به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

تمرین ۲ بخش اول

نام و نام خانودگی: علی عطاءاللهی

شماره دانشجویی: ۸۱۰۱۹۹۴۶۱

اسفند ماه ۱۴۰۲

فهرست

پاسخ سوال اول ۳

پاسخ بخش اول – پیش پردازش مجموعه داده ۳

پاسخ بخش دوم – ساخت بردار جانمایی اول – TERM FREQENCY ۴

پاسخ بخش سوم – ساخت بردار جانمایی دوم – TF-IDF ۴

پاسخ بخش چهارم – ساخت بردار جانمایی سوم - PPMI ۶

پاسخ بخش پنجم - آموزش مدل ۷

پاسخ سوال اول

پاسخ بخش اول – پیش پردازش مجموعه داده

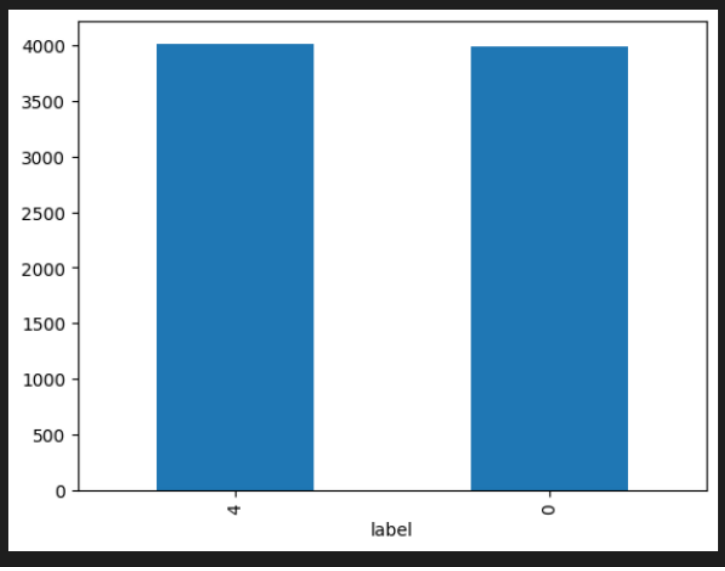
ابتدا با استفاده از کتابخانه panda فایل tweets.csv را خواندیم و سپس همان طور که خواسته شده بود از هر کلاس ۵۰۰۰ نمونه گرفتیم و بقیه را دور ریختیم. سپس همه آنها را به dataframe تبدیل کرده عملیات preprocess را روی آنها انجام دادیم و شامل مراحل توکنایز کردن و پاک کردن punctuation و stop words و stemming می‌شود. این مراحل باعث می‌شوند که متن، به یک شکل ساده‌تر و قابل فهم‌تر تبدیل شود و اطلاعات غیرضروری مانند علائم نگارشی و کلمات پرتکرار حذف شوند. استم‌کردن همچنین با تبدیل کلمات به شکل پایه‌ی آنها، باعث سازوکار ساده‌تری برای مدل‌های پردازش متنی می‌شود. این فرآیند بهبود قابلیت فهم و دقت مدل‌های پردازش زبان را ارتقاء می‌دهد.

حال با استفاده از train_test_split ۲۰ درصد از نمونه گرفته شده را برای ارزیابی قرار می‌دهیم (در قسمت ارزیابی می‌توانیم از روش cross validation استفاده کنیم و دوباره این آزمایش را تکرار کرده و ۲۰ درصد آزمون را قسمت دیگری از دیتا قرار بدهیم).

```python
def preprocess_data(file_path, random_state=42, test_size=0.2, encoding='latin-1', encoding_errors='ignore', remove_stopwords=True, stemming=False):
    data = pd.read_csv(file_path, header=None, names=['label', 'id', 'date', 'query', 'user', 'text'], encoding=encoding, encoding_errors=encoding_errors)

    classes = data['label'].unique()

    sampled_data = {c: data[data['label'] == c].sample(n=5000, random_state=random_state) for c in classes}

    combined_data = pd.concat(sampled_data.values())

    combined_data['tokens'] = combined_data['text'].apply(preprocess_text, remove_stopwords=remove_stopwords, stemming=stemming)

    train_data, eval_data = train_test_split(combined_data, test_size=test_size, random_state=random_state)

    return train_data, eval_data

def preprocess_text(text, remove_stopwords=True, stemming=False):
    tokens = nltk.word_tokenize(text.lower())

    tokens = [token for token in tokens if token.isalnum()]

    if remove_stopwords:
        stop_words = set(stopwords.words('english'))
        tokens = [token for token in tokens if token not in stop_words]

    if stemming:
        stemmer = PorterStemmer()
        tokens = [stemmer.stem(token) for token in tokens]

    return tokens
```

```
train_data['label'].value_counts().plot(kind='bar')
```
0.1s

```
<Axes: xlabel='label'>
```



