04_news_text_classification

September 29, 2021

1 Text classification and sentiment analysis

Once text data has been converted into numerical features using the natural language processing techniques discussed in the previous sections, text classification works just like any other classification task.

In this notebook, we will apply these preprocessing technique to news articles, product reviews, and Twitter data and teach various classifiers to predict discrete news categories, review scores, and sentiment polarity.

1.1 Imports

[3]:
```
%matplotlib inline
import warnings
from collections import Counter, OrderedDict
from pathlib import Path

import numpy as np
import pandas as pd
from pandas.io.json import json_normalize
import pyarrow as pa
import pyarrow.parquet as pq
from fastparquet import ParquetFile
from scipy import sparse
from scipy.spatial.distance import pdist, squareform

# Visualization
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter, ScalarFormatter
import seaborn as sns

# spacy, textblob and nltk for language processing
from textblob import TextBlob, Word

# sklearn for feature extraction & modeling
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score,␣
 ↪confusion_matrix
from sklearn.externals import joblib

import lightgbm as lgb

import json
from time import clock, time
```

[4]:
```
plt.style.use('fivethirtyeight')
warnings.filterwarnings('ignore')
```

1.2 News article classification

We start with an illustration of the Naive Bayes model for news article classification using the BBC articles that we read as before to obtain a DataFrame with 2,225 articles from 5 categories.

1.2.1 Read BBC articles

[5]:
```
path = Path('data', 'bbc')
files = path.glob('**/*.txt')
doc_list = []
for i, file in enumerate(files):
    topic = file.parts[-2]
    article = file.read_text(encoding='latin1').split('\n')
    heading = article[0].strip()
    body = ' '.join([l.strip() for l in article[1:]])
    doc_list.append([topic, heading, body])
```

[6]:
```
docs = pd.DataFrame(doc_list, columns=['topic', 'heading', 'body'])
docs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2225 entries, 0 to 2224
Data columns (total 3 columns):
topic      2225 non-null object
heading    2225 non-null object
body       2225 non-null object
dtypes: object(3)
memory usage: 52.2+ KB
```

1.2.2 Create stratified train-test split

We split the data into the default 75:25 train-test sets, ensuring that the test set classes closely mirror the train set:

```
[7]: y = pd.factorize(docs.topic)[0]
X = docs.body
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1,␣
 ↪stratify=y)
```

1.2.3 Vectorize text data

We proceed to learn the vocabulary from the training set and transforming both dataset using the CountVectorizer with default settings to obtain almost 26,000 features:

```
[8]: vectorizer = CountVectorizer()
X_train_dtm = vectorizer.fit_transform(X_train)
X_test_dtm = vectorizer.transform(X_test)
```

```
[9]: X_train_dtm.shape, X_test_dtm.shape
```

```
[9]: ((1668, 25919), (557, 25919))
```

1.2.4 Train Multi-class Naive Bayes model

```
[10]: nb = MultinomialNB()
nb.fit(X_train_dtm, y_train)
y_pred_class = nb.predict(X_test_dtm)
```

1.2.5 Evaluate Results

We evaluate the multiclass predictions using accuracy to find the default classifier achieved almost 98%:

Accuracy

```
[12]: accuracy_score(y_test, y_pred_class)
```

```
[12]: 0.9766606822262118
```

Confusion matrix

```
[13]: pd.DataFrame(confusion_matrix(y_true=y_test, y_pred=y_pred_class))
```

[13]:

	0	1	2	3	4
0	98	0	0	2	0
1	0	128	0	0	0
2	0	0	102	2	0
3	2	0	5	121	0
4	0	0	1	1	95