Intelligent Email Filtering: Exploring Naive Bayes for Spam and Subtle Ham Detection

2025-06-23

Problem 1: Spam and Ham Classification

Introduction

This project investigates email classification by distinguishing between two primary categories:

- **Ham:** Legitimate (non-spam) emails
- **Spam:** Unsolicited and potentially malicious bulk emails

The datasets used include:

- **Easy Ham:** Clearly benign emails
- **Hard Ham:** Legitimate emails that are more difficult to distinguish from spam
- **Spam:** Emails containing spam characteristics

Two Naive Bayes classifiers—Multinomial and Bernoulli—were implemented to analyze their effectiveness in classifying these emails.

A. Data Exploration

Initial inspection reveals key differences:

- **Easy Ham:** Shorter messages, clear structure, and fewer HTML or marketing terms.
- **Hard Ham:** More complex messages, occasional links, and vocabulary overlapping with spam.
- **Spam:** Rich in HTML content, includes numbers in email addresses, many links, and terms like "free", "offer", "win".

B. Data Splitting

Each dataset was labeled appropriately. After labeling, the datasets were split into training and testing sets separately for each category. The feature matrix X contained the email content, and the target vector y contained the corresponding labels.

Problem 2: Preprocessing

We applied text preprocessing using `CountVectorizer` to tokenize and vectorize the text data:

- Unique tokens were assigned to words in the dataset.
- We used `fit_transform()` for training data and `transform()` for test data.
- Default settings were retained for simplicity.

Problem 3: Easy Ham Classification

Multinomial Naive Bayes

1. Accuracy:

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

- TP: Correctly predicted spam
- TN: Correctly predicted easy ham
- FP: Ham misclassified as spam
- FN: Spam misclassified as ham

Calculated using:

$$\text{accuracy\_score}(y\_test\_eh\_spam, \text{predictMN\_eh})$$

2. Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. Recall:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Both metrics are computed using Scikit-learn functions.

Confusion Matrix:

```
eh_MN_confusion_matrix = confusion_matrix(y_test_eh_spam, predictMN_eh)
```

```
Easy ham Accuracy: 0.9777
Easy ham Precision: 0.9910
Easy ham Recall: 0.8730

Easy ham Confusion Matrix(Multinomial):
```

	Positive prediction	Negative prediction	Total
Actual Positive	637	1	638
Actual Negative	16	110	126
Total	653	111	764

Figure 1: Multinomial Naive Bayes for Easy Ham vs Spam

Bernoulli Naive Bayes

```
Easy ham Accuracy: 0.9123
Easy ham Precision: 0.9683
Easy ham Recall: 0.4841

Easy ham Confusion Matrix(Bernoulli):
```

	Positive prediction	Negative prediction	Total
Actual Positive	636	2	638
Actual Negative	65	61	126
Total	701	63	764

Figure 2: Bernoulli Naive Bayes for Easy Ham vs Spam

Problem 4: Hard Ham Classification

Multinomial Naive Bayes

```
Hard ham Accuracy: 0.9259
Hard ham Precision: 0.9308
Hard ham Recall: 0.9603

Hard ham Confusion Matrix(Multinomial):
```

	Positive prediction	Negative prediction	Total
Actual Positive	54	9	63
Actual Negative	5	121	126
Total	59	130	189

Figure 3: Multinomial Naive Bayes for Hard Ham vs Spam

Bernoulli Naive Bayes

```
Hard ham Accuracy: 0.8995
Hard ham Precision: 0.8741
Hard ham Recall: 0.9921

Hard ham Confusion Matrix(Bernoulli):
```

	Positive prediction	Negative prediction	Total
Actual Positive	45	18	63
Actual Negative	1	125	126
Total	46	143	189

Figure 4: Bernoulli Naive Bayes for Hard Ham vs Spam

Discussion

The experiments show that:

- Both classifiers perform better on easy ham compared to hard ham.
- The hard ham dataset likely reduces accuracy due to similarities with spam and smaller sample size.
- The Multinomial Naive Bayes classifier consistently outperforms Bernoulli in both cases.

- Bernoulli ignores word frequency, which limits its effectiveness for text-heavy data like emails.

Multinomial Naive Bayes benefits from capturing term frequency, making it more suitable for spam filtering in this context.