پاسخ بخش دوم – ساخت بردار جانمایی اول – TERM FREQENCY

در اینجا با استفاده از CountVectorizer که از توابع sklearn.feature_extraction.text می‌باشد tf محاسبه شد (محدودیتی در استفاده از کتابخانه مطرح نشد. همچنین در بخش سوم کد این قسمت مجدد پیاده‌سازی می‌شود).

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
tf_vectors_train = vectorizer.fit_transform(train_data['tokens'].apply(' '.join))
tf_vectors_test = vectorizer.transform(eval_data['tokens'].apply(' '.join))
```
0.2s

پاسخ بخش سوم – ساخت بردار جانمایی دوم – TF-IDF

پیاده‌سازی:

برای حل مسئله محاسبه ویژگی‌های Tf-idf برای متون توییت، ابتدا هر جمله را tokenize می‌کنیم و کلمات را به مجموعه کلمات اضافه می‌کنیم تا تمام کلمات train در آن حضور داشته باشند. سپس برای هر کلمه یک index در نظر می‌گیریم تا یک دیکشنری شامل تمام کلمات train همراه با ایندکس آن‌ها داشته باشیم.

حال باید tf و idf را حساب کنیم و ضرب tf در idf را برای کلمات جمله بدست آوریم. سپس ایندکس کلمه را بدست آورده و در خانه متناظر آن در بردار ویژگی را برابر مقدار بدست آمده قرار می‌دهیم. این فرآیند در تابع tf_idf پیاده شده است.

$$TF - IDF = TF * IDF$$

برای محاسبه idf، ابتدا می‌شماریم که در داکیومنت یک کلمه چندبار تکرار شده است که در تابع count_dict پیاده شده است. از خروجی آن می‌توان برای بدست آوردن inverse_doc_freq استفاده کرد. یعنی تعداد کل متن توییت‌ها تقسیم بر تعداد وقوع کلمه در کل داکیومنت.

$$IDF = \log\left(\frac{document\ number}{1 + number\ of\ texts\ have\ word}\right)$$

برای محاسبه tf، تعداد وقوع یک کلمه تقسیم بر تعداد کل کلمات متن توییت را می‌توان در نظر گرفت که طول نظرات کمتر مهم باشد. البته در اسلایدهای درس اینطور حساب می‌کند که از تعداد وقوع کلمه در متن توییت لگاریتم می‌گیرد.

$$TF = \frac{number\ of\ appeareance\ of\ word\ in\ text}{total\ words\ number\ in\ text}$$

حال برای تمام جملات داده‌ی train تابع tf_idf را فراخوانی کرده و embedding کلمات آن‌ها را در یک لیست ذخیره می‌کنیم. دقت شود برای هر جمله یک وکتور به سایز کل کلمات داخل دیتاست داریم. یعنی این وکتورها اسپارس هستند چون خیلی از کلمات در نیامدند، پس tf_idf متناظر با آن‌ها را صفر در نظر می‌گیریم.

برای داده test هم همین کار را می‌کنیم. نکته مهم اینکه ممکن است کلماتی در تست باشند که در train نباشند. برای این کلمات هم tf_idf را صفر در نظر می‌گیریم. در پیاده‌سازی برای هندل کردن این روش از موضوع try, except در تابع tf_idf استفاده می‌کنیم.

