Assignment 4: Spam classification using Naïve Bayes

- Neha Devi Shakya: 13h
- Sarvesh Meenowa: 13h

1. Preprocessing: Note that the email files contain a lot of extra information, besides the actual message. Ignore that for now and run on the entire text. Further down (in the higher-grade part), you will be asked to filter out the headers and footers.

a. Load the datasets:

We decided to include the emails from all the forlders given in the link.

```
# use sklearn load_files to have data on easy ham vs spam
emails_easy_spam = load_files("./data/raw/easy_ham_vs_spam_data/", encoding = "latin-1")

# use sklearn load_files to have data on hard ham vs spam
emails_hard_spam = load_files("./data/raw/hard_ham_vs_spam_data/", encoding="latin-1")
```

Below is the distribution of emails are in each category:

Easy Ham	Hard Ham	Spam
6452	501	2399

b. Split the spam and the ham datasets in a training set and a test set:

```
x_easy_spam_train, x_easy_spam_test, y_easy_spam_train, y_easy_spam_test =
train_test_split(emails_easy_spam.data, emails_easy_spam.target, test_size = 0.2,
random_state = 0)
x_hard_spam_train, x_hard_spam_test, y_hard_spam_train, y_hard_spam_test =
train_test_split(emails_hard_spam.data, emails_hard_spam.target, test_size = 0.2,
random_state = 0)
```

---

2. For this question you will write a function in Python with the following name and arguments:

def run_naiveBayes(ham_train, ham_test, spam_train, spam_test):

Below is the code for the function:

```
def run_naiveBayes(ham_train, ham_test, spam_train, spam_test, model_name):
    if model_name == "MultinomialNB":
        # Transform the data into vectors with CountVectorizer
        CV = CountVectorizer()
        x_train = CV.fit_transform(ham_train)
        x_test = CV.transform(ham_test)
        y_train = spam_train.astype("int")

        # Train the MultinomialNB classifier
        model = MultinomialNB().fit(x_train, y_train)

    elif model_name == "BernoulliNB":
        # Transform the data into vectors with CountVectorizer with binary = True
        CV = CountVectorizer(binary = True)
        x_train = CV.fit_transform(ham_train)
        x_test = CV.transform(ham_test)
        y_train = spam_train.astype("int")

        # Train the BernouilliNB classifier
        model = BernoulliNB().fit(x_train, y_train)

    # predict spam or ham
    y_pred = model.predict(x_test)
    # calculate accuracy score
    acc = accuracy_score(spam_test, y_pred)
    # create confusion matrix
    cm = confusion_matrix(spam_test, y_pred)
    # create classification report
    cr = classification_report(spam_test,y_pred)

    return acc, cm, cr
```

Discuss the differences between Multinomial Naive Bayes and Bernouilli Naive Bayes

The Bernoulli Naive Nayes classifier performs the classifications of the emails based on words(features) occuring or not. In general,it treats features(words) as binary values whereas the Multinomial Naive Bayes classifier calculates the frequency with which a feature(a word in context of this classification) appears.

For instance, if we take two snippets of two emails :

- Text 1 : "Invest in crypto crypto crypto"
- Text 2 : "Invest in crypto"

Bernoulli NB would classify Text 1 and Text 2 similarly since it checks if the feature "crypto" is there but doesn't account for the frequency, which Multinomial NB will account for. The way the classifiers work differs, which has an impact on how well they perform. From the preceding example, it is plausible explanation why the multinomial classifier outperforms the bernoulli classifier as we see later in part 3.

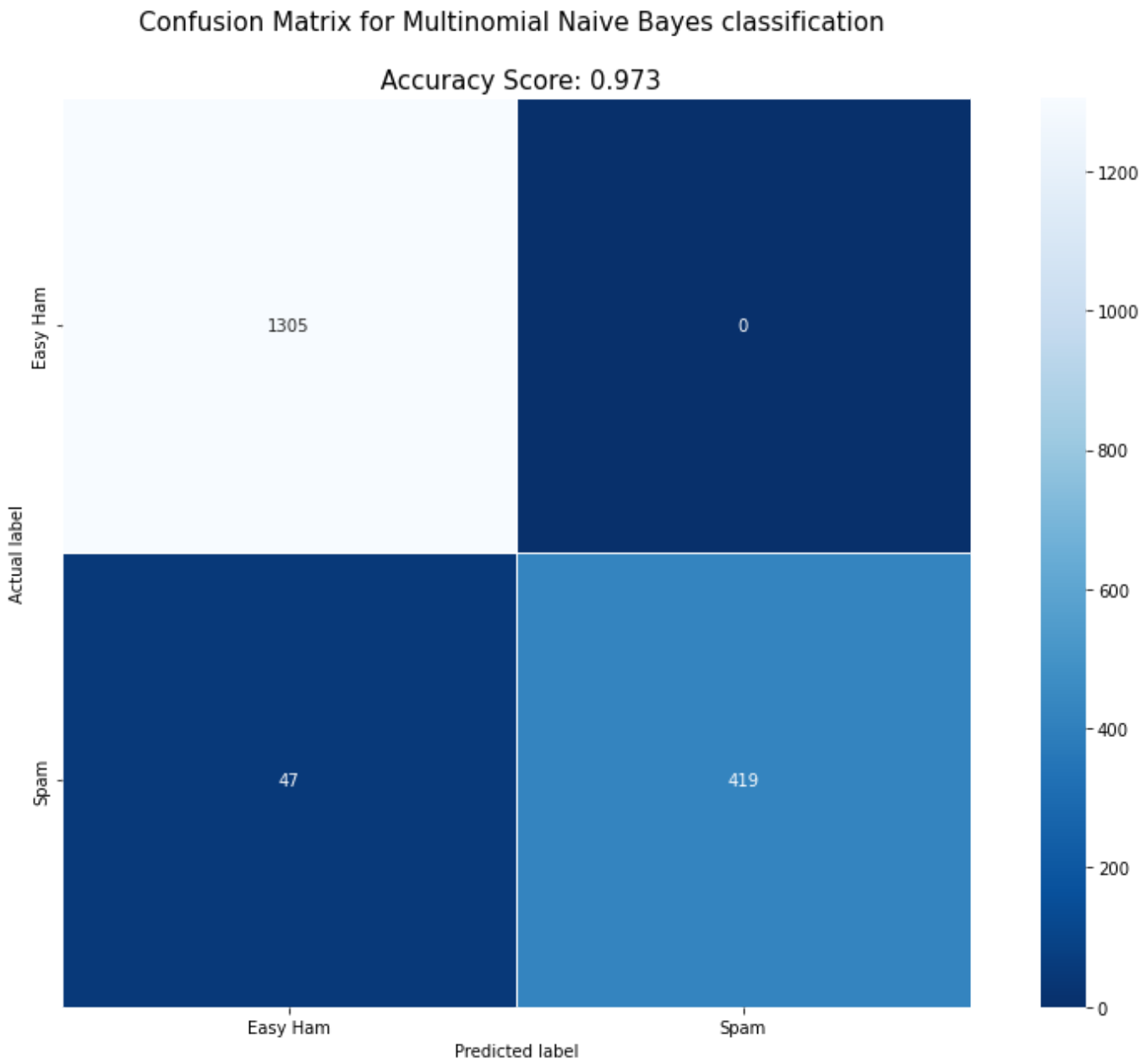
For the Bernoulli NB classifier we dicided to stick to the default value for the parameter **binarize = 0.0**.

