DAT565/DIT407 Assignment 3

Giacomo Guidotto
gusguigi@student.gu.se

Leong Jia Yi, Janna
gusleoji@student.gu.se

2024-09-26

1 Spam & Ham: a human comparison

Firstly, we discuss the difference between spam and ham emails. Spam emails are often more incoherent than ham emails, containing more grammatical errors and seemingly random capitalization. Moreover, spam and ham emails differ in their purpose. Spam emails are more frequently asking the user to buy something or promise something too good to be true. Secondly, we talk about the difference between easy and hard ham emails. For easy ham emails, the purpose of the emails is more personal, clarifying a doubt or just continuing a conversation. However, many hard-ham emails try to sell a product or a service based on a subscription. Thus, as the email is formatted in a way that is trying to sell the receiver something, similar to the intention of some spam emails, it can be harder to distinguish hard ham from spam.

2 Dataset preparation: data splitting and preprocessing

After extracting and parsing the text files from the dataset provided in tarball format, the "chardet" module was employed to detect their character encodings and convert them into strings. These encoded strings were subsequently incorporated into a pandas "DataFrame".

Afterward, we created the two datasets, namely "hard" and "easy" by concatenating the "hard_ham" with the "spam" and the "easy_ham" with the "spam", respectively. In this step, we created a new dataset column with a boolean value indicating if the text was spam.

Then, using sklearn's train_test_split function, we set the seed and applied the train-test split to the datasets with an 80-20 proportion to prioritize the training data.

With the partitions done, the last step to classify the strings was to count-vectorize them. We needed to transform every email into a vector of frequencies for each word in the dataset dictionary i.e., the set of all distinct words. Firstly, we call the fit_transform function on the easy ham and spam training data. This fits the data, assigning numerical values to the words. Then it transforms the data, counting the frequency of each word and outputting a vector where each word is mapped to its frequency. Secondly, we call the transform function on the easy ham and spam test data. This will map the known words to their assigned indices and count them, ignoring unknown words. The same process applies to the hard ham and spam training, and its corresponding test data.

3 Easy ham results

For the Multinomial Naive Bayesian Classifier, the accuracy is 0.98, the precision is 0.98 and the recall is 1.00. The confusion matrix is in table 1.

	Predicted Positive (Ham)	Predicted Negative (Spam)
Actual Positive (Ham)	518	1
Actual Negative (Spam)	13	79

Table 1: Confusion matrix for easy ham using Multinomial Naive Bayesian Classifier

For the Bernoulli Naive Bayesian Classifier, the accuracy is 0.92, the precision is 0.92 and the recall is 1.00. The confusion matrix is in table 2.

	Predicted Positive (Ham)	Predicted Negative (Spam)
Actual Positive (Ham)	519	0
Actual Negative (Spam)	48	44

Table 2: Confusion matrix for easy ham using Bernoulli Naive Bayesian Classifier

4 Hard ham results

For the Multinomial Naive Bayesian Classifier, the accuracy is 0.97, the precision is 0.98 and the recall is 0.92. The confusion matrix is in table 3.

	Predicted Positive (Ham)	Predicted Negative (Spam)
Actual Positive (Ham)	48	4
Actual Negative (Spam)	1	98

Table 3: Confusion matrix for hard using Multinomial Naive Bayesian Classifier

For the Bernoulli Naive Bayesian Classifier, the accuracy is 0.93, the precision is 0.98 and the recall is 0.83. The confusion matrix is in table 4.

	Predicted Positive (Ham)	Predicted Negative (Spam)
Actual Positive (Ham)	43	9
Actual Negative (Spam)	1	98

Table 4: Confusion matrix for hard using Bernoulli Naive Bayesian Classifier

5 Easy vs Hard results

The MNB classifier performs generally better on the easy ham dataset than the hard dataset where the model on the easy ham dataset had a higher accuracy, higher recall, and equal precision. This is probably because there are more significant differences between easy ham and spam emails, allowing the model to distinguish and classify ham from spam emails better.

