به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

پـزدازش زبـان هـای طبـیـعـی

CA1

فاطمه سلیقه

۸۱۰۱۹۸۳۰۶

بهمن ماه ۱۳۹۸

سوال ۱-پیش پردازش

در این قسمت از کتابخانه parsivar استفاده شده است . به این صورت که داده ها را نرمال کرده و متن را به جملات تقسیم کرده (sentence segmentation) و سپس نشانه گذاری ها و کاراکترهای اضافه را حذف نموده و اعداد را حذف کرده .

```
def preprocess(paths,entry):
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
    sentences = []
#     classi = []
    for txt in paths:
        file = open('train/'+entry+'/'+txt, encoding="utf8")
        text = file.read()
        file.close()
        #remove numbers
        text = re.sub(r"\d+", "", text)
        #normalize
        my_normalizer.normalize(text)
        #sentence segmentation
        sents = my_tokenizer.tokenize_sentences(my_normalizer.normalize(text))
        #remove punctuations
        for sent in sents:
            no_punct = ""
            for char in sent:
                if char not in punctuations:
                    no_punct = no_punct + char
            sentences.append(no_punct)
```

هم چنین در هنگام تقسیم جملات به کلمات lemmatization را انجام می دهیم :

```
  for word in words:
      x=lemmatizer.lemmatize(word)
```

سوال ۲- ایجاد مدل های زبانی

در ابتدا ۸۰ درصد از فایل های هر موضوع خبری را به تابع preprocess داده تا جملات پیش پردازش شده را بدهد.

در این قسمت برای ایجاد مدل های زبانی می توان از کتابخانه nltk استفاده کرد . در این قسمت هم به پیاده سازی مدل های زبانی پرداخته و هم استفاده از کتابخانه nltk بررسی شده است .

برای مدل های خواسته شده دومدلی که براساس کلمه هستند به این صورت عمل شده که جملات را به کلمه تقسیم می کنیم .

```
my_tokenizer.tokenize_words(sent)
```

و برای مدل های بر اساس حروف هر کلمه را به حرف ها تقسیم می کنیم .

```
[char for char in sent]
```

مدل یکتایی کلمه :

در این مدل لازم است تا هر کلمه را بررسی کنیم که چند بار در داده ها تکرار شده است . سپس برای محاسبه احتمال هر کلمه تعداد باری که کلمه تکرار شده است تقسیم بر تعداد کل کلمات می کنیم .

برای این کار یک دیکشنری ایجاد می کنیم که به هر کلمه تعداد باری که تکرار شده را نسبت می دهیم .

مدل دوتایی کلمه :

برای این قسمت تعداد باری که یک کلمه بعد از کلمه دیگر آمده را شمارش می کنیم . در این قسمت هم یک دیکشنری در نظر می گیریم که به هر دو کلمه که پشت سر هم آمده اند تعداد تکرارشان را نسبت می دهد . سپس تقسیم بر تعداد کل دو کلمه ای های موجود می کنیم تا احتمال هردو کلمه به دست آید .

مدل یکتایی حرف:

در این قسمت مانند یکتایی کلمه است با این تفاوت که تعداد باری که یک حرف تکرار شده را می شماریم .

مدل دوتایی حرف:

این مدل مانند مدل دوتایی کلمه است با این تفاوت که تعداد باری که دو حرف پشت سر هم تکرار شده اند را می شماریم .

حال برای پیاده سازی دو تابع unigram , bigram نوشته شده که در unigram تعداد باری که یک توکن تکرار شده را می شمارد و احتمالش را حساب می کند و در bigram تعداد باری که دو توکن پشت سر هم تکرار شده اند را می شمارد و احتمال را حساب می کند .

```
class unigram:
    def __init__(self,sentences):
        self.corpus_length = 0
        self.unigram_frequencies = dict()
        for sentence in sentences:
            for word in sentence:
                self.unigram_frequencies[word] = self.unigram_frequencies.get(word, 0) + 1
                self.corpus_length += 1
    def unigram_probability(self,word):
        return float(self.unigram_frequencies.get(word, 0) + 1)/float(self.corpus_length+len(self.unigram_frequencies)+1)
```

```
class bigram(unigram):
    def __init__(self,sentences):
        unigram.__init__(self, sentences)
        self.corpus_length = 0
        self.bigram_frequencies = dict()
        self.unique_bigrams = set()
        for sentence in sentences:
            previous_word = None
            for word in sentence:
                if previous_word != None:
                    self.bigram_frequencies[(previous_word, word)] = self.bigram_frequencies.get((previous_word, word),0) + 1
                    self.unique_bigrams.add((previous_word, word))
                previous_word = word
    def bigram_probability(self,word,previous_word):
        return float(self.bigram_frequencies.get((previous_word, word), 0) + 1)/float(self.unigram_frequencies.get(previous_w
```