---

3. Run the function you wrote in the previous question on

i. Spam versus easy-ham

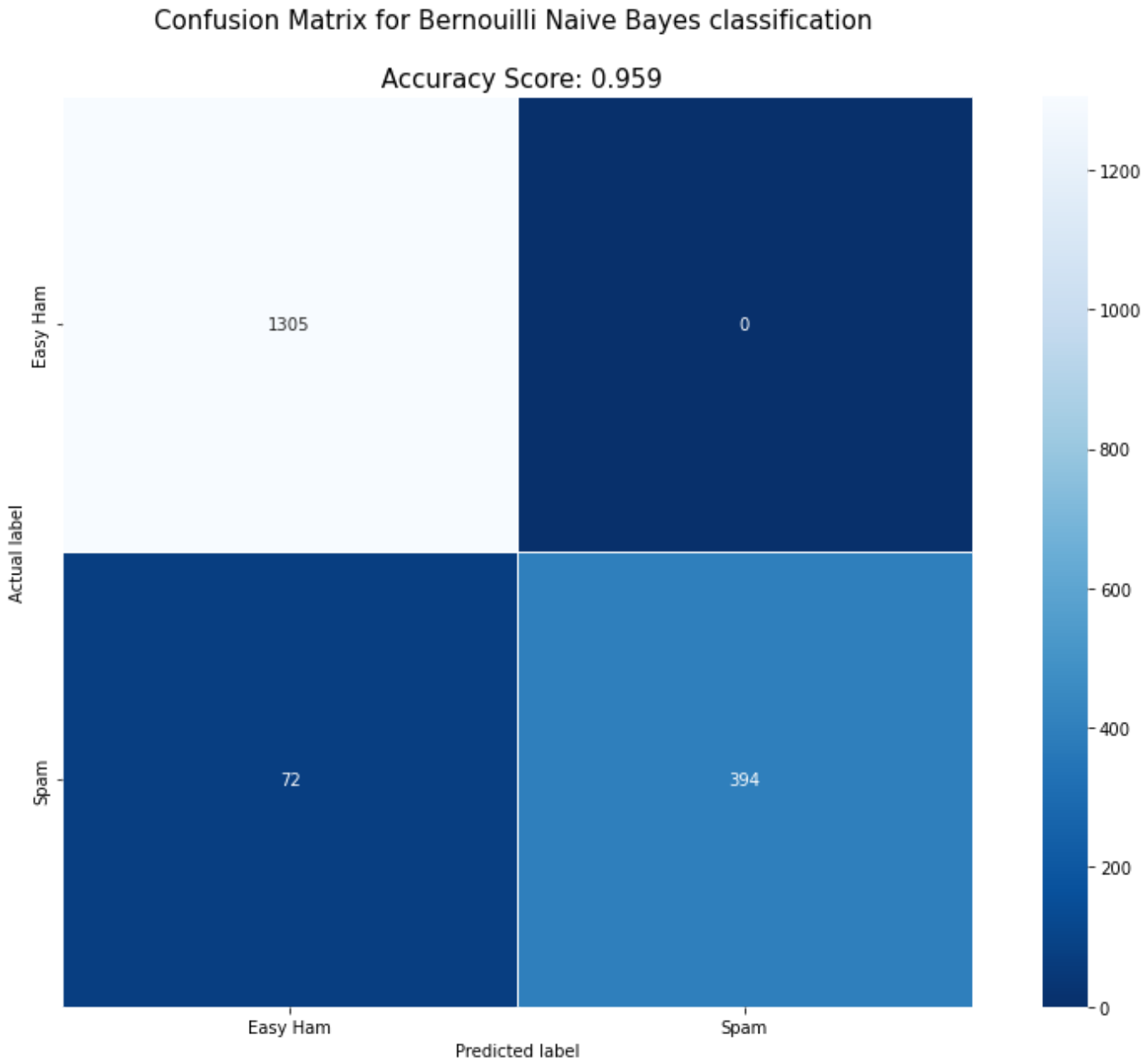
Multinomial Naive Bayes



With the Multinomial Naive Bayes classifier, we obtained an accuracy of 0.97 when classifying easy ham and spam. However, the value of this accuracy may be misleading since the classification accuracy fails on classification problems with a skewed class distribution(more hams than spams), which means we have to use other evaluation metrics such as precision and recall.

To obtain precision and recall, we use the classification report function from sklearn. We consider the precision and recall for the positive class, i.e. spam. The precision is very high at 1.00, which means it has no false positives i.e. there are no actual hams are classified as spams. The model has lower recall which means that it has more false negatives(FN = 47), i.e. spams are misclassified as hams. There is a trade-off between precision and recall. However, to know which one (precision or recall) we want to be higher depends on the context and which one would consider worse, having more spams classified as hams or more hams classified as spams.

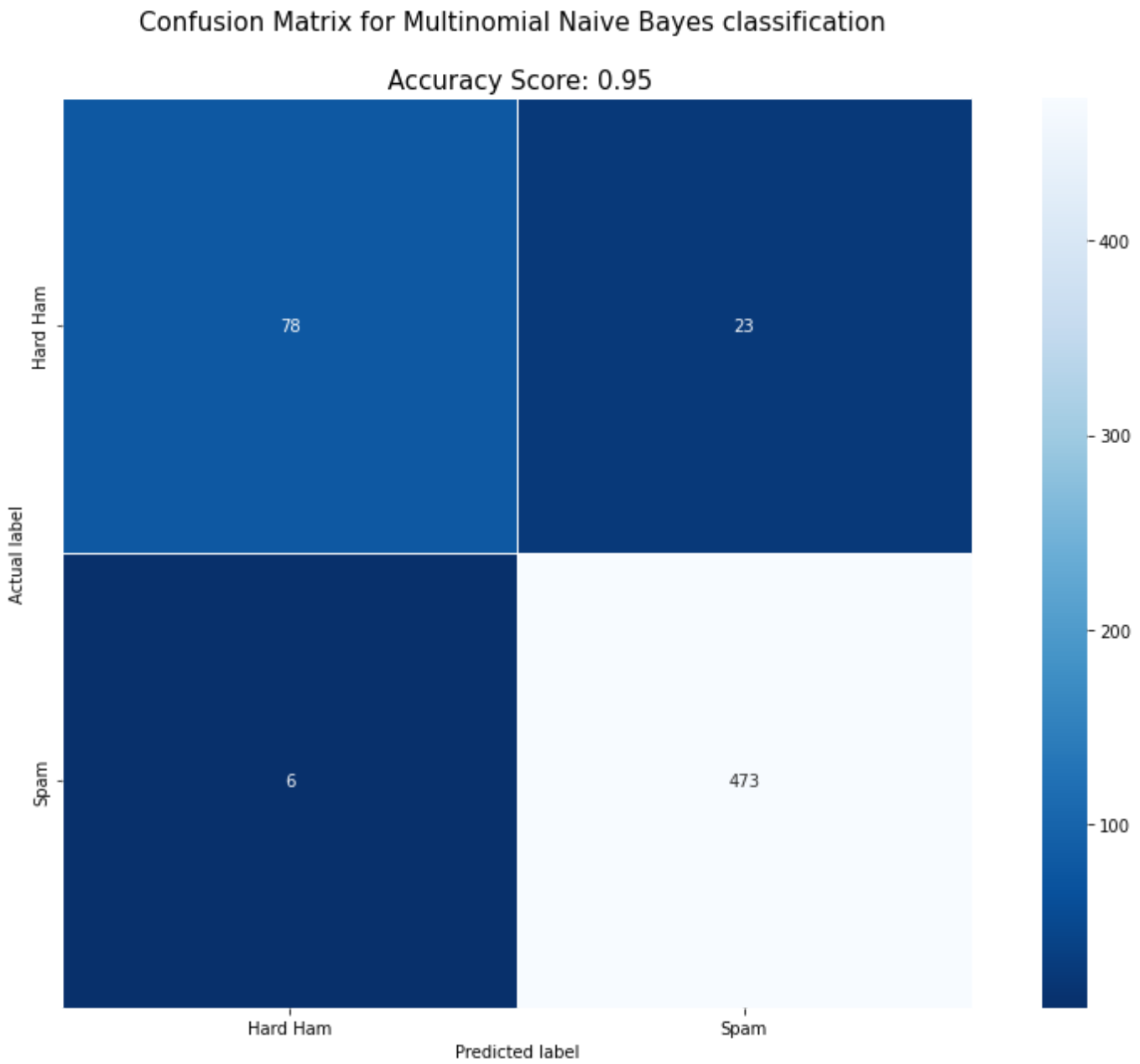
Bernouilli Naive Bayes



Despite having a relatively high accuracy (0.96), the Bernoulli NB classifier has a relatively low recall for the positive class, i.e. spam; that is, it has a high number of false-negative (FN = 72), i.e. spams are misclassified as easy hams.

