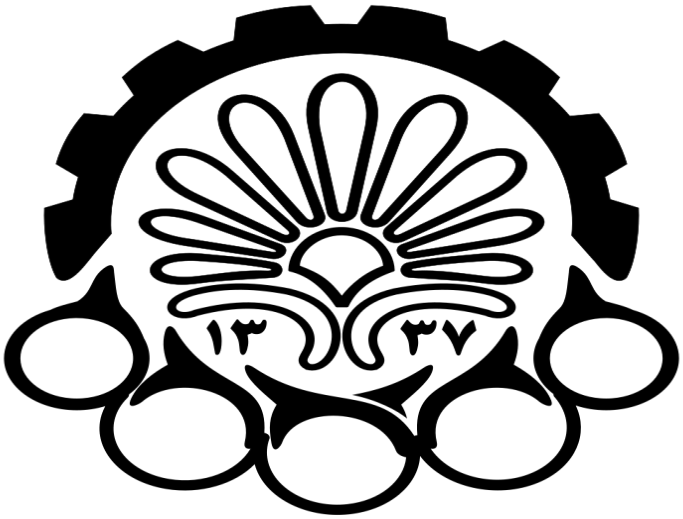


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

درس پردازش زبان طبیعی

استاد ممتازی

نیما پری فرد

۴۰۲۱۳۱۰۱۷

فهرست تمرین اول پردازش زبان طبیعی

نیما پری فرد ۴۰۲۱۱۳۱۰۷

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 مشترک و مع انتخابی اعضای کمیسیون برنامه، بودجه و محاسبات مجلس و شورای هماهنگی وزارت بهداش...	7
1	به گزارش خبرنگار فوتبال و فوتسال گروه ورزشی باشگاه خبرنگاران جوان، آیین پرچمان با حضور در زمین صنایع دفاع به دوبیدن به دور زمین پرداختند. حمین حمینی با وجود آمید دیدگی چشم در تمرینات تیمش حاضر شد. با...	0
2	بهروز اکرمی، 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، معید 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، نفوذ ناپذیری، کمبود و کمری خدمات از جمله مهمترین معاشل عملکردی و کالبدی محلات دارای بافت فرسوده هستند، گفت: عده راه حل...	0
6	یک زن موثری 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 گوشت قرمز را دارند بهتر است چربی‌های آن را جدا کنند و بهترین روش طبخ برای گوشتها بخارپز...	7
8	به گزارش حوزه دفاعی- امنیتی باشگاه خبرنگاران؛ مردار مرتیپ پناه‌از عید قمعود جزایری با اشاره به پاره‌ای اسناد مربوط به اتاقهای فکر آمریکا و رژیم صهیونیستی که دالت بر به نتیجه رسیدن فشار بر جمهوری ...	3
9	بوستون دایناامیکو تا به حال توانسته رباتهایش را به کارهای عجیب و غریب زیادی وادار کند. با این وجود، مهندسین این کمپ‌ن- معروف مثی کردند که با ویدیوی کوتاهی، قابلیتهای حرکتی منحصر به فرد ربات امپات...	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.

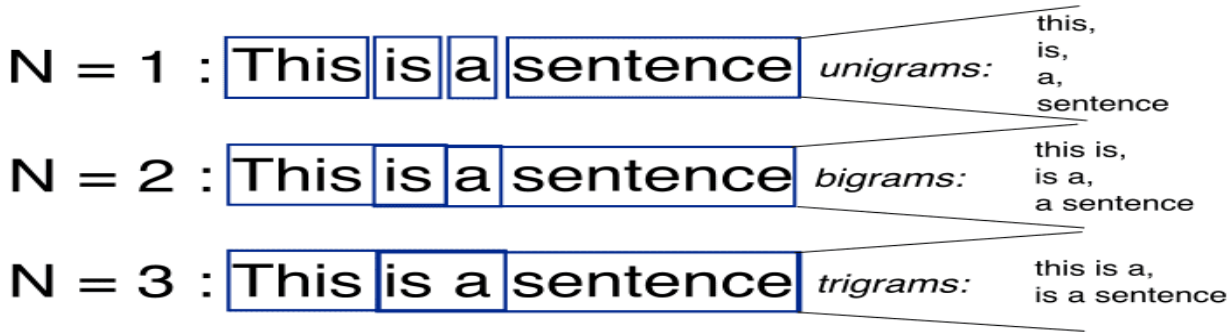
```python
from hazm import Normalizer, word_tokenize
# Create a normalizer object
normalizer = Normalizer()   normalizer: <hazm.normalizer

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
    self.number_of_minus_n_grams = sum(self.counts_minus_one_grams.values())
    self.number_of_n_grams = sum(self.counts.values())
```

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.

```python
def probability(self, token, context):
    context = tuple(context)
    ngram = context + (token,)
    if self.n == 1:
        if ngram in self.counts:
            return self.counts[ngram] / self.number_of_n_grams
    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.

