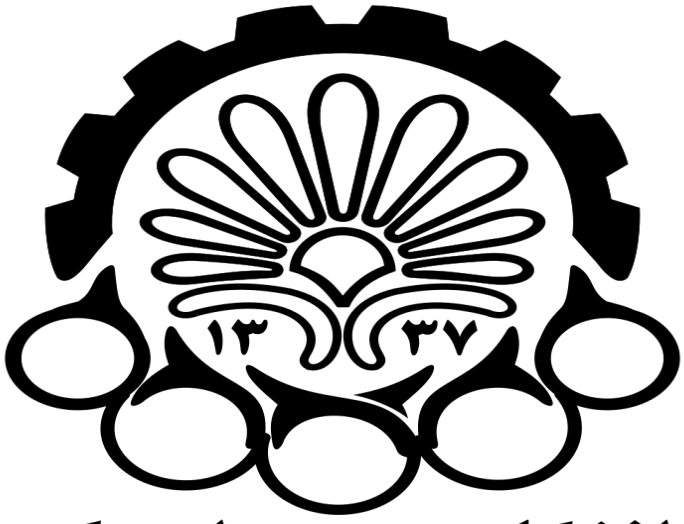


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس پردازش زبان طبیعی

استاد ممتازی

نیما پری فرد

۴۰۲۱۳۱۰۱۷

فهرست تمرین اول پردازش زبان طبیعی

بخش ۱

مشاهده داده و اعمال پیش پردازش های لازم

	content	label
0	به گزارش خبرنگار حوزه بهداشت و درمان گروه علمی پزشکی باشگاه خبرنگاران جوان؛ محمد آقاجانی روز دوشنبه در نشست هفتگی و هم اندیشی اعضای کمیسیون برنامه، بودجه و محاسبات مجلس و شورای معاونین وزارت بهداش...	7
1	به گزارش خبرنگار فوتبال و فوتسال گروه ورزشی باشگاه خبرنگاران جوان، آبی پوشان با حضور در زمین صنایع دفاع به دوبدن به دور زمین پرداختند. همین حبیبی با وجود آمید دیدگی چشم در تمرینات تیمش حاضر شد. با...	6
2	بهروز اکولی، در گفتگو با خبرنگار اجتماعی باشگاه خبرنگاران گفت: مجموعه وزارت تعاون، کار و رفاه اجتماعی از راه‌های مختلف از جمله ارائه تسهیلات واحدهای اقتصادی را حمایت می‌کند و در این راستا اداره کل...	0
3	به گزارش خبرنگار حوزه شهری گروه اجتماعی باشگاه خبرنگاران جوان، سعید دهقان مدیر اشتغال و کارآفرینی اداره کل تعاون و رفاه استان تهران در پ-ن-دهمین جلسه کارگروه تخصصی اشتغال تهران با اشاره به اینکه ا...	0
4	به گزارش باشگاه خبرنگاران و به نقل از روابط عمومی تماشاخانه ایرانشهر، این ۴ اثر نمایشی که از نیمه دوم فروردین ماه سال جاری، اجراهای خود را آغاز کرده‌اند، طی روزهای گذشته میزبان تعدادی از هنرمندان ...	5
5	به گزارش گروه اجتماعی باشگاه خبرنگاران، فتح اللهی با بیان این که ریزه‌انگیز، نقود نابینایبری، کمبود و کمری خدمات از جمله مهمترین مسائل عملکردی و کالبدی محلات دارای بافت فرسوده هستند، گفت: عده راه حله...	0
6	یک زن میلادی به تازگی موفق به دریافت یک پروتز دست با عقلات و اعصاب شده است. این زن اولین فرد در جهان است که پروتز دستی با این شکل دریافت می‌کند. عملکرد این دست به نحوی است که به صاحب خود احساس لمس...	4
7	پیام فرحبخش متخصص تغذیه در گفتگو با خبرنگار بهداشت و درمان باشگاه خبرنگاران، افزود: اگر افراد قصد استفاده از کوئنه قرمز را دارند بهتر است چربی‌های آن را جدا کنند و بهترین روش طبخ برای کوئنه‌ها بخارپز...	7
8	به گزارش حوزه دفاعی- امنیتی باشگاه خبرنگاران؛ سردار مرتضی بامدار عضو معاون جزایری با اشاره به یازده‌ای امداد مربوط به اتاقهای فکر آفریکا و رژیم صهیونیستی که دالند بر به نتیجه رسیدن فشار بر جمهوری ...	3
9	یوستون دابنامیکو تا به حال توانسته رسانه‌هایق را به کارهای عجیب و غریب زیادی وادار کند. با این وجود، مهندسین این کمپ‌ن- معروف هی کردند که با ویدیوی کوتاهی، قابلیت‌های حرکتی منحصر به فرد رسان امیدان...	4

داده قرار داده شده از دو ستون تشکیل شده است که یکی شامل متن مورد نظر و دیگری دارای برچسب مورد نظر است.

ساختار سه داده به این صورت است:

```
Shape of the train data: (13314, 2)
Shape of the validation data: (1480, 2)
Shape of the test data: (1644, 2)
```

به علت فارسی بودن متن از کتاب خانه Hazm برای پیش پردازش ممتون استفاده کردم.

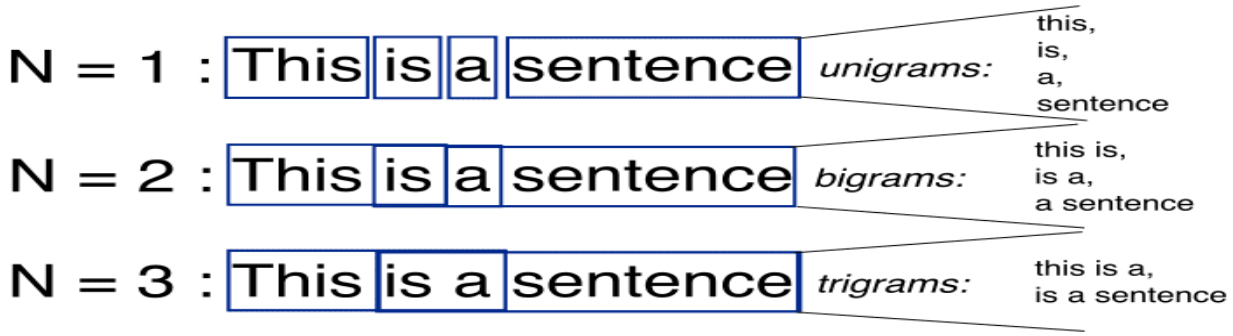
```python
from hazm import Normalizer, word_tokenize
# Create a normalizer object
normalizer = Normalizer()   normalizer: <hazm.normalizer

def remove_u200c(text):
    return text.replace('\u200c', '')

def preprocess_text(text):
    # Normalize the text
    text = normalizer.normalize(text)
    text = normalizer.remove_specials_chars(text)

    # Tokenize the text
    words = word_tokenize(text)
    words = [remove_u200c(word) for word in words]
    return words
```