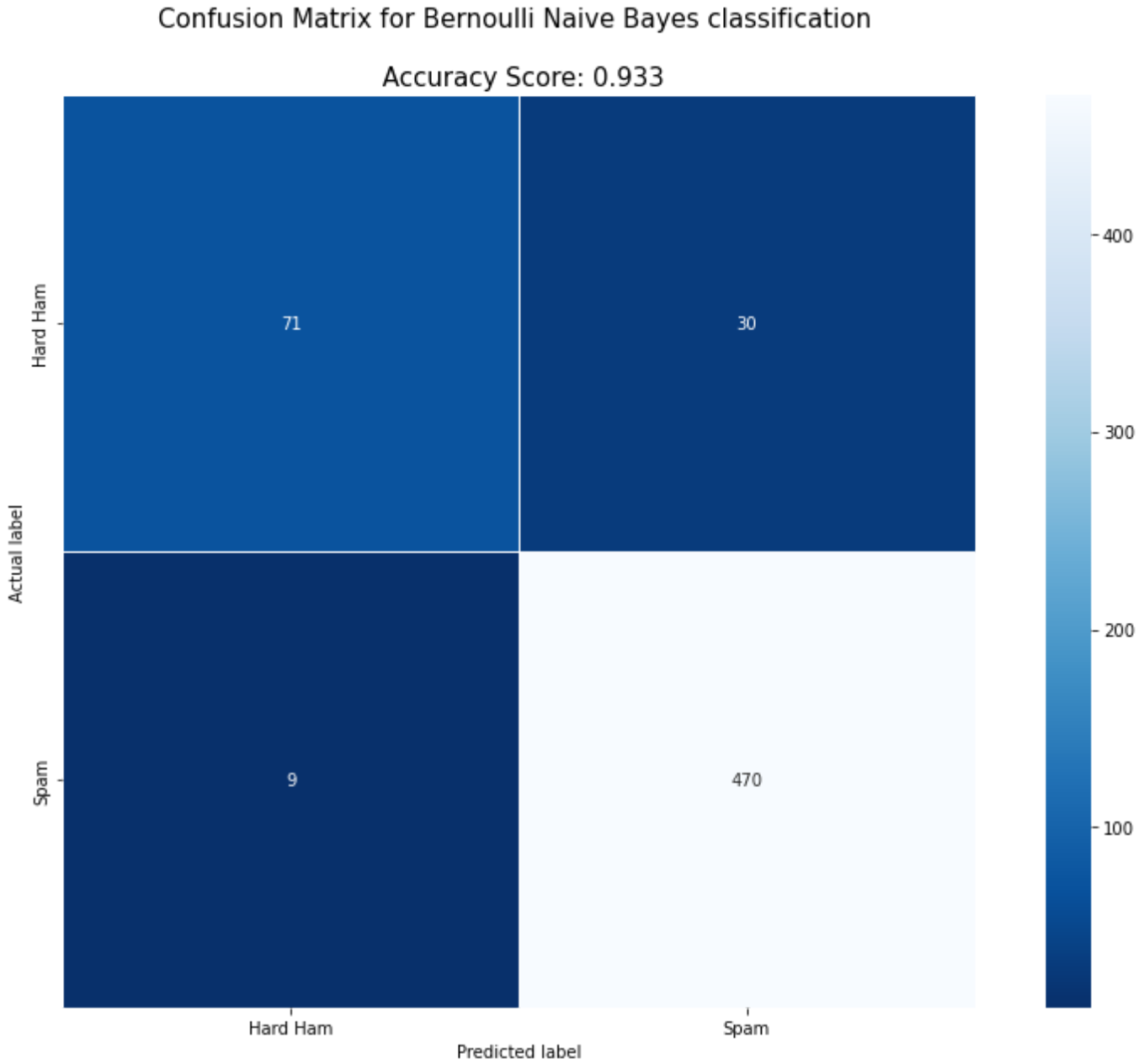
ii. Spam versus hard-ham

Multinomial Naive Bayes



Multinomial Naive Bayes performs well accuracy wise (0.95) and has both precision and recall close to 1, which is the ideal scenario in the classification of hard ham and spam. This time the accuracy score is closer to the f1-score (or precision/recall) since the dataset is less imbalanced than easy ham and spam.

Bernouilli Naive Bayes



Bernoulli NB has an accuracy of 0.93 in hard ham and spam classification. The model has a very high recall and lower precision, meaning it has more false positives, i.e. it misclassifies hard hams as spams.

Overall Result

Classifier	Ham Type	Part 3						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.97	0.97	0.99	1.00	0.82	0.98	0.89
	Hard	0.91	0.90	0.91	0.84	0.95	0.84	0.93
Bernoulli Naive Bayes	Easy	0.90	0.90	0.90	0.99	0.4	0.94	0.55
	Hard	0.85	0.90	0.83	0.66	0.96	0.76	0.89

Overall, Multinomial Naive Bayes performs better than Bernoulli Naive Bayes in classifying easy ham and hard ham versus spam, which was expected due to how the algorithms operate, as mentioned above.

---

4. To avoid classification based on common and uninformative words it is common to filter these out.

Below is the code snippet for the function:

```
def get_frequent_words(corpus, order, n = None):
    vector = CountVectorizer().fit(corpus)
    bag_of_words = vector.transform(corpus)
    words_sum = bag_of_words.sum(axis = 0)
    words_frequency = [(word, words_sum [0, idx]) for word, idx in
vector.vocabulary_.items()]

    if order.lower() == "common":
        words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse =
True)
    elif order.lower() == "uncommon":
        words_frequency = sorted(words_frequency, key = lambda x: x[1], reverse =
False)

    return words_frequency[:n]
```

a. Argue why this may be useful. Try finding the words that are too common/uncommon in the dataset.

Those words are known as stopwords. They occur in abundance in any human language. Stopwords refer to the words in any language which does not add much meaning to a sentence. By removing the low-level information from our dataset, we can give more focus to the important information. As a result, removing stopwords may aid performance by reducing the number of tokens available to only the most important ones. Also, removing stopwords decreases the data size, and the training time also decreases with no discernible effect on the model "s accuracy.

While libraries such as NLTK and Spacey have a corpus of stopwords which generalizes common words used in different languages, it will remove words such as "the", "is", and "and". However, in sklearn, for CountVectorizer (which we will use in 3 (b), we have an in-built parameter to control stopwords because they either :

* occurred in too many documents (max_df)

* occurred in too few documents (min_df)

* were cut off by feature selection (max_features).

