HW#9 - Email Classification

PRASHANT TOMAR

04/21/2023

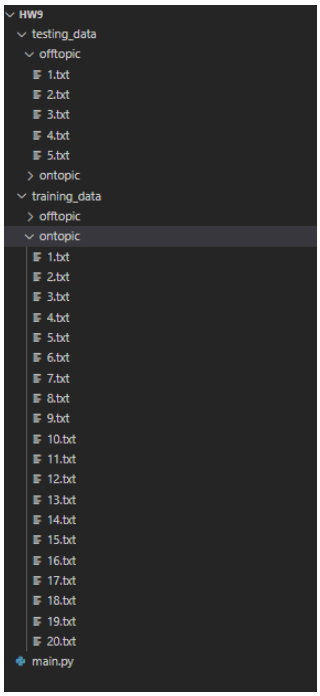
Q1 - Create two datasets, Testing and Training.

Answer

In the root folder of the source code, I have established two main directories: one for training data and the other for testing data. Each of these directories contains two more subdirectories named ”on topic” and ”off topic”, which correspond to our email dataset. To create the on-topic email dataset, I utilized my official email address from Old Dominion University, which includes the university’s newsletters and seminar invitations. For the off-topic dataset, I utilized my personal email account, where I receive various advertisements such as new movie releases and release notes . For training, I have produced at least 20 on-topic and 20 off-topic email datasets.

The same process was followed for the testing dataset, but with only 5 on-topic and 5 off-topic datasets. By doing so, we will have a total of 10 training datasets, making it easier to evaluate the results of our classification model.

Figure 1: Folder Structure



Q2 - Naive Bayes classifier

Answer

The text documents were classified using the code for classification from the book Programming Collective Intelligence. To do this, the files from the training folder were read and a list called list_train_data was created. This list contained items with a text property and a class property. The email text was added to the text property, and the class property was set to either ”on topic” or ”off topic”.

After generating a complete list of 40 items, the sample train function from the code was used to train all the items by iterating over the list and sending each item to the training function.