```python
def perplexity(self, test_data):
    log_prob_sum = 0
    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)
        log_prob_sum += np.log(prob)
    return np.exp(-log_prob_sum / len(test_data))
```

از این کد برای محاسبه perplexity استفاده 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.

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=0.000001):
        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 self.n == 1:
            if ngram in self.counts:
                return self.counts[ngram] / self.number_of_n_grams
        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
            interpolation = self.discount * context_count / self.number_of_minus_n_grams
            return max(ngram_count - self.discount, 0) / context_count + interpolation * self.p_bg
        return  self.p_bg
```

Executed at 2024.04.17 13:07:51 in 70ms

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__.Ba
for tokens in train['content']:   train: DataFrame (13314, 2)
    backoff_model.update_counts(tokens)   backoff_model: <__main__.BackoffSmoothing object at 0x0000019071372 4D0>
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.1]   background_prob: l
perplexity_dict = {}   perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 117202.1542707947, (0
for p_bg in background_prob:   background_prob: list (11)
    backoff_model.p_bg = p_bg   p_bg: 0.1    backoff_model: <__main__.BackoffSmoothing object at 0x000001907137248
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)   perplexity_dict: {(0.001, 0.001): 27899

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)   best_p_bg: 0.1    perplexity_dict: {(0.001, 0.001): 2
backoff_model.p_bg = best_p_bg   p_bg: 0.1    backoff_model: <__main__.BackoffSmoothing object at 0x0000019071372
# Evaluate the model
for i, content in enumerate(test['content']):   test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens   test_tokens: list (363)    token:
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.perplexity(test_tokens)
Executed at 2024.04.17 12:27:27 in 40s 251ms
```

```
Perplexity of the 1496 BackoffSmoothing model on the test data: 1463.4896999273092
Perplexity of the 1497 BackoffSmoothing model on the test data: 1947.8821729767596
Perplexity of the 1498 BackoffSmoothing model on the test data: 1271.7372004414879
Perplexity of the 1499 BackoffSmoothing model on the test data: 1672.2160587243343
Perplexity of the 1500 BackoffSmoothing model on the test data: 1397.72954294788
Perplexity of the 1501 BackoffSmoothing model on the test data: 945.3459046602057
Perplexity of the 1502 BackoffSmoothing model on the test data: 1356.6886544153624
Perplexity of the 1503 BackoffSmoothing model on the test data: 1989.3090517372782
Perplexity of the 1504 BackoffSmoothing model on the test data: 1155.4848876592107
Perplexity of the 1505 BackoffSmoothing model on the test data: 1852.6280821036162
Perplexity of the 1506 BackoffSmoothing model on the test data: 2359.628220011928
Perplexity of the 1507 BackoffSmoothing model on the test data: 965.3777187107368
Perplexity of the 1508 BackoffSmoothing model on the test data: 1193.40228908016
Perplexity of the 1509 BackoffSmoothing model on the test data: 1707.6937503762372
```

بایگرم با back off smoothing

```python
# bigrams
# Train the model with BackoffSmoothing
backoff_model = BackoffSmoothing(2, 0.1)  backoff_model: <__main__.BackoffSmoothing object at 0x000001
for tokens in train['content']:  train: DataFrame (13314, 2)
    backoff_model.update_counts(tokens)  backoff_model: <__main__.BackoffSmoothing object at 0x000001
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.1]  backgrou
perplexity_dict = {}  perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 117202.1542
for p_bg in background_prob:  background_prob: list (11)
    backoff_model.p_bg = p_bg  p_bg: 0.1  backoff_model: <__main__.BackoffSmoothing object at 0x0000
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)  perplexity_dict: {(0.001, 0.0

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.1  perplexity_dict: {(0.001,
backoff_model.p_bg = best_p_bg  p_bg: 0.1  backoff_model: <__main__.BackoffSmoothing object at 0x000
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens  test_tokens: list (363)
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.perplexity(t
```

Executed at 2024.04.17 12:31:10 in 3m 31s 705ms

```
Perplexity of the 1630 BackoffSmoothing model on the test data: 42.9834578884768
Perplexity of the 1631 BackoffSmoothing model on the test data: 46.545220206917314
Perplexity of the 1632 BackoffSmoothing model on the test data: 55.00547145827366
Perplexity of the 1633 BackoffSmoothing model on the test data: 62.0005292237633
Perplexity of the 1634 BackoffSmoothing model on the test data: 55.97499210535065
Perplexity of the 1635 BackoffSmoothing model on the test data: 49.80043260603905
Perplexity of the 1636 BackoffSmoothing model on the test data: 50.989681158745704
Perplexity of the 1637 BackoffSmoothing model on the test data: 39.52420223976061
Perplexity of the 1638 BackoffSmoothing model on the test data: 61.20852671430907
Perplexity of the 1639 BackoffSmoothing model on the test data: 85.57248393963096
Perplexity of the 1640 BackoffSmoothing model on the test data: 68.27408186582947
Perplexity of the 1641 BackoffSmoothing model on the test data: 76.74241371768042
Perplexity of the 1642 BackoffSmoothing model on the test data: 30.548339525233217
Perplexity of the 1643 BackoffSmoothing model on the test data: 65.21304679393566
```

ترایگرم با back off smoothing