Executed at 2024.04.11 12:16:15 in 2s 417ms

بخش ۲

تعاریف

مدل‌های زبانی n-gram روشی برای پیش‌بینی احتمال ظاهر شدن یک کلمه در متن بر اساس کلماتی که قبل از آن آمده‌اند هستند. تصور کنید که شما سعی دارید کلمه بعدی در یک جمله را حدس بزنید. یک مدل n-gram یک دنباله از n کلمه (که n هر عددی می‌تواند باشد) را در نظر می‌گیرد و تجزیه و تحلیل می‌کند که چه اندازه این دنباله در یک نمونه متن بزرگ ظاهر می‌شود. این به آن کمک می‌کند تا احتمالات مختلف کلمات بعدی را تخصیص دهد.



مدل‌های زبانی n-gram، با وجود کاربردی که دارند، ممکن است با مشکلاتی مواجه شوند هنگامی که با کلمات یا عباراتی غیر دیده شده سر و کار دارند. این به دلیل این است که آن‌ها بر اساس تعداد ظاهر شدن دنباله‌ها در داده‌های آموزشی حساب می‌کنند. اگر یک دنباله خاص قبلاً مشاهده نشده باشد، مدل احتمال آن را صفر می‌دهد، که هرگونه پیش‌بینی مربوط به آن دنباله را غیرممکن می‌کند.

تکنیک‌های هموارسازی این مشکل را با تنظیم احتمالات اختصاص داده شده توسط مدل‌های n-gram حل می‌کنند.

Back-off

$$P(w_i|w_{i-1}) = \begin{cases} \frac{\#(w_{i-1},w_i)}{\#(w_{i-1})} & \text{if } \#(w_{i-1},w_i)>0 \\ P_{BG} & \text{Otherwise} \end{cases}$$

$$P(w_i) = \begin{cases} \frac{\#(w_i)}{N} & \text{if } \#(w_i)>0 \\ 1/V & \text{Otherwise} \end{cases}$$

Absolute Discounting

$$P(w_i|w_{i-1}) = \frac{\#(w_{i-1},w_i) - \delta}{\#(w_{i-1})} + \alpha P_{BG}$$

$$\alpha = \frac{\delta}{\#(w_{i-1})} \cdot B$$

B : the number of times $\#(w_i, w_{i-1}) > 0$
(the number of times that we applied discounting)

$$P(w_i|w_{i-1}) = \frac{\max(\#(w_{i-1},w_i) - \delta, 0)}{\#(w_{i-1})} + \alpha P_{BG}$$

پیاده سازی مدل زبانی n-grams

در کلاس زیر مدل پایه n-grams را پیاده سازی کردم.

```
class NgramLanguageModel:
    def __init__(self, n):
        self.n = n
        self.counts = {}
        self.counts_minus_one_grams = {}
        self.vocab = set()
```

```
def update_counts(self, tokens):
    n = self.n
    for i in range(len(tokens) - n + 1):
        ngram = tuple(tokens[i:i + n])
        self.counts[ngram] = self.counts.get(ngram, 0) + 1
        for token in ngram:
            self.vocab.add(token)

    n = self.n - 1
    for i in range(len(tokens) - n + 1):
        ngram = tuple(tokens[i:i + n])
        self.counts_minus_one_grams[ngram] = self.counts_minus_one_grams.get(ngram, 0) + 1
```

در متد بالا در کلاس n-grams دیکشنری تشکیل دادم و با توجه به شماره n رشته ها با طول n و n-1 به دست آوردم تا برای محاسبه احتمال از آن استفاده کنم.

این قسمت با داده های آموزش انجام شد.

```
def probability(self, token, context):
    context = tuple(context)
    ngram = context + (token,)
    if context in self.counts_minus_one_grams:
        context_count = self.counts_minus_one_grams[context]
        if ngram in self.counts:
            return self.counts[ngram] / context_count
    return 0
```

در این قسمت احتمال را جهت استفاده در مدل زبانی حساب کردم البته در کد های هموارسازی قسمت بعد هر دو بازنویسی شده اند.

```
def perplexity(self, test_data):
    ppl = 1
    total_ngrams = 0
    perplexity = []
    for i in range(len(test_data) - self.n + 1):
        context = tuple(test_data[i:i + self.n - 1])
        token = test_data[i + self.n - 1]
        prob = self.probability(token, context)
        ppl *= (1/prob)
        total_ngrams += 1
        if i % 10 == 0:
            perplexity.append(ppl ** (1/len(test_data)))
            ppl = 1
        # if len(test_data) < 100 or len(test_data) - i < 100:
        #     perplexity.append(ppl ** (1/len(test_data)))
    perplex = 1
    for ppl in perplexity:
        perplex *= ppl
    return perplex
```

از این کد برای محاسبه perplexity استفاده کردم.

شاید کمی عجیب برسد اما به علت بزرگ شدن اعداد با ضرب ان ها برای هر ده رشته حساب کردم و بردم زیر رادیکال تا اعداد زیاد بزرگ نشوند و در نهایت اخر سر در هم ضرب کردم.