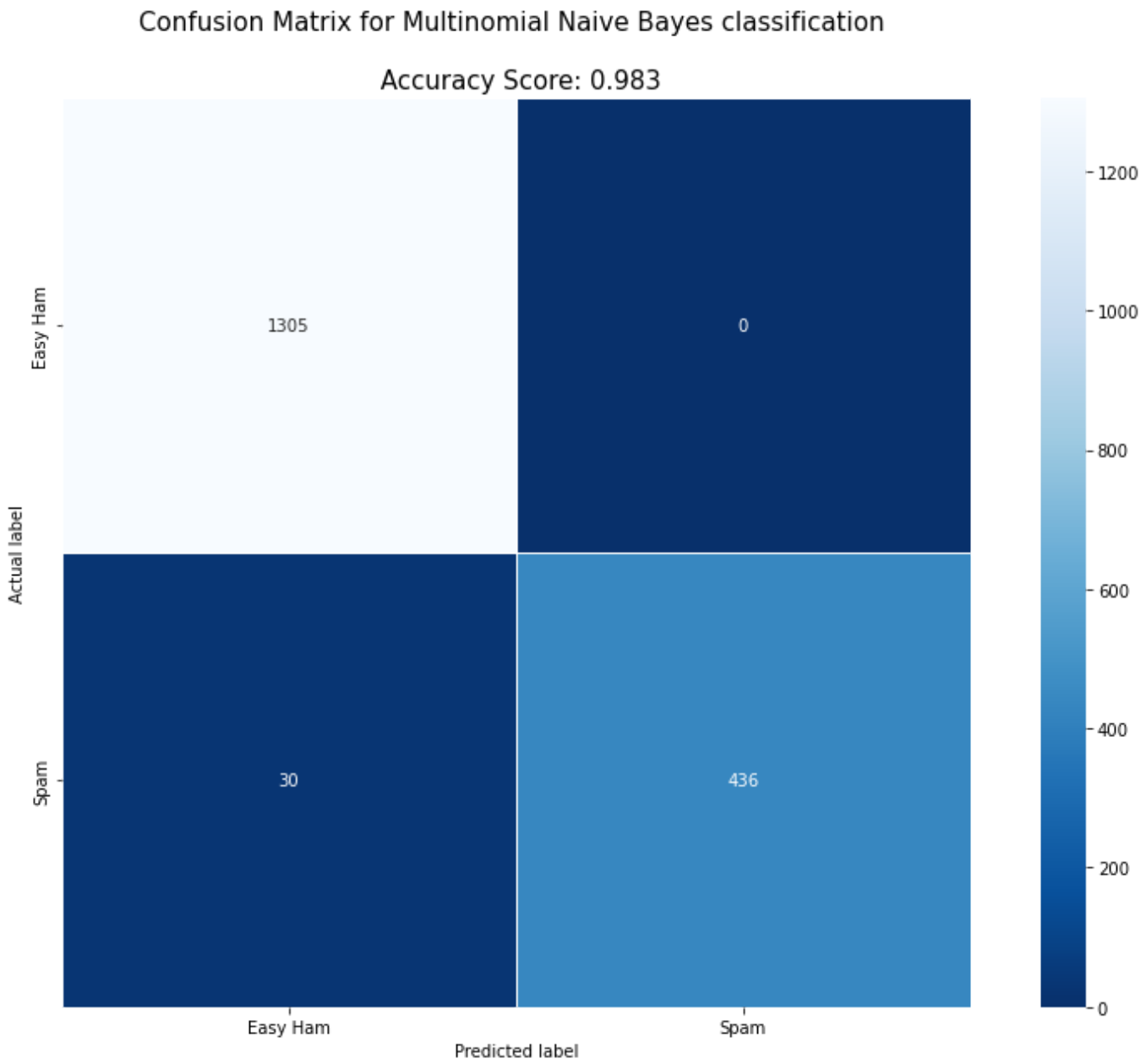
If we use the stop_words = "english" parameter in CountVectorize, sklearn acknowledges that there are several known issues with 'english' and considers an alternative such as using (min_df,max_df) = (0,0.7). In 3 (b), we tried for the CountVectorizer() with (min_df,max_df) = (0,0.7); however, if we want to optimize the model, we can perform hyperparameter tuning to obtain min_df and max_df values, which give better results in terms of precision, recall or f1-score.

The most uncommon words were strings of numbers and letters i.e. meaningless words. Some most common words were com and net which could indicate the presence of links in the emails.

b. Use the parameters in Sklearn's CountVectorizer to filter out these words. Run the updated program on your data and record how the results differ from 3.

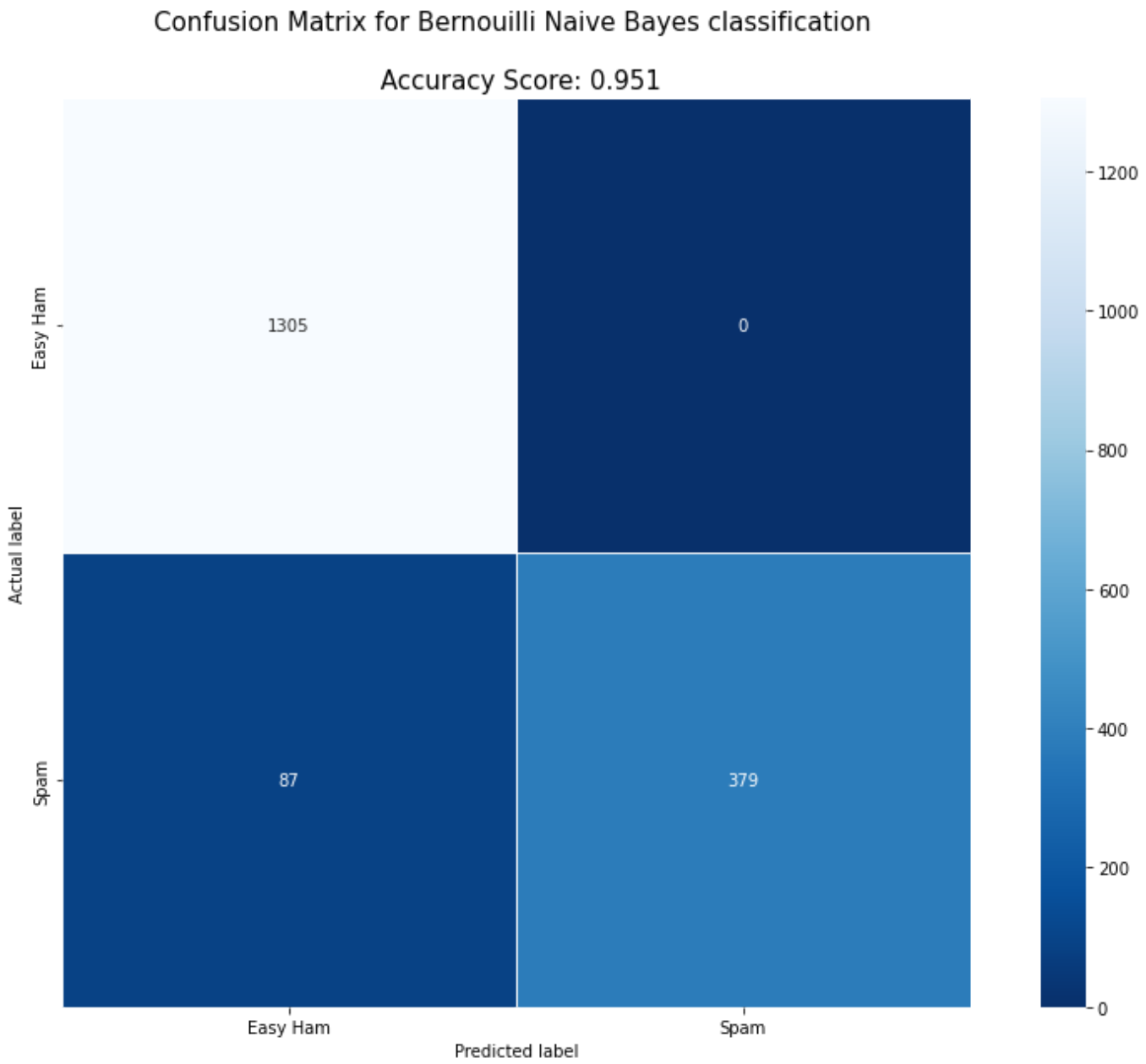
i. Spam versus easy-ham

Multinomial Naive Bayes



With the Multinomial Naive Bayes classifier, we obtained an accuracy of 0.98 when classifying easy ham and spam. We consider the precision and recall for the positive class, i.e. spam. The precision is at 1.00, which means it has no false positives that are actual hams are classified as spams. The model has lower recall which means that it has more false negatives(FN = 30), i.e. spams are misclassified as hams.

