

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 give information than ham emails. 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 pre-processing

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[1], 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, spam training, and test data.

3 Easy ham results

For the Multinomial Naive Bayesian Classifier, the accuracy is 0.98, the precision is 1.00 and the recall is 0.86. 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 0.99 and the recall is 0.48. 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 0.96 and the recall is 0.99. 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 0.92 and the recall is 0.99. 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 Discussion: Easy vs Hard Ham results

References

- [1] scikit-learn. *train_test_split*. Retrieved 2023-09-21. nd. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

A Code

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

A.1 Problem 1: Spam & Ham

```
1 import pandas as pd
2 import tarfile
3 import chardet
```

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

```
1 def extract_dataset(tarball_path):
2     target_list = []
3     errors_count = 0
4     with tarfile.open(tarball_path, 'r:bz2') as
5         tar:
6         for member in tar.getmembers():
7             # skip if directory
8             if not member.isfile():
9                 continue
10
11             # read bytes
12             extracted_file = tar.
13                 extractfile(member)
14             if extracted_file is None:
15                 continue
16             raw_data = extracted_file.read
17                 ()
18
19             # detect encoding
20             detection = chardet.detect(
21                 raw_data)
22             encoding = detection['encoding
23                 '']
24             confidence = detection['
25                 confidence']
26
27         try:
```

```

22         text = raw_data.decode
                (encoding)
23         print(f'parsed_file_{
                member.name}"_with_
                encoding_{encoding}
                _({confidence*_
                100:.2f})%_of_
                confidence}')
24
25     except (UnicodeDecodeError,
                TypeError): # fallback to
                utf-8 if detected encoding
                fails
26         text = raw_data.decode
                ('utf-8', errors='
                replace')
27         print(f'parsed_file_{
                member.name}"_with_
                encoding_utf-8_(
                fallback,_encoding_
                {encoding}_failed)')
                )
28         errors_count += 1
29
30     target_list.append(text)
31
32     print(f'parsed_{len(target_list)}_
                files_with_{(1-(errors_count/_
                len(target_list)))*100:.0f}%_of_
                accuracy_{errors_count}_errors')
33     print("-" * 100)
34     return pd.DataFrame(target_list,
                columns=['text'])

```

Apply the function to the three tarballs

```

1 easy_ham = extract_dataset("data/20021010_easy_ham.tar
    .bz2")
2 hard_ham = extract_dataset("data/20021010_hard_ham.tar
    .bz2")
3 spam = extract_dataset("data/20021010_spam.tar.bz2")
4
5 easy_ham.head()

```

Create the two environment datasets: easy and hard

```

1 easy_ham['is_spam'] = 0
2 hard_ham['is_spam'] = 0
3 spam['is_spam'] = 1
4
5 easy = pd.concat([easy_ham, spam], ignore_index=True)
6 hard = pd.concat([hard_ham, spam], ignore_index=True)

```

```

7
8 easy.head()

    Apply the train-test split

1 from sklearn.model_selection import train_test_split
2
3 easy_train, easy_test = train_test_split(easy,
    test_size=0.2, random_state=42)
4 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

```

1 from sklearn.feature_extraction.text import
    CountVectorizer
2
3 cv = CountVectorizer()
4 easy_train_cv = cv.fit_transform(easy_train['text'])
5 easy_test_cv = cv.transform(easy_test['text'])
6
7 hard_train_cv = cv.fit_transform(hard_train['text'])
8 hard_test_cv = cv.transform(hard_test['text'])
9
10 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)

```

1 from sklearn.naive_bayes import MultinomialNB
2 mnb = MultinomialNB()
3 mnb.fit(easy_train_cv, easy_train['is_spam'])
4
5 easy_mnb_pred = mnb.predict(easy_test_cv)

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

1 from sklearn.metrics import accuracy_score,
    precision_score, recall_score, confusion_matrix
2
3 easy_mnb_accuracy = accuracy_score(easy_test['is_spam'],
    easy_mnb_pred)
4 easy_mnb_precision = precision_score(easy_test['
    is_spam'], easy_mnb_pred)

