Week 6 Tutorial 1 Text Classification using Naive Bayes

August 20, 2022

1 Introduction

Text Classification is one of the widely use from a task of natural language processing (NLP). It is also one of the important and typical task in pattern recognition. Text classification aims to automatically classify text documents into one or more defined categories which can be a web page, library book, media articles, gallery etc. Some examples of text classification can be:

- Understanding audience sentiment from social media,
- Detection of spam and non-spam emails,
- Auto tagging of customer queries, and
- Categorization of news articles into defined topics.

In this tutorial, we would like to demonstrate how we can categorize different new articles into defined topics using sklearn library.

2 Dataset

The dataset used is the 20 Newsgroup dataset (http://qwone.com/~jason/20Newsgroups/). The 20 Newsgroups data set is a collection of about 20000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The dataset was originally collected by Ken Lang. The 20 newsgroups dataset has become a popular data for experiments in text applications of machine learning techniques including text classification and text clustering.

The dataset used in this tutorial is extracted from the 20 Newsgroups dataset sorted by date in which duplicates and some headers removed. The dataset contains 18846 documents. The data is categorized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware), while others are highly unrelated (e.g misc.forsale / soc.religion.christian). Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter:

- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.ibm.pc.hardware
- comp.sys.mac.hardware
- comp.windows.x

- rec.autos
- rec.motorcycles
- rec.sport.baseball

- rec.sport.hockey

- sci.crypt
- sci.electronics
- sci.med
- sci.space
- misc.forsale

- talk.politics.misc
- talk.politics.guns
- talk.politics.mideast
- talk.religion.misc
- alt.atheism
- soc.religion.christian

3 Classification of news articles into different topics using Naive Bayes models

3.1 Retrieving data

[1]:
```
from sklearn.datasets import fetch_20newsgroups
#categories = ['alt.atheism', 'talk.religion.misc','comp.graphics', 'sci.space']
```

[2]:
```
#newsgroups = fetch_20newsgroups(subset='all', categories=categories)
newsgroups = fetch_20newsgroups(subset='all')
```

3.2 Exploring the dataset

[3]:
```
newsgroups.data[1]
```

[3]:
```
'From: mblawson@midway.ecn.uoknor.edu (Matthew B Lawson)\nSubject: Which high-
performance VLB video card?\nSummary: Seek recommendations for VLB video
card\nNntp-Posting-Host: midway.ecn.uoknor.edu\nOrganization: Engineering
Computer Network, University of Oklahoma, Norman, OK, USA\nKeywords: orchid,
stealth, vlb\nLines: 21\n\n  My brother is in the market for a high-performance
video card that supports\nVESA local bus with 1-2MB RAM.  Does anyone have
suggestions/ideas on:\n\n  - Diamond Stealth Pro Local Bus\n\n  - Orchid
Farenheit 1280\n\n  - ATI Graphics Ultra Pro\n\n  - Any other high-performance
VLB card\n\n\nPlease post or email.  Thank you!\n\n  - Matt\n\n-- \n    |
Matthew B. Lawson <------------> (mblawson@essex.ecn.uoknor.edu)  |   \n  --+--
"Now I, Nebuchadnezzar, praise and exalt and glorify the King  --+-- \n    |
of heaven, because everything he does is right and all his ways  |   \n    |
are just." - Nebuchadnezzar, king of Babylon, 562 B.C.           |   \n'
```

[4]:
```
list(newsgroups.target_names)
```

```
[4]: ['alt.atheism',
      'comp.graphics',
      'comp.os.ms-windows.misc',
      'comp.sys.ibm.pc.hardware',
      'comp.sys.mac.hardware',
      'comp.windows.x',
      'misc.forsale',
      'rec.autos',
      'rec.motorcycles',
      'rec.sport.baseball',
      'rec.sport.hockey',
      'sci.crypt',
      'sci.electronics',
      'sci.med',
      'sci.space',
      'soc.religion.christian',
      'talk.politics.guns',
      'talk.politics.mideast',
      'talk.politics.misc',
      'talk.religion.misc']
```

```
[5]: newsgroups.filenames.shape
```

```
[5]: (18846,)
```

```
[6]: newsgroups.target.shape
```

```
[6]: (18846,)
```

3.3 Feature extraction

This step will calculate a bag of words and their frequencies. We use the CountVectorizer from sklearn to do this.

```
[7]: # import the CountVectorizer
     from sklearn.feature_extraction.text import CountVectorizer
```