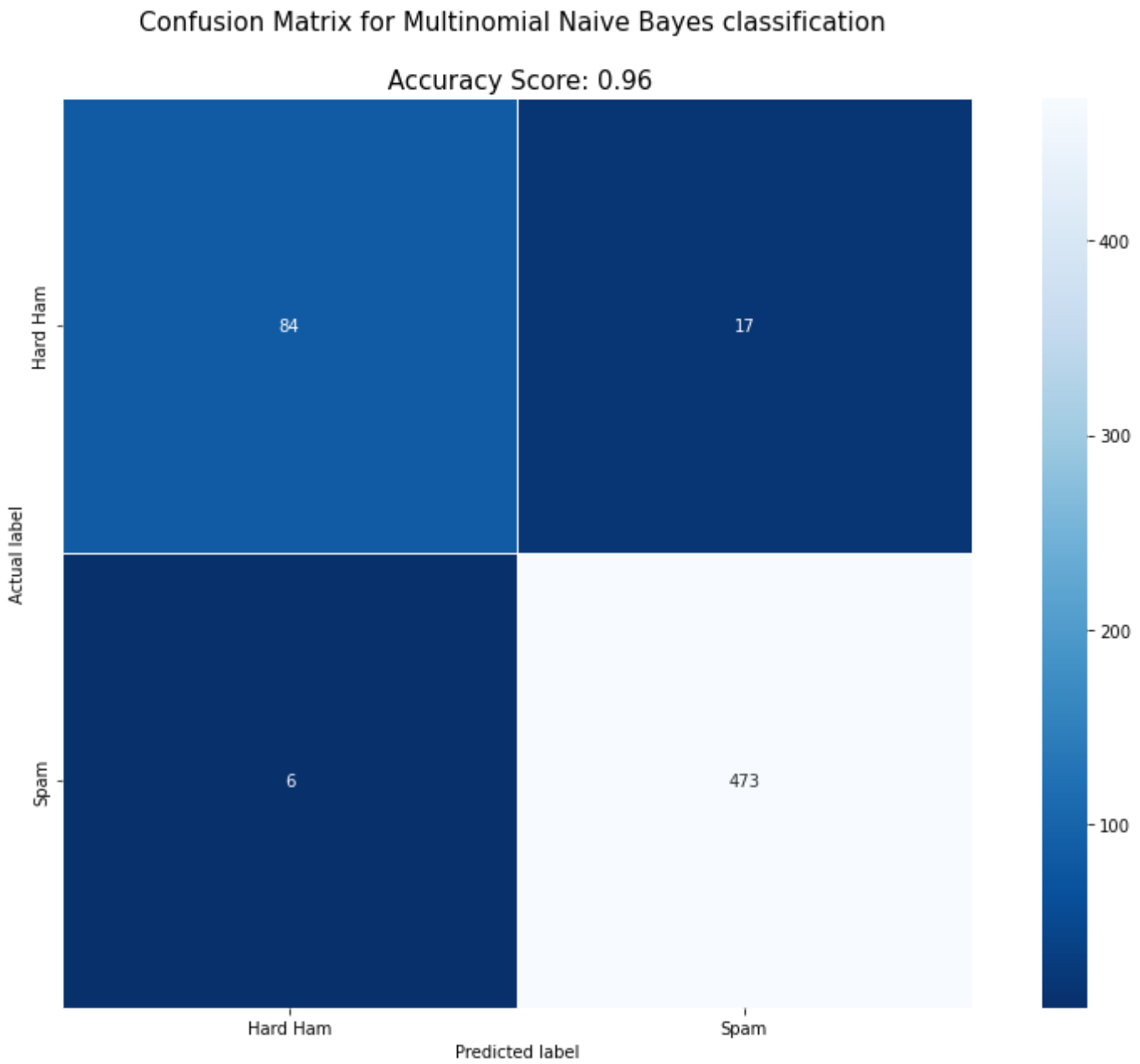
Bernouilli Naive Bayes



Despite having a relatively high accuracy (0.95), the Bernoulli NB classifier has a relatively low recall for the positive class, i.e. spam; that is, it has a high number of false-negative (FN = 87), i.e. spams are misclassified as easy hams.

