lesson6_visualization

April 27, 2018

```
In [1]: # Run this every time you open the spreadsheet
        %load_ext autoreload
        %autoreload 2
        from collections import Counter
        import lib
        import random
```

0.1 Optional Exercise: Add bigram capabilities to the classifier!

So far our Naive Bayes classifier scores an Average F1 score of 66.9% on the test set. Let's see if we can improve on that by incorporating bigrams!

```
In [2]: def add_bigrams(tweet):
        # Currently, tweet has an attribute called tweet.tokenList which is a list of tokens.
        # You want to add a new attribute to tweet called tweet.bigramList which is a list of 
        # Each bigram should be a pair of strings. You can define the bigram like this: bigram
        # In Python, this is called a tuple. You can read more about tuples here: https://www.

        ##### YOUR CODE STARTS HERE #####

            tweet.bigramList = [(tweet.tokenList[i], tweet.tokenList[i + 1]) for i in range(le

        ##### YOUR CODE ENDS HERE #####


        tweets, test_tweets = lib.read_data()
        for tweet in tweets + test_tweets:
            add_bigrams(tweet)
        print("Checking if bigrams are correct...")
        for tweet in tweets + test_tweets:
            assert tweet._bigramList == tweet.bigramList, "Error in your implementation of the
        print("Bigrams are correct.\n")

        prior_probs, token_probs = lib.learn_nb(tweets)
        predictions = [(tweet, lib.classify_nb(tweet, prior_probs, token_probs)) for tweet in 
        lib.evaluate(predictions)
```

```
Checking if bigrams are correct...
Bigrams are correct.
```

```
Energy
Precision:  60.0
Recall:  67.5
F1:  63.529411764705884

Food
Precision:  84.39716312056737
Recall:  92.24806201550388
F1:  88.14814814814815

Medical
Precision:  75.0
Recall:  46.15384615384615
F1:  57.14285714285714

None
Precision:  82.66666666666667
Recall:  78.48101265822785
F1:  80.51948051948052

Water
Precision:  83.33333333333333
Recall:  50.0
F1:  62.5

Average F1:  70.36797951503834
```

0.2 Re-run the classifier and get evaluation score

This notebook uses our implementation of the Naive Bayes classifier, but it's very similar to what you implemented yesterday. If you're interested in the details, take a look at the `learn_nb` and `classify_nb` functions in `lib.py` in the `sailors2017` directory.

```
In [3]: tweets, test_tweets = lib.read_data()
        prior_probs, token_probs = lib.learn_nb(tweets)
        predictions = [(tweet, lib.classify_nb(tweet, prior_probs, token_probs)) for tweet in t
        lib.evaluate(predictions)
```

```
Energy
Precision:  60.0
Recall:  67.5
F1:  63.529411764705884

Food
Precision:  84.39716312056737
Recall:  92.24806201550388
```

```
F1:  88.14814814814815

Medical
Precision:  75.0
Recall:  46.15384615384615
F1:  57.14285714285714

None
Precision:  82.66666666666667
Recall:  78.48101265822785
F1:  80.51948051948052

Water
Precision:  83.33333333333333
Recall:  50.0
F1:  62.5

Average F1:  70.36797951503834
```

0.3 Inspecting the Classifier

After implementing and training a classifier, you often want to inspect what kind of things it has learned and how it is making predictions on individual examples. This can help you make sure that you implemented everything correctly and it might give you ideas on how to further improve the classifier.