```python
# trigrams
# Train the model with BackoffSmoothing
backoff_model = BackoffSmoothing(3, 0.1)  backoff_model: <__main__.BackoffSmoothing object a
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, 0.01, 0.1]
perplexity_dict = {}  perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 1
for p_bg in background_prob:  background_prob: list (11)
    backoff_model.p_bg = p_bg  p_bg: 0.1  backoff_model: <__main__.BackoffSmoothing object
    perplexity_dict[p_bg] = backoff_model.perplexity(validation_tokens)  perplexity_dict: {(

#choose the best hyperparameter
best_p_bg = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.1  perplexity_dict
backoff_model.p_bg = best_p_bg  p_bg: 0.1  backoff_model: <__main__.BackoffSmoothing objec
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens  test_tokens: l
    print(f"Perplexity of the {i} BackoffSmoothing model on the test data: {backoff_model.per
```

Executed at 2024.04.17 12:40:40 in 9m 29s 577ms

```
Perplexity of the 1622 BackoffSmoothing model on the test data: 9.527049673352415
Perplexity of the 1623 BackoffSmoothing model on the test data: 12.725339904332008
Perplexity of the 1624 BackoffSmoothing model on the test data: 11.64016558411867
Perplexity of the 1625 BackoffSmoothing model on the test data: 11.32751743475167
Perplexity of the 1626 BackoffSmoothing model on the test data: 12.421405264539487
Perplexity of the 1627 BackoffSmoothing model on the test data: 9.105096805234012
Perplexity of the 1628 BackoffSmoothing model on the test data: 7.588061746762389
Perplexity of the 1629 BackoffSmoothing model on the test data: 13.2738422433352
Perplexity of the 1630 BackoffSmoothing model on the test data: 9.419725935627453
Perplexity of the 1631 BackoffSmoothing model on the test data: 11.137598513887669
Perplexity of the 1632 BackoffSmoothing model on the test data: 11.364816797565963
Perplexity of the 1633 BackoffSmoothing model on the test data: 11.875808312764068
Perplexity of the 1634 BackoffSmoothing model on the test data: 10.972342086937111
Perplexity of the 1635 BackoffSmoothing model on the test data: 12.65959752623195
```

یونیگرم با absolute discounting smoothing

```
# unigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(1)  absolute_discounting_model: <__main__.AbsoluteDiscounting object at
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscounting object at
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.1]  background_prob: list (11)
discounts = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]  discounts: list (6)
perplexity_dict = {}  perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 117202.1542707947, (0.001, 0.1
for p_bg in background_prob:  background_prob: list (11)
    for discount in discounts:  discounts: list (6)
        absolute_discounting_model.p_bg = p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteDiscounting obj
        absolute_discounting_model.discount = discount  discount: 0.7  absolute_discounting_model: <__main__.AbsoluteDis
        perplexity_dict[(p_bg, discount)] = absolute_discounting_model.perplexity(validation_tokens)  perplexity_dict: {(

#choose the best hyperparameter
best_p_bg, best_discount = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.1
absolute_discounting_model.p_bg = best_p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteDiscounting object
absolute_discounting_model.discount = best_discount  discount: 0.7  absolute_discounting_model: <__main__.AbsoluteDiscou
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens  test_tokens: list (363)  token: 'ج'  co
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_discounting_model.perplexity(test
    absolute_discounting_model.applied_discounting = 1
Executed at 2024.04.17 13:10:05 in 2m 5s 549ms
```

```
Perplexity of the 1620 Absolute Discounting model on the test data: 1594.1505614924934
Perplexity of the 1621 Absolute Discounting model on the test data: 1718.3330327652002
Perplexity of the 1622 Absolute Discounting model on the test data: 1517.5201883796667
Perplexity of the 1623 Absolute Discounting model on the test data: 894.7547275179481
Perplexity of the 1624 Absolute Discounting model on the test data: 1905.1508797597915
Perplexity of the 1625 Absolute Discounting model on the test data: 1263.3158841247405
Perplexity of the 1626 Absolute Discounting model on the test data: 1537.6914410324887
Perplexity of the 1627 Absolute Discounting model on the test data: 2200.036286120213
Perplexity of the 1628 Absolute Discounting model on the test data: 2858.0115482453734
Perplexity of the 1629 Absolute Discounting model on the test data: 1183.6642714941195
Perplexity of the 1630 Absolute Discounting model on the test data: 1655.9376026261177
Perplexity of the 1631 Absolute Discounting model on the test data: 1436.0703521883365
Perplexity of the 1632 Absolute Discounting model on the test data: 1617.2734426627062
Perplexity of the 1633 Absolute Discounting model on the test data: 1659.0463830505607
```

بایگرم با absolute discounting smoothing