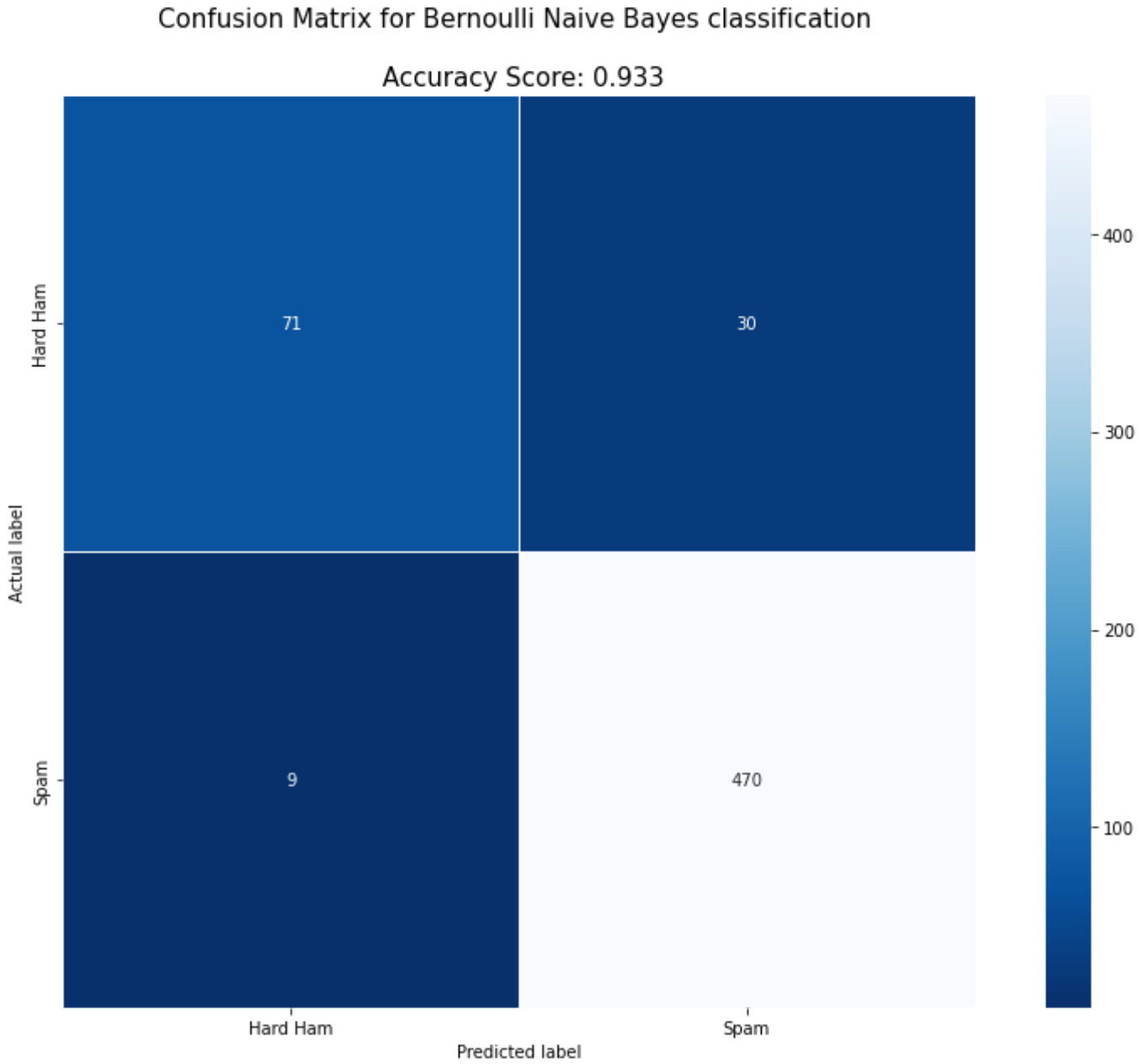
ii. Spam versus hard-ham

Multinomial Naive Bayes



Multinomial Naive Bayes performs well accuracy wise (0.96) and has both precision and recall close to 1, which is the ideal scenario in the classification of hard ham and spam. This time the accuracy score is closer to the f1-score (or precision/recall) since the dataset is less imbalanced than easy ham and spam.

Bernouilli Naive Bayes



Bernoulli NB has an accuracy of 0.93 in hard ham and spam classification. The model has a very high recall and lower precision, meaning it has more false positives, i.e. it misclassifies hard hams as spams.

Overall Results

Similar to part 3, Multinomial Naive Bayes overall performs better than Bernoulli Naive Bayes in classifying easy ham and hard ham versus spam.

Part 3 vs Part 4

Classifier	Ham Type	Part 3						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.97	0.97	1.00	1.00	0.90	0.98	0.95
	Hard	0.95	0.93	0.95	0.77	0.99	0.84	0.97
Bernoulli Naive Bayes	Easy	0.96	0.95	1.00	1.00	0.85	0.97	0.92
	Hard	0.93	0.89	0.94	0.70	0.98	0.78	0.96

Classifier	Ham Type	Part 4						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.98	0.98	1.00	1.00	0.94	0.99	0.97
	Hard	0.96	0.93	0.97	0.83	0.99	0.88	0.98
Bernoulli Naive Bayes	Easy	0.95	0.94	1.00	1.00	0.81	0.97	0.98
	Hard	0.93	0.89	0.94	0.70	0.98	0.78	0.96

When comparing the precision and recall values of the models for the positive class, i.e. spam, we can see that the model improves from parts 3 to 4 for Multinomial Naive Bayes. However, the Bernoulli Naive Bayes model for easy hamps and spams has a slightly worse recall score in part 4 after removing the stopwords.

---

5. [For higher grades] Filter out the headers and the footers of the emails before you run on them. The format may vary somewhat between emails, which can make this a bit tricky, so perfect filtering is not required. Run your program again and answer the following questions:

We use this package from python to deal with emails (https://docs.python.org/3/library/email.message.html)

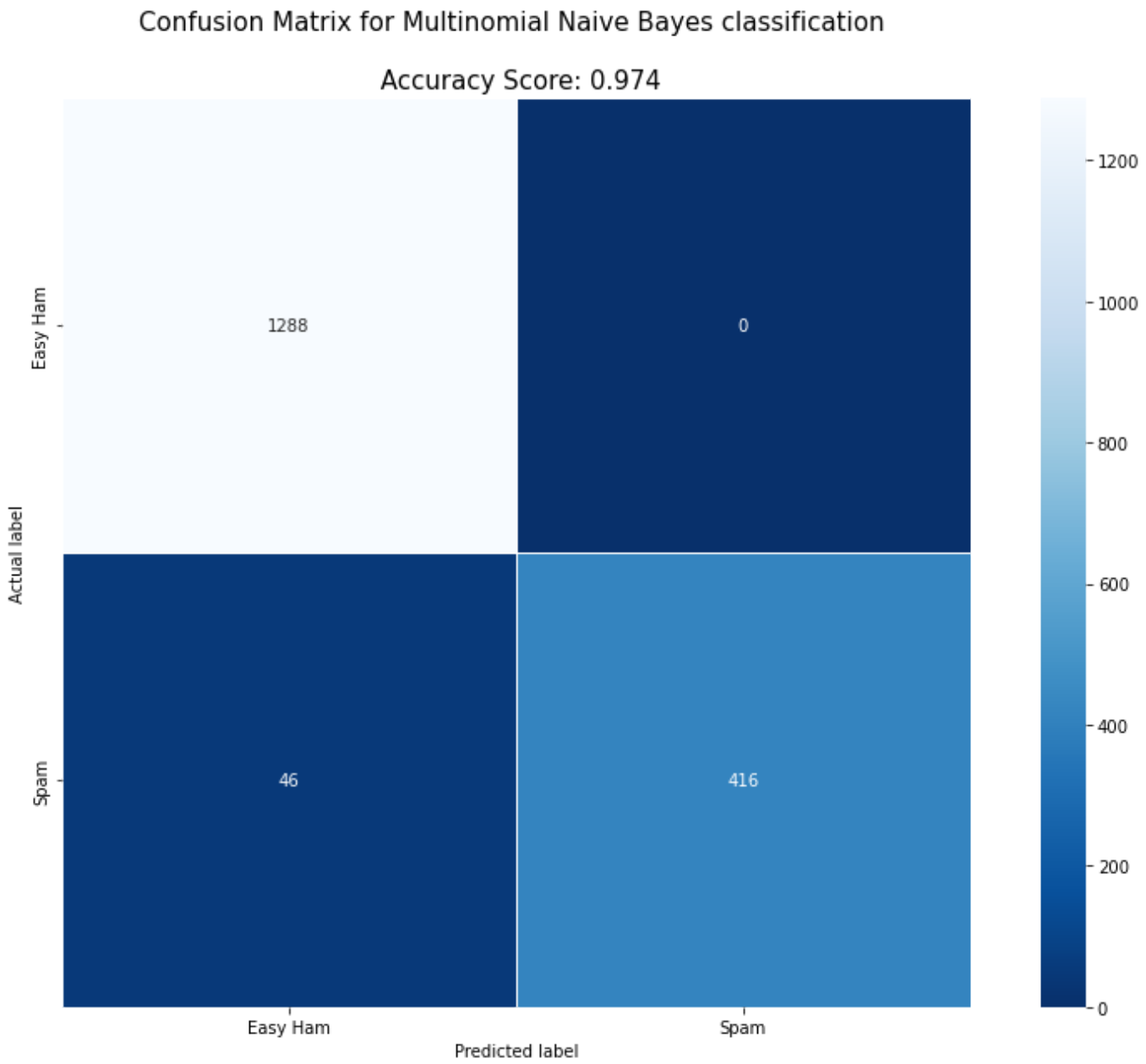
Below is the code snippet for the function:

```
def get_email_body(emailobj):
    # if email is not multipart, return just just regular payload
    # else return the first text/plain body

    if not emailobj.is_multipart():
        return emailobj.get_payload()

    for payload in emailobj.get_payload():
        # If the message comes with a signature it can be that this
        # payload itself has multiple parts, so just return the
        # first one
        if payload.is_multipart():
            return get_email_body(payload)

        body = payload.get_payload()

        if payload.get_content_type() == "text/plain":
            return body
```

When we filter out the headers and footers, we obtain some NoneType objects, hence we convert the filtered data into dataframes along with the targets so that we can drop those rows.