Furthermore, we can also see that the false negatives are generally higher in the hard ham results as compared to the easy ham results. This means that more ham emails are being misidentified as spam in the hard ham dataset. This is consistent in both the MNB classifier and the BNB classifier. We have identified some reasons for this. Firstly, this could be because, in the hard ham dataset, there are almost double the number of spam emails than ham emails. However, in the easy dataset, there are many more ham emails compared to spam emails. As a result, the classifiers using the hard ham dataset might be more inclined to categorize an email as a spam than a ham. Another possible reason is that for the hard ham dataset, it is much harder to tell the difference between a spam and a ham. Thus, it is more easy to wrongly classify a hard ham as a spam, giving rise to more false negatives.

A Code

Following is the Python code that we used to present the information in this report:

A.1 Problem 1: Spam & Ham

```
import pandas as pd
import tarfile
import chardet
```

The following function extracts the text files with the right encoding for the tarballs and loads them into a data frame

```
def extract_dataset(tarball_path):
        target_list = []
        errors_count = 0
        with tarfile.open(tarball_path, 'r:bz2') as
            tar:
                for member in tar.getmembers():
                        # skip if directory
                        if not member.isfile():
                                continue

                        # read bytes
                        extracted_file = tar.
                            extractfile(member)
                        if extracted_file is None:
                                continue
                        raw_data = extracted_file.read()

                        # detect encoding
                        detection = chardet.detect(raw_data)
                        encoding = detection['encoding']
                        confidence = detection['confidence']

                        try:
                                text = raw_data.decode(encoding)
                                print(f'parsed file "{member.name}" with encoding {encoding} ({confidence * 100:.2f}% of confidence)')

                        except (UnicodeDecodeError, TypeError): # fallback to utf-8 if detected encoding fails
                                text = raw_data.decode('utf-8', errors='replace')
                                print(f'parsed file "{member.name}" with encoding utf-8 (fallback, encoding {encoding} failed)')
                                errors_count += 1

                        target_list.append(text)

                print(f'parsed {len(target_list)} files with {(1 - (errors_count / len(target_list))) * 100:.0f}% of accuracy ({errors_count} errors)')
                print("-" * 100)
                return pd.DataFrame(target_list, columns=['text'])
```

Apply the function to the three tarballs

```
easy_ham = extract_dataset("data/20021010_easy_ham.tar.bz2")
hard_ham = extract_dataset("data/20021010_hard_ham.tar
    .bz2")
spam = extract_dataset("data/20021010_spam.tar.bz2")

easy_ham.head()
```

Create the two environment datasets: easy and hard

```
easy_ham['is_spam'] = 0
hard_ham['is_spam'] = 0
spam['is_spam'] = 1

easy = pd.concat([easy_ham, spam], ignore_index=True)
hard = pd.concat([hard_ham, spam], ignore_index=True)

easy.head()
```

Apply the train-test split

```
from sklearn.model_selection import train_test_split

easy_train, easy_test = train_test_split(easy,
    test_size=0.2, random_state=42)
hard_train, hard_test = train_test_split(hard,
    test_size=0.2, random_state=42)
```

A.2 Problem 2: Preprocessing

Count-vectorize the text data

```
from sklearn.feature_extraction.text import
    CountVectorizer

cv = CountVectorizer()
easy_train_cv = cv.fit_transform(easy_train['text'])
easy_test_cv = cv.transform(easy_test['text'])

hard_train_cv = cv.fit_transform(hard_train['text'])
hard_test_cv = cv.transform(hard_test['text'])

easy_train_cv.shape, easy_test_cv.shape, hard_train_cv
    .shape, hard_test_cv.shape
```

A.3 Problem 3: Easy Ham

Apply the two classifiers to the ‘easy‘ dataset

A.3.1 Multinomial Naive Bayes Classifier (MNB)

```
from sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB()
mnb.fit(easy_train_cv, easy_train['is_spam'])
```

```python

easy_mnb_pred = mnb.predict(easy_test_cv)
```