```
# bigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(2)  absolute_discounting_model: <__main__.AbsoluteDiscounting obje
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscounting obj
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.1]  background_prob: list
discounts = [0.001, 0.01, 0.1, 0.3, 0.5, 0.7]  discounts: list (6)
perplexity_dict = {}  perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 117202.1542707947, (0.0
for p_bg in background_prob:  background_prob: list (11)
    for discount in discounts:  discounts: list (6)
        absolute_discounting_model.p_bg = p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteDiscount
        absolute_discounting_model.discount = discount  discount: 0.7  absolute_discounting_model: <__main__.Abso
        perplexity_dict[(p_bg, discount)] = absolute_discounting_model.perplexity(validation_tokens)  perplexity_di

#choose the best hyperparameter
best_p_bg, best_discount = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.1
absolute_discounting_model.p_bg = best_p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteDiscounting
absolute_discounting_model.discount = best_discount  discount: 0.7  absolute_discounting_model: <__main__.Absolut
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens  test_tokens: list (363)  token: 'ج'
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_discounting_model.perplexi
    absolute_discounting_model.applied_discounting = 1
Executed at 2024.04.17 12:49:39 in 6m 15s 948ms
Perplexity of the 1617 Absolute Discounting model on the test data: 375.9073815995182
Perplexity of the 1618 Absolute Discounting model on the test data: 295.08150073758856
Perplexity of the 1619 Absolute Discounting model on the test data: 290.72765456290927
Perplexity of the 1620 Absolute Discounting model on the test data: 279.28438042243937
Perplexity of the 1621 Absolute Discounting model on the test data: 266.58385730564953
Perplexity of the 1622 Absolute Discounting model on the test data: 268.5766728860507
Perplexity of the 1623 Absolute Discounting model on the test data: 128.15520352987272
Perplexity of the 1624 Absolute Discounting model on the test data: 737.0724680461012
Perplexity of the 1625 Absolute Discounting model on the test data: 237.65478079731946
Perplexity of the 1626 Absolute Discounting model on the test data: 486.38788476486417
Perplexity of the 1627 Absolute Discounting model on the test data: 360.4456118261212
Perplexity of the 1628 Absolute Discounting model on the test data: 599.0355172157214
Perplexity of the 1629 Absolute Discounting model on the test data: 226.35919650335197
Perplexity of the 1630 Absolute Discounting model on the test data: 341.2578063079189
```

ترایگرم با absolute discounting smoothing

```
# trigram
# Train the model with AbsoluteDiscounting
absolute_discounting_model = AbsoluteDiscounting(3)  absolute_discounting_model: <__main__.AbsoluteDiscounti
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscount
background_prob = [0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01, 0.1]  background_pro
discounts = [0.001, 0.01, 0.1, 0.3, 0.5 , 0.7]  discounts: list (6)
perplexity_dict = {}  perplexity_dict: {(0.001, 0.001): 278997.95927367604, (0.001, 0.01): 117202.154270794
for p_bg in background_prob:  background_prob: list (11)
    for discount in discounts:  discounts: list (6)
        absolute_discounting_model.p_bg = p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteD
        absolute_discounting_model.discount = discount  discount: 0.7  absolute_discounting_model: <__mai
        perplexity_dict[(p_bg, discount)] = absolute_discounting_model.perplexity(validation_tokens)  perple

#choose the best hyperparameter
best_p_bg, best_discount = min(perplexity_dict, key=perplexity_dict.get)  best_p_bg: 0.1
absolute_discounting_model.p_bg = best_p_bg  p_bg: 0.1  absolute_discounting_model: <__main__.AbsoluteDisc
absolute_discounting_model.discount = best_discount  discount: 0.7  absolute_discounting_model: <__main__.
# Evaluate the model
for i, content in enumerate(test['content']):  test: DataFrame (1644, 2)
    test_tokens = [token for token in content]  # Flatten the list of tokens  test_tokens: list (363)  tok
    print(f"Perplexity of the {i} Absolute Discounting model on the test data: {absolute_discounting_model.p
    absolute_discounting_model.applied_discounting = 1
Executed at 2024.04.17 19:33:50 in 11m 42s 964ms

