Spam Classifier with Naive Bayes

September 11, 2021

[26]:
```
from pandas import DataFrame
import os
import io
import numpy
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import  MultinomialNB
```

[30]:
```
def readFiles(path):
    # using the os.walk function to find all of the files in a directory, builds up the full path name for each individual file in that directory, and then it reads it in.

    for root, dirnames, filenames in os.walk(path):
        for filename in filenames:
            path = os.path.join(root, filename)
            isMessageBody = False
            lines = []
            # While it's reading it in, it actually skips the header for each email and just goes straight to the text,
            # and it does that by looking for the first blank line here line=="\n". It knows that everything after the first empty line
            # is actually the message body and everything in front of that first empty line is just a bunch of header information
            # that I don't actually want to train my spam classifier on.
            f = io.open(path, 'r', encoding='latin1')
            for line in f:
                if isMessageBody:
                    lines.append(line)
                elif line == '\n':
                    isMessageBody = True
            f.close()
            message = '\n'.join(lines)
            #So it gives me back both the full path to each file and the body of the message.
            yield path, message
```

[31]:
```
#DataFrameFromDirectory is a function I wrote up here. Basically it says I have␣
 ↪a path to a directory and I know it's a given classification, spam or ham,
# and what I'm gonna do is call this readFiles function that I also wrote that␣
 ↪will iterate through every single file in a directory and
# gives me back both the full path to each file and the body of the message..
def readDataFromDirectory(path,classification):
    rows = []
    index = []
    for filename, message in readFiles(path):
        rows.append({'message': message, 'class': classification})
        index.append(filename)

    return DataFrame(rows, index=index)
```

[31]:

[32]:
```
#So what I have at the end of the day here is a data frame object, basically a␣
 ↪database with two columns that contains body, message bodies, and whether␣
 ↪it's spam or not.
data = DataFrame({'message': [], 'class': []})

data = data.append(readDataFromDirectory('emails/spam', 'spam'))
data = data.append(readDataFromDirectory('emails/ham', 'ham'))
```

[33]:
```
"""So the first few entries in our data frame look like this for each path to a␣
 ↪given file full of emails.
We have a classification and we have the message body.
"""

data.head()
```

[33]:
```
                                                    message  \
emails/spam/00249.5f45607c1bffe89f60ba1ec9f878039a  Dear Homeowner,\n\n
\n\nInterest Rates are at …
emails/spam/00373.ebe8670ac56b04125c25100a36ab0510  ATTENTION: This is a MUST
for ALL Computer Use…
emails/spam/00214.1367039e50dc6b7adb0f2aa8aba83216  This is a multi-part message
in MIME format.\n…
emails/spam/00210.050ffd105bd4e006771ee63cabc59978  IMPORTANT
INFORMATION:\n\n\n\nThe new domain n…
emails/spam/00033.9babb58d9298daa2963d4f514193d7d6  This is the bottom line.  If
you can GIVE AWAY…

                                                   class
emails/spam/00249.5f45607c1bffe89f60ba1ec9f878039a  spam
emails/spam/00373.ebe8670ac56b04125c25100a36ab0510  spam
emails/spam/00214.1367039e50dc6b7adb0f2aa8aba83216  spam
```

```
emails/spam/00210.050ffd105bd4e006771ee63cabc59978  spam
emails/spam/00033.9babb58d9298daa2963d4f514193d7d6  spam
```

[34]:
```
"""
So, we're going to use the MultinomialNB function from Scikit-learn to actually␣
 ↪perform Naive Bayes
on this data that we have.

What that is is basically a list of all the words in each email and the number␣
 ↪of times that word occurs.
So that's what this CountVectorizer thing does.

"""
vectorizer = CountVectorizer()

'''
data["message"].values-->This syntax means take the message column from my data␣
 ↪frame and take all the values from it,
and I'm gonna call vectorizer.fit_transform.

What that does is it basically tokenizes or converts all of the individual␣
 ↪words seen in my data into numbers, into values, and it will then count up␣
 ↪how many times
each word occurs.

So this is a more compact way of representing how many times each word occurs␣
 ↪in an email.

Instead of actually preserving the words themselves, I'm representing those␣
 ↪words as different values
in a sparse matrix, which is basically saying that I'm treating each word as a␣
 ↪number, as a numerical index into an array.


So what that does is it just, in plain English, it splits each message up into␣
 ↪a list of words
that are in it and how many times each word occurs. So we're calling that␣
 ↪counts.

It's basically that information of how many times each word occurs in each␣
 ↪individual message,

'''
counts = vectorizer.fit_transform(data['message'].values)

'''
```

```
targets is the actual classification data for each email that I've encountered.
'''
targets = data['class'].values

'''
So once we build a multinomial Naive Bayes classifier it needs two inputs.
It needs the actual data that we're training on and the targets for each thing.

What that is is basically a list of all the words in each email and the number␣
 ↪of times that word occurs.

And I can call classifier.fit using my MultinomialNB function
to actually create a model using Naive Bayes that will predict whether new␣
 ↪emails are spam or not
based on the information I gave it.
'''
classifier = MultinomialNB()
classifier.fit(counts, targets)
```

[34]:
```
MultinomialNB()
```

[35]:
```
#manually creating example
examples = ['Free Viagra now!!!', "Hi Bob, how about a game of golf tomorrow?"]

'''
First thing we need to do is convert these messages into the same format that I␣
 ↪train my model on.

So I'm gonna use that same vectorizer that I created when creating the model
to convert each message into a list of words and their frequencies where the␣
 ↪words are represented by
positions in an array.
'''
example_counts = vectorizer.transform(examples)

'''
Then once I've done that transformation, I can actually use the predict␣
 ↪function on my classifier
on that array of examples that have transformed into lists of words and see␣
 ↪what we come up with.
'''


predictions = classifier.predict(example_counts)
'''
So given this array of two input message,free Viagra now and hi Bob,
```

```
it's telling me that the first result came back as spam and the second result␣
 ↪came back as ham.
Which is what I would expect
'''
predictions
```

[35]:
```
array(['spam', 'ham'], dtype='<U4')
```

[ ]:
```

```