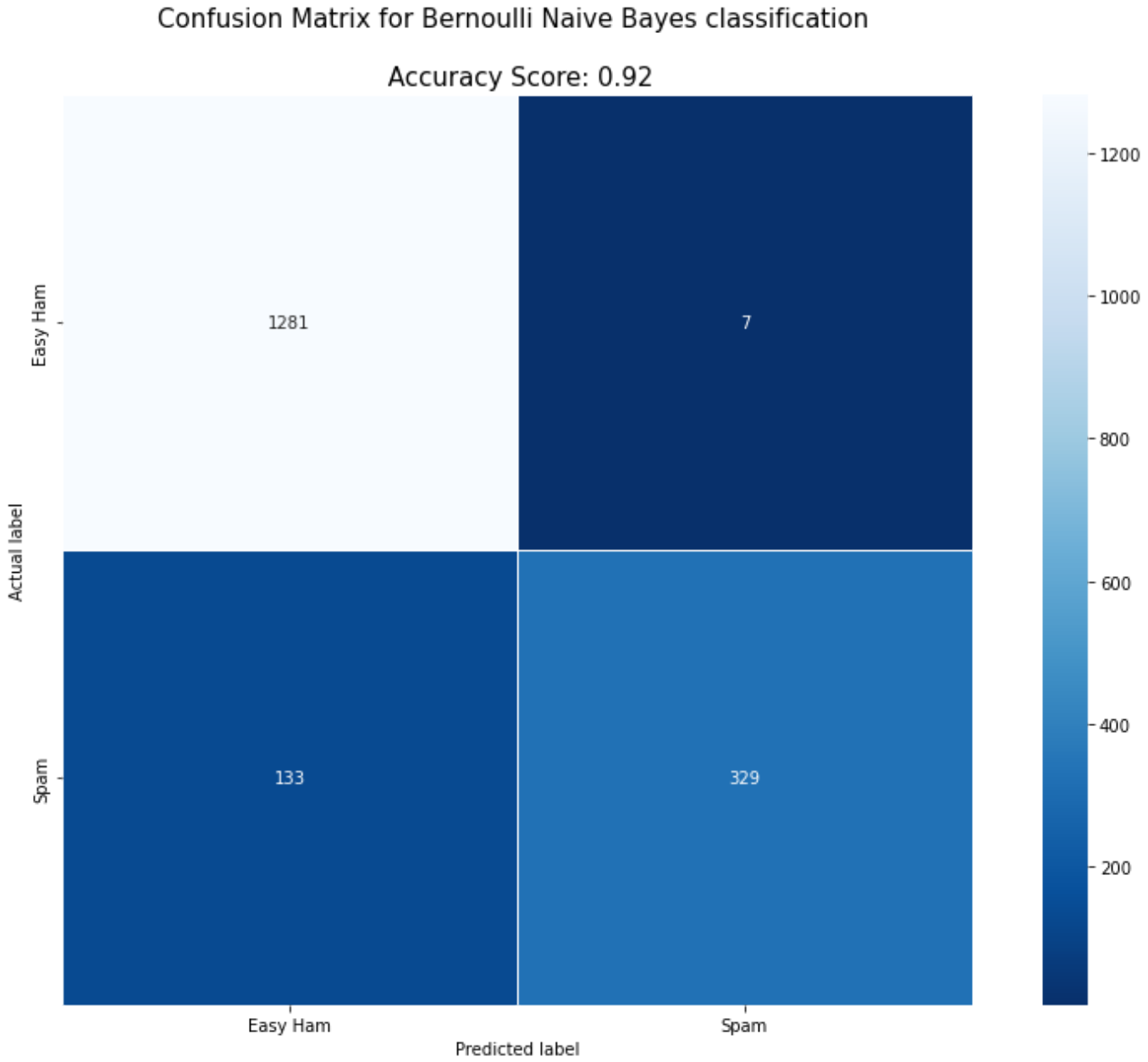
i. Spam versus easy-ham

Multinomial Naive Bayes



We obtained an accuracy of 0.97 when classifying easy ham and spam using the Multinomial Naive Bayes classifier. Like before we consider the precision and recall for the positive class, i.e. spam. The precision is 1.00, which means it has no false positives i.e. actual hams classified as spams. The model has lower recall which means that it has more false negatives(FN = 46), i.e. spams are misclassified as hams.

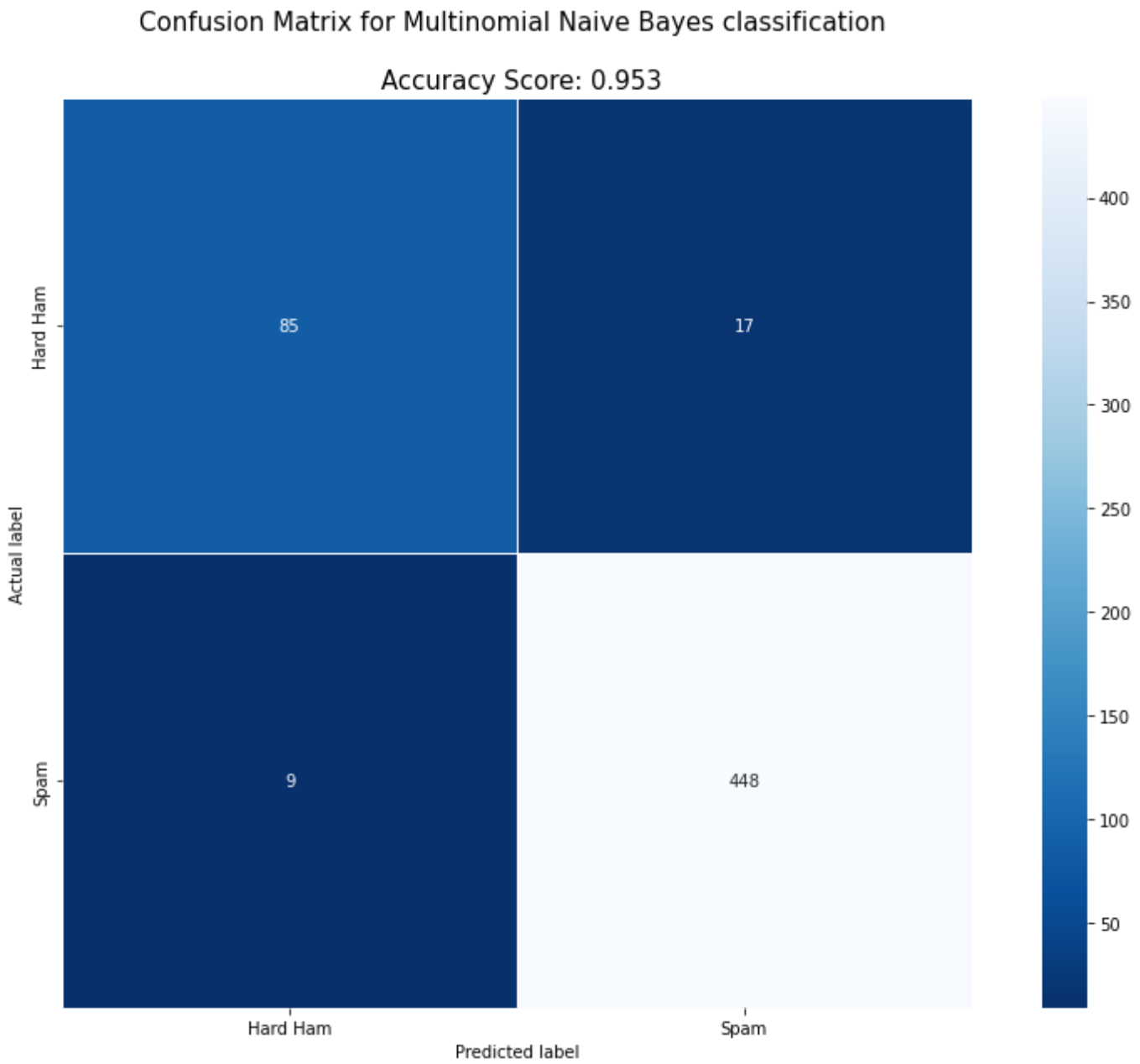
Bernouilli Naive Bayes



Despite having a relatively high accuracy (0.92), the Bernoulli NB classifier has a relatively low recall for the positive class, i.e. spam; that is, it has a high number of false-negative (FN = 133), i.e. spams are misclassified as easy hams.

