותם הן לא מופיעות הרבה ולא נפוצות מאוד במסמכי committee

הסיבה היא ש

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term} + 1})$$

מהגדרת ה idf הייתי מצפה שיהיה לי מילים לא נפוצים מאוד כי

כל sentence שמכיל את הקולוקציה הוא יקטין את ה idf

אז הייתה מצפה לקבל מילים לא נמצאים בהרבה משפטים.

(4

כן יש הרבה הבדלים בולטים גם עבור Committee וגם עבור Plenary

עבור Frequency קיבלנו הרבה דברים חסרים משמעות כמו סימי פסוק ,כינווים,ביטויים בלי משמעות כמו ("את זה")

לעומת זאת עבור tf-idf קיבלנו דברים יותר עם משמעות וגם פחות סימני פסוק וביטויים חסרים משמעות

הסיבה לשונה זה היא פשוט :

עבור " Frequency" אנחנו לא דואגים לשום דבר חוץ ממספר הופעות

אבל מספר הופעות של קולקציה לא אומר שהיא בעלת משמעות

למשל המילה "אני" ,הרבה פועלים יכולים להופיע אחרי "אני"(אני רוצה , אני לא ,אני מבין...)

אז לכן יש סיכוי גדול לקבל קולקציה מורכבת מ "אני " ואז עוד פועל

אבל לא מבינים כלום מזה

וגם יש הרבה סימני פסיוק ב courps שלנו אז היה סיכוי גדול לקבל

מילה עם סמין פסוק כקולקציה שגם היא חסרת משמעות

עובר " tf-idf"

הגדרת ה idf מקטינה את ה tf-idf למילים שהם מופעים בהרבה משפטים

וגם לפי הגדרת ה tf אנחנו יודעים שכל שהקולוקציה שלנו מופיעה יותר במשפט מסוים לעומת קולקציות אחרות אז ה tf שלה יגדל

אז אם יש לנו משפט מאורך 4 שמכיל 2 מופעים של קולקציה מסוימת

אז ה tf שלה יהיה גדול יחסית ואז ה tf-idf גם יגדל

לכן אנחנו ברוב מדברים על קולקציות שלא מופיעות בהרבה משפטים אלה רק בכמה מהם לכן קשה לנבא אותם ויש סיכוי טוב שהם יהיו בעל משמעות כי הן מופיעות במשפטים מסויימים הרבה פעמים ,אבל לא בכל משפט(הם למשל מופעים רק כאשר חברי הכנסת דברו על זה לכן הן בסיכוי גדול בעלות משמעות)

(5

כן קיבלנו הרבה משפטים הגיונים

למשל המשפט הזה

```
Original sentence: אנחנו צריכים [*] טובי המשפטנים .
Committee sentence: אנחנו צריכים לעשות טובי המשפטנים .
Committee tokens: לעשות
Probability of committee sentence in committee corpus: -36.12609147798951
Probability of committee sentence in plenary corpus: -37.9191957521126
This sentence is more likely to appear in corpus: committee
Plenary sentence: אנחנו צריכים להיות טובי המשפטנים .
Plenary tokens: להיות
Probability of plenary sentence in plenary corpus: -37.62249157048029
Probability of plenary sentence in committee corpus: -35.82491746247989
This sentence is more likely to appear in corpus: committee
```

```
Original sentence: אבל הנושא הוא [*] אם אתה בעד [*] ההתנתקות או נגד .
Committee sentence: אבל הנושא הוא לא אם אתה בעד . ההתנתקות או נגד .
Committee tokens: לא,.
Probability of committee sentence in committee corpus: -93.5909457858865
Probability of committee sentence in plenary corpus: -91.79176008043375
This sentence is more likely to appear in corpus: plenary
Plenary sentence: אבל הנושא הוא נושא אם אתה בעד או ההתנתקות או נגד .
Plenary tokens: נושא,או
Probability of plenary sentence in plenary corpus: -96.34187521145287
Probability of plenary sentence in committee corpus: -98.44607876033703
This sentence is more likely to appear in corpus: plenary
```

אבל קיבלנו גם משפטיים שלא הוגיינים כמו

```
Original sentence: שעה הוראת לא [*] [*] ההצמדה .
Committee sentence: שעה הוראת לא בוקר . ההצמדה .
Committee tokens: .,בוקר
Probability of committee sentence in committee corpus: -50.82319839175626
Probability of committee sentence in plenary corpus: -53.12171957654795
This sentence is more likely to appear in corpus: committee
Plenary sentence: שעה הוראת לא חברי לא ההצמדה .
Plenary tokens: לא,חברי
Probability of plenary sentence in plenary corpus: -52.550841201249796
Probability of plenary sentence in committee corpus: -50.77893100046406
This sentence is more likely to appear in corpus: committee
```

הסיבה לקבלת משפטיים אינם הגיונים יכולה להיות מהרבה דברים

יכול להיות שמה אנחנו מנבאים הוא כן טוב עם מילים לפניו אבל לא מותאם עם מה שאחריוו,או שאין מספיק data ואז יהיהunderfitting למשל יש לנו שני מילים אחורה שאף אחת מהם לא מופיעה ב corpus אף פעם אז זה יהיה רנדומלי כי אין על פי מה להחליט

6)בטח שזה תלוי ב corpus יכול להיות של corpus זה כן טוב אבל לאחר לא

אבל בדרך כלל אני חושב שכן יהיה יותר טוב כי אנחנו עכשיוו לוקחים 3 מילות לפני ואז אם לוקחים 4 זה יתן לנו יותר משמעות למילה שאנחנו מנבאים

כי המשפט יהיה יותר קוהרנטי, ואז בהסתברות גדולה נקבל באמת מילה שהיא בעלת משמעות ביחד למילים שלפני