 Perplexity of the 1585 Absolute Discounting model on the test data: 2167.9137470693863
 Perplexity of the 1586 Absolute Discounting model on the test data: 1598.308712479526
 Perplexity of the 1587 Absolute Discounting model on the test data: 1362.1242051752747
 Perplexity of the 1588 Absolute Discounting model on the test data: 1024.2288031632222
 Perplexity of the 1589 Absolute Discounting model on the test data: 1181.9817102829666
 Perplexity of the 1590 Absolute Discounting model on the test data: 431.19962072075947
 Perplexity of the 1591 Absolute Discounting model on the test data: 2664.0076759063168
 Perplexity of the 1592 Absolute Discounting model on the test data: 1507.2694085845371
 Perplexity of the 1593 Absolute Discounting model on the test data: 1570.6018872797108
 Perplexity of the 1594 Absolute Discounting model on the test data: 1025.23884651199
 Perplexity of the 1595 Absolute Discounting model on the test data: 1567.5476558630044
```

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.

```
best_pg = backoff_model.p_bg
backoff_model = BackoffSmoothing(1, best_pg)  # 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 token in test_class_tokens[i]]  # 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, best_pg)  # 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 token in test_class_tokens[i]]  # 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, best_pg)  # 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 token in test_class_tokens[i]]  # 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 2024.04.17 13:04:33 in 8m 19s 238ms

 Perplexity of the class 0 BackoffSmoothing model on the test data with unigram: 1542.8926711581457
```

```
Perplexity of the class 0 BackoffSmoothing model on the test data with unigram: 1542.8926711581457
Perplexity of the class 1 BackoffSmoothing model on the test data with unigram: 1532.953566968718
Perplexity of the class 2 BackoffSmoothing model on the test data with unigram: 1472.0811466477771
Perplexity of the class 3 BackoffSmoothing model on the test data with unigram: 1407.9579632140478
Perplexity of the class 4 BackoffSmoothing model on the test data with unigram: 1794.698318055619
Perplexity of the class 5 BackoffSmoothing model on the test data with unigram: 1661.640768213993
Perplexity of the class 6 BackoffSmoothing model on the test data with unigram: 1585.387790135717
Perplexity of the class 7 BackoffSmoothing model on the test data with unigram: 1499.954491517723
Perplexity of the class 0 BackoffSmoothing model on the test data with bigram: 56.68213315685762
Perplexity of the class 1 BackoffSmoothing model on the test data with bigram: 53.88316560411561
Perplexity of the class 2 BackoffSmoothing model on the test data with bigram: 60.37978734350282
Perplexity of the class 3 BackoffSmoothing model on the test data with bigram: 59.59687057936534
Perplexity of the class 4 BackoffSmoothing model on the test data with bigram: 60.629069572613176
Perplexity of the class 5 BackoffSmoothing model on the test data with bigram: 55.33387665704807
Perplexity of the class 6 BackoffSmoothing model on the test data with bigram: 57.48375385422127
Perplexity of the class 7 BackoffSmoothing model on the test data with bigram: 58.72718173248872
Perplexity of the class 0 BackoffSmoothing model on the test data with trigram: 11.034032875448188
Perplexity of the class 1 BackoffSmoothing model on the test data with trigram: 10.140965909552628
Perplexity of the class 2 BackoffSmoothing model on the test data with trigram: 11.41679192101374
Perplexity of the class 3 BackoffSmoothing model on the test data with trigram: 11.47190111986754
Perplexity of the class 4 BackoffSmoothing model on the test data with trigram: 11.730435264340004
Perplexity of the class 5 BackoffSmoothing model on the test data with trigram: 11.176922243466613
Perplexity of the class 6 BackoffSmoothing model on the test data with trigram: 11.447511036218364
Perplexity of the class 7 BackoffSmoothing model on the test data with trigram: 11.287732088498048
```

```
best_pg = absolute_discounting_model.p_bg  best_pg: 0.75  absolute_discounting_model: <__main__.AbsoluteDiscounting object at 0x000001902340C7990>  p_bg
best_discount = 0.75  best_discount: 0.7
absolute_discounting_model = AbsoluteDiscounting(1,best_discount, best_discount)  # Change the number to the desired n-gram  absolute_discounting_model: <
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscounting object at 0x000001902340C7990>  tokens: ,
for i in labels:  labels: list (8)
    test_tokens = [token for token in test_class_tokens[i]]  # Flatten the list of tokens  test_tokens: list (363)  token: 'ج'  test_class_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,best_discount, best_discount)  # Change the number to the desired n-gram  absolute_discounting_model: <
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscounting object at 0x000001902340C7990>  tokens: ,
for i in labels:  labels: list (8)
    test_tokens = [token for token in test_class_tokens[i]]  # Flatten the list of tokens  test_tokens: list (363)  token: 'ج'  test_class_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,best_discount, best_discount)  # Change the number to the desired n-gram  absolute_discounting_model: <
for tokens in train['content']:  train: DataFrame (13314, 2)
    absolute_discounting_model.update_counts(tokens)  absolute_discounting_model: <__main__.AbsoluteDiscounting object at 0x000001902340C7990>  tokens: ,
for i in labels:  labels: list (8)
    test_tokens = [token for token in test_class_tokens[i]]  # Flatten the list of tokens  test_tokens: list (363)  token: 'ج'  test_class_tokens: ...
    print(f"Perplexity of the class {i} absolute discounting model on the test data with trigram: {absolute_discounting_model.perplexity(test_tokens)}")
Executed at 2024.04.17 19:19:52 in 13m 21s 796ms
```