```

```

5 easy_mnb_recall = recall_score(easy_test['is_spam'],
    easy_mnb_pred)
6 easy_mnb_confusion_matrix = confusion_matrix(easy_test
    ['is_spam'], easy_mnb_pred)
7
8 print(f'Accuracy: {easy_mnb_accuracy:.2f}')
9 print(f'Precision: {easy_mnb_precision:.2f}')
10 print(f'Recall: {easy_mnb_recall:.2f}')
11 print(f'Confusion Matrix:\n{easy_mnb_confusion_matrix}
    ')
12
13 from sklearn.naive_bayes import BernoulliNB
14
15 bnb = BernoulliNB()
16 bnb.fit(easy_train_cv, easy_train['is_spam'])
17 easy_bnb_pred = bnb.predict(easy_test_cv)

```

A.3.2 Bernoulli Naive Bayes Classifier (BNB)

```

1 from sklearn.naive_bayes import BernoulliNB
2
3 bnb = BernoulliNB()
4 bnb.fit(easy_train_cv, easy_train['is_spam'])
5 easy_bnb_pred = bnb.predict(easy_test_cv)

```

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

```

1 easy_bnb_accuracy = accuracy_score(easy_test['is_spam'],
    easy_bnb_pred)
2 easy_bnb_precision = precision_score(easy_test['
    is_spam'], easy_bnb_pred)
3 easy_bnb_recall = recall_score(easy_test['is_spam'],
    easy_bnb_pred)
4 easy_bnb_confusion_matrix = confusion_matrix(easy_test
    ['is_spam'], easy_bnb_pred)
5
6 print(f'Accuracy: {easy_bnb_accuracy:.2f}')
7 print(f'Precision: {easy_bnb_precision:.2f}')
8 print(f'Recall: {easy_bnb_recall:.2f}')
9 print(f'Confusion Matrix:\n{easy_bnb_confusion_matrix}
    ')

```

A.4 Problem 4: Hard Ham

Apply the two classifiers to the 'hard' dataset

A.4.1 Multinomial Naive Bayes Classifier (MNB)

```

1 mnb = MultinomialNB()
2 mnb.fit(hard_train_cv, hard_train['is_spam'])
3 hard_mnb_pred = mnb.predict(hard_test_cv)

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

1 hard_mnb_accuracy = accuracy_score(hard_test['is_spam'],
    hard_mnb_pred)
2 hard_mnb_precision = precision_score(hard_test['
    is_spam'], hard_mnb_pred)
3 hard_mnb_recall = recall_score(hard_test['is_spam'],
    hard_mnb_pred)
4 hard_mnb_confusion_matrix = confusion_matrix(hard_test
    ['is_spam'], hard_mnb_pred)
5
6 print(f'Accuracy: {hard_mnb_accuracy:.2f}')
7 print(f'Precision: {hard_mnb_precision:.2f}')
8 print(f'Recall: {hard_mnb_recall:.2f}')
9 print(f'Confusion Matrix:\n{hard_mnb_confusion_matrix}
    ')

```

A.4.2 Bernoulli Naive Bayes Classifier (BNB)

```

1 bnb = BernoulliNB()
2 bnb.fit(hard_train_cv, hard_train['is_spam'])
3 hard_bnb_pred = bnb.predict(hard_test_cv)

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

1 hard_bnb_accuracy = accuracy_score(hard_test['is_spam'],
    hard_bnb_pred)
2 hard_bnb_precision = precision_score(hard_test['
    is_spam'], hard_bnb_pred)
3 hard_bnb_recall = recall_score(hard_test['is_spam'],
    hard_bnb_pred)
4 hard_bnb_confusion_matrix = confusion_matrix(hard_test
    ['is_spam'], hard_bnb_pred)
5
6 print(f'Accuracy: {hard_bnb_accuracy:.2f}')
7 print(f'Precision: {hard_bnb_precision:.2f}')
8 print(f'Recall: {hard_bnb_recall:.2f}')
9 print(f'Confusion Matrix:\n{hard_bnb_confusion_matrix}
    ')

```