به نام خدا

آمار و احتمالات مهندسی – پروژه اول

فرشته باقری – 810100089

پیش پردازش ها: برای دو فایل train و test روی ستون های title و description پیش پردازش های زیر انجام شده است:

1. نرمالایز کردن با استفاده از کتابخانه هضم: با این کار، فاصله های اضافی بین کلمات و حروف تکراری (مانند "م" در "سلاممممم") حذف می‌شوند.

2. حذف علائم نگارشی و اعداد: این موارد وابسته به ژانر خاصی از کتاب نیستند و احتمالاً فقط برای یک کتاب خاص به کار می‌روند. با حذف آنها، پیشبینی بهبود می‌یابد.

3. حذف کلمات توقف (Stop Words): این کلمات، کلماتی هستند که معنی خاصی را منتقل نمی‌کنند یا در اکثر متون تکرار می‌شوند. حذف آنها می‌تواند دقت پیشبینی را افزایش دهد.

4. ریشه‌یابی کلمات: بعضی کلمات به دلیل چسبیدن ضمیر یا صرف فعل‌ها با زمان و فاعل‌های متفاوت، تغییر می‌کنند. با تبدیل کلمات به ریشه‌ی آنها، دقت پیشبینی افزایش می‌یابد. برای این کار از تابع stemmer کتابخانه هضم استفاده شده است.

در تابع preprocess، بعد از انجام پیش پردازش های بالا، کلمات به صورت رشته‌های concat شده برگردانده می‌شوند. سپس، با ترکیب ستون‌های title و description و ذخیره آنها به عنوان preprocessed_text در دیتافریم، برای تشکیل bow و پیشبینی از داده‌های تست استفاده می‌شود.

سپس، ماتریس bow و normalized_bow تشکیل می‌شود تا احتمال حضور هر کلمه در یک دسته‌بندی را داشته باشیم.

در تمام پیاده سازی ها در حالتی که احتمال تعلق کتاب به هر 6 ژانر یکسان باشد -1 برگردانده می شود.

چون تعداد کتاب های هر 6 ژانر در فایل train با هم برابر است میتوان در پیاده سازی قانون بیز پارت p(c) را در نظر نگرفت.

پیاده‌سازی ساده: ممکن است در هر تست، برخی کلمات وجود داشته باشند که در یک دسته‌بندی خاص یا کل bow وجود نداشته باشند. در این صورت، می‌توانیم احتمال آنها را صفر در نظر بگیریم یا از آنها صرف نظر کنیم. اگر صفر را در نظر بگیریم، با ضرب شدن در سایر احتمال‌ها، کل عبارت را صفر می‌کنند و دقت پیشبینی کاهش می‌یابد. از طرفی، اگر از آنها صرف نظر کنیم، ممکن است دقت را پایین بیاورند. به دلیل مشکلات بالا، از روشی به نام additive smoothing استفاده می‌کنیم.

1.1 گذاشتن احتمال صفر -> در این حالت احتمال متعلق بودن به یک ژانر هر کتاب صفر می شود و عملا دقت ما صفر است .(prediction_1) در این قسمت از روش جمع لگاریتم ها هم نمی شود استفاده کرد چون لگاریتم برای اعداد مثبت تعریف شده است.

```
results = []
for index, row in test_df.iterrows():
    text = row['processed_text']
    category_scores = []

    for category_index in range(len(normalized_BOW)):
        word_scores = []

        for word in text.split():
            if word in normalized_BOW.columns:
                word_score = normalized_BOW.loc[category_index, word]
                word_scores.append(word_score)
            else:
                word_scores.append(0)

        category_score = np.prod(word_scores)
        category_scores.append(category_score)

    max_score_index = np.argmax(category_scores)
    predicted_category = normalized_BOW.loc[max_score_index, 'categories']
    results.append(predicted_category)
```

1.2 در نظر نگرفتن صفر ها -> فایل prediction_2

```
results = []
for index, row in test_df.iterrows():
    text = row['processed_text']
    category_scores = []

    for category_index in range(len(normalized_BOW)):
        word_scores = []

        for word in text.split():
            if word in normalized_BOW.columns:
                word_score = normalized_BOW.loc[category_index, word]
                if word_score !=0:
                    word_scores.append(word_score)

        category_score = np.prod(word_scores)
        category_scores.append(category_score)

    max_score_index = np.argmax(category_scores)
    predicted_category = normalized_BOW.loc[max_score_index, 'categories']
    results.append(predicted_category)
```

