# 04_news_text_classification

September 29, 2021

# 1 Classifying news articles with Naive Bayes

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.

## 1.1 Imports

[1]:

```
%matplotlib inline

from pathlib import Path

import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix
```

## 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

[2]:

```
DATA_DIR = Path('..', 'data')
```

[3]:

```
path = DATA_DIR / 'bbc'
files = sorted(list(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])
```

[4]:

```
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):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   topic    2225 non-null   object
 1   heading  2225 non-null   object
 2   body     2225 non-null   object
dtypes: object(3)
memory usage: 52.3+ 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:

[5]:

```
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:

[6]:

```
vectorizer = CountVectorizer()
X_train_dtm = vectorizer.fit_transform(X_train)
X_test_dtm = vectorizer.transform(X_test)
```

[7]:

```
X_train_dtm.shape, X_test_dtm.shape
```

[7]:

```
((1668, 25951), (557, 25951))
```

### 1.2.4 Train Multi-class Naive Bayes model

[8]:

```
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

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

```
[9]: 0.9712746858168761
```

### Confusion matrix

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

```
[10]:      0   1    2    3   4
      0  120   0    6    0   2
      1    0  94    2    0   1
      2    1   0  103    0   0
      3    0   0    1  127   0
      4    0   1    2    0  97
```