```
Perplexity of the class 0 absolute discounting model on the test data with unigram: 1509.462758026619
Perplexity of the class 1 absolute discounting model on the test data with unigram: 1492.8817388604186
Perplexity of the class 2 absolute discounting model on the test data with unigram: 1432.0579462140952
Perplexity of the class 3 absolute discounting model on the test data with unigram: 1380.7551107375364
Perplexity of the class 4 absolute discounting model on the test data with unigram: 1730.5980756779888
Perplexity of the class 5 absolute discounting model on the test data with unigram: 1584.5291751243863
Perplexity of the class 6 absolute discounting model on the test data with unigram: 1543.3112467043977
Perplexity of the class 7 absolute discounting model on the test data with unigram: 1472.3280618481779
Perplexity of the class 0 absolute discounting model on the test data with bigram: 136.32575731513438
Perplexity of the class 1 absolute discounting model on the test data with bigram: 114.8476926340882
Perplexity of the class 2 absolute discounting model on the test data with bigram: 134.31743386477729
Perplexity of the class 3 absolute discounting model on the test data with bigram: 122.00519813590334
Perplexity of the class 4 absolute discounting model on the test data with bigram: 199.77117472949598
Perplexity of the class 5 absolute discounting model on the test data with bigram: 184.15880350023605
Perplexity of the class 6 absolute discounting model on the test data with bigram: 145.994065693024
Perplexity of the class 7 absolute discounting model on the test data with bigram: 134.55255200514083
Perplexity of the class 0 absolute discounting model on the test data with trigram: 442.61781338675206
Perplexity of the class 1 absolute discounting model on the test data with trigram: 293.39479344420903
Perplexity of the class 2 absolute discounting model on the test data with trigram: 545.2975815933171
Perplexity of the class 3 absolute discounting model on the test data with trigram: 489.09196533509675
Perplexity of the class 4 absolute discounting model on the test data with trigram: 542.0826011284618
Perplexity of the class 5 absolute discounting model on the test data with trigram: 409.1698117215197
Perplexity of the class 6 absolute discounting model on the test data with trigram: 487.7485630210677
Perplexity of the class 7 absolute discounting model on the test data with trigram: 516.1759752244685
```

بخش ج)

مواردی که از نتایج به دست آمده می توان تحلیل کرد.

- 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. البته در trigram برای absolute discounting این اتفاق نیافتاد دلیلش هم شاید به خاطر مدل هموار سازی است که به ما می دهد.
- توقع داشتم هموار سازی absolute discounting نتایج بهتری نسبت به backoff داشته باشد اما برای این مجموعه داده backoff برای من بهتر جواب داد.
- 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.
- بعد همان طور که در بخش اخر گفته می شود که perplexity به تنهایی نمی تواند کافی باشد باید نظر انسانی هم در نظر گرفته شود.

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.

```python
# 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
```

```python
# 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.

```python
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
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

def compute_tfidf(dataframe):
    dataframe_copy = dataframe.copy()
    dataframe_copy['content'] = dataframe_copy['content'].astype(str)

    # Initialize the TF-IDF Vectorizer
    tfidf_vectorizer = TfidfVectorizer()

    # Fit and transform the 'content' column to a TF-IDF matrix
    tfidf_matrix = tfidf_vectorizer.fit_transform(dataframe_copy['content'])

    # Create a DataFrame with the TF-IDF scores
    tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.get_feature_names_out())

    # Return the TF-IDF DataFrame
    return tfidf_df


tfidf_dataframe_train = compute_tfidf(train)   tfidf_dataframe_train: DataFrame (13314, 70716)    train: DataFrame (13314, 2)
tfidf_dataframe_test = compute_tfidf(test)   tfidf_dataframe_test: DataFrame (1644, 24003)    test: DataFrame (1644, 2)
```
Executed at 2024.04.17 14:29:09 in 13s 147ms

```python
def weighted_average_vector(words, tfidf_df, model, vector_size, content_index):
    weights = [tfidf_df.at[content_index, word] if word in tfidf_df.columns else 0 for word in words]
    vectors = [check_existence(word[2:-1], model, vector_size) * weight for word, weight in zip(words, weights)]
    weighted_vector = np.sum(vectors, axis=0) / sum(weights) if sum(weights) > 0 else np.zeros(vector_size)
    return weighted_vector

# Applying the function using .apply()
averaged_tfidf_train_vector = pd.DataFrame()   averaged_tfidf_train_vector: DataFrame (13314, 200)
averaged_tfidf_train_vector['weighted_vectors'] = train.apply(lambda row: weighted_average_vector(row['content'], tfidf_dataframe_train, model, 200, row.name),
 axis=1)   averaged_tfidf_train_vector: DataFrame (13314, 200)    train: DataFrame (13314, 2)
```
Executed at 2024.04.17 14:36:49 in 6m 49s 974ms

```python
# Applying the function using .apply()
averaged_tfidf_test_vector = pd.DataFrame()   averaged_tfidf_test_vector: DataFrame (1644, 200)
averaged_tfidf_test_vector['weighted_vectors'] = test.apply(lambda row: weighted_average_vector(row['content'], tfidf_dataframe_test, model, 200, row.name), axis=1)
```
Executed at 2024.04.17 14:40:10 in 39s 102ms

```
averaged_tfidf_train_vector   averaged_tfidf_train_vector: DataFrame (13314, 200)
```
Executed at 2024.04.17 14:49:05 in 58ms