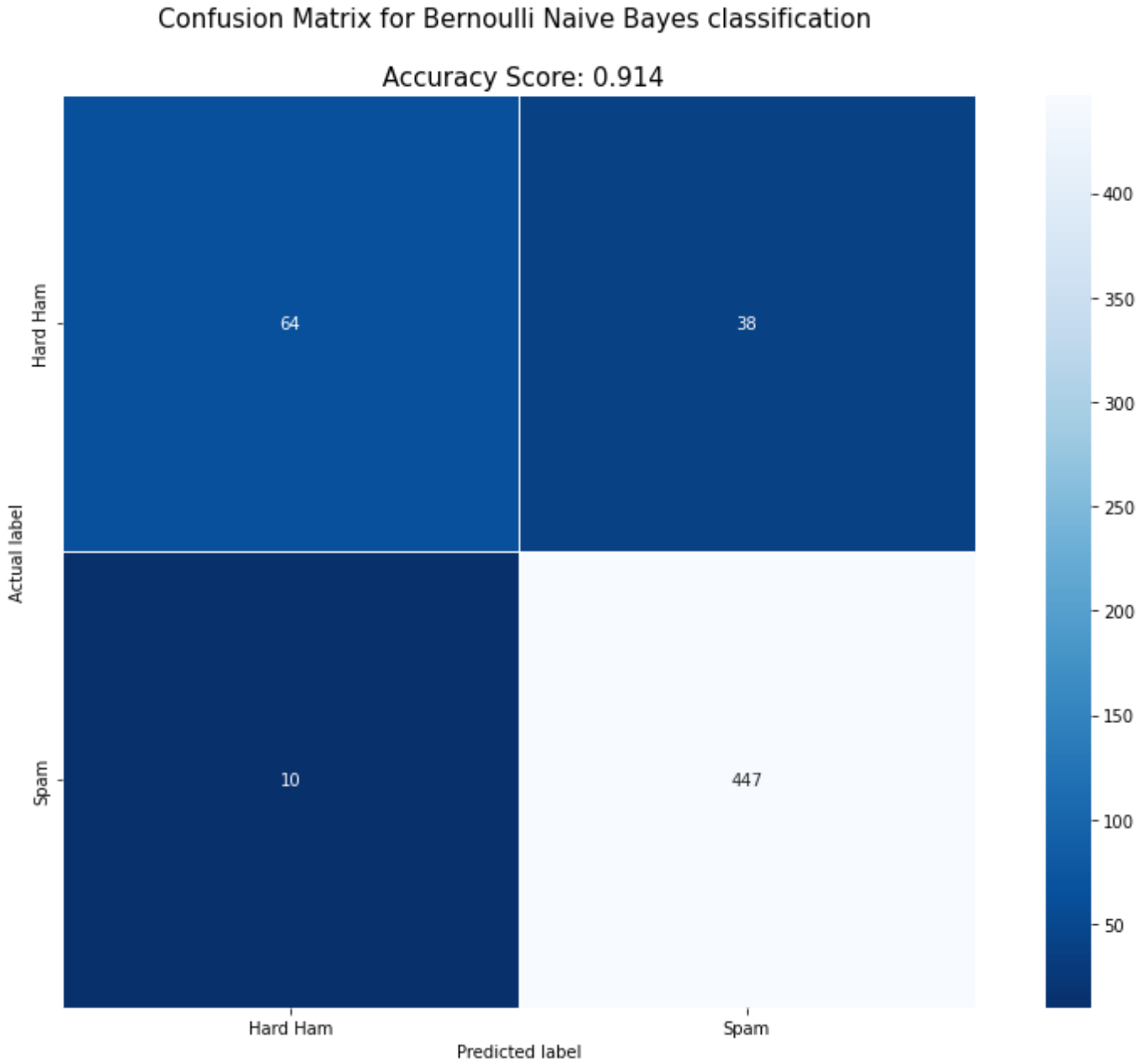
ii. Spam versus hard-ham

Multinomial Naive Bayes



Multinomial Naive Bayes performs well accuracy wise(0.95) and has both precision and recall close to 1, which is the ideal scenario in the classification of hard ham and spam. As seen before since the dataset is less imbalanced than easy ham and spam, the accuracy score is closer to the f1-score (or precision/recall) .

Bernouilli Naive Bayes



Bernoulli NB has an accuracy of 0.91 in hard ham and spam classification. The model has a very high recall and lower precision, meaning it has more false positives, i.e. it misclassifies hard hams as spams.

Overall results

Again, overall, Multinomial Naive Bayes performs better than Bernoulli Naive Bayes in classifying easy ham and hard ham versus spam.

a. Does the result improve from 3 and 4?

Classifier	Ham Type	Part 3						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.97	0.97	1.00	1.00	0.90	0.98	0.95
	Hard	0.95	0.93	0.95	0.77	0.99	0.84	0.97
Bernoulli Naive Bayes	Easy	0.96	0.95	1.00	1.00	0.85	0.97	0.92
	Hard	0.93	0.89	0.94	0.70	0.98	0.78	0.96

Classifier	Ham Type	Part 4						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.98	0.98	1.00	1.00	0.94	0.99	0.97
	Hard	0.96	0.93	0.97	0.83	0.99	0.88	0.98
Bernoulli Naive Bayes	Easy	0.95	0.94	1.00	1.00	0.81	0.97	0.98
	Hard	0.93	0.89	0.94	0.70	0.98	0.78	0.96

Classifier	Ham Type	Part 5						
		accuracy	precision		recall		f1-score	
			0	1	0	1	0	1
Multinomial Naive Bayes	Easy	0.97	0.97	1.00	1.00	0.90	0.98	0.95
	Hard	0.95	0.90	0.96	0.83	0.98	0.87	0.97
Bernoulli Naive Bayes	Easy	0.92	0.91	0.98	0.99	0.71	0.95	0.82
	Hard	0.91	0.86	0.92	0.63	0.98	0.73	0.95

By comparing the precision and recall for the positive class, i.e. spam, we can see that the models have reduced for both Multinominal Naive Bayes and Bernoulli Naive Bayes classifiers. By extracting the email body, we can see that the header and footer of the emails did have some necessary information to help classify the email as spam or ham.

b. The split of the data set into a training set and a test set can lead to very skewed results. Why is this, and do you have suggestions on remedies?

Our training dataset is imbalanced, and most of the data is from the ham class. The class distribution is skewed, so most machine learning algorithms will perform poorly and require modification to avoid simply predicting the majority class in all cases.

For instance, in our dataset, we consider spam and ham emails. The training dataset consists mainly of ham emails, and the model might be more inclined to predict ham since there are more ham emails than spam emails.

There are many ways to handle imbalanced datasets; below are a few suggestions :

- Using SMOTE(Synthetic Minority Oversampling Technique) generates synthetic samples for your imbalanced text data.
- Balancing the data with Oversampling techniques or undersampling the majority class.
- Use the stratified parameters when splitting the training and test set so that the output has a balanced distribution
- Assign class_weights as a parameter to the model's fit method, for instance, increasing weights of minority classes.

c. What do you expect would happen if your training set were mostly spam messages while your test set were mostly ham messages?

The class distribution would again be imbalanced. However, this time, the results will be more skewed towards the opposite direction (more likely to predict spam rather than ham), i.e. there would be many more false positives where hams will be classified as spams.
