DAT565/DIT407 Assignment 3

Henning Anevik Joar Forsberg

2024-10-08

Problem 1: Spam and Ham

Data exploration

Starting by extracting the emails by using the library tarfile. We extracted the files from their bz2 format and now we are going to export the emails to individual text files A to be able to analyze their content and so one can supply an answer to question A. A sample of a mail can be seen in Figure 1

Figure 1: Sample of a mail in .txt file

Starting by looking at the readme.html provided by spamassassin its stated that the easy_ham emails don't frequently contain spammish signatures which can be lots of capital letters, HTML and keyword like "adult-only" etc. Hard_ham on the other hand have several aspects close to spam [3]. So some of the signatures of spam can be found in the emails, which makes them harder to detect as non spam. And the spam file includes spam.

Data splitting

Start with creating python lists that include spam and easy-ham and the other will also include spam and hard-ham. The following code have resembles to the code seen in "Problem 1" but for more structured code this was the approach used. After the mails are put into the list the data splitting is performed. As mentioned in the lecture the trainset consist of 75% of the mails and only 25% for the test B.

Problem 2: Preprocessing

Started with importing the library from Scikit-Learn that had the class CountVectorizer. Then applied fit_transform to the train data. The fit_transform perform two actions. It learns the models scaling parameters on the training set where it learns the different parameters and their connections. The fit function part calculates the mean and variance of the different features found in the data. Secondly it transform the data into a matrix that is used later by the two classifiers in problem 3 & 4. The transform function is applied on the test dataset [2].

The fit method that comes with the library sklearn is used to train the model on the dataset we provide. If we would have used fit on the test data it would have lead to the actual test becoming over fitted and resulted in an invalid evaluation. The transform function simply put use what it would have learn from the training data and apply the knowledge on the test data.

The preprocessing can be seen in C where CountVectorizer, fit_transform and transform is used separately on Easy/Hard Ham.

Problem 3: Easy Ham

With the test and training data gathered from the preprocessing we apply the two different Naïve Bayesian classifiers. The Multinomial Naive Bayesian & the Bernoulli Naive Bayesian Classifier seen in D.

Table 1: The Multinomial Naive Bayesian Classifier (Easy Ham)

Metric		**Value**
Accuracy		96.85%
Precision		98.10%
Recall		82.40%
Confusion Matrix	636	2
	22	103

Table 2: The Bernoulli Naive Bayesian Classifier (Easy Ham)

Metric	**Value**	
Accuracy	90.69%	
Precision	100.00%	
Recall	43.20%	
Confusion Matrix	638	0
	71	54

Both classifiers performed well but had some differences when looking the their metrics. Looking at 1 & 2 one can see that their precision where close but the metrics that was most different was accuracy and recall. MultinomialNB had an accuracy of 96.85% which is around 6% better then BernoulliNB that had an accuracy of 90.68%. The reason behind this difference is that MultinomialNB is better regarding text classification where word count and/or frequency is important, while BernoulliNB works a bit more binary focus on a single keyword. So it can miss the bigger picture which the MultinomialNB captures [1]. Looking at recall MultinomialNB had a value of 82.40% & BernoulliNB only had 43.20%. The recall showcases how many time and actual spam was correctly identified. Meaning that MultinomialNB performed way better then the other classifier.

Problem 4: Hard Ham

Same as done in problem 3. Only difference is that the Hard Ham is used instead of the Easy Ham E.

Table 3: The Multinomial Naive Bayesian Classifier (Hard Ham)

Metric	**Value**	
Accuracy	92.55%	
Precision	93.02%	
Recall	96.00%	
Confusion Matrix	54	9
	5	120

Table 4: The Bernoulli Naive Bayesian Classifier (Hard Ham)

Metric	**Value**	
Accuracy	84.04%	
Precision	81.46%	
Recall	98.40%	
Confusion Matrix	35	28
	2	123

Looking at 3 & 4 we can directly see that MultinomialNB wins this time as well but performed worse in the recall metric. The reason why we can probalby see the recall win for BernoulliNB in recall is because of the feature where its

better when looking into single keywords, which can be found more often in the Hard Ham (more spam signatures). The MultinomialNB achieves a better balance between that resulted in an higher overall performance. This trend can be seen when also looking at the data generated from the hard ham test. The classifiers on the hard ham performed a little worse which points that the ham is somewhat more complex. The Bernoulli Naive Bayes classifier has high recall but are having problems with many false positives. Both classifiers perform better on easy ham in terms of precision vs the hard ham.