References

1. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy˙score.html
2. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision˙score.html
3. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall˙score.html
4. https://en.wikipedia.org/wiki/Confusion˙matrix

Appendix

Setting up

```
#Importing necessary packages
import numpy as np
import pandas as pd
import os
import tarfile

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

```
#Connect to drive to be able to import files
from google.colab import drive
drive.mount('/content/drive')
```

```
#File path
file_path_easy_ham = '/content/drive/MyDrive/20021010_easy_ham.tar.bz2'
file_path_hard_ham = '/content/drive/MyDrive//20021010_hard_ham.tar.bz2'
file_path_spam = '/content/drive/MyDrive//20021010_spam.tar.bz2'
```

Extracting tarfiles to working directory

```
#Extracting all tarfiles to current woring directory

#All tarfiles
tarfile_list = [file_path_easy_ham, file_path_hard_ham, file_path_spam]

for file in tarfile_list:
#Open the tarfiles we have recieved for this assignment
  with tarfile.open(file, "r") as tf:
    #Extracting files to current working directory
    tf.extractall()
```

Reading the files

```
#Defining generic function to read and decode the files
def read_files(folder_name):

  data = []
```

```
    for filename in sorted(os.listdir(folder_name)):
        #Joining the file path together
        filepath = os.path.join(folder_name, filename)

        #Try reading with encoding utf-8
        try:
            with open(filepath, 'r', encoding='utf-8') as file:
                content = file.read()
                data.append(content)
        except UnicodeDecodeError:
            #If utf-8 decoding fails, try iso-8859-1
            try:
                with open(filepath, 'r', encoding='iso-8859-1') as file:
                    content = file.read()
                    data.append(content)
            except UnicodeDecodeError:
                #If that also fails, try ascii
                try:
                    with open(filepath, 'r', encoding='ascii') as file:
                        content = file.read()
                        data.append(content)
                except Exception as e:
                  #If none of these decodings work, show errors
                  print(f"Failed to decode {filename}: {e}")

    return data
```

```
#Reading the files
easy_ham = read_files('easy_ham')
hard_ham = read_files('hard_ham')
spam = read_files('spam')
print('Reading the files is done')
```

```
#Look at the lengths of the data
print(f'Length of Easy ham data: {len(easy_ham)}')
print(f'Length of Hard ham data: {len(hard_ham)}')
print(f'Length of Spam data: {len(spam)}')
```

```
#Looking at data
print(easy_ham[1])
print(hard_ham[7])
print(spam[7])
```

SPAM and HAM

```python
#List of data frames and corresponding labels
df_list = [easy_ham, hard_ham, spam]
labels = ['easy_ham', 'hard_ham', 'spam']

#Loop to create a dataframe
df_eh, df_hh, df_spam = [
#Converting to dataframe by renaming the 1st column as email and assigning a new column to each dataframe
    pd.DataFrame(df).rename(columns={0: 'email'}).assign(label=label)
    for df, label in zip(df_list, labels)]
```

```python
#Looking at the data, it is now divided into email and a corresponding label
print("Easy Ham DataFrame:\n", df_eh.head(100))
print("Hard Ham DataFrame:\n", df_hh.head(100))
print("Spam DataFrame:\n", df_spam.head(100))
```

```python
#Creating CSV files if we want to export the data
df_eh.to_csv("eh_content.csv")
df_hh.to_csv("hh_content.csv")
df_spam.to_csv("spam_content.csv")
```

Data Splitting

```python
#Data splitting
#X contains the full emails, while y contains the corresponding label
#can test with some different text sizes and random states

#Train-test split for easy ham
X_train_eh, X_test_eh, y_train_eh, y_test_eh = train_test_split(df_eh['email'], df_eh['label'], random_state=50)

#Train-test split for hard ham
X_train_hh, X_test_hh, y_train_hh, y_test_hh = train_test_split(df_hh['email'], df_hh['label'], random_state=50)

#Train-test split for spam
X_train_spam, X_test_spam, y_train_spam, y_test_spam = train_test_split(df_spam['email'], df_spam['label'], random_state=50)
```

```python
#Creating vectorizer for assignment 3 (easy ham) and assignment 4 (hard ham)
count_eh = CountVectorizer()
count_hh = CountVectorizer()
```

```

#Training data

#Concatenating the spam training data with the two
    different ham datas
X_train_eh_spam = pd.concat([X_train_spam, X_train_eh
    ],ignore_index=True)
X_train_hh_spam = pd.concat([X_train_spam, X_train_hh
    ],ignore_index=True)

y_train_eh_spam = pd.concat([y_train_spam, y_train_eh
    ],ignore_index=True)
y_train_hh_spam = pd.concat([y_train_spam, y_train_hh
    ],ignore_index=True)

#CountVectorizer to transform the train data
X_train_eh_spam = count_eh.fit_transform(
    X_train_eh_spam)
X_train_hh_spam = count_hh.fit_transform(
    X_train_hh_spam)