0.3.1 Most discriminative words

Let's first look again at the most discriminative words for each category, i.e. the words that maximize P(category | word), for each category.

```
In [4]: lib.most_discriminative(tweets, token_probs, prior_probs)
```

```
MOST DISCRIMINATIVE TOKENS:

TOKEN                P(Energy|token)
dark                 0.8029
powers               0.8029
generator            0.7654
batteries            0.7559
class                0.7534
sandysucks           0.7534
flashlights          0.7345
masks                0.7334
11/3                 0.6736
cleaner              0.6707

TOKEN                P(Food|token)
```

canned	0.9784
non-perishable	0.9767
serve	0.9663
perishable	0.9562
cook	0.9511
soup	0.9489
sandwiches	0.9489
rice	0.9441
thanksgiving	0.9441
meal	0.9383

TOKEN	P(Medical|token)
meds	0.8229
aid	0.8008
ups	0.7360
medications	0.7360
prescription	0.7360
4t-5t	0.7360
ointment	0.7360
medicine	0.7360
kits	0.6596
pull	0.6596

TOKEN	P(None|token)
..	0.9531
everyone	0.8955
last	0.8809
feel	0.8809
im	0.8618
irene	0.8604
...	0.8601
tropical	0.8314
halloween	0.8314
finally	0.8314

TOKEN	P(Water|token)
bottled	0.9059
gallon	0.8307
jugs	0.7970
water	0.7873
gallons	0.7266
flood	0.6625
pallets	0.6625
spring	0.6625
feet	0.6625
parks	0.6625

These five lists show you which words are most predictive of the five categories. For example, the word *bottled* is a very strong indicator that the tweet is about water or the word *canned* is a very strong indicator that the tweet is about food.

Many of you used several of these words in your rule-based classifiers in week 1. It's reassuring (and exciting!) to see that the Naive Bayes classifier learned that these words are good indicators of the categories as well.

0.3.2 Confusion matrix

Another useful type of visualization is a so-called confusion matrix. A confusion matrix shows you for each true category c how many of the tweets in c were classified into the five different categories. (In this way it tells you which categories are "confused" for others by the classifier).

```
In [5]: lib.show_confusion_matrix(predictions)
```

```
<IPython.core.display.HTML object>
```

In the matrix, the **rows** correspond to the **true category** and the **columns** correspond to the **predicted category**.

For example, this matrix shows you that of all the 79 tweets in the category *None*, 13 were incorrectly classified as *Energy*, 3 as *Food*, and 1 as *Medical*. 62 of them were actually correctly classified as *None*.

0.3.3 Visualizing individual tweets

It can also be really useful to visualize the probabilities of each token in an individual tweet. This can help you understand why a classifier made a correct or wrong prediction. We've implemented a visualization for you so that you can use to inspect how the classifier works on individual tweets.

```
In [6]: # The following code visualizes a random tweet from the test data.
        # Hover your mouse over the words!

        random_tweet = random.choice(test_tweets)
        lib.visualize_tweet(random_tweet, prior_probs, token_probs)
```

```
<IPython.core.display.HTML object>
```

The color of each word tells you for which category $P(\text{token} \mid \text{category})$ is the highest. When you move the mouse over a word, it shows you the actual values of $P(\text{token} \mid \text{category})$ for each category that the classifier uses to make its predictions.

You can also have the classifier make a prediction on your own tweets. Change the text in `my_tweet` below and run the cell below to see what the classifier would predict.

```
In [7]: my_tweet = "I urgently need some bottled water."

        lib.visualize_tweet(lib.Tweet(my_tweet, "?", ""), prior_probs, token_probs)
```

```
<IPython.core.display.HTML object>
```

0.4 Error analysis: Figuring out remaining errors

Often, one wants to know in which scenarios a classifier makes mistakes. This can be really useful when you want to improve your classifier.

In this exercise, try to break the Naive Bayes classifier. Use the cell above and try to come up with a tweet which should be classified as *Food* but which is assigned a different category. Once you find such a tweet, use the visualization to figure out why the classifier gets this example wrong.

Repeat this exercise for all the other categories. Based on your observations, do you have any ideas on how to further improve the classifier?

```
In [10]: my_tweet = "Milks are the batteries powering the life."  # Food but assigned to energ

         lib.visualize_tweet(lib.Tweet(my_tweet, "?", ""), prior_probs, token_probs)
```

```
<IPython.core.display.HTML object>
```