References

[1] Valentin Calomme. *Difference between Bernoulli and Multinomial Naive Bayes*. Retrieved 2024-10-09. 2018. URL: https://datascience.stackexchange.com/questions/27624/difference-between-bernoulli-and-multinomial-naive-bayes.

[2] Chetna Khanna. *What and why behind $fit_transform()and transform()in scikit-learn$!*. Retrieved 2024-10-09. 2020. URL: https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe.

[3] Justin Mason. *Spam Assassin public mail corpus*. Retrieved 2023-11-10. 2004. URL: https://spamassassin.apache.org/old/publiccorpus/.

A Data exploration

```
import tarfile
import os

for file in os.listdir("/content/"):
  if file.endswith(".bz2"):
    with tarfile.open(file, "r:bz2") as tar:
      tar.extractall(path = "tmp")

extracted_folders = os.listdir("/content/tmp")
print("Extracted␣folders:", extracted_folders)

import email
from email import policy
from email.parser import BytesParser
from email.header import decode_header

for folder in extracted_folders:
  path = "/content/tmp/" + folder
  file_list = os.listdir(path)

#Opening one folder at the time and then reading and
    ↪ writing each file to a .txt based on the folder
    ↪ name
```

```
#In the end genereates three .txt that can be read and analyzed for question A
  for file_name in file_list:
    with open(path + "/" + file_name, "rb") as file:
      with open(folder + ".txt", "a", encoding="utf-8") as f:
        msg = BytesParser(policy=policy.default).parse(file)

        if msg["subject"] is None:
          continue
        else:
          f.write(f"Subject: {msg['subject']}\n")
        f.write(f"From: {msg['from']}\n")

        if msg['to'] is None:
          continue
        else:
          f.write(f"To: {msg['to']}\n")
        body = msg.get_body(preferencelist=('plain',))

        if body is not None:
          try:
              content = body.get_content()
              charset = body.get_content_charset() or "utf-8"
              #Attempt to decode using the detected charset if not fallback will apply
              content = content.encode("ascii", "ignore").decode(charset, errors="replace")
              f.write(f"Body:\n{content}\n")
          except LookupError:
              f.write("Body: [Could not decode the body due to unknown encoding]\n")
          #Email seperator for easy reading
          f.write("##########################\n")
```

B Data splitting

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score, confusion_matrix

#Create 2 list which is gonna hold mail from easy and
```

```
    ↪ hard, both the list will also contain spam for
    ↪ the data splitting.
lst_ez_ham = []
lst_hd_ham = []
lst_spam = []

encodings = ["utf-8", "ascii", "iso-8859-1", "7-bit",
    ↪ "8-bit"]

for folder in extracted_folders:
  for file in os.listdir("/content/tmp/" + folder +
      ↪ "/"):
    for encoding in encodings:
      try:
        with open("/content/tmp/" + folder + "/" +
            ↪ file, "rb") as f:
          data = f.readlines()
          data = [line.decode(encoding,
              ↪ errors='ignore') for line in data]
          content = ''.join(data)  # Join lines into
              ↪ a single string
          break
      except (UnicodeDecodeError, LookupError):
        continue  # Try the next encoding if current
            ↪ fails

    if folder == "easy_ham":
      lst_ez_ham.append(content)
    elif folder == "hard_ham":
      lst_hd_ham.append(content)
    else:
      lst_spam.append(content)

#Rule of thumb: if we do a simple train-test split,
    ↪ use 75% of data for the training set and 25%
    ↪ for the test set