قبل از بررسی نتایج باید عرض کنم مدل های زبانی با داده آموزش آموزش دیده شده اند و کد آن در قسمت زیر است.

```
backoff_model = BackoffSmoothing(1, 0.1)  # C
for tokens in train['content']:  train: DataF
    backoff_model.update_counts(tokens)  back
```

پیاده سازی هموار سازی backoff و absolute

```python
class BackoffSmoothing(NgramLanguageModel):
    def __init__(self, n, p_bg=0.001):
        super().__init__(n)
        self.p_bg = p_bg


    def probability(self, token, context):
        p = super().probability(token, context)
        if p == 0:
            p = self.p_bg
        return p
```

در این جا هموارسازی back off پیاده سازی کردم و از کلاس ngram قبلی ارث بری کردم.

```python
class AbsoluteDiscounting(NgramLanguageModel):
    def __init__(self, n, discount=0.5, p_bg=0.1):
        super().__init__(n)
        self.discount = discount
        self.p_bg = p_bg
        self.applied_discounting = 1

    def probability(self, token, context):
        context = tuple(context)
        ngram = context + (token,)
        if context in self.counts_minus_one_grams:
            context_count = self.counts_minus_one_grams[context]
            ngram_count = self.counts.get(ngram, 0)
            if context_count > 0:
                self.applied_discounting += 1
            alpha = self.discount * self.applied_discounting / context_count
            return max(ngram_count - self.discount, 0) / context_count + alpha * self.p_bg
        return  self.p_bg
```

تحلیل و گزارش نتایج

بخش الف)

معیار ارزیابی برای تمام متون داده آزمون با مدل های unigram, bigram, trigram با هر دو هموارسازی پیاده سازی شده تست شده.

همین طور از داده ولیدیشن برای انتخاب بهترین ابرپارامتر استفاده کردم

یونیگرم با back off smoothing

```
# unigram
# Train the model with BackoffSmoothing
backoff_model = BackoffSmoothing(1, 0.1)  # Change the number to the desired n-gram   backoff_model: <__main
for tokens in train['content']:   train: DataFrame (13314, 2)
    backoff_model.update_counts(tokens)   backoff_model: <__main__.BackoffSmoothing object at 0x000002DD26B2
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]   background_prob: list (9
perplexity_dict = {}   perplexity_dict: {0.001: 9.606647917876415, 0.002: 2.2438517864097247, 0.003: 1.07470
for p_bg in background_prob:   background_prob: list (9)
    backoff_model.p_bg = p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing object at 0x000002D
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)   perplexity_dict: {0.001: 9.6066479

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)   best_p_bg: 0.009    perplexity_dict: {0.001: 9.6
backoff_model.p_bg = best_p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing object at 0x000002
# Evaluate the model
for i, content in enumerate(test['content']):   test: DataFrame (1644, 2)
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of tokens   test_t
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.perplexity(test_t
Executed at 2024.04.11 17:45:53 in 45s 854ms
```

```
Perplexity of the 1633 BackoffSmoothing model on the test data: 24629.05138374151
Perplexity of the 1634 BackoffSmoothing model on the test data: 22548.478192349354
Perplexity of the 1635 BackoffSmoothing model on the test data: 23174.37276126121
Perplexity of the 1636 BackoffSmoothing model on the test data: 18993.586401307686
Perplexity of the 1637 BackoffSmoothing model on the test data: 17741.437051344612
Perplexity of the 1638 BackoffSmoothing model on the test data: 21671.26863985651
Perplexity of the 1639 BackoffSmoothing model on the test data: 20591.76475430103
Perplexity of the 1640 BackoffSmoothing model on the test data: 23206.29540020043
Perplexity of the 1641 BackoffSmoothing model on the test data: 18187.977957622003
Perplexity of the 1642 BackoffSmoothing model on the test data: 25782.779663556463
Perplexity of the 1643 BackoffSmoothing model on the test data: 26346.327140591056
```

بایگرم با back off smoothing

```
# bigrams
# Train the model with BackoffSmoothing
backoff_model = BackoffSmoothing(2, 0.1)  # Change the number to the desired n-gram   backoff
for tokens in train['content']:   train: DataFrame (13314, 2)
    backoff_model.update_counts(tokens)   backoff_model: <__main__.BackoffSmoothing object at
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]   backgrou
perplexity_dict = {}   perplexity_dict: {0.001: 9.606647917876415, 0.002: 2.2438517864097247,
for p_bg in background_prob:   background_prob: list (9)
    backoff_model.p_bg = p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing obje
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)   perplexity_dict: {0

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)   best_p_bg: 0.009    perplexity_di
backoff_model.p_bg = best_p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing obj
# Evaluate the model
for i, content in enumerate(test['content']):   test: DataFrame (1644, 2)
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.pe
Executed at 2024.04.11 17:46:43 in 49s 726ms
```

```
Perplexity of the 1633 BackoffSmoothing model on the test data: 129.45651927478866
Perplexity of the 1634 BackoffSmoothing model on the test data: 130.4922104801294
Perplexity of the 1635 BackoffSmoothing model on the test data: 128.72313799051804
Perplexity of the 1636 BackoffSmoothing model on the test data: 136.54554165379952
Perplexity of the 1637 BackoffSmoothing model on the test data: 127.7399717874297
Perplexity of the 1638 BackoffSmoothing model on the test data: 128.09209609808877
Perplexity of the 1639 BackoffSmoothing model on the test data: 135.8809490234155
Perplexity of the 1640 BackoffSmoothing model on the test data: 126.62236112472443
Perplexity of the 1641 BackoffSmoothing model on the test data: 126.71875247454422
Perplexity of the 1642 BackoffSmoothing model on the test data: 129.31577245348296
Perplexity of the 1643 BackoffSmoothing model on the test data: 129.58215715960307
```

ترایگرم با back off smoothing

```
# trigrams
# Train the model with BackoffSmoothing
backoff_model = BackoffSmoothing(3, 0.1)  # Change the number to the desired n-gram   backoff_mod
for tokens in train['content']:   train: DataFrame (13314, 2)
    backoff_model.update_counts(tokens)   backoff_model: <__main__.BackoffSmoothing object at 0x0
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]   background_pr
perplexity_dict = {}   perplexity_dict: {0.001: 9.606647917876415, 0.002: 2.2438517864097247, 0.0
for p_bg in background_prob:   background_prob: list (9)
    backoff_model.p_bg = p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing object a
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)   perplexity_dict: {0.001

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)   best_p_bg: 0.009    perplexity_dict:
backoff_model.p_bg = best_p_bg   p_bg: 0.009    backoff_model: <__main__.BackoffSmoothing object
# Evaluate the model
for i, content in enumerate(test['content']):   test: DataFrame (1644, 2)
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of toke
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.perplex
Executed at 2024.04.11 17:47:37 in 54s 53ms
```

```
Perplexity of the 1633 BackoffSmoothing model on the test data: 107.08294271850711
Perplexity of the 1634 BackoffSmoothing model on the test data: 104.70031207287622
Perplexity of the 1635 BackoffSmoothing model on the test data: 106.07148128642085
Perplexity of the 1636 BackoffSmoothing model on the test data: 107.2494320124516
Perplexity of the 1637 BackoffSmoothing model on the test data: 103.47639603432914
Perplexity of the 1638 BackoffSmoothing model on the test data: 106.90216040772451
Perplexity of the 1639 BackoffSmoothing model on the test data: 109.18605936108322
Perplexity of the 1640 BackoffSmoothing model on the test data: 106.8099156550074
Perplexity of the 1641 BackoffSmoothing model on the test data: 105.40984502069853
Perplexity of the 1642 BackoffSmoothing model on the test data: 106.20187148982913
Perplexity of the 1643 BackoffSmoothing model on the test data: 105.85542357799929
```

