ml-assignment-mnb

February 24, 2024

[1]:
```python
from sklearn.feature_extraction.text import CountVectorizer

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

vectorizer = CountVectorizer(stop_words='english', max_features=1000)
dtm = vectorizer.fit_transform(text)

print(vectorizer.get_feature_names_out())
print(dtm.toarray())
```

```
['document' 'second']
[[1 0]
 [2 1]
 [0 0]
 [1 0]]
```

Question 2: Purpose of CountVectorizer in sklearn

CountVectorizer is a crucial component in natural language processing (NLP) workflows, particularly when dealing with text data. Its primary purpose is to convert raw text into numerical feature vectors that can be used as input to machine learning models. This transformation is necessary because most machine learning algorithms require numerical input to operate effectively.

Text to Numerical Representation: CountVectorizer takes a corpus of text documents (such as sentences, paragraphs, or entire documents) as input and transforms them into a numerical representation. Each document is represented as a vector, where each element of the vector corresponds to the frequency of a particular word in the document.

Document-Term Matrix (DTM): The output of CountVectorizer is often referred to as a Document-Term Matrix (DTM). This matrix has rows corresponding to documents in the corpus and columns corresponding to individual terms (words or tokens). The values in the matrix represent the frequency of each term in each document.

Feature Extraction: CountVectorizer extracts features from text data, where each feature corresponds to a unique term present in the corpus. These features can then be used as input to machine learning models for tasks such as classification, clustering, or regression.

Sparse Matrix Representation: Since text data often contains a large number of unique terms (or vocabulary), the resulting DTM can be very high-dimensional. However, most documents only contain a small subset of the entire vocabulary. CountVectorizer typically represents the DTM as a sparse matrix to efficiently store and manipulate this data, saving memory and computational resources.

Question 3: Explain what is stop_words in CountVectorizer. (10 pts)

Removal of Common Words: Stop words are common words that often appear frequently in text but typically do not carry much meaning or contribute to the overall context. Examples include articles ("the", "a", "an"), prepositions ("in", "on", "at"), and conjunctions ("and", "but", "or"). By removing these words before vectorization, we can focus on the more important words that are more indicative of the content of the text.

Improvement of Model Performance: Removing stop words can lead to more meaningful and informative features, which can improve the performance of machine learning models. This is because stop words can introduce noise into the data and may not contribute significantly to the task at hand.

Customization: CountVectorizer allows users to specify their own list of stop words or use built-in lists for common languages such as English. This flexibility enables users to tailor the vectorization process to their specific needs and domain of application.

In summary, CountVectorizer plays a crucial role in converting text data into a numerical format suitable for machine learning models. Additionally, the ability to remove stop words further enhances the quality of the resulting feature representation. Exercise 2

[29]:
```python
#Exercise 2
#import the required libraries
import pandas as pd

# Load the datasets
goodware_df = pd.read_csv('goodware_r.csv')
malware_df = pd.read_csv('malware_r.csv')

# Preview the first few rows of each dataset to understand their structure
goodware_df_head = goodware_df.head()
malware_df_head = malware_df.head()

goodware_df_head, malware_df_head
```

```
C:\Users\kusum\AppData\Local\Temp\ipykernel_5028\243875899.py:4: DtypeWarning:
Columns (0,1,2,3,4,5,8,12,13,15,16,17,18,19,21,22,23,24,25,26,27,28,29,30,31,32,
33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59
,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,8
6,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109
,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129
,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149
,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169
```

```
,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189
,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209
,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229
,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249
,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,269
,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289
,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,305,306,307,308,309
,310,311,312,313,314,315,316,317,318,319,320,321,322,323,324,325,326,327,328,329
,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,345,346,347,348,349
,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,365,366,367,368,369
,370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385,386,387,388,389
,390,391,392,393,394,395,396,397,398,399,400,401,402,403,404,405,406,407,408,409
,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,425,426,427,428,429
,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449
,450,451,452,453,454,455,456,457,458,459,460,461,462,463,464,465,466,467,468,469
,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,485,486,487,488,489
,490,491,492,493,494,495,496,497,498,499,500,501,502,503,504,505,506,507,508,509
,510,511,512,513,514,515,516,517,518,519,520,521,522,523,524,525,526,527,528,529
,530,531,532,533,534,535,536,537,538,539,540,541,542,543,544,545,546,547,548,549
,550,551,552,553,554,555,556,557,558,559,560,561,562,563,564,565,566,567,568,569
,570,571,572,573,574,575,576,577,578,579,580,581,582,583,584,585,586,587,588,589
,590,591,592,593,594,595,596,597,600,606) have mixed types. Specify dtype option
on import or set low_memory=False.
  goodware_df = pd.read_csv('goodware_r.csv')
```