1-10 | 13314 rows × 200 columns | pd.Dataframe ↗ | CSV

	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	19
54	-0.108319	0.007247	0.129991	0.062604	0.090599	0.189800	0.120029	0.204518	0.093822	0.045869	-0.092711	-0.035211	0.082029	0.180364	-0.094961	-0.22
89	-0.077122	0.062508	0.020513	-0.021803	0.040539	0.060304	0.072395	0.111859	0.091946	0.016544	-0.050197	-0.074023	0.059599	0.089849	-0.007602	-0.06
53	-0.072452	0.028088	0.065671	0.009953	0.067122	0.145910	0.134338	0.157480	0.033569	-0.001953	-0.090035	-0.038973	0.060554	0.159910	-0.020167	-0.23
34	-0.014847	0.026338	0.032968	0.014014	0.023027	0.039076	0.034654	0.034527	0.030257	0.008301	-0.030636	-0.006380	0.023158	0.047930	0.002099	-0.06
28	-0.030543	0.024148	0.048124	0.018505	0.019907	0.072389	0.079145	0.096556	0.054721	0.002409	-0.040711	-0.024341	0.050985	0.055208	-0.025623	-0.08
05	-0.120479	0.080791	0.177580	-0.082854	0.059184	0.121064	0.026119	0.156347	0.081463	-0.069832	-0.112872	-0.153016	0.143367	0.034473	0.008465	-0.12
22	-0.041282	0.085210	0.056023	-0.023701	0.032646	0.102815	0.095081	0.102933	0.067557	-0.019813	-0.036759	-0.076365	0.072670	0.062185	-0.014034	-0.08
8…	-0.026487	0.030501	0.049046	0.030578	0.021651	0.081918	0.084804	0.054746	0.076372	0.017496	-0.073573	-0.019154	0.022958	0.079713	-0.005821	-0.10
75	-0.025417	0.034797	0.056174	0.034580	0.046804	0.091115	0.085944	0.136951	0.090449	0.002809	-0.033646	-0.004243	0.016499	0.066563	0.023303	-0.10
59	-0.040113	0.006391	0.059389	0.021155	0.031199	0.099799	0.087533	0.110901	0.031705	0.016261	-0.042042	-0.017539	0.014248	0.079584	-0.034607	-0.07

```python
# Define parameter grids for each classifier
param_grids = {  param_grids: {'KNN': {'n_neighbors': [3, 5, 7, 9, 11, 20]}, 'RandomForest': {'max_features': ['sqrt', 'log2
    'KNN': {'n_neighbors': [3, 5, 7, 9, 11, 20]},
    'RandomForest': {'n_estimators': [50, 100, 200], 'max_features': ['sqrt', 'log2']},
    'SVM': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
}

classifiers = {  classifiers: {'KNN': KNeighborsClassifier(), 'RandomForest': RandomForestClassifier(), 'SVM': SVC()}
    'KNN': KNeighborsClassifier(),  KNeighborsClassifier: <class 'sklearn.neighbors._classification.KNeighborsClassifier'>
    'RandomForest': RandomForestClassifier(),  RandomForestClassifier: <class 'sklearn.ensemble._forest.RandomForestClassifi
    'SVM': SVC()  SVC: <class 'sklearn.svm._classes.SVC'>
}

best_scores = {}  best_scores: {'KNN': 0.343022294272205, 'RandomForest': 0.40093114508783156, 'SVM': 0.5042064600659115}
best_params = {}  best_params: {'KNN': {'n_neighbors': 20}, 'RandomForest': {'max_features': 'sqrt', 'n_estimators': 200}, '
best_estimators = {}  best_estimators: {'KNN': KNeighborsClassifier(n_neighbors=20), 'RandomForest': RandomForestClassifier(
```

```python
# Perform Grid Search for each classifier
for name, classifier in classifiers.items():  classifiers: {'KNN': KNeighborsClassifier(), 'RandomForest': RandomFores
    grid_search = GridSearchCV(classifier, param_grids[name], cv=5, n_jobs=-1)  grid_search: GridSearchCV(cv=5, estima
    grid_search.fit(averaged_tfidf_train_vector, train['label'])  grid_search: GridSearchCV(cv=5, estimator=SVC(), n_j
    best_scores[name] = grid_search.best_score_  best_scores: {'KNN': 0.343022294272205, 'RandomForest': 0.40093114508
    best_params[name] = grid_search.best_params_  best_params: {'KNN': {'n_neighbors': 20}, 'RandomForest': {'max_feat
    best_estimators[name] = grid_search.best_estimator_  best_estimators: {'KNN': KNeighborsClassifier(n_neighbors=20)
    print(f"Best parameters for {name}: ", grid_search.best_params_)  name: 'SVM'  grid_search: GridSearchCV(cv=5, e
    print(f"Best cross-validation score for {name}: {grid_search.best_score_:.3f}")  name: 'SVM'  grid_search: GridS

# Determine the classifier with the best score
best_classifier_name = max(best_scores, key=best_scores.get)  best_classifier_name: 'SVM'  best_scores: {'KNN': 0.34
print(f"Best classifier is {best_classifier_name} with a score of {best_scores[best_classifier_name]:.3f}")  best_clas

# Use the best estimator to make predictions on the test data
best_classifier = best_estimators[best_classifier_name]  best_classifier: SVC(C=10)  best_estimators: {'KNN': KNeigh
predictions = best_classifier.predict(averaged_tfidf_test_vector)  predictions: ndarray (1644,)  best_classifier: SV

print(classification_report(test['label'], predictions))  test: DataFrame (1644, 2)  predictions: ndarray (1644,)
```