حال برای هر کدام از موضوع های خبری ۴ مدل بیان شده را ایجاد کرده و برای کلاسبندی استفاده می کنیم .

نکته ای که وجود دارد آن است که همان طور که در کدهای بالا می بینیم از روش های smoothing در اینجا روشadd one استفاده شده است .

سوال ۳-معیار سرگشتگی

الف)

در این قسمت دو تابع نوشته شده است یکی معیار سرگشتگی برای unigram را محاسبه می کند و دیگری معیار سرگشتگی برای bigram .

حال لازم است تا از ۲۰ درصد باقیمانده فایل ها برای محاسبه معیار سرگشتگی استفاده کنیم . فایل ها را برای هر مدل جدا گانه بررسی می کنیم . به این صورت که برای اینکه بفهمیم کدام نوع مدل از مدل های یکتایی کلمه و دوتایی کلمه و..... بهتر هستند لازم است تا دقت هر کدام را محاسبه کنیم .

برای نوع اول هر فایل را به مدل های ساخته شده برای هر موضوع خبری از ان نوع می دهیم و معیار سرگشتگی را محاسبه می کنیم هر کدام کمتر بود یعنی آن فایل متعلق به آن موضوع خبری است . به همین ترتیب همه داده های اعتبار سنجی را بررسی می کنیم تا بتوانیم ماتریس درهم ریختگی را محاسبه کنیم. و همین طور برای نوع دوم و سوم و چهارم .

```
def unigram_perplexity(model,testset):
    perplexity = 0
    N = 0
    for sent in testset:
        for word in sent:
            N += 1
            perplexity = perplexity + math.log(model.unigram_probability(word),2)
    perplexity = perplexity*(-1/float(N))

    perplexity = pow(perplexity, 2)
    return perplexity
```

```
def bigram_perplexity(model,testset):
    perplexity = 0
    N = 0
    for sent in testset:
        previous_word = None
        for word in sent:
            if(previous_word != None):
                N += 1
                perplexity = perplexity + math.log(model.bigram_probability(word,previous_word),2)
            previous_word = word
    perplexity = perplexity*(-1/float(N))
    perplexity = pow(perplexity, 2)
    return perplexity
```

برای جلوگیری از بینهایت شدن تابع سرگشتگی از log استفاده شده است .

ب)

برای اعتبار سنجی یک تابع به نام validation نوشته شده که نام فایل ها و اینکه هر فایل متعلق به کدام موضوع خبری است و نوع مدل مورد نظر را ورودی می گیرد و به عنوان خروجی موضوع پیش بینی شده از هر فایل را خروجی می دهد .

مدل یکتایی کلمه :

```
Accuracy: 0.771186

array([[44,  5, 13, 16,  0,  6],
       [ 0, 74,  6,  0,  0,  4],
       [11, 12, 54,  3,  1,  0],
       [ 8,  7,  6, 54,  0,  4],
       [ 1,  1,  1,  1, 83,  1],
       [ 0,  1,  0,  0,  0, 55]], dtype=int64)
```

$$precision = \frac{TP}{TP + Fp}$$

$$avrage_{precision} = \frac{\frac{44}{64} + \frac{74}{100} + \frac{54}{80} + \frac{54}{74} + \frac{83}{84} + \frac{55}{70}}{6} = 0.76$$