یونیگرم با absolute discounting smoothing

```python
# unigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(1)  # Change the number to the desired n-gram
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.A
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]  background
perplexity_dict = {}  perplexity_dict: {0.001: 9.606647917876415, 0.002: 2.2438517864097247,
for p_bg in background_prob:  background_prob: list (9)
    absolute_discounting_model.p_bg = p_bg  p_bg: 0.009  absolute_discounting_model: <__mai
    perplexity_dict[p_bg] = absolute_discounting_model.perplexity(validation_tokens)  perplex

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.009  perplexity_dic
absolute_discounting_model.p_bg = best_p_bg  p_bg: 0.009  absolute_discounting_model: <__ma
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of t
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_disco
    absolute_discounting_model.applied_discounting = 1
```

Executed at 2024.04.11 17:48:18 in 40s 578ms

```
Perplexity of the 1631 Absolute Discounting model on the test data: 24720.552158799746
Perplexity of the 1632 Absolute Discounting model on the test data: 20128.788765806952
Perplexity of the 1633 Absolute Discounting model on the test data: 25124.219367465786
Perplexity of the 1634 Absolute Discounting model on the test data: 23133.475033659266
Perplexity of the 1635 Absolute Discounting model on the test data: 24043.816833485096
Perplexity of the 1636 Absolute Discounting model on the test data: 19199.822286760158
Perplexity of the 1637 Absolute Discounting model on the test data: 18330.386496460647
Perplexity of the 1638 Absolute Discounting model on the test data: 23030.387215356634
Perplexity of the 1639 Absolute Discounting model on the test data: 21624.136677333347
Perplexity of the 1640 Absolute Discounting model on the test data: 23325.374817298165
Perplexity of the 1641 Absolute Discounting model on the test data: 19624.74792198748
Perplexity of the 1642 Absolute Discounting model on the test data: 25370.31294115192
```

بایگرم با absolute discounting smoothing

```python
# bigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(2)  # Change the number to the desired n-gram
for tokens in train['content']:
    absolute_discounting_model.update_counts(tokens)
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]
perplexity_dict = {}
for p_bg in background_prob:
    absolute_discounting_model.p_bg = p_bg
    perplexity_dict[p_bg] = absolute_discounting_model.perplexity(validation_tokens)

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)
absolute_discounting_model.p_bg = best_p_bg
# Evaluate the model
for i, content in enumerate(test['content']):
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_discounting_model.perplexity(test_tokens)}")
    absolute_discounting_model.applied_discounting = 1
```

Executed at 2024.04.11 19:23:00 in 1m 10s 96ms

```
Perplexity of the 1630 Absolute Discounting model on the test data: 200.63553807331652
Perplexity of the 1631 Absolute Discounting model on the test data: 66.87754204780532
Perplexity of the 1632 Absolute Discounting model on the test data: 97.02760025732012
Perplexity of the 1633 Absolute Discounting model on the test data: 114.8722929789822
Perplexity of the 1634 Absolute Discounting model on the test data: 74.37279153883293
Perplexity of the 1635 Absolute Discounting model on the test data: 70.65847278519504
Perplexity of the 1636 Absolute Discounting model on the test data: 92.58917704228367
Perplexity of the 1637 Absolute Discounting model on the test data: 117.72418173463623
Perplexity of the 1638 Absolute Discounting model on the test data: 100.31118334895213
Perplexity of the 1639 Absolute Discounting model on the test data: 100.55858700970184
Perplexity of the 1640 Absolute Discounting model on the test data: 121.24734022000386
Perplexity of the 1641 Absolute Discounting model on the test data: 149.9186950498523
Perplexity of the 1642 Absolute Discounting model on the test data: 128.72661612462113
Perplexity of the 1643 Absolute Discounting model on the test data: 70.08636418323348
```

ترایگرم با absolute discounting smoothing

```
# trigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(3)  # Change the number to the desired n-gram   absolu
for tokens in train['content']:   train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)   absolute_discounting_model: <__main__.AbsoluteDi
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009]   background_prob: li
perplexity_dict = {}   perplexity_dict: {0.001: 9.606647917876415, 0.002: 2.2438517864097247, 0.003: 1.
for p_bg in background_prob:   background_prob: list (9)
    absolute_discounting_model.p_bg = p_bg   p_bg: 0.009    absolute_discounting_model: <__main__.Absol
    perplexity_dict[p_bg] = absolute_discounting_model.perplexity(validation_tokens)   perplexity_dict:

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)   best_p_bg: 0.009    perplexity_dict: {0.001
absolute_discounting_model.p_bg = best_p_bg   p_bg: 0.009    absolute_discounting_model: <__main__.Absol
# Evaluate the model
for i, content in enumerate(test['content']):   test: DataFrame (1644, 2)
    test_tokens = [token for sublist in content for token in sublist]  # Flatten the list of tokens   t
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_discounting_m
    absolute_discounting_model.applied_discounting = 1
Executed at 2024.04.11 19:23:57 in 56s 764ms
```

```
Perplexity of the 1630 Absolute Discounting model on the test data: 63.98119045304431
Perplexity of the 1631 Absolute Discounting model on the test data: 47.66728027778998
Perplexity of the 1632 Absolute Discounting model on the test data: 49.10019229467069
Perplexity of the 1633 Absolute Discounting model on the test data: 52.70734095137774
Perplexity of the 1634 Absolute Discounting model on the test data: 40.86068145537037
Perplexity of the 1635 Absolute Discounting model on the test data: 50.07964198712437
Perplexity of the 1636 Absolute Discounting model on the test data: 45.63965920098171
Perplexity of the 1637 Absolute Discounting model on the test data: 51.60714951665999
Perplexity of the 1638 Absolute Discounting model on the test data: 51.34714683570967
Perplexity of the 1639 Absolute Discounting model on the test data: 51.94332303079716
Perplexity of the 1640 Absolute Discounting model on the test data: 53.5564036257472
Perplexity of the 1641 Absolute Discounting model on the test data: 57.000107303841936
Perplexity of the 1642 Absolute Discounting model on the test data: 59.653747667522275
Perplexity of the 1643 Absolute Discounting model on the test data: 46.11230049561908
```