3.3.1 Let's see an example first

```
[8]: document = [
          'This is the first document.',
          'This document is the second document.',
          'And this is the third one.',
          'Is this the first document?',
      ]
```

```
[9]: cv = CountVectorizer()
```

```
[10]: cv = CountVectorizer()
      cv.fit_transform(document)

      cv.vocabulary_
```

```
[10]: {'this': 8,
       'is': 3,
       'the': 6,
       'first': 2,
       'document': 1,
       'second': 5,
       'and': 0,
       'third': 7,
       'one': 4}
```

```
[11]: cv.fit_transform(document).toarray()
```

```
[11]: array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
             [0, 2, 0, 1, 0, 1, 1, 0, 1],
             [1, 0, 0, 1, 1, 0, 1, 1, 1],
             [0, 1, 1, 1, 0, 0, 1, 0, 1]], dtype=int64)
```

Output a bag of words and their frequencey

```
[12]: word_list = cv.get_feature_names()
      count_list = cv.fit_transform(document).toarray().sum(axis=0)

      print(dict(zip(word_list,count_list)))
```

```
{'and': 1, 'document': 4, 'first': 2, 'is': 4, 'one': 1, 'second': 1, 'the': 4,
'third': 1, 'this': 4}
```

3.3.2 Feature Extraction for the 20NewsGroup dataset

```
[13]: features = cv.fit_transform(newsgroups.data)
```

```
[14]: features.shape
```

```
[14]: (18846, 173762)
```

```
[15]: features.toarray()
```

```
[15]: array([[0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
             …,
             [0, 0, 0, …, 0, 0, 0],
             [0, 0, 0, …, 0, 0, 0],
        [0, 0, 0, …, 0, 0, 0]], dtype=int64)
```

3.4 Split these feature vectors into a trainning and testing sets

[16]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features,
                                                    newsgroups.target,
                                                    train_size = 0.8,
                                                    random_state=0)
```

[17]:
```
X_train.shape
```

[17]:
```
(15076, 173762)
```

[18]:
```
X_test.shape
```

[18]:
```
(3770, 173762)
```

[19]:
```
y_train.shape
```

[19]:
```
(15076,)
```

[20]:
```
y_test.shape
```

[20]:
```
(3770,)
```

3.5 Build a Multinomial Naive Bayes model

[21]:
```
from sklearn.naive_bayes import MultinomialNB
```

[22]:
```
model = MultinomialNB(alpha=1)
```

3.6 Training the model and make a prediction

[23]:
```
model.fit(X_train,y_train)
```

[23]:
```
MultinomialNB(alpha=1)
```

[24]:
```
y_pred = model.predict(X_test)
```

[25]:
```
print(y_test)
print(y_pred)
```

```
[ 2  6 10 … 13 12 13]
[12 13 10 … 13  3 13]
```

3.7 Evaluate the model

[26]:
```
from sklearn import metrics
from sklearn.metrics import classification_report
```

[27]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred, average =␣
 ↪'weighted'))
print("Recall:",metrics.recall_score(y_test, y_pred, average = 'weighted'))
print("F1-score:",metrics.f1_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy: 0.8557029177718833
Precision: 0.8727827934235892
Recall: 0.8557029177718833
F1-score: 0.8444185008277632
```

Classification report

[28]:
```
report = classification_report(y_test, y_pred, target_names = newsgroups.
 ↪target_names)
```

[29]:
```
print(report)
```

```
                          precision    recall  f1-score   support

             alt.atheism       0.86      0.94      0.90       163
           comp.graphics       0.67      0.92      0.77       190
 comp.os.ms-windows.misc       0.98      0.21      0.35       200
comp.sys.ibm.pc.hardware       0.64      0.83      0.72       196
   comp.sys.mac.hardware       0.88      0.86      0.87       201
          comp.windows.x       0.74      0.90      0.81       198
            misc.forsale       0.94      0.71      0.81       206
               rec.autos       0.90      0.94      0.92       177
         rec.motorcycles       0.99      0.90      0.94       189
      rec.sport.baseball       0.94      0.98      0.96       171
        rec.sport.hockey       0.97      0.96      0.97       233
               sci.crypt       0.83      0.98      0.90       190
         sci.electronics       0.88      0.80      0.84       207
                 sci.med       0.95      0.91      0.93       203
               sci.space       0.89      0.95      0.92       191
  soc.religion.christian       0.80      0.95      0.87       198
      talk.politics.guns       0.86      0.96      0.91       155
   talk.politics.mideast       0.92      0.99      0.95       196
      talk.politics.misc       0.89      0.90      0.89       170
      talk.religion.misc       0.94      0.48      0.63       136

                accuracy                           0.86      3770
               macro avg       0.87      0.85      0.84      3770
            weighted avg       0.87      0.86      0.84      3770
```