#Easy Ham Split datasplit
X_ez = lst_ez_ham + lst_spam
y_ez = [0] * len(lst_ez_ham) + [1] * len(lst_spam)
    ↪ #Labels
X_ez_train, X_ez_test, y_ez_train, y_ez_test =
    ↪ train_test_split(X_ez, y_ez, test_size=0.25,
    ↪ random_state=64, stratify=y_ez)

#Hard Ham Split datasplit
X_hd = lst_hd_ham + lst_spam
y_hd = [0] * len(lst_hd_ham) + [1] * len(lst_spam)
    ↪ #Labels
X_hd_train, X_hd_test, y_hd_train, y_hd_test =
```

```
    ↪ train_test_split(X_hd, y_hd, test_size=0.25,
    ↪ random_state=64, stratify=y_hd)
```

C Preprocessing

```
from sklearn.feature_extraction.text import
    ↪ CountVectorizer
vectorizer = CountVectorizer()
x_ez_train = vectorizer.fit_transform(X_ez_train)
x_ez_test = vectorizer.transform(X_ez_test)
x_ez_train = vectorizer.fit_transform(X_ez_train)
x_ez_test = vectorizer.transform(X_ez_test)
```

D Easy Ham

```
from sklearn.naive_bayes import MultinomialNB,
    ↪ BernoulliNB

clf = MultinomialNB()
clf.fit(x_ez_train, y_ez_train)
testA_predictions = clf.predict(x_ez_test)

accuracy = accuracy_score(y_ez_test,
    ↪ testA_predictions)
precision = precision_score(y_ez_test,
    ↪ testA_predictions)
recall = recall_score(y_ez_test, testA_predictions)
print("The␣Multinomial␣Naive␣Bayesian␣Classifier")
print(f"Accuracy:␣{accuracy␣*␣100:.2f}%")
print(f"Precision:␣{precision␣*␣100:.2f}%")
print(f"Recall:␣{recall␣*␣100:.2f}%")

conf_matrix = confusion_matrix(y_ez_test,
    ↪ testA_predictions)
print("Confusion␣Matrix:")
print(conf_matrix)
print("")
############################

clfB = BernoulliNB()
clfB.fit(x_ez_train, y_ez_train)
testB_predictions = clfB.predict(x_ez_test)

accuracy = accuracy_score(y_ez_test,
    ↪ testB_predictions)
precision = precision_score(y_ez_test,
    ↪ testB_predictions)
```

```
recall = recall_score(y_ez_test, testB_predictions)
print("The Bernoulli Naive Bayesian Classifier")
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")

conf_matrix = confusion_matrix(y_ez_test,
    ↪ testB_predictions)
print("Confusion Matrix:")
print(conf_matrix)
```

E Hard Ham

```
clf = MultinomialNB()
clf.fit(x_hd_train, y_hd_train)
testA_predictions = clf.predict(x_hd_test)
accuracy = accuracy_score(y_hd_test,
    ↪ testA_predictions)
precision = precision_score(y_hd_test,
    ↪ testA_predictions)
recall = recall_score(y_hd_test, testA_predictions)

print("The Multinomial Naive Bayesian Classifier")
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")

conf_matrix = confusion_matrix(y_hd_test,
    ↪ testA_predictions)
print("Confusion Matrix:")
print(conf_matrix)
print("")
################################

clfB = BernoulliNB()
clfB.fit(x_hd_train, y_hd_train)
testB_predictions = clfB.predict(x_hd_test)
accuracy = accuracy_score(y_hd_test,
    ↪ testB_predictions)
precision = precision_score(y_hd_test,
    ↪ testB_predictions)
recall = recall_score(y_hd_test, testB_predictions)

print("The Bernoulli Naive Bayesian Classifier")
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision * 100:.2f}%")
print(f"Recall: {recall * 100:.2f}%")

```

```
conf_matrix = confusion_matrix(y_hd_test,
    ↪ testB_predictions)
print("Confusion Matrix:")
print(conf_matrix)
```