DAT565/DIT407 Assignment 2

Chunqiu Xia
Chunqiu@student.chalmers.se

Rami Ghalib
ghalib@chalmers.se

November 24, 2024

Abstract

This report is aimed to provide us with a comprehensive understanding of key data science and machine learning concepts through hands-on practice. The focus is on using the Bag of Words model for preprocessing and applying two types of Naive Bayes classifiers—Multinomial and Bernoulli—to classify email datasets as spam or ham. Through data exploration, we will understand the different characteristics of spam and ham emails, as well as the distinctions between easy and hard ham. The assignment emphasizes key skills such as data preprocessing, feature extraction, correct train-test splitting, and handling various encoding formats. By evaluating the model performance based on accuracy, precision, recall, and confusion matrices, we will gain deeper insights into the effectiveness of different machine learning algorithms in text classification tasks.

1 Problem 1: Spam and Ham

1.1 Question A

After exploring the three datasets(**Examples below are all from 0001 files in the three datasets**), we could divide the task into two parts. One part is to distinguish between spam and ham, the other part is to distinguish between easy ham and hard ham.

For the distinguish between spam and ham, spam emails usually contain a lot of promotional content, such as in the example "Life Insurance - Why Pay More?" email. It uses exaggerated promotional phrases like "free" and "save up to 70%" with the primary goal of leading the recipient to click certain links or purchase services. In addition, spam emails often use complex HTML structures, insert numerous advertisements, and frequently use images and embedded links to attract user clicks. The sender's information is often unreliable, usually appearing as auto-generated, and the email headers often show a complex route, potentially passing through multiple servers to evade filters.

Next part is about the difference between easy ham and hard ham, easy Ham emails are relatively straightforward in content and simple in structure, without containing a lot of advertisements or unnecessary links. Their purpose is to effectively convey information, such as discussions about software development issues. These types of emails are typically circulated within specific groups, are professional in content, and have a clear objective.

Hard Ham emails, like the "CNET Technology News Subscription," have certain characteristics that make them resemble spam. They contain a lot of advertising content, use HTML structures, and include many promotional links and information, making the content quite complex. Although these emails have promotional aspects, they are not sent randomly but are based on user subscriptions and provide valuable news or product information. Thus, while they may resemble spam in terms of format, they hold value for the recipient. This makes it challenging for classifiers to differentiate them from spam.

1.2 Question B

In this part, we are requested to split the datasets for training and testing parts. In this situation, we could use function 'train_test_split' to split the datasets [2]. After running the codes, we could print out the number of the training emails and testing emails to see whether the process is true. Figure 1 proves it is correct.

```
D:\Anaconda\python.exe D:\PCwork\DAT565_A2\pythonProject3\Assignment3\Problem1.py
Training set size: 2641
Test set size: 661
```

Figure 1: Distribution of Training and Test Set Sizes After Splitting

This means that the process of splitting works well!

2 Problem 2: Preprocessing

In this part, we started by initializing 'CountVectorizer' to transform text data into a Bag of Words representation. Then, we used the 'fit_transform()' method to convert the emails in the training set into a feature matrix, allowing the model to learn all unique words from the training data and generate corresponding word frequency vectors. Next, we used the 'transform()' method on the test set to convert the test emails into vectors based on the same vocabulary, ensuring that both the training and test data share the same feature space. In this assignment, we processed the entire email content without separating the header from the body, which means we simply loaded the complete email text and passed it to 'CountVectorizer' [2] for consistent processing.

To conclude, we ran the codes and got the outputs as follow:

```
D:\Anaconda\python.exe D:\PCwork\DAT565_A2\pythonProject3\Assignment3\Problem2.py
Training set shape: (2641, 98875)
Test set shape: (661, 98875)
```

Figure 2: Preprocessed Training and Test Set Sizes

Still it seems that we have preprocessed well!