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 34
total : 451
accuracy percentage : 7.53

Process finished with exit code 0
```

1.3 روش دیگر -> می توان به جای احتمال های صفر یک عدد بسیار کوچک قرار داد طوری که تاثیرش در تغییر داده ها زیاد نباشد و از مشکلات احتمال صفر هم جلوگیری کرد.(prediction_3)

```
results = []
for index, row in test_df.iterrows():
    text = row['processed_text']
    category_scores = []

    for category_index in range(len(normalized_BOW)):
        word_scores = []

        for word in text.split():
            if word in normalized_BOW.columns:
                word_score = normalized_BOW.loc[category_index, word]
                if word_score ==0:
                    word_score = 0.0000000000001
                word_scores.append(word_score)
        else:
            word_scores.append(0.0000000000001)



        category_score = np.prod(word_scores)
        category_scores.append(category_score)

    max_score_index = np.argmax(category_scores)
    predicted_category = normalized_BOW.loc[max_score_index, 'categories']
    results.append(predicted_category)
```

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 89
total : 451
accuracy percentage : 19.7

Process finished with exit code 0
```

1.4 اگر در این مرحله به جای ضرب احتمال ها از جمع احتمال های استفاده کنیم دقت به صورت زیر به دست میاید: (prediction_4)

```
        category_score = np.sum(np.log(word_scores))
        category_scores.append(category_score)
```

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 201
total : 451
accuracy percentage : 44.5

Process finished with exit code 0
```

پرسش اول:

Additive smoothing : یک تکنیک استفاده شده در احتمال و آمار است که برای مدیریت مشکل احتمال‌های صفر استفاده می‌شود. این تکنیک به طور معمول در پردازش زبان طبیعی و وظایف یادگیری ماشین مانند طبقه‌بندی متن یا مدل‌سازی زبان استفاده می‌شود.
هدف اصلی این روش، حل مشکل احتمال‌های صفر است که ممکن است در هنگام برآورد احتمال‌ها از داده‌های محدود به وجود آید. در بسیاری از موارد، هنگام محاسبه احتمال‌ها، ممکن است رویدادهای ناشناخته یا کمیابی وجود داشته باشند که منجر به احتمال‌های صفر می‌شوند. این مشکل می‌تواند در استفاده از این احتمال‌ها در محاسبات یا مدل‌های بعدی مشکل‌ساز شود.
تساوی افزایشی با اضافه کردن یک مقدار ثابت کوچک (معمولاً ۱) به تعداد مشاهدات دیده شده هر رویداد، این مشکل را حل می‌کند. با این کار، اطمینان حاصل می‌شود که هیچ یک از برآوردهای احتمال صفر نیست و همچنین جریان احتمال را از رویدادهای دیده نشده یا کمیاب به سمت آن‌ها تغییر می‌دهد. به این ترتیب، حتی اگر یک رویداد در train مشاهده نشده باشد، هنوز به آن یک احتمال غیر صفر اختصاص داده می‌شود.

استفاده از additive smoothing چندین مزیت دارد:

۱. جلوگیری از احتمال‌های صفر: با اضافه کردن یک مقدار ثابت، جلوی وقوع احتمال‌های صفر را می‌گیرد که می‌تواند در محاسبات یا مدل‌های بعدی مشکل‌ساز شود.

۲. مدیریت رویدادهای ناشناخته: تساوی افزایشی احتمال‌های غیر صفر را به رویدادهای ناشناخته اختصاص می‌دهد، این امر به مدل امکان پیش‌بینی مناسب حتی برای رویدادهایی که در داده‌های آموزش وجود ندارند را می‌دهد.

به طور کلی، تساوی افزایشی یک تکنیک مفید برای مدیریت مشکل احتمال‌های صفر است و باعث بهبود قابلیت اطمینان و تعمیم‌پذیری مدل‌های احتمالاتی در برنامه‌های مختلف می‌شود.

```python
for index, row in test_df.iterrows():
    text = row['processed_text']
    category_scores = []
    for category_index in range(len(normalized_BOW)):
        word_scores = []

        for word in text.split():
            if word in normalized_BOW.columns:
                word_score = normalized_BOW.loc[category_index, word]
                if word_score ==0:
                    alpha = 1
                    row_total_count = BOW.loc[category_index].drop('categories').sum()
                    word_score = alpha / (row_total_count+ alpha)
                word_scores.append(word_score)
            else:
                alpha = 1
                row_total_count = BOW.loc[category_index].drop('categories').sum()
                word_score = alpha / (row_total_count + alpha)
                word_scores.append(word_score)

        # category_score = np.sum(np.log(word_scores))
        category_score = np.prod(word_scores)
        category_scores.append(category_score)

    max_score_index = np.argmax(category_scores)
    predicted_category = normalized_BOW.loc[max_score_index, 'categories']
    results.append(predicted_category)
```

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 124
total : 451
accuracy percentage : 27.4

Process finished with exit code 0
```

اگر این روش را با لگاریتم ترکیب کنیم دقت بهتری به ما خواهد داد.