بخش ب)

برای گزارش بر حسب برچسب کلاس اول داده های برچسب با هم اجماع کردم.

```
labels = list(range(0, 8))   labels: list (8)
test_class_tokens = {}   test_class_tokens: 'مهمت' , '؛' , 'جوان' , 'خبرنگاران'
for i in labels:   labels: list (8)
    test_label = test[test['label'] == i]   test_label: DataFrame (209
    test_tokens = []   test_tokens: list (1391)
    for j, content in enumerate(test_label['content']):   test_label:
        for token in content:   content: 'ش' , 'برگزار' , 'اصولگرایان' , 'ائتلاف'
            test_tokens.append(token)   test_tokens: list (1391)    to
    test_class_tokens[i] = test_tokens   test_class_tokens: 'مهمت' , '؛' ,
Executed at 2024.04.11 17:52:55 in 404ms
```

در نهایت با استفاده سه مدل زبانی perplexity را برای هر برچسب بر حسب داده آزمون گزارش کردم.

```python
backoff_model = BackoffSmoothing(1, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    backoff_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} BackoffSmoothing model on the test data with unigram: {backoff_model.perplexity(test_tokens)}")

backoff_model = BackoffSmoothing(2, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    backoff_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} BackoffSmoothing model on the test data with bigram: {backoff_model.perplexity(test_tokens)}")

backoff_model = BackoffSmoothing(3, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    backoff_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} BackoffSmoothing model on the test data with trigram: {backoff_model.perplexity(test_tokens)}")
```

Executed at 2024.04.11 17:55:24 in 2m 28s 104ms

```
Perplexity of the class 0 BackoffSmoothing model on the test data with unigram: 23469.19837795475
Perplexity of the class 1 BackoffSmoothing model on the test data with unigram: 22115.30239346858
Perplexity of the class 2 BackoffSmoothing model on the test data with unigram: 23974.493816102735
Perplexity of the class 3 BackoffSmoothing model on the test data with unigram: 23501.98584015653
Perplexity of the class 4 BackoffSmoothing model on the test data with unigram: 23996.02799200902
Perplexity of the class 5 BackoffSmoothing model on the test data with unigram: 22691.33894779525
Perplexity of the class 6 BackoffSmoothing model on the test data with unigram: 22766.249373112743
Perplexity of the class 7 BackoffSmoothing model on the test data with unigram: 23530.599177726588
Perplexity of the class 0 BackoffSmoothing model on the test data with bigram: 133.9155073071851
Perplexity of the class 1 BackoffSmoothing model on the test data with bigram: 135.635756051459
Perplexity of the class 2 BackoffSmoothing model on the test data with bigram: 131.5027269770164
Perplexity of the class 3 BackoffSmoothing model on the test data with bigram: 133.64350563474213
Perplexity of the class 4 BackoffSmoothing model on the test data with bigram: 131.78788010046748
Perplexity of the class 5 BackoffSmoothing model on the test data with bigram: 133.90253284857906
Perplexity of the class 6 BackoffSmoothing model on the test data with bigram: 131.75114008603828
Perplexity of the class 7 BackoffSmoothing model on the test data with bigram: 136.00569066082338
Perplexity of the class 0 BackoffSmoothing model on the test data with trigram: 109.16904450405046
Perplexity of the class 1 BackoffSmoothing model on the test data with trigram: 108.24620141592746
Perplexity of the class 2 BackoffSmoothing model on the test data with trigram: 108.90835100513111
Perplexity of the class 3 BackoffSmoothing model on the test data with trigram: 109.16041220887217
Perplexity of the class 4 BackoffSmoothing model on the test data with trigram: 109.10528917505209
Perplexity of the class 5 BackoffSmoothing model on the test data with trigram: 108.9201329987683
Perplexity of the class 6 BackoffSmoothing model on the test data with trigram: 108.62875559887819
Perplexity of the class 7 BackoffSmoothing model on the test data with trigram: 109.20940365490034
```

```python
absolute_discounting_model = AbsoluteDiscounting(1, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    absolute_discounting_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} absolute discounting model on the test data with unigram: {absolute_discounting_model.perplexity(test_tokens)}")

absolute_discounting_model = AbsoluteDiscounting(2, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    absolute_discounting_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} absolute discounting model on the test data with bigram: {absolute_discounting_model.perplexity(test_tokens)}")

absolute_discounting_model = AbsoluteDiscounting(3, 0.009)  # Change the number to the desired n-gram
for tokens in train['content']:
    absolute_discounting_model.update_counts(tokens)
for i in labels:
    test_tokens = [token for sublist in test_class_tokens[i] for token in sublist]  # Flatten the list of tokens
    print(f"Perplexity of the class {i} absolute discounting model on the test data with trigram: {absolute_discounting_model.perplexity(test_tokens)}")
```

```
Perplexity of the class 0 absolute discounting model on the test data with unigram: 11613.852134939816
Perplexity of the class 1 absolute discounting model on the test data with unigram: 5801.445271772141
Perplexity of the class 2 absolute discounting model on the test data with unigram: 4263.068187292759
Perplexity of the class 3 absolute discounting model on the test data with unigram: 3029.7950647961757
Perplexity of the class 4 absolute discounting model on the test data with unigram: 2322.6069065183287
Perplexity of the class 5 absolute discounting model on the test data with unigram: 1886.2402303981612
Perplexity of the class 6 absolute discounting model on the test data with unigram: 1705.340063604118
Perplexity of the class 7 absolute discounting model on the test data with unigram: 1573.8520720940642
Perplexity of the class 0 absolute discounting model on the test data with bigram: 2.10017268653048
Perplexity of the class 1 absolute discounting model on the test data with bigram: 0.6026463160233755
Perplexity of the class 2 absolute discounting model on the test data with bigram: 0.33630156154100127
Perplexity of the class 3 absolute discounting model on the test data with bigram: 0.2225603751998875
Perplexity of the class 4 absolute discounting model on the test data with bigram: 0.15629842741231834
Perplexity of the class 5 absolute discounting model on the test data with bigram: 0.13182924608816168
Perplexity of the class 6 absolute discounting model on the test data with bigram: 0.11582213574015533
Perplexity of the class 7 absolute discounting model on the test data with bigram: 0.10217033196376603
Perplexity of the class 0 absolute discounting model on the test data with trigram: 3.127281679154286
Perplexity of the class 1 absolute discounting model on the test data with trigram: 2.1279325201628962
Perplexity of the class 2 absolute discounting model on the test data with trigram: 2.057302501400128
Perplexity of the class 3 absolute discounting model on the test data with trigram: 1.7697578177884998
Perplexity of the class 4 absolute discounting model on the test data with trigram: 1.731710331014706
Perplexity of the class 5 absolute discounting model on the test data with trigram: 1.552586773716083
Perplexity of the class 6 absolute discounting model on the test data with trigram: 1.4586198613237171
Perplexity of the class 7 absolute discounting model on the test data with trigram: 1.404046591130652
```