[29]:
```
(  BaseOfCode BaseOfData Characteristics DllCharacteristics   Entropy  \
 0       4096      40960             783              32768  7.999997
 1       4096      28672             271              32768  7.870771
 2       4096     131072             303                  0   7.99977
 3       4096    1646592             259                  0  5.590701
 4       8192     786432             258              34112  6.812076

   FileAlignment FormatedTimeDateStamp  \
 0           512       2/18/2016 20:08
 1           512       12/5/2009 20:50
 2           512       4/18/2011 15:54
 3          4096       3/28/2011 11:59
 4           512       12/22/2015 9:04

                                           Identify ImageBase  \
 0                                              NaN   4194304
 1                      [['Nullsoft PiMP Stub -> SFX']]   4194304
 2  [['Microsoft Visual C++ v6.0'], ['Microsoft Vi…   4194304
 3  [['Microsoft Visual C++ 8'], ['VC8 -> Microsof…   4194304
 4  [['Microsoft Visual C# / Basic .NET'], ['Micro…   4194304

                                       ImportedDlls  … Unnamed: 605  \
```

```
0  ['ADVAPI32.dll', 'COMCTL32.DLL', 'GDI32.dll', …  …          NaN
1  ['KERNEL32.dll', 'USER32.dll', 'GDI32.dll', 'S…  …          NaN
2  ['OLEAUT32.dll', 'ole32.dll', 'USER32.dll', 'S…  …          NaN
3  ['VERSION.dll', 'COMCTL32.dll', 'WINMM.dll', '…  …          NaN
4                                    ['mscoree.dll']  …          NaN

  Unnamed: 606 Unnamed: 607 Unnamed: 608 Unnamed: 609 Unnamed: 610  \
0          NaN          NaN          NaN          NaN          NaN
1          NaN          NaN          NaN          NaN          NaN
2          NaN          NaN          NaN          NaN          NaN
3          NaN          NaN          NaN          NaN          NaN
4          NaN          NaN          NaN          NaN          NaN

  Unnamed: 611 Unnamed: 612 Unnamed: 613 Unnamed: 614
0          NaN          NaN          NaN          NaN
1          NaN          NaN          NaN          NaN
2          NaN          NaN          NaN          NaN
3          NaN          NaN          NaN          NaN
4          NaN          NaN          NaN          NaN

[5 rows x 615 columns],
  BaseOfCode  BaseOfData  Characteristics  DllCharacteristics   Entropy  \
0       4096       69632              783                   0  5.981249
1       4096     1851392              783                   0  6.081747
2       4096       40960              783                   0  5.586422
3    1359872     2138112              783                   0  7.969464
4       4096       40960              783               32768  7.999900

   FileAlignment FirstSeenDate  \
0            512      1/1/1970
1            512      1/1/1970
2            512      1/1/1970
3            512      1/1/1970
4            512      1/1/1970

                                          Identify  ImageBase  \
0                               powerbasic/win 8.00    4194304
1                                               NaN    4194304
2                                               NaN    4194304
3  upx 2.93 - 3.00 [lzma] -> markus oberhumer, la…    4194304
4                                               NaN    4194304

                                       ImportedDlls  …  \
0  comdlg32.dll gdi32.dll kernel32.dll ole32.dll …  …
1  comctl32.dll comdlg32.dll gdi32.dll kernel32.d…  …
2  comdlg32.dll kernel32.dll msvcrt.dll msvcrt.dl…  …
3  kernel32.dll advapi32.dll comdlg32.dll gdi32.d…  …
```

```
4  advapi32.dll comctl32.dll gdi32.dll kernel32.d…  …

  PointerToSymbolTable                                      SHA1        Size  \
0                   0.0  b0068836a40e6a43c6b546fcb709237e5aa223d1     76288.0
1                   0.0  5741708cd785f13b44267883e3f2fd2fa51fc23f   2558464.0
2                   0.0  507fe5d8244f33d29d427468efca4ce406f23666    178688.0
3                   0.0  e51a7811464be1acadf6e72ba3a66aba0da438cd    806816.0
4                   0.0  0e046d9903c313ffeeb0d6392335437fe881b1f5  50689096.0

   SizeOfCode  SizeOfHeaders  SizeOfImage  SizeOfInitializedData  \
0     64855.0         1024.0      86016.0                 2560.0
1   1843888.0         1024.0    2600960.0               500348.0
2     33792.0         1024.0   33759232.0               177664.0
3    778240.0         4096.0    2166784.0                28672.0
4     35840.0         1024.0     303104.0                38912.0

   SizeOfOptionalHeader  SizeOfUninitializedData TimeDateStamp
0                 224.0                   1500.0       12345.0
1                 224.0                  21476.0           0.0
2                 224.0               33557504.0           0.0
3                 224.0                1355776.0           0.0
4                 224.0                 110080.0           0.0

[5 rows x 28 columns])
```

[44]:

```python
import os
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
```

[47]:

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import csv

#Define the parameter to store the path for the script to read data. And define␣
↪the parameters to store the labels and text to be vectorized.
goodware_path = 'goodware_r.csv'
malware_path = 'malware_r.csv'

