DAT565/DIT407 Assignment 3

Connell Hagen
connellh@student.chalmers.se

Måns Redin
mansre@student.chalmers.se

2024-09-19

1 Problem 1: Spam & Ham

1.1 Data Exploration

The spam emails frequently used sentences that started with a command. For example "Get this..., Experience this..., etc." The spam emails frequently uses exclamation marks, uppercase letters, and also had a lot of links in them. The ham emails are generally classified by them being directed to a specific person, involving sentimental expressions. The spam emails are often very explicit, and usually want you to buy something.

2 Problem 2: Preprocessing

The method applied to this problem was from the CountVectorizer library in Python. The three different operations were the fit, transform and the fit_transform. For the training dataset the fit_transform operation was applied, which is a combination of the fit and the transform operation. The fit operation goes though the training dataset and creates an array that has features for the different words in the emails. The transform operation is going through the dataset and counting the amount of appearances of each specific word, and stores it in the array constructed by the fit operation. Since the fit_transform operation was already made on the training dataset, which is a subset of the total dataset, the only operation needed for the testing dataset was the transform operation. The same applies to the type of emails, but for the labelEncoder operation. The labelEncoder operation makes binary classification columns for the different types of emails.

3 Problem 3: Easy Ham

In this part of the problem the accuracy, precision, and recall was desired. Equation 1, 2 and 3 defines these three measurements.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

These were computed for the trained model, using both a Multinomial Naive Bayes classifier and a Bernoulli Naive Bayes classifier. Table 1 shows the accuracy, precision and recall for the two classifiers for the "easy ham" dataset.

	Multinomial	Bernoulli
Accuracy	0.969	0.917
Precision	0.966	0.914
Recall	0.998	0.995

Table 1: The accuracy, precision and recall for the Multinomial and Bernoulli Naive Bayes classifier for the "easy ham" dataset

A confusion matrix was desired for the different Naive Bayes classifiers. Table 2 shows the scores of the Bernoulli Naive Bayes classifier.

	Predicted positive	Predicted negative	Total
Actual positive	597	3	600
Actual negative	56	51	107
Total	653	54	707

Table 2: Confusion matrix for the Bernoulli Naive Bayes classifier for the "easy ham" dataset

Table 3 shows the results for the Multinomial Naive Bayes classifier.

	Predicted positive	Predicted negative	Total
Actual positive	599	1	600
Actual negative	21	86	107
Total	620	87	707

Table 3: Confusion matrix for the Multinominal Naive Bayes classifier for the "easy ham" dataset

4 Problem 4: Hard Ham

The same approach as in the previous exercise, but now for the "hard ham" dataset instead of the "easy ham". Table 4 shows the accuracy, precision and recall values for the "hard ham" dataset.

The confusion matrices for the two classifiers for the "hard ham" dataset can be seen in table 5 and table 6.

4.1 Discussion

The overall accuracy and recall for the hard ham dataset were worse than for the easy ham dataset on both model types. It was harder for the models to tell what

	Multinomial	Bernoulli
Accuracy	0.950	0.875
Precision	0.979	0.972
Recall	0.870	0.648

Table 4: The accuracy, precision and recall for the Multinomial and Bernoulli Naive Bayes classifier for the "hard ham" dataset

was hard ham than easy ham because the hard ham had features like HTML and advertisements within commonly. The recall (See equation 3) especially was lower, which can be explained by the fact that there were a lot of false negatives. This is probably because the hard ham was frequently newsletters or other mass distributed emails that share many characteristics with spam emails. The precision, in contrast, actually was better for the hard ham dataset. This is because there were not many false positives. This can be explained by the fact that there are many spam emails that are obviously spam, but among the hard ham emails, there were not many that were obviously not spam. This causes the mistakes to shift towards false negatives instead of false positives when it comes to identifying spam.

	Predicted positive	Predicted negative	Total
Actual positive	35	19	54
Actual negative	1	105	106
Total	36	122	160

Table 5: Confusion matrix for the Bernoulli Naive Bayes classifier for the "hard ham" dataset

	Predicted positive	Predicted negative	Total
Actual positive	47	7	54
Actual negative	1	105	106
Total	48	112	160

Table 6: Confusion matrix for the Multinomial Naive Bayes classifier for the "hard ham" dataset

A Code

```python
import numpy as np
import pandas as pd
import os
import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB

# takes a directory of files, and a label, and returns a
# list of 2-tuples of each file's contents, and the given label
def get_email_content(dir, label):
    files = []
    for filename in os.listdir(dir):
        path = os.path.join(dir, filename)
        if os.path.isfile(path):
            try:
                with open(path, encoding="utf-8") as f:
                    files += [(clean_email(f.read()), label)]
            except:
                pass
    return files

# if we want to clean the emails we can modify this function
# however, our proposed way of removing the headers actually
# decreased the accuracy of the models
def clean_email(content):
    # remove_header = '\\n\\n'
    # email_parts = re.split(remove_header, content)
    # if (len(email_parts) == 1):
    #     return email_parts[0]
    # else:
    #     return email_parts[1]
    return content

def get_confusion_values(y_test_inv, y_pred_inv, label):
    tp = ((y_test_inv == label) & (y_pred_inv == label)).sum()
    fp = ((y_test_inv != label) & (y_pred_inv == label)).sum()
    fn = ((y_test_inv == label) & (y_pred_inv != label)).sum()
    tn = ((y_test_inv != label) & (y_pred_inv != label)).sum()
    return tp, fp, fn, tn

def get_apr_values(tp, fp, fn, tn):
    accuracy = (tp + tn) / (tp + fp + fn + tn)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    return accuracy, precision, recall

SEED = 12345678
easy_ham_dir = 'assignment_3/easy_ham'
hard_ham_dir = 'assignment_3/hard_ham'
spam_dir = 'assignment_3/spam'

easy_ham_files = get_email_content(easy_ham_dir, "Ham")
hard_ham_files = get_email_content(hard_ham_dir, "Ham")
spam_files = get_email_content(spam_dir, "Spam")

# to switch between the easy_ham and hard_ham dataset,
# swap out either easy_ham_files or hard_ham_files for the
# other here
records = hard_ham_files + spam_files

df = pd.DataFrame.from_records(records, columns=["Email", "Type"])
df_train, df_test = train_test_split(df, random_state=SEED, test_size=0.25)

cv = CountVectorizer()
x_train = cv.fit_transform(df_train["Email"])
x_test = cv.transform(df_test["Email"])

le = LabelEncoder()
y_train = le.fit_transform(df_train["Type"])
y_test = le.transform(df_test["Type"])
y_test_inv = le.inverse_transform(y_test)

bnb = BernoulliNB()
bnb.fit(x_train, y_train)
y_pred_bnb = bnb.predict(x_test)
y_pred_bnb_inv = le.inverse_transform(y_pred_bnb)
```

```
acc_bnb, pre_bnb, rec_bnb = get_apr_values(*
    get_confusion_values(y_test_inv, y_pred_bnb_inv, "
    Ham"))

mnb = MultinomialNB()
mnb.fit(x_train, y_train)
y_pred_mnb = mnb.predict(x_test)
y_pred_mnb_inv = le.inverse_transform(y_pred_mnb)
acc_mnb, pre_mnb, rec_mnb = get_apr_values(*
    get_confusion_values(y_test_inv, y_pred_mnb_inv, "
    Ham"))
```