بخش ج)

مواردی که متوجه شدم از نتایج این دو مورد است.

- با افزایش n در مدل زبانی perplexity کاهش پیدا می کند. دلیل ان هم وقتی تعداد (n) اضافه می‌کنیم، در واقع اطلاعات متون بیشتری به مدل زبان ارائه می‌دهیم. این متون اضافی به مدل کمک می‌کنند تا ساختار و جریان زبان را بهتر درک کند و در نتیجه، دقت پیش‌بینی کلمه بعدی در یک

دنباله را بالا ببرد. به عبارت دیگر، با افزودن بیشترین میزان متون، مدل بیشتر اطمینان پیدا می‌کند و با این اطلاعات بیشتر، perplexity کاهش می‌یابد.

- هموار سازی absolute discouting هموار سازی معقول تری نسبت به backoff است و معمولا نتایج بهتری دارد. شاید چون هشیار تر عمل هموار سازی را انجام می دهد.
- در داده تست داده برچسب های هر داده تقریبا با هر مدل perplexity مشابه دارند.
- مورد بعد این است که unigram اصلا نتایج جالبی به همراه ندارد.

بخش ۳

تعاریف

Vec۲Word یک تکنیک محبوب در پردازش زبان طبیعی (NLP) برای یادگیری تعبیه کلمات است که نمایش‌های برداری عمیق از کلمات را در یک فضای برداری پیوسته ارائه می‌دهد.

دو معماری اصلی برای آموزش مدل‌های Vec۲Word وجود دارد: Continuous Bag of Words (CBOW) و Skip-gram.

۱. CBOW: Continuous Bag of Words (CBOW) به توجه به متن پیشین کلمه هدف را پیش بینی می‌کند. این معماری یک متن از کلمات محیط را به عنوان ورودی می‌گیرد و سعی می‌کند کلمه هدف را پیش بینی کند. این معماری برای مجموعه داده‌های کوچکتر و کلمات متداول موثر است.

۲. Skip-gram: Skip-gram به عنوان یک معماری دیگر، کلمات محیطی را با توجه به یک کلمه هدف پیش‌بینی می‌کند. این معماری یک کلمه هدف را به عنوان ورودی می‌گیرد و سعی می‌کند کلمات محیط را پیش بینی کند. Skip-gram برای مجموعه داده‌های بزرگتر و گرفتن معنی کلمات کم‌تر موثر است.

هر دو CBOW و Skip-gram از یک شبکه عصبی با یک لایه پنهان برای یادگیری تعبیه کلمات استفاده می‌کنند. در طول آموزش، شبکه عصبی تعبیه کلمات را تنظیم می‌کند تا احتمال پیش‌بینی کلمات محیط (Skip-gram) یا کلمات هدف (CBOW) را به درستی بیشینه کند.

TF-IDF یک آمار عددی است که اهمیت یک کلمه در یک سند نسبت به یک مجموعه از اسناد یا متن مشخص می‌کند. این مفهوم به طور معمول در بازیابی اطلاعات و استخراج متن به عنوان یک روش برای تعیین ارتباط یک واژه با یک سند در یک مجموعه استفاده می‌شود.

در زیر تجزیه و تحلیل نحوه محاسبه TF-IDF آمده است:

1. TF این مقدار نشان می‌دهد که چقدر یک واژه در یک سند تکرار شده است. این به عنوان نسبت تعداد بارهایی که یک واژه در یک سند ظاهر می‌شود به تعداد کل واژگان در سند محاسبه می‌شود. در واقع، این مقدار اهمیت یک واژه درون سند را نشان می‌دهد.

tf: term frequency. frequency count (usually log-transformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. IDF: این مقدار نشان می‌دهد که یک واژه در سراسر کل مجموعه متن چقدر مهم است. این به عنوان لگاریتم نسبت تعداد کل اسناد در مجموعه به تعداد اسنادی که حاوی واژه هستند محاسبه می‌شود. واژگانی که در بسیاری از اسناد ظاهر می‌شوند ارزش IDF کمتری دارند، در حالی که واژگانی که در تعداد کمی از اسناد ظاهر می‌شوند، ارزش IDF بیشتری دارند.

- Idf: inverse document frequency:

$$\mathrm{idf}_i = \log\left(\frac{N}{\mathrm{df}_i}\right)$$

Total # of docs in collection

# of docs that have word i

3. TF-IDF: در نهایت، امتیاز TF-IDF برای یک واژه در یک سند با ضرب کردن مقادیر TF و IDF محاسبه می‌شود. این موجب می‌شود که واژگانی که درون سند متکرر ولی در کل مجموعه کمیاب هستند، وزن بیشتری داشته باشند و اهمیت آن‌ها در سند بیشتر در نظر گرفته شود

tf-idf value for word t in document d: $w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$

با افزایش امتیاز TF-IDF یک واژه در یک سند، اهمیت بیشتری به آن واژه درون سند نسبت داده می‌شود.

پیاده سازی میانگین حسابی word2vec

طبق توصیه صورت سوال از کتابخانه genism برای پیاده سازی word2vec استفاده کردم.