3 Problem 3: Easy Ham

In this problem, we are requested to distinguish between easy ham emails and spam emails. In problem 1, we have split the datasets into training datasets and testing datasets. In problem 2, we have preprocessed the datasets. Thus, we could straightly use the easy ham and spam labels and characteristic matrices to train and test the two classifiers [1]. However, what is worth mentioning is that we ought to filter the data for harm ham emails in both labels and matrices. We use function 'isin()' to do the filtering job.

```
D:\Anaconda\python.exe D:\PCwork\DAT565_A2\pythonProject3\Assignment3\Problem3.py
Multinomial Naive Bayes Classifier Results:
Accuracy: 0.9755
Precision: 1.0000
Recall: 0.8500
Confusion Matrix:
[[511   0]
 [ 15  85]]

Bernoulli Naive Bayes Classifier Results:
Accuracy: 0.9002
Precision: 0.9535
Recall: 0.4100
Confusion Matrix:
[[509   2]
 [ 59  41]]
```

Figure 3: Classification Results for Easy Ham and Spam Emails

Multinomial Naive Bayes Classifier

- Accuracy is 0.9755;
- precision is 1
- recall is 0.85

	Predicted Spam	**Predicted Ham**	**Total**
Actual Spam	511 (TP)	0 (FN)	511
Actual Ham	15 (FP)	85 (TN)	100
Total	526	85	611

Table 1: Confusion Matrix for Multinomial Naive Bayes Classifier - Easy Ham vs Spam

Bernoulli Naive Bayes Classifier

- Accuracy is 0.9002;
- Precision of it is 0.9535;
- Recall of it is 0.41;

	Predicted Spam	Predicted Ham	Total
Actual Spam	509 (TP)	2 (FN)	511
Actual Ham	59 (FP)	41 (TN)	100
Total	568	43	611

Table 2: Confusion Matrix for Bernoulli Naive Bayes Classifier - Easy Ham vs Spam

4 Problem 4: Hard Ham

In this section, we are requested to train the two models with the hard ham dataset and the spam dataset. From these two datasets, we could still use the method from Problem 3, filtering the data of easy ham in both labels and matrices. Then, after running the codes, the outputs are as follows:

The difference between hard ham and easy ham lies in the complexity of distinguishing the datasets from spam. This was evident when comparing the BNB classifier between Problems 3 and 4. For Problem 3, the BNB achieved 90.02 % accuracy, 95.35% precision, and 41% recall. On the other hand, the BNB in Problem 4 had an accuracy of 88.2%, 89.64% precision, and 88.2% recall. The accuracy and precision are slightly higher for easy ham than hard ham, but there is a huge difference in recall. This indicates that the model misses a lot of spam emails in Problem 3.

For the MNB classifier, the accuracy and precision are marginally better for hard ham than easy ham. Furthermore, there is a significant difference in recall: 98.18% (hard ham) vs. 85% (easy ham), albeit the difference is smaller than for the BNB classifier.

Hence, the performance in classifying hard ham vs. easy ham is better for both classifiers. MNB is particularly efficient in classifying hard ham, likely due to its ability to leverage high-frequency-based distinctions.

```
D:\Anaconda\python.exe D:\PCwork\DAT565_A2\pythonProject3\Assignment3\Problem4.py
Multinomial Naive Bayes Classifier Results:
Accuracy: 0.9818
Precision: 0.9822
Recall: 0.9818
Confusion Matrix:
[[561   0]
 [ 12  88]]

Bernoulli Naive Bayes Classifier Results:
Accuracy: 0.8820
Precision: 0.8964
Recall: 0.8820
Confusion Matrix:
[[561   0]
 [ 78  22]]
```

Figure 4: Classification Results for Hard Ham and Spam Emails

Multinomial Naive Bayes Classifier

- Accuracy is 0.9818;
- Precision is 0.9822;
- Recall is 0.9818;

	Predicted Spam	**Predicted Ham**	**Total**
Actual Spam	561 (TP)	0 (FN)	561
Actual Ham	12 (FP)	88 (TN)	100
Total	573	88	661

Table 3: Confusion Matrix for Multinomial Naive Bayes Classifier - Hard Ham vs Spam

Bernoulli Naive Bayes Classifier

- Accuracy is 0.8820;
- Precision is 0.8964;
- Recall is 0.8820;

	Predicted Spam	**Predicted Ham**	**Total**
Actual Spam	561 (TP)	0 (FN)	561
Actual Ham	78 (FP)	22 (TN)	100
Total	639	22	661

Table 4: Confusion Matrix for Bernoulli Naive Bayes Classifier - Hard Ham vs Spam

References

[1] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. *Scikit-Learn: Machine Learning in Python*. Retrieved from https://scikit-learn.org/stable/. 2011.