```
    def vectorize(self):
        self.preprocess_data()
        self.create_index_dict()
        self.calculate_word_count()
        self.generate_tfidf_vectors()
        return self.tfidf_vectors_train, self.tfidf_vectors_test
```

[7] ✓ 0.3s

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ali18\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
vectorizer = TfIdfVectorizer(train_data, eval_data)
tfidf_vectors_train, tfidf_vectors_test = vectorizer.vectorize()
```

[8] ✓ 31.7s

پاسخ بخش چهارم – ساخت بردار جانمایی سوم - PPMI

پیاده‌سازی:

روش PMI به طور کلی میزان همرخداد بودن دو پدیده را محاسبه می‌کند که می‌تواند با استفاده از کلمات یا مثال‌های دیگر تعریف شود. این روش ابتدا میزان همرخداد بودن هر کلمه با لیبل مثبت و منفی منفی (همان لیبل چهار و صفر که چهار یعنی happy و مثبت و صفر یعنی sad و منفی) را محاسبه می‌کند. سپس برای تمام کلمات موجود در متن توییت، این محاسبات انجام می‌شود و مقادیر صفر برای کلماتی که در متن توییت وجود ندارند، در نظر گرفته می‌شود. در نهایت، یک بردار ویژگی اسپارس به دست می‌آید که می‌تواند به عنوان ورودی برای آموزش classifier استفاده شود.

$$PMI_{happy} = \frac{P(word, happy)}{P(word)P(happy)}$$

$$PMI_{sad} = \frac{P(word, sad)}{P(word)P(sad)}$$

برای استفاده از دو بردار PMI به عنوان ویژگی‌ها برای classifier، ابتدا این دو بردار را با هم ترکیب می‌کنیم تا یک بردار واحد به دست آید. این کار انجام می‌شود تا classifier بتواند از این ویژگی‌ها برای آموزش استفاده کند. پیاده‌سازی این روش بهترین پیاده‌سازی است که تاکنون مشاهده شده است. البته روش‌های دیگری مانند میانگین‌گیری از بردارهای PMI مثبت و یا استفاده از تفاضل بردارها نیز وجود دارد. این موارد همگی قابل اجرا هستند.

برای محاسبه احتمالات مربوط به متن توییت‌ها، ابتدا نسبت متن‌های مثبت و منفی را نسبت به کل داده‌ها محاسبه می‌کنیم. سپس با استفاده از تعداد متن‌های مثبت و منفی، احتمالات P(happy) و P(sad) را محاسبه می‌کنیم. برای محاسبه احتمال وقوع یک کلمه در متن توییت، تعداد متن‌هایی که حاوی آن کلمه هستند را محاسبه می‌کنیم. برای محاسبه احتمالات هم‌رخداد بین یک کلمه و حالت مثبت یا منفی، ابتدا داده‌های مربوط به هر کلاس را جدا می‌کنیم و سپس برای هر کدام از این دسته‌ها، تعداد تکرار کلمات را محاسبه می‌کنیم. سپس با تقسیم تعداد تکرار کلمه در هر دسته بر تعداد کل متن‌های آن دسته، احتمالات P(word,happy) و P(word,sad) را محاسبه می‌کنیم. برای محاسبه احتمالات هم‌رخداد بین هر کلمه و حالت مثبت یا منفی برای تمام کلمات موجود در متن توییت، از تابع PPMI استفاده می‌کنیم که ورودی آن متن توییت است. این تابع ابتدا دو وکتور با سایز تعداد کلمات در داده با مقدار اولیه صفر می‌سازد. سپس برای هر کلمه در جمله، تابع calculate_PPMI را فراخوانی کرده و مقادیر آن را در دو وکتور ذخیره می‌کند. در نهایت، این دو وکتور به عنوان خروجی تابع برگردانده می‌شوند. برای مدیریت کلماتی که در داده train وجود ندارند، از تکنیک try, except استفاده می‌شود.