#Testing data

#Apply the same method on the test data
X_test_eh_spam = pd.concat([X_test_spam, X_test_eh],
    ignore_index=True)
X_test_hh_spam = pd.concat([X_test_spam, X_test_hh],
    ignore_index=True)

y_test_eh_spam = pd.concat([y_test_spam, y_test_eh],
    ignore_index=True)
y_test_hh_spam = pd.concat([y_test_spam, y_test_hh],
    ignore_index=True)

#CountVectorizer to transform the test data
X_test_eh_spam = count_eh.transform(X_test_eh_spam)
X_test_hh_spam = count_hh.transform(X_test_hh_spam)
```

```
#Print the shapes of the transformed test data
print("\nShape of X_test_eh_spam matrix:",
    X_test_eh_spam.shape)
print("Shape of X_test_hh_spam matrix:",
    X_test_hh_spam.shape)

#Print the first rows of the transformed test data for
     easy ham an hard ham
print("First 5 rows of the transformed test data (
    X_test_eh_spam):")
print(X_test_eh_spam.toarray()[:5])

```

```
print("First 5 rows of the transformed test data (
    X_test_hh_spam):")
print(X_test_hh_spam.toarray()[:5])
```

MultinominalNB for Easy Ham and Spam

```
#MultinominalNB - Easy ham and spam

#clf short for classifyer
clfMN_eh = MultinomialNB()
clfMN_eh.fit(X_train_eh_spam, y_train_eh_spam)

predictMN_eh = clfMN_eh.predict(X_test_eh_spam)
print(predictMN_eh)
print()

#Accuracy
#accuracy_score(y_true, y_predicted)
eh_MN_accuracy = accuracy_score(y_test_eh_spam,
    predictMN_eh)
print(f"Easy ham Accuracy: {eh_MN_accuracy:.4f}")

#Precision
#Can also use label 'easy_ham'
eh_MN_precision = precision_score(y_test_eh_spam,
    predictMN_eh, pos_label='spam')
print(f"Easy ham Precision: {eh_MN_precision:.4f}")

#Recall
#Can also use label 'easy_ham'
eh_MN_recall = recall_score(y_test_eh_spam,
    predictMN_eh, pos_label='spam')
print(f"Easy ham Recall: {eh_MN_recall:.4f}")

#Confusion matrix
eh_MN_confusion_matrix = confusion_matrix(
    y_test_eh_spam, predictMN_eh)
#Creating a confusion matrix that is easier to read
rows = ['Actual Positive', 'Actual Negative']
columns = ['Positive prediction', 'Negative prediction
    ']
confusion_MN_eh = pd.DataFrame(eh_MN_confusion_matrix,
     index=rows, columns=columns)
#Calculating the sums
confusion_MN_eh_total_columns = confusion_MN_eh.sum()
confusion_MN_eh_total_rows = confusion_MN_eh.sum(axis
    =1)
#Adding sums to matrix
confusion_MN_eh["Total"] = confusion_MN_eh_total_rows
confusion_MN_eh.loc["Total"] =
```

```
    confusion_MN_eh_total_columns
confusion_MN_eh.loc["Total", "Total"] = confusion_MN_eh.iloc[:-1, :-1].sum().sum()

print()
print("Easy ham Confusion Matrix:")
confusion_MN_eh.astype(int)
```

BernoulliNB for Easy Ham and Spam

```
#Bernoulli - Easy ham and spam

clfBN_eh = BernoulliNB()
clfBN_eh.fit(X_train_eh_spam, y_train_eh_spam)

predictBN_eh = clfBN_eh.predict(X_test_eh_spam)
print(predictBN_eh)
print()

#Accuracy
#accuracy_score(y_true, y_predicted)
eh_BN_accuracy = accuracy_score(y_test_eh_spam, predictBN_eh)
print(f"Easy ham Accuracy: {eh_BN_accuracy:.4f}")

#Precision
#Can also use label 'easy_ham'
eh_BN_precision = precision_score(y_test_eh_spam, predictBN_eh, pos_label='spam')
print(f"Easy ham Precision: {eh_BN_precision:.4f}")

#Recall
#Can also use label 'easy_ham'
eh_BN_recall = recall_score(y_test_eh_spam, predictBN_eh, pos_label='spam')
print(f"Easy ham Recall: {eh_BN_recall:.4f}")

#Confusion matrix
eh_BN_confusion_matrix = confusion_matrix(y_test_eh_spam, predictBN_eh)

#Creating a confusion matrix that is easier to read
rows = ['Actual Positive', 'Actual Negative']
columns = ['Positive prediction', 'Negative prediction']
confusion_BN_eh = pd.DataFrame(eh_BN_confusion_matrix, index=rows, columns=columns)