A Python Code for Analysis

The following code was used for data processing and analysis. For the first part, the codes used to solve the problem are as follow:
Problem 1:

```
import tarfile
import os
import pandas as pd
import warnings
from sklearn.model_selection import train_test_split

warnings.filterwarnings("ignore", category=DeprecationWarning)
def extract_tar_files(file_path, extract_path):
    with tarfile.open(file_path, 'r:bz2') as tar:
        tar.extractall(path=extract_path)
def load_emails_from_folder(folder_path):
    emails = []
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            file_path = os.path.join(root, file)
            with open(file_path, 'r', encoding='latin1') as f:
                emails.append(f.read())
    return emails

easy_ham_path = 'D:/program files/DATA3/20021010_easy_ham.tar.bz2'
hard_ham_path = 'D:/program files/DATA3/20021010_hard_ham.tar.bz2'
spam_path = 'D:/program files/DATA3/20021010_spam.tar.bz2'
extract_dir = 'D:/program files/DATA3/Afterya'
os.makedirs(extract_dir, exist_ok=True)
extract_tar_files(easy_ham_path, extract_dir + '/easy_ham')
extract_tar_files(hard_ham_path, extract_dir + '/hard_ham')
extract_tar_files(spam_path, extract_dir + '/spam')
easy_ham_emails = load_emails_from_folder(extract_dir +
    '/easy_ham')
hard_ham_emails = load_emails_from_folder(extract_dir +
    '/hard_ham')
spam_emails = load_emails_from_folder(extract_dir + '/spam')
data = easy_ham_emails + hard_ham_emails + spam_emails
labels = ["easy_ham"] * len(easy_ham_emails) + ["hard_ham"] *
    len(hard_ham_emails) + ["spam"] * len(spam_emails)
df = pd.DataFrame({'email': data, 'label': labels})
# Train-test split
train_set, test_set = train_test_split(df, test_size=0.2,
    random_state=42, stratify=df['label'])
train_set.to_csv('D:/program files/DATA3/train_set.csv',
    index=False)
test_set.to_csv('D:/program files/DATA3/test_set.csv', index=False)
print(f"Training set size: {len(train_set)}")
print(f"Test set size: {len(test_set)}")
```

Problem 2:

```
    import pandas as pd
import scipy.sparse
from sklearn.feature_extraction.text import CountVectorizer

train_set = pd.read_csv('D:/program files/DATA3/train_set.csv')
test_set = pd.read_csv('D:/program files/DATA3/test_set.csv')
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(train_set['email'])
X_test_counts = vectorizer.transform(test_set['email'])
print(f"Training set shape: {X_train_counts.shape}")
print(f"Test set shape: {X_test_counts.shape}")
scipy.sparse.save_npz('D:/program files/DATA3/X_train_counts.npz',
    X_train_counts)
scipy.sparse.save_npz('D:/program files/DATA3/X_test_counts.npz',
    X_test_counts)
```