# Lists to store labels and text data
labels = []
text = []

# List of filenames to process
filenames = [goodware_path, malware_path]

#Read the content from each file and create labels for them
for filename in filenames:
```

```
    # Determine if the file is goodware or malware
    label = "1" if "good" in filename else "-1"

    # Open and process the file
    with open(filename, encoding="utf8") as f:
        content = csv.reader(f, delimiter="\t")
        next(content)  # Skip the header
        for line in content:
            # Convert line to string and clean it
            line_str = ' '.join(line).replace(',', ' ').replace('"', ' ')
            # Append the processed line and label to the lists
            text.append(line_str)
            labels.append(label)

# Initialize CountVectorizer with the specified parameters
vectorizer = CountVectorizer(stop_words='english', max_features=1000)

# Apply CountVectorizer to the text data
dtm = vectorizer.fit_transform(text)

# Convert the document-term matrix to a pandas DataFrame
df = pd.DataFrame(dtm.toarray(), index=labels, columns=vectorizer.
 ↪get_feature_names_out())
df.index.name = "labels"

# Save the dataframe to a CSV file
df.to_csv('MalwareMatrix.csv')
```

Question 4: Randomly split data into training (70%) and test set (30%), and then apply multinomial Naive Bayes classifier (Use functions from Scikit library [Link]). (20 pts)

[52]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB


# Load the MalwareMatrix.csv file into a DataFrame
df = pd.read_csv('MalwareMatrix.csv')

# Extract labels and features
labels = df['labels']
features = df.drop('labels', axis=1)


# Split data into training (70%) and test set (30%)
X_train, X_test, y_train, y_test = train_test_split(features,␣
 ↪labels,test_size=0.3, random_state=42)
# Apply Multinomial Naive Bayes classifier
```

```
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)
```

[52]: MultinomialNB()

[54]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
# Predictions on the training set
y_train_pred = nb_classifier.predict(X_train)
# Predictions on the test set
y_test_pred = nb_classifier.predict(X_test)

# Evaluate the classifier on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_classification_rep = classification_report(y_train, y_train_pred)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)

# Evaluate the classifier on the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_classification_rep = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

# Display results for the training set
print("Training Set Results:")
print(f"Classification Accuracy: {train_accuracy:.2f}")
print("Training Classification Report:")
print(train_classification_rep)
print("Training Confusion Matrix:")
print(train_conf_matrix)

# Display results for the test set
print("\nTesting Set Results:")
print(f"Classification Accuracy: {test_accuracy:.2f}")
print("Testing Classification Report:")
print(test_classification_rep)
print("Testing Confusion Matrix:")
print(test_conf_matrix)
```

```
Training Set Results:
Classification Accuracy: 0.71
Training Classification Report:
              precision    recall  f1-score   support

          -1       0.73      0.75      0.74      7091
           1       0.69      0.67      0.68      5952

    accuracy                           0.71     13043
```

```
   macro avg       0.71      0.71      0.71     13043
weighted avg       0.71      0.71      0.71     13043

Training Confusion Matrix:
[[5331 1760]
 [1963 3989]]

Testing Set Results:
Classification Accuracy: 0.70
Testing Classification Report:
              precision    recall  f1-score   support

          -1       0.72      0.74      0.73      3039
           1       0.68      0.66      0.67      2552

    accuracy                           0.70      5591
   macro avg       0.70      0.70      0.70      5591
weighted avg       0.70      0.70      0.70      5591

Testing Confusion Matrix:
[[2242  797]
 [ 862 1690]]
```

[ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Load the MalwareMatrix.csv file into a DataFrame
df = pd.read_csv('MalwareMatrix.csv')

# Extract labels and features
labels = df['labels']
features = df.drop('labels', axis=1)

# Split data into training (70%) and test set (30%)
X_train, X_test, y_train, y_test = train_test_split(features, labels,␣
 ↪test_size=0.3, random_state=42)

# Apply Multinomial Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

# Predictions on the training set
y_train_pred = nb_classifier.predict(X_train)
# Predictions on the test set
y_test_pred = nb_classifier.predict(X_test)
```

```python
# Evaluate the classifier on the training set
train_accuracy = accuracy_score(y_train, y_train_pred)
train_classification_rep = classification_report(y_train, y_train_pred)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)
# Evaluate the classifier on the test set
test_accuracy = accuracy_score(y_test, y_test_pred)
test_classification_rep = classification_report(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)
# Display results for the training set
print("Training Set Results:")
print(f"Classification Accuracy: {train_accuracy:.2f}")
print("Training Classification Report:")
print(train_classification_rep)
print("Training Confusion Matrix:")
print(train_conf_matrix)
# Display results for the test set
print("\nTesting Set Results:")
print(f"Classification Accuracy: {test_accuracy:.2f}")
print("Testing Classification Report:")
print(test_classification_rep)
print("Testing Confusion Matrix:")
print(test_conf_matrix)
```

[ ]:

```python

```