HW#7 - Email Classification

Ethan Landers

Due: Sunday, November 24, 2024 by 11:59 PM

Q1

Q: What topic did you decide to classify on?

I decided to classify my emails on whether they were from Old Dominion University (on-topic) or not (off-topic).

To create the dataset, I wrote a script called q1.py that generated the necessary folder structure and blank files for organizing the email content. The base folder, *emails_dataset*, contains two sub folders: *testing* and *on_topic*. These sub folders are further divided into *off_topic* and *on_topic* categories.

For the *training* dataset, I generated 20 blank files for each category (*on_topic* and *off_topic*), and for the *testing* dataset, I generated five blank files for each category. I used a consistent naming scheme to ensure clarity and easy identification of each file based on its dataset type and category. Below is an excerpt of the code showing how the file paths were structured:

```
...

file_path = os.path.join(category_folder,
f"{dataset_type}_{category}_{i}.txt")

...

base_folder = "emails_dataset"
dataset_types = ["training", "testing"]
categories = ["on_topic", "off_topic"]
```

This organization allowed me to systematically populate the files with the content of my emails, ensuring a balanced and well-structured dataset for classification.

Q2

The goal of Q2 was to classify emails as ”on_topic” or ”off_topic” using a Naive Bayes classifier. To achieve this, I implemented a script, q2.py, which performs training, testing, and evaluation of the classifier.

This implementation is based on code provided in the Google Colab notebook for this assignment, which can be found here. Specifically, the notebook provided the foundational functions for feature extraction and probability calculations.

One key component of the classifier is the getwords() function, adapted from the notebook. This function processes the text of each email by:

- Splitting it into words.
- Filtering out words that are too short or long.
- Creating a dictionary of unique words from the email.

This dictionary serves as the set of features for each email, forming the input of the Naive Bayes classifier.

I defined two key functions, test_classifier() and train_classifier(), which are not part of the original notebook. Each take three parameters: the Naive Bayes classifier, the folder containing the relevant dataset, and the category being trained or tested.

- The training function reads emails from the training dataset and passes their text to the classifier's train() method along with the corresponding category.
- The testing function reads emails from the testing dataset, predicts their category using the classifier, and compares the predictions with the actual labels. It stores the results, indicating whether each prediction was correct.

Q: For those emails that the classifier got wrong, what factors might have caused the classifier to be incorrect? You will need to look at the text of the email to determine this.

The classifier performed perfectly on the testing dataset, correctly classifying all emails as either as ”on_topic” or ”off_topic.” The results were printed in the terminal and saved to the CSV file named ”classification_results.csv.” Below are the results of the classifier:

```
                 File Name     Actual  Predicted  Correct
0   testing_on_topic_4.txt   on_topic   on_topic     True
1   testing_on_topic_5.txt   on_topic   on_topic     True
2   testing_on_topic_1.txt   on_topic   on_topic     True
3   testing_on_topic_2.txt   on_topic   on_topic     True
4   testing_on_topic_3.txt   on_topic   on_topic     True
5  testing_off_topic_2.txt  off_topic  off_topic     True
6  testing_off_topic_3.txt  off_topic  off_topic     True
```

```
7  testing_off_topic_1.txt  off_topic  off_topic     True
8  testing_off_topic_4.txt  off_topic  off_topic     True
9  testing_off_topic_5.txt  off_topic  off_topic     True
```

Q3

Table 1: Classification Results Confusion Matrix

		Actual	
		on_topic	off_topic
Predicted	on_topic	5 (TP)	0 (FP)
	off_topic	0 (FN)	5 (TN)

Q: Based on the results in the confusion matrix, how well did the classifier perform?

Based on the results in the confusion matrix, the classifier performed perfectly, achieving 100% accuracy. All 10 test emails were classified correctly, with no false positives or false negatives. This indicates that the model effectively distinguished between ”on_topic” and ”off_topic” emails within the test dataset.

Q: Would you prefer an email classifier to have more false positives or more false negatives? Why?

The choice between tolerating more false positives or false negatives depends on a variety of factors:

- **False Positives:** If the classifier produces more false positives, it means that irrelevant emails (off_topic) are mistakenly categorized as relevant (on_topic). This could lead to wasted time reviewing off-topic emails but ensures that all relevant emails are identified.
- **False Negatives:** If the classifier produces more false negatives, it means that some relevant emails (on_topic) are mistakenly categorized as irrelevant (off_topic). This risks missing important messages, which could be problematic if critical information is lost.

For most email classification tasks, minimizing false negatives would often be preferred. Missing a critical email (false negative) has more serious implications than having to sort through a few off-topic ones (false positive).

References

- Document Filtering, `https://github.com/odu-cs432-websci/public/blob/main/432_PCI_Ch06.ipynb`

- Module-12 Document-Filtering, `https://docs.google.com/presentation/d/1OpfBDl2YEE7AONVeKUyHA-J7a1mRjncD7cen8F6BG1A/edit#slide=id.g7f83ebe645_0_15`