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

$$avrage_{recall} = \frac{\frac{44}{84} + \frac{74}{84} + \frac{54}{81} + \frac{54}{79} + \frac{83}{88} + \frac{55}{56}}{6} = 0.78$$

$$F1 = 2\frac{precision * recall}{precision + recal} = 0.76$$

مدل دوتایی کلمه:

```
Accuracy: 0.542373

array([[ 5, 46,  4,  2,  0, 27],
       [ 0, 81,  0,  0,  0,  3],
       [ 0, 63, 16,  0,  0,  2],
       [ 1, 40,  0, 23,  0, 15],
       [ 0,  5,  0,  0, 79,  4],
       [ 0,  4,  0,  0,  0, 52]], dtype=int64)
```

$$avrage_{precision} = \frac{\frac{5}{6} + \frac{81}{239} + \frac{16}{20} + \frac{23}{25} + \frac{79}{79} + \frac{52}{103}}{6} = 0.73$$

$$avrage_{recall} = \frac{\frac{5}{6} + \frac{81}{239} + \frac{16}{20} + \frac{23}{25} + \frac{79}{79} + \frac{52}{103}}{6} = 0.73$$

$$F1 = \frac{2 * 0.73 * 0.73}{0.73 + 0.73} = 0.73$$

مدل یکتایی حرف:

```
Accuracy: 0.182203

array([[84,  0,  0,  0,  0,  0],
       [84,  0,  0,  0,  0,  0],
       [81,  0,  0,  0,  0,  0],
       [79,  0,  0,  0,  0,  0],
       [88,  0,  0,  0,  0,  0],
       [54,  0,  0,  0,  0,  2]], dtype=int64)
```

$$avrage_{precision} = \frac{\frac{84}{470} + \frac{2}{2}}{6} = 0.19$$

$$avrage_{recall} = \frac{\frac{84}{84} + \frac{2}{56}}{6} = 0.17$$

$$F1 = 0.179$$

مدل دوتایی حرف:

```
Accuracy: 0.118644

array([[ 0,  0,  0,  0,  0, 84],
       [ 0,  0,  0,  0,  0, 84],
       [ 0,  0,  0,  0,  0, 81],
       [ 0,  0,  0,  0,  0, 79],
       [ 0,  0,  0,  0,  0, 88],
       [ 0,  0,  0,  0,  0, 56]], dtype=int64)
```

$$avrage_{precision} = \frac{\frac{56}{470}}{6} = 0.01$$

$$avrage_{recall} = \frac{\frac{56}{56}}{6} = 0.16$$

$$F1 = 0.018$$

با توجه به نتایج به دست آمده مشاهده می کنیم که نتایج مدل های مربوط به کلمه بهتر از مدل های مربوط به حرف است . این مورد واضح است زیرا تشخیص از روی کلمات دقت را بالاتر می برد . اما در مورد unigram,bigram از انجایی که در یک داده اموزش مشترک امتحان می کنیم و احتمال ها در bigram پایین تر از unigram است زیرا طول عبارات افزایش می یابد . بنابراین مدل یکتایی کلمه بهتر از مدل دوتایی کلمه است .

سوال ۴

همانطور که در قسمت قبل ملاحظه نمودیم مدل اول یعنی مدل یکتایی کلمه ازهمه بهتر بود . پس برای تست از مدلاول استفاده می کنیم .

در این قسمت هم یک تابع test نوشته شده است که داده ها را label می زند و نتایج در فایل Result.csv ذخیره می شود .

```
,file,predictclass
0,10038.txt,finance
1,10066.txt,finance
2,10135.txt,culture
3,10168.txt,finance
4,10190.txt,social
5,10212.txt,culture
6,10217.txt,finance
7,10262.txt,social
8,10264.txt,finance
9,10325.txt,politic
10,10338.txt,finance
11,10357.txt,finance
12,10359.txt,culture
13,10380.txt,politic
14,10454.txt,finance
15,10460.txt,politic
16,10501.txt,finance
17,10508.txt,culture
18,10531.txt,culture
19,11218.txt,social
20,11255.txt,finance
21,11274.txt,culture
22,11283.txt,finance
23,11301.txt,culture
24,11349.txt,finance
25,11398.txt,finance
26,11444.txt,finance
27,11844.txt,finance
28,11852.txt,technology
29,11904.txt,culture
```