سپس برای اینکه وکتور PPMI جملات داده train دو بعدی شود، از تابع reshape استفاده می‌شود تا وکتور PPMI مربوط به حالت‌های مثبت و منفی را کنار هم قرار دهد. برای داده test نیز همین محاسبات انجام می‌شود. در نهایت، با استفاده از یک مدل Naive Bayes از کتابخانه scikit-learn و با استفاده از ویژگی‌های بدست آمده از داده train و لیبل‌های مربوط به آن‌ها، مدل

آموزش داده می‌شود. سپس با استفاده از این مدل آموزش دیده، لیبل متن توییت‌های test را با دادن ویژگی‌های مربوط به آن‌ها پیش‌بینی می‌کنیم.

```
ppmi_vectorizer = PPMIVectorizer(train_data, eval_data)
ppmi_vectors_train, ppmi_vectors_test = ppmi_vectorizer.vectorize()
```
✓ 59.3s

پاسخ بخش پنجم - آموزش مدل

ابتدا کلاس naïve_bayes را می‌نویسیم که این کد یک تابع تعریف می‌کند که از مدل Naive Bayes چندجمله‌ای از کتابخانه scikit-learn برای آموزش و ارزیابی استفاده می‌کند. در این تابع، مدل بر روی داده‌های آموزش آموزش داده شده و سپس با استفاده از داده‌های تست، پیش‌بینی برچسب‌ها انجام می‌شود. سپس با استفاده از معیارهای مهمی مانند precision, recall, f1-score و accuracy عملکرد مدل بررسی می‌شود. این امتیازها اطلاعاتی مفید درباره کارایی مدل در پیش‌بینی برچسب‌های صحیح ارائه می‌دهند.

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

def apply_naive_bayes(vectorizer_train, vectorizer_test, train_labels, test_labels):
    naive_bayes_classifier = MultinomialNB()
    naive_bayes_classifier.fit(vectorizer_train, train_labels)
    y_pred = naive_bayes_classifier.predict(vectorizer_test)
    print(classification_report(test_labels, y_pred))
```
✓ 0.0s

نتایج TF :

```
print("TF Vectors:")
apply_naive_bayes(tf_vectors_train, tf_vectors_test, train_data['label'], eval_data['label'])
```
[29] ✓ 0.0s

```
TF Vectors:
              precision    recall  f1-score   support

           0       0.69      0.75      0.72      1012
           4       0.72      0.65      0.69       988

    accuracy                           0.70      2000
   macro avg       0.71      0.70      0.70      2000
weighted avg       0.70      0.70      0.70      2000
```

نتایج TF-IDF :

```
print("TF-IDF Vectors:")
apply_naive_bayes(tfidf_vectors_train, tfidf_vectors_test, train_data['label'], eval_data['label'])
```

[30] ✓ 1.1s

```
TF-IDF Vectors:
              precision    recall  f1-score   support

           0       0.69      0.76      0.72      1012
           4       0.73      0.66      0.69       988

    accuracy                           0.71      2000
   macro avg       0.71      0.71      0.71      2000
weighted avg       0.71      0.71      0.71      2000
```

نتایج PPMI :

```
print("PPMI Vectors:")
apply_naive_bayes(ppmi_vectors_train, ppmi_vectors_test, train_data['label'], eval_data['label'])
```

[31] ✓ 1.0s

```
PPMI Vectors:
              precision    recall  f1-score   support

           0       0.70      0.78      0.73      1012
           4       0.74      0.65      0.69       988

    accuracy                           0.72      2000
   macro avg       0.72      0.72      0.71      2000
weighted avg       0.72      0.72      0.71      2000
```

بررسی نتایج: دقت روش TF-IDF از TF بیشتر بوده که به خاطر این است که مقدار IDF را در embedding تاثیر داده ایم و اطلاعات بیشتری داریم. به همین ترتیب دقت PPMI از TF-IDF بهتر است که خاطر این است که لیبل را در embedding تاثیر داده ایم که این یعنی اطلاعات بیشتری داشته ایم ولی در TF-IDF به لیبل کاری نداشته‌ایم.

البته ممکن است در شرایطی این برتری ها تغییر کند و مثلا PPMI دقت کمتری داشته باشد که علت آن overfit کردن مدل می‌شود ولی به طور کلی انتظار می‌رود PPMI دقت بهتری داشته باشد. علت اینکه نتایج بهتر نشده این است که بنظرم می‌توانستیم سمپل‌های بیشتری بگیریم چون تعداد feature های ما زیاد است. با خواندن لیبل‌گذاری ها هم حس کردم برخی از آنها درست لیبل نخورده‌اند که می‌توانستیم از لیبل‌گذاری‌های بهتری نیز استفاده کنیم.