Executed at 2024.04.17 15:11:56 in 14m 53s 613ms

```
Best parameters for KNN:  {'n_neighbors': 20}
Best cross-validation score for KNN: 0.343
Best parameters for RandomForest:  {'max_features': 'sqrt', 'n_estimators': 200}
Best cross-validation score for RandomForest: 0.401
Best parameters for SVM:  {'C': 10, 'kernel': 'rbf'}
Best cross-validation score for SVM: 0.504
Best classifier is SVM with a score of 0.504
              precision    recall  f1-score   support

           0       0.44      0.46      0.45       217
           1       0.48      0.42      0.45       156
           2       0.54      0.53      0.54       197
           3       0.45      0.50      0.47       227
           4       0.53      0.53      0.53       244
           5       0.55      0.62      0.58       256
           6       0.71      0.59      0.65       138
           7       0.54      0.50      0.52       209

    accuracy                           0.52      1644
   macro avg       0.53      0.52      0.52      1644
weighted avg       0.52      0.52      0.52      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 for KNN:  {'n_neighbors': 20}
Best cross-validation score for KNN: 0.343
Best parameters for RandomForest:  {'max_features': 'sqrt', 'n_estimators': 200}
Best cross-validation score for RandomForest: 0.401
Best parameters for SVM:  {'C': 10, 'kernel': 'rbf'}
Best cross-validation score for SVM: 0.504
Best classifier is SVM with a score of 0.504
              precision    recall  f1-score   support

           0       0.44      0.46      0.45       217
           1       0.48      0.42      0.45       156
           2       0.54      0.53      0.54       197
           3       0.45      0.50      0.47       227
           4       0.53      0.53      0.53       244
           5       0.55      0.62      0.58       256
           6       0.71      0.59      0.65       138
           7       0.54      0.50      0.52       209

    accuracy                           0.52      1644
   macro avg       0.53      0.52      0.52      1644
weighted avg       0.52      0.52      0.52      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 شدیدی پیدا کرد. شاید یکی از دلایلش این است word2vec یک مدل عمیق است و مدل های عمیق مشاهده شده که نتایج بهتری تحویل میدهند. و اینکه در word2vec یک

فرایند اموزش را طی کرده تا اموزش ببیند بردار های مناسب را برگرداند و که منسب معنای کلمه باشد و نمی توان انتظار داشت با وزن دار کردن میانگین نتایج بهبود پیدا کند.

بخش ۴

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 از فرمول دیگری perplexity استفاده کردم که معادل همان عکس صفحه قبل است.

```
def perplexity(self, test_data):
    log_prob_sum = 0
    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)
        log_prob_sum += np.log(prob)
    return np.exp(-log_prob_sum / len(test_data))
```

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 0.5206812652068127
f1 score of wighted vector with knn 0.5233737501517393
```