Confusion matrix

```
[30]: from sklearn.metrics import confusion_matrix

      confusion_matrix(y_test, y_pred)
```

```
[30]: array([[153,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,
                0,   1,   4,   0,   0,   0,   4],
             [  0, 175,   0,   6,   1,   1,   1,   0,   0,   0,   0,   3,   0,
                0,   1,   0,   0,   1,   1,   0],
             [  0,  32,  43,  50,   7,  47,   0,   1,   0,   1,   0,   6,   5,
                1,   3,   1,   1,   0,   2,   0],
             [  0,  10,   0, 163,   9,   4,   3,   0,   0,   0,   1,   4,   1,
                0,   1,   0,   0,   0,   0,   0],
             [  1,   4,   0,   9, 172,   4,   1,   0,   0,   0,   0,   2,   6,
                0,   1,   0,   0,   1,   0,   0],
             [  0,  12,   1,   4,   0, 178,   0,   0,   0,   0,   0,   2,   0,
                0,   1,   0,   0,   0,   0,   0],
             [  0,   8,   0,  11,   3,   3, 147,   8,   1,   2,   0,   6,   7,
                2,   3,   1,   1,   2,   1,   0],
             [  0,   0,   0,   0,   0,   0,   2, 166,   1,   2,   0,   0,   2,
                0,   0,   0,   2,   0,   2,   0],
             [  0,   0,   0,   1,   0,   1,   1,   7, 170,   1,   0,   0,   0,
                0,   2,   1,   1,   1,   3,   0],
             [  0,   1,   0,   0,   0,   0,   0,   0,   0, 167,   2,   0,   0,
                0,   0,   1,   0,   0,   0,   0],
             [  0,   0,   0,   0,   0,   0,   0,   0,   0,   4, 224,   0,   0,
                0,   0,   2,   1,   1,   1,   0],
             [  0,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0, 186,   0,
                0,   0,   0,   0,   0,   2,   0],
             [  0,  13,   0,   8,   4,   0,   1,   1,   0,   0,   1,  10, 165,
                0,   4,   0,   0,   0,   0,   0],
             [  1,   4,   0,   2,   0,   0,   0,   1,   0,   0,   0,   0,   2,
              185,   4,   2,   2,   0,   0,   0],
             [  0,   1,   0,   0,   0,   1,   1,   0,   0,   0,   0,   0,   0,
                1, 182,   2,   0,   2,   1,   0],
             [  2,   0,   0,   1,   0,   0,   0,   0,   0,   0,   0,   0,   0,
                4,   0, 189,   0,   1,   1,   0],
             [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
                0,   0,   1, 149,   2,   2,   0],
             [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,   0,
                0,   0,   1,   0, 194,   0,   0],
             [  0,   1,   0,   0,   0,   0,   0,   0,   0,   1,   0,   3,   0,
                0,   1,   0,   7,   4, 153,   0],
             [ 21,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   1,   0,
                2,   1,  32,   9,   2,   3,  65]], dtype=int64)
```

3.8 Finding the best parameter alpha for the model using GridSearchCV

GridSearchCV is an exhaustive search with cross validation over specified parameter values to determine the optimal value for a given model. The parameters of the model used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

```
[31]: #Import the Grid Search

      from sklearn.model_selection import GridSearchCV
```

```
[32]: model = MultinomialNB()
```

Parameters setting for alpha = 10, 1, 0.1, 0.001, 0.0001

```
[33]: params = {'alpha': [10, 1, 1e-1, 1e-2, 1e-3]}
```

Running the Grid Search on parameters and fit the trainning data

```
[34]: # 10-fold
      #grs = GridSearchCV(model, param_grid=params, cv = 10)

      # 5-fold default
      grs = GridSearchCV(model, param_grid=params)
```

```
[35]: grs.fit(X_train, y_train)
```

```
[35]: GridSearchCV(estimator=MultinomialNB(),
                   param_grid={'alpha': [10, 1, 0.1, 0.01, 0.001]})
```

Output the optimal values

```
[36]: print("Best Hyper Parameters:",grs.best_params_)
```

```
Best Hyper Parameters: {'alpha': 0.001}
```

Make a prediction and calculate metrics

```
[37]: y_pred=grs.predict(X_test)
```

```
[38]: print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
      print("Precision:",metrics.precision_score(y_test, y_pred, average =␣
       ↪'weighted'))
      print("Recall:",metrics.recall_score(y_test, y_pred, average = 'weighted'))
      print("F1-score:",metrics.f1_score(y_test, y_pred, average = 'weighted'))
```

```
Accuracy: 0.8949602122015915
Precision: 0.9011003567458483
Recall: 0.8949602122015915
F1-score: 0.893045682368305
```

```
[ ]:
```