Problem 3:

```
    import pandas as pd
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, confusion_matrix
import scipy.sparse

train_set = pd.read_csv('D:/program files/DATA3/train_set.csv')
test_set = pd.read_csv('D:/program files/DATA3/test_set.csv')
train_set = train_set[train_set['label'].isin(['easy_ham',
    'spam'])]
test_set = test_set[test_set['label'].isin(['easy_ham', 'spam'])]
X_train_counts = scipy.sparse.load_npz('D:/program
    files/DATA3/X_train_counts.npz')
X_test_counts = scipy.sparse.load_npz('D:/program
    files/DATA3/X_test_counts.npz')
X_train_counts = X_train_counts[train_set.index, :]
X_test_counts = X_test_counts[test_set.index, :]
y_train = train_set['label']
y_test = test_set['label']

multinomial_nb = MultinomialNB()
bernoulli_nb = BernoulliNB()
multinomial_nb.fit(X_train_counts, y_train)
y_pred_mnb = multinomial_nb.predict(X_test_counts)
accuracy_mnb = accuracy_score(y_test, y_pred_mnb)
precision_mnb = precision_score(y_test, y_pred_mnb,
    pos_label='spam', zero_division=1)
recall_mnb = recall_score(y_test, y_pred_mnb, pos_label='spam',
    zero_division=1)
confusion_mnb = confusion_matrix(y_test, y_pred_mnb)

print("Multinomial Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy_mnb:.4f}")
print(f"Precision: {precision_mnb:.4f}")
print(f"Recall: {recall_mnb:.4f}")
print("Confusion Matrix:")
print(confusion_mnb)

bernoulli_nb.fit(X_train_counts, y_train)
```

```
y_pred_bnb = bernoulli_nb.predict(X_test_counts)
accuracy_bnb = accuracy_score(y_test, y_pred_bnb)
precision_bnb = precision_score(y_test, y_pred_bnb,
    pos_label='spam', zero_division=1)
recall_bnb = recall_score(y_test, y_pred_bnb, pos_label='spam',
    zero_division=1)
confusion_bnb = confusion_matrix(y_test, y_pred_bnb)

print("\nBernoulli Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy_bnb:.4f}")
print(f"Precision: {precision_bnb:.4f}")
print(f"Recall: {recall_bnb:.4f}")
print("Confusion Matrix:")
print(confusion_bnb)
```

Problem 4:

```
    import pandas as pd
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, confusion_matrix
import scipy.sparse

train_set = pd.read_csv('D:/program files/DATA3/train_set.csv')
test_set = pd.read_csv('D:/program files/DATA3/test_set.csv')
X_train_counts = scipy.sparse.load_npz('D:/program
    files/DATA3/X_train_counts.npz')
X_test_counts = scipy.sparse.load_npz('D:/program
    files/DATA3/X_test_counts.npz')
y_train = train_set['label'].replace('easy_ham', 'hard_ham')
y_test = test_set['label'].replace('easy_ham', 'hard_ham')

multinomial_nb = MultinomialNB()
bernoulli_nb = BernoulliNB()
multinomial_nb.fit(X_train_counts, y_train)
y_pred_mnb = multinomial_nb.predict(X_test_counts)
accuracy_mnb = accuracy_score(y_test, y_pred_mnb)
precision_mnb = precision_score(y_test, y_pred_mnb,
    average='weighted', zero_division=1)
recall_mnb = recall_score(y_test, y_pred_mnb, average='weighted',
    zero_division=1)
confusion_mnb = confusion_matrix(y_test, y_pred_mnb)

print("Multinomial Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy_mnb:.4f}")
print(f"Precision: {precision_mnb:.4f}")
print(f"Recall: {recall_mnb:.4f}")
print("Confusion Matrix:")
print(confusion_mnb)

bernoulli_nb.fit(X_train_counts, y_train)
y_pred_bnb = bernoulli_nb.predict(X_test_counts)
accuracy_bnb = accuracy_score(y_test, y_pred_bnb)
precision_bnb = precision_score(y_test, y_pred_bnb,
    average='weighted', zero_division=1)
recall_bnb = recall_score(y_test, y_pred_bnb, average='weighted',
    zero_division=1)
confusion_bnb = confusion_matrix(y_test, y_pred_bnb)

print("\nBernoulli Naive Bayes Classifier Results:")
print(f"Accuracy: {accuracy_bnb:.4f}")
```

```
print(f"Precision: {precision_bnb:.4f}")
print(f"Recall: {recall_bnb:.4f}")
print("Confusion Matrix:")
print(confusion_bnb)
```