پرسش دوم:
اگر متن توضیحات یک کتاب بسیار طولانی باشد با ضرب احتمال های هر کلمه در هم عبارتی بسیار کوچک و نزدیک به صفر تولید می شود که می تواند محاسبات را سخت کند یا دقت را کاهش دهد به همین دلیل می توانیم محاسبات را از فضای خطی به فضای لگاریتمی ببریم.
لگاریتم این خاصیت را دارد که ضرب را به جمع تبدیل می کند. پس ما می توانیم به جای اینکه ضرب احتمال ها را با هم مقایسه کنیم، جمع لگاریتم آنها را مقایسه کنیم (اگر مبنای لگاریتم بزرگتر از یک باشد جهت نامساوی حفظ می شود)

فاز امتیازی: در تمام پیشبینی های بالا از دو پیش پردازش تبدیل کلمات به ریشه ها و حذف کلمات توقف استفاده شده است. برای مقایسه میزان افزایش دقت در کد بخش 1.4 این پیش پردازش ها را حذف می کنیم و تاثیر آن را میبینیم:

بدون هیچ کدام از این دو پیش پردازش: (prediction_6)

```python
def preprocess_text(text):
    normalized_text = normalizer.normalize(text)
    punct_normalized = re.sub(r'[^\w\s]', '', normalized_text)
    tokens = hazm.word_tokenize(punct_normalized)
    # tokens_without_stopwords = [token for token in tokens if token not in sw]
    tokens_without_numbers = [re.sub(r'\d+', '', token) for token in tokens]
    non_empty_tokens = [token for token in tokens_without_numbers if token.strip() != '']
    # stemmed_tokens = [stemmer.stem(token) for token in non_empty_tokens]
    return ' '.join(non_empty_tokens)
```

در این حالت دقت ما از 44 درصد به 32 درصد کاهش می یابد:

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 145
total : 451
accuracy percentage : 32.1

Process finished with exit code 0
```

فقط با تبدیل کلمات به ریشه: (prediction_7)

نسبت به حالت قبل (بدون این دو پیش پردازش) 8 درصد کاهش دارد.

```python
def preprocess_text(text):
    normalized_text = normalizer.normalize(text)
    punct_normalized = re.sub(r'[^\w\s]', '', normalized_text)
    tokens = hazm.word_tokenize(punct_normalized)
    # tokens_without_stopwords = [token for token in tokens if token not in sw]
    tokens_without_numbers = [re.sub(r'\d+', '', token) for token in tokens]
    non_empty_tokens = [token for token in tokens_without_numbers if token.strip() != '']
    stemmed_tokens = [stemmer.stem(token) for token in non_empty_tokens]
    return ' '.join(stemmed_tokens)
```

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 111
total : 451
accuracy percentage : 24.6

Process finished with exit code 0
```

فقط با حذف stop words : (prediction_8)

```python
def preprocess_text(text):
    normalized_text = normalizer.normalize(text)
    punct_normalized = re.sub(r'[^\w\s]', '', normalized_text)
    tokens = hazm.word_tokenize(punct_normalized)
    tokens_without_stopwords = [token for token in tokens if token not in sw]
    tokens_without_numbers = [re.sub(r'\d+', '', token) for token in tokens_without_stopwords]
    non_empty_tokens = [token for token in tokens_without_numbers if token.strip() != '']
    # stemmed_tokens = [stemmer.stem(token) for token in non_empty_tokens]
    return ' '.join(non_empty_tokens)
```

در این حالت نسبت به حالت ترکیب این دو پیش پردازش 4 درصد افزایش دقت داریم :)

نسبت به حالت بدون این پیش پردازش ها، 16 درصد افزایش دقت دارد.

```
"C:\Users\my asus\Desktop\CA0\venv\Scripts\python.exe" "C:/Users/my asus/Desktop/CA0/main.py"
correct predictions : 217
total : 451
accuracy percentage : 48.1

Process finished with exit code 0
```