#Calculating the sums
confusion_BN_eh_total_columns = confusion_BN_eh.sum()
```

```
confusion_BN_eh_total_rows = confusion_BN_eh.sum(axis=1)
#Adding sums to matrix
confusion_BN_eh["Total"] = confusion_BN_eh_total_rows
confusion_BN_eh.loc["Total"] = confusion_BN_eh_total_columns
confusion_BN_eh.loc["Total", "Total"] = confusion_BN_eh.iloc[:-1, :-1].sum().sum()

print()
print("Easy ham Confusion Matrix:")
confusion_BN_eh.astype(int)
```

MultinominalNB for Hard Ham and Spam

```
#MultinominalNB - Hard ham and spam

#clf short for classifyer
clfMN_hh = MultinomialNB()
clfMN_hh.fit(X_train_hh_spam, y_train_hh_spam)

predictMN_hh = clfMN_hh.predict(X_test_hh_spam)
print(predictMN_hh)
print()

#Accuracy
#accuracy_score(y_true, y_predicted)
hh_MN_accuracy = accuracy_score(y_test_hh_spam, predictMN_hh)
print(f"Hard ham Accuracy: {hh_MN_accuracy:.4f}")

#Precision
#Can also use label 'hard_ham'
hh_MN_precision = precision_score(y_test_hh_spam, predictMN_hh, pos_label='spam')
print(f"Hard ham Precision: {hh_MN_precision:.4f}")

#Recall
#Can also use label 'hard_ham'
hh_MN_recall = recall_score(y_test_hh_spam, predictMN_hh, pos_label='spam')
print(f"Hard ham Recall: {hh_MN_recall:.4f}")

#Confusion matrix
hh_MN_confusion_matrix = confusion_matrix(y_test_hh_spam, predictMN_hh)

#Creating a confusion matrix that is easier to read
rows = ['Actual Positive', 'Actual Negative']
columns = ['Positive prediction', 'Negative prediction
```

```
    ']
confusion_MN_hh = pd.DataFrame(hh_MN_confusion_matrix,
     index=rows, columns=columns)

#Calculating the sums
confusion_MN_hh_total_columns = confusion_MN_hh.sum()
confusion_MN_hh_total_rows = confusion_MN_hh.sum(axis
    =1)
#Adding sums to matrix
confusion_MN_hh["Total"] = confusion_MN_hh_total_rows
confusion_MN_hh.loc["Total"] =
    confusion_MN_hh_total_columns
confusion_MN_hh.loc["Total", "Total"] =
    confusion_MN_hh.iloc[:-1, :-1].sum().sum()

print()
print("Hard ham Confusion Matrix:")
confusion_MN_hh.astype(int)
```

BernoulliNB for Hard Ham and Spam

```
BernoulliNB - Hard ham and spam

#clf short for classifyer
clfBN_hh = BernoulliNB()
clfBN_hh.fit(X_train_hh_spam, y_train_hh_spam)

predictBN_hh = clfBN_hh.predict(X_test_hh_spam)
print(predictBN_hh)
print()

#Accuracy
#accuracy_score(y_true, y_predicted)
hh_BN_accuracy = accuracy_score(y_test_hh_spam,
    predictBN_hh)
print(f"Hard ham Accuracy: {hh_BN_accuracy:.4f}")

#Precision
#Can also use label 'hard_ham'
hh_BN_precision = precision_score(y_test_hh_spam,
    predictBN_hh, pos_label='spam')
print(f"Hard ham Precision: {hh_BN_precision:.4f}")

#Recall
#Can also use label 'hard_ham'
hh_BN_recall = recall_score(y_test_hh_spam,
    predictBN_hh, pos_label='spam')
print(f"Hard ham Recall: {hh_BN_recall:.4f}")

#Confusion matrix
```

```
hh_BN_confusion_matrix = confusion_matrix(y_test_hh_spam, predictBN_hh)

#Creating a confusion matrix that is easier to read
rows = ['Actual Positive', 'Actual Negative']
columns = ['Positive prediction', 'Negative prediction']
confusion_BN_hh = pd.DataFrame(hh_BN_confusion_matrix, index=rows, columns=columns)

#Calculating the sums
confusion_BN_hh_total_columns = confusion_BN_hh.sum()
confusion_BN_hh_total_rows = confusion_BN_hh.sum(axis=1)
#Adding sums to matrix
confusion_BN_hh["Total"] = confusion_BN_hh_total_rows
confusion_BN_hh.loc["Total"] = confusion_BN_hh_total_columns
confusion_BN_hh.loc["Total", "Total"] = confusion_BN_hh.iloc[:-1, :-1].sum().sum()

print()
print("Hard ham Confusion Matrix:")
confusion_BN_hh.astype(int)
```