Compute the accuracy, precision, recall, and confusion matrix for the MNB classifier-vectorize the text data

```python
tp = ((easy_test['is_spam'] == 0) & (easy_mnb_pred == 0)).sum()
fp = ((easy_test['is_spam'] == 1) & (easy_mnb_pred == 0)).sum()
fn = ((easy_test['is_spam'] == 0) & (easy_mnb_pred == 1)).sum()
tn = ((easy_test['is_spam'] == 1) & (easy_mnb_pred == 1)).sum()
acc = (tp + tn) / (tp + fp + tn + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)

print(f"Accuracy:␣{acc:.2f}")
print(f"Precision:␣{precision:.2f}")
print(f"Recall:␣{recall:.2f}")
print(f"Confusion␣Matrix:\n{tp}␣{fn}\n{fp}␣{tn}")
```

A.3.2 Bernoulli Naive Bayes Classifier (BNB)

```python
from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()
bnb.fit(easy_train_cv, easy_train['is_spam'])
easy_bnb_pred = bnb.predict(easy_test_cv)
```

Compute the accuracy, precision, recall, and confusion matrix for the MNB classifier-vectorize the text

```python
tp = ((easy_test['is_spam'] == 0) & (easy_bnb_pred == 0)).sum()
fp = ((easy_test['is_spam'] == 1) & (easy_bnb_pred == 0)).sum()
fn = ((easy_test['is_spam'] == 0) & (easy_bnb_pred == 1)).sum()
tn = ((easy_test['is_spam'] == 1) & (easy_bnb_pred == 1)).sum()
acc = (tp + tn) / (tp + fp + tn + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)

print(f"Accuracy:␣{acc:.2f}")
print(f"Precision:␣{precision:.2f}")
print(f"Recall:␣{recall:.2f}")
print(f"Confusion␣Matrix:\n{tp}␣{fn}\n{fp}␣{tn}")
```

A.4 Problem 4: Hard Ham

Apply the two classifiers to the ‘hard‘ dataset

A.4.1 Multinomial Naive Bayes Classifier (MNB)

```python
mnb = MultinomialNB()
mnb.fit(hard_train_cv, hard_train['is_spam'])
hard_mnb_pred = mnb.predict(hard_test_cv)
```

Compute the accuracy, precision, recall, and confusion matrix for the MNB classifier

```python
tp = ((hard_test['is_spam'] == 0) & (hard_mnb_pred == 0)).sum()
fp = ((hard_test['is_spam'] == 1) & (hard_mnb_pred == 0)).sum()
fn = ((hard_test['is_spam'] == 0) & (hard_mnb_pred == 1)).sum()
tn = ((hard_test['is_spam'] == 1) & (hard_mnb_pred == 1)).sum()
acc = (tp + tn) / (tp + fp + tn + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)

print(f"Accuracy: {acc:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"Confusion Matrix:\n{tp} {fn}\n{fp} {tn}")
```

A.4.2 Bernoulli Naive Bayes Classifier (BNB)

```python
bnb = BernoulliNB()
bnb.fit(hard_train_cv, hard_train['is_spam'])
hard_bnb_pred = bnb.predict(hard_test_cv)
```

Compute the accuracy, precision, recall, and confusion matrix for the BNB classifier

```python
tp = ((hard_test['is_spam'] == 0) & (hard_bnb_pred == 0)).sum()
fp = ((hard_test['is_spam'] == 1) & (hard_bnb_pred == 0)).sum()
fn = ((hard_test['is_spam'] == 0) & (hard_bnb_pred == 1)).sum()
tn = ((hard_test['is_spam'] == 1) & (hard_bnb_pred == 1)).sum()
acc = (tp + tn) / (tp + fp + tn + fn)
precision = tp / (tp + fp)
recall = tp / (tp + fn)

```

```python
print(f"Accuracy: {acc:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"Confusion Matrix:\n{tp} {fn}\n{fp} {tn}")
```