Once the model was trained, the classify function was used to classify the test data by sending it into the function. The function returned the classification based on the trained model classification. Below is the complete implementation.

```
import sqlite3 as sqlite
import re
import math
import os


def getwords(doc):
    splitter = re.compile('\W+')
    words = [s.lower() for s in splitter.split(doc)
             if len(s) > 2 and len(s) < 20]
    uniq_words = dict([(w, 1) for w in words])
    return uniq_words


class basic_classifier:

    def __init__(self, getfeatures, filename=None):
        self.fc = {}
        self.cc = {}
        self.getfeatures = getfeatures

    def incf(self, f, cat):
        self.fc.setdefault(f, {})
        self.fc[f].setdefault(cat, 0)
        self.fc[f][cat] += 1

    def incc(self, cat):
        self.cc.setdefault(cat, 0)
```

```
29          self.cc[cat] += 1
30
31      def fcount(self, f, cat):
32          if f in self.fc and cat in self.fc[f]:
33              return float(self.fc[f][cat])
34          return 0.0
35
36      def catcount(self, cat):
37          if cat in self.cc:
38              return float(self.cc[cat])
39          return 0
40
41      def totalcount(self):
42          return sum(self.cc.values())
43
44      def categories(self):
45          return self.cc.keys()
46
47      def train(self, item, cat):
48          features = self.getfeatures(item)
49          for f in features:
50              self.incf(f, cat)
51          self.incc(cat)
52
53      def fprob(self, f, cat):
54          if self.catcount(cat) == 0:
55              return 0
56          return self.fcount(f, cat)/self.catcount(cat)
57
58      def weightedprob(self, f, cat, prf, weight=1.0, ap=0.5):
59          basicprob = prf(f, cat)
60          totals = sum([self.fcount(f, c) for c in self.categories()])
61          bp = ((weight*ap)+(totals*basicprob))/(weight+totals)
62          return bp
63
64
65  class naivebayes(basic_classifier):
66
67      def __init__(self, getfeatures):
68          basic_classifier.__init__(self, getfeatures)
69          self.thresholds = {}
70
71      def docprob(self, item, cat):
72          features = self.getfeatures(item)
73          p = 1
74          for f in features:
75              p *= self.weightedprob(f, cat, self.fprob)
```

```
        return p

    def prob(self, item, cat):
        catprob = self.catcount(cat)/self.totalcount()
        docprob = self.docprob(item, cat)
        return docprob*catprob

    def setthreshold(self, cat, t):
        self.thresholds[cat] = t

    def getthreshold(self, cat):
        if cat not in self.thresholds:
            return 1.0
        return self.thresholds[cat]

    def classify(self, item, default=None):
        probs = {}
        max = 0.0
        for cat in self.categories():
            probs[cat] = self.prob(item, cat)
            if probs[cat] > max:
                max = probs[cat]
                best = cat
        for cat in probs:
            if cat == best:
                continue
            if probs[cat]*self.getthreshold(best) > probs[best]:
                return default
        return best


all_training_ontopic_files = os.listdir('training_data\ontopic')
all_training_offtopic_files = os.listdir('training_data\offtopic')
all_testing_ontopic_files = os.listdir('testing_data\ontopic')
all_testing_offtopic_files = os.listdir('testing_data\offtopic')

list_train_data = []

for item in all_training_ontopic_files:
    fd = open('training_data\ontopic\\' + item, 'r', encoding="utf8")
    text = fd.read()
    train_data = {'text': text, 'class': 'on topic'}
    list_train_data.append(train_data)

for item in all_training_offtopic_files:
    fd = open('training_data\offtopic\\' + item, 'r', encoding="utf8")
    text = fd.read()
```

```
123     train_data = {'text': text, 'class': 'off topic'}
124     list_train_data.append(train_data)
125
126
127 def sampletrain(cl):
128     for item in list_train_data:
129         cl.train(item['text'], item['class'])
130
131
132 cl = naivebayes(getwords)
133 sampletrain(cl)
134
135 list_testing_ontopic_data = []
136 for item in all_testing_ontopic_files:
137     fd = open('testing_data\ontopic\\' + item, 'r', encoding="utf8")
138     text = fd.read()
139     train_data = {'text': text, 'class': 'on topic'}
140     list_testing_ontopic_data.append(train_data)
141
142 list_testing_offtopic_data = []
143 for item in all_testing_offtopic_files:
144     fd = open('testing_data\offtopic\\' + item, 'r', encoding="utf8")
145     text = fd.read()
146     train_data = {'text': text, 'class': 'off topic'}
147     list_testing_offtopic_data.append(train_data)
148
149 print(cl.classify(list_testing_ontopic_data[0]['text'], default='
    unknown'))
150 print(cl.classify(list_testing_ontopic_data[1]['text'], default='
    unknown'))
151 print(cl.classify(list_testing_ontopic_data[2]['text'], default='
    unknown'))
152 print(cl.classify(list_testing_ontopic_data[3]['text'], default='
    unknown'))
153 print(cl.classify(list_testing_ontopic_data[4]['text'], default='
    unknown'))
154
155 print(cl.classify(list_testing_offtopic_data[0]['text'], default='
    unknown'))
156 print(cl.classify(list_testing_offtopic_data[1]['text'], default='
    unknown'))
157 print(cl.classify(list_testing_offtopic_data[2]['text'], default='
    unknown'))
158 print(cl.classify(list_testing_offtopic_data[3]['text'], default='
    unknown'))
159 print(cl.classify(list_testing_offtopic_data[4]['text'], default='
```

```
    unknown'))
```

Listing 1: Email Classification Using Naive Bayes Classifier

After running the code, the output was displayed in the terminal window. The results were accurate, and the classifier was able to correctly classify all the data. You can find the results in the table on the next page.

Table 1: Classification of the Email

Email	Actual Result	Predicted Result
1	On Topic	On Topic
2	On Topic	On Topic
3	On Topic	On Topic
4	On Topic	On Topic
5	On Topic	On Topic
6	Off Topic	Off Topic
7	Off Topic	Off Topic
8	Off Topic	Off Topic
9	Off Topic	Off Topic
10	Off Topic	Off Topic

Q3 - Confusion Matrix

Answer

As evident from the Q2 results, I have obtained a 100% accuracy rate. Please refer to the confusion matrix presented below.

Table 2: Example Confusion Matrix from Wikipedia

		Actual	
		On Topic	Off Topic
Predicted	On Topic	5 (TP)	0 (FP)
	Off Topic	0 (FN)	5 (TN)

Upon examining the confusion matrix, it is apparent that the Bayes classifier performed exceptionally well and achieved 100% accuracy. However, it was expected that there would be some variation in the results as I had included off-topic data from my personal email address in the training dataset as mentioned in Q1. Specifically, I had received a newsletter from a different university during the time I was applying to Old Dominion University, and I was hoping that this off-topic email would be classified as on-topic.

References

- Code Ref from Chap 6, `https://learning.oreilly.com/library/view/programming-collective-intelligence/9780596529321/`