پیش پردازش های که برای این سوال انجام دادم.

```
# Create a stemmer object
stemmer = Stemmer()  stemmer: <hazm.stemmer.Stemmer object at 0x000001B8CC228AD0>    Stemmer: <class

# Create a lemmatizer object
lemmatizer = Lemmatizer()  lemmatizer: <hazm.lemmatizer.Lemmatizer object at 0x000001B8CC237850>

# Get the list of Persian stopwords
stopwords = stopwords_list()  stopwords: list (389)

def preprocess_text(words):
    # Remove stopwords, apply stemming and lemmatization
    words = [lemmatizer.lemmatize(stemmer.stem(word)) for word in words if word not in stopwords]
    return words

train['content'] = train['content'].apply(preprocess_text)  train: DataFrame (13314, 2)    train: Dat
test['content'] = test['content'].apply(preprocess_text)  test: DataFrame (1644, 2)    test: DataFram
Executed at 2024.04.11 21:25:56 in 2m 4s 435ms
```

```
# Train Word2Vec model
model = Word2Vec(train['content'], min_count=1, sg=1, vector_size=200)
Executed at 2024.04.11 18:58:25 in 1m 38s 730ms
```

میانگین حسابی بردار های آموزش و تست استفاده کردم و از knn برای دسته استفاده کردم.

```
test_avg_vectors = test['content'].apply(lambda words: np.mean([check_existence(word, model, 200) for word in words], axis=0))
val_avg_vectors = val['content'].apply(lambda words: np.mean([check_existence(word, model, 200) for word in words], axis=0))
Executed at 2024.04.11 14:19:26 in 2s 765ms
```

از روش gridsearch برای جستجو پارامتز بهینه استفاده کردم.

```python
# Define the parameter grid for the KNN classifier
param_grid = {'n_neighbors': [3, 5, 7, 9, 11, 20, 30 , 45]}  param_grid: {'n_neighbors':

# Create a GridSearchCV object
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5, n_jobs=-1)  grid_se

# Fit the GridSearchCV object to the training data
grid_search.fit(list(train_avg_vectors), train['label'])  grid_search: GridSearchCV(cv=5

# Print the best parameters found by GridSearchCV
print("Best parameters: ", grid_search.best_params_)  grid_search: GridSearchCV(cv=5, es

# Use the best estimator to make predictions on the test data
best_knn = grid_search.best_estimator_  best_knn: KNeighborsClassifier(n_neighbors=20)
predictions = best_knn.predict(list(test_avg_vectors))  predictions: ndarray (1644,)

# Print the classification report for the test data predictions
print(classification_report(test['label'], predictions))  test: DataFrame (1644, 2)    p
```

نتایج زیر به دست آمد.

```
Best parameters:  {'n_neighbors': 11}
              precision    recall  f1-score   support

           0       0.82      0.72      0.77       217
           1       0.84      0.76      0.80       156
           2       0.91      0.87      0.89       197
           3       0.74      0.84      0.79       227
           4       0.88      0.91      0.89       244
           5       0.90      0.91      0.91       256
           6       0.99      0.94      0.97       138
           7       0.81      0.86      0.83       209

    accuracy                           0.85      1644
   macro avg       0.86      0.85      0.86      1644
weighted avg       0.86      0.85      0.85      1644
```

پیاده سازی وزنی word2vec با وزن های TF-IDF

پیاده سازی tf idf با توجه به توضیح در قسمت توضیحات به این شرح است.

```
# Calculate term frequency
def term_frequency(doc):
    counts = Counter(doc)
    return {word: count/len(doc) for word, count in counts.items()}
```

```
def inverse_document_frequency(docs):
    idf = {}
    all_words = set(word for doc in docs for word in doc)
    for word in all_words:
        contains_word = map(lambda doc: word in doc, docs)
        idf[word] = log(len(docs)/(1 + sum(contains_word)))
    return idf
```

```
def tf_idf(docs):
    word2weight = {}
    idf = inverse_document_frequency(docs)
    for doc in docs:
        tf = term_frequency(doc)
        for word, freq in tf.items():
            word2weight[word] = freq * idf[word]
    return word2weight
```

البته به علت کند بودن فرایند بسیار زیاد طول کشید برای همین از sklearn برای انجام ان استفاده کردم.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer(max_features=10000, stop_words='english')  vectorizer:

train_vectors = vectorizer.fit_transform(train_class_content['Value'])  train_vecto

# Transform the content of the test data
test_vectors = vectorizer.transform(test_class_content['Value'])  test_vectors: <8x
```
Executed at 2024.04.11 22:13:50 in 4s 913ms

```
Best parameters:  {'n_neighbors': 11}
              precision    recall  f1-score   support

           0       0.67      0.75      0.71       217
           1       0.83      0.67      0.74       156
           2       0.89      0.81      0.85       197
           3       0.68      0.85      0.76       227
           4       0.85      0.89      0.87       244
           5       0.84      0.81      0.82       256
           6       0.97      0.85      0.91       138
           7       0.88      0.79      0.84       209

    accuracy                           0.81      1644
   macro avg       0.83      0.80      0.81      1644
weighted avg       0.82      0.81      0.81      1644
```

تحلیل نتایج

نتایج بخش الف

```
Best parameters:  {'n_neighbors': 11}
              precision    recall  f1-score   support

           0       0.82      0.72      0.77       217
           1       0.84      0.76      0.80       156
           2       0.91      0.87      0.89       197
           3       0.74      0.84      0.79       227
           4       0.88      0.91      0.89       244
           5       0.90      0.91      0.91       256
           6       0.99      0.94      0.97       138
           7       0.81      0.86      0.83       209

    accuracy                           0.85      1644
   macro avg       0.86      0.85      0.86      1644
weighted avg       0.86      0.85      0.85      1644
```

نتایج بخش ب

```
Best parameters:  {'n_neighbors': 11}
              precision    recall  f1-score   support

           0       0.67      0.75      0.71       217
           1       0.83      0.67      0.74       156
           2       0.89      0.81      0.85       197
           3       0.68      0.85      0.76       227
           4       0.85      0.89      0.87       244
           5       0.84      0.81      0.82       256
           6       0.97      0.85      0.91       138
           7       0.88      0.79      0.84       209

    accuracy                           0.81      1644
   macro avg       0.83      0.80      0.81      1644
weighted avg       0.82      0.81      0.81      1644
```

توقع داشتم که روش دوم نتایج بهتری را ارائه دهد چون در واقع بر اساس یک ضریب میانگین حسابی استفاده می کرد اما مشاهده کردیم که نتایج افت پیدا کرد. شاید یکی از دلایلش این است word2vec یک مدل عمیق است و مدل های عمیق مشاهده شده که نتایج بهتری تحویل میدهند.

بخش ٤

پیاده سازی perplexity

Perplexity یک معیار استفاده می‌شود برای ارزیابی عملکرد یک مدل زبان در پیش‌بینی یک دنباله کلمات. این معیار نشان می‌دهد که مدل زبان چقدر خوب در پیش‌بینی یک نمونه متن یا یک دنباله کلمات عمل می‌کند. به زبان ساده، perplexity نشان می‌دهد که مدل زبان چقدر شگفت‌زده یا گیج می‌شود وقتی که سعی در پیش‌بینی کلمه بعدی در یک دنباله دارد.

۱. محاسبه: Perplexity با استفاده از توزیع احتمالی که توسط مدل زبان برای یک دنباله کلمات داده شده پیش‌بینی می‌شود. این معیار برعکس احتمال مجموعه آزمایشی است، با توجه به تعداد کلمات معمولی‌سازی می‌شود. به طور ریاضی، perplexity (PP) به شکل زیر محاسبه می‌شود:

$$P(S) = P(w1, w2, \dots, wn)$$

$$Perplexity(S) = P(w1, w2, \dots, wn)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w1, w2, \dots, wn)}}$$

$$Perplexity(S) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(wi|w1, w2, \dots, wi-1)}}$$

Goal: giving higher probability to frequent texts

$\Longrightarrow$ minimizing the perplexity of the frequent texts

۲. تفسیر: یک perplexity کمتر نشان می‌دهد که مدل اطمینان بیشتری دارد و بهتر در پیش‌بینی دنباله کلمات عمل می‌کند. یک perplexity به اندازه یک به معنای این است که مدل دنباله کلمات را به‌طور کامل پیش‌بینی می‌کند که عملیاتی نیست. برعکس احتمال مرتبط است. مقدار کمتر نشان‌دهنده احتمال بالاتری است که مدل به درستی دنباله کلمات را پیش‌بینی کرده است. Perplexity را مثل انتخاب پاسخ در یک سوال چندگزینه تصور کنید. مقدار کم نشان می‌دهد که مدل دارای مجموعه کوچکی از کلمات احتمالی برای انتخاب

است، در حالی که یک perplexity بالا نشان‌دهنده وجود یک انتخاب گسترده‌تر از امکانات است، که باعث می‌شود پیش‌بینی کلمه صحیح دشوارتر شود.

۳. ارزیابی: Perplexity معمولاً برای مقایسه مدل‌های زبانی مختلف یا تنظیمات مختلف همان مدل استفاده می‌شود. این معمولاً در طول فرایند آموزش برای نظارت بر عملکرد مدل و هدایت تنظیمات هایپرپارامتر استفاده می‌شود. علاوه بر این، perplexity می‌تواند برای مقایسه عملکرد مدل‌های زبانی در وظایف یا مجموعه داده‌های مختلف استفاده شود.

با این حال، مهم است بدانید که perplexity همیشه قابل فهم نیست و ممکن است با ارزیابی انسانی از پردازش زبان یا هماهنگی، هماهنگی نداشته باشد. بنابراین، اگرچه perplexity یک معیار مفید برای ارزیابی مدل‌های زبانی است، اما باید با تکنیک‌های ارزیابی دیگر مانند ارزیابی انسانی یا معیارهای وظیفه‌ای، ترکیب شود تا درک جامعی از عملکرد مدل بدست آید.

۴. پیاده سازی: در قسمت ۱ پیاده سازی انجام شده بود با این حال توضیح آن را نیز در این قسمت آوردیم. پیاده سازی طبق همین عکس است فقط بعد هر ده بار محاسبه بر عکس احتمال عدد زیر رادیکال آن را جایی ذخیره کردم چون اعداد خیلی بزرگ می شد و نامپای به بی نهایت نسبت می داد ولی در مقدار آن هیچ تاثیری ندارد.

```python
def perplexity(self, test_data):
    ppl = 1
    total_ngrams = 0
    perplexity = []
    for i in range(len(test_data) - self.n + 1):
        context = tuple(test_data[i:i + self.n - 1])
        token = test_data[i + self.n - 1]
        prob = self.probability(token, context)
        ppl *= (1/prob)
        total_ngrams += 1
        if i % 10 == 0:
            perplexity.append(ppl ** (1/len(test_data)))
            ppl = 1
        # if len(test_data) < 100 or len(test_data) - i < 100:
        #     perplexity.append(ppl ** (1/len(test_data)))
    perplex = 1
    for ppl in perplexity:
        perplex *= ppl
    return perplex
```

پیاده سازی accuracy و f1 score در حالت چند کلاسه

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

صحت یا همان accuracy در حالت چند کلاسه تفاوتی ندارد.

اما f1 به اسین گونه است برای هر هر کلاس فرض می کنیم که بقیه کلاس ها منفی هستند و خودش مثبت این کار برای همه کلاس ها انجام می دهیم و در اخر میانگین می گیریم.

```python
# P4 implementation of multiclass Accuracy and F1 score
def accuracy(y_true, y_pred):
    correct = np.sum(y_true == y_pred)
    total = len(y_true)
    return correct / total

def f1_score(y_true, y_pred):
    # Calculate precision and recall for each class
    classes = np.unique(y_true)
    f1_scores = []
    for cls in classes:
        tp = np.sum((y_true == cls) & (y_pred == cls))
        fp = np.sum((y_true != cls) & (y_pred == cls))
        fn = np.sum((y_true == cls) & (y_pred != cls))
        precision = tp / (tp + fp) if (tp + fp) > 0 else 0
        recall = tp / (tp + fn) if (tp + fn) > 0 else 0
        f1 = 2 * precision * recall / (precision + recall) if (precision + recall) > 0 else 0
        f1_scores.append(f1)
    # Calculate the average F1-score
    return np.mean(f1_scores)
```

```
Accuracy of wighted vector with knn 0.8083941605839416
f1 score of wighted vector with knn 0.8117163323117935
```