Assignment 3

Henning Anevik, Joar Forsberg

September 25 2024

1 Problem 1

Starting by extracting the emails by using the librarie tarfile.

[24]:
```
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)
```

```
Extracted folders: ['easy_ham', 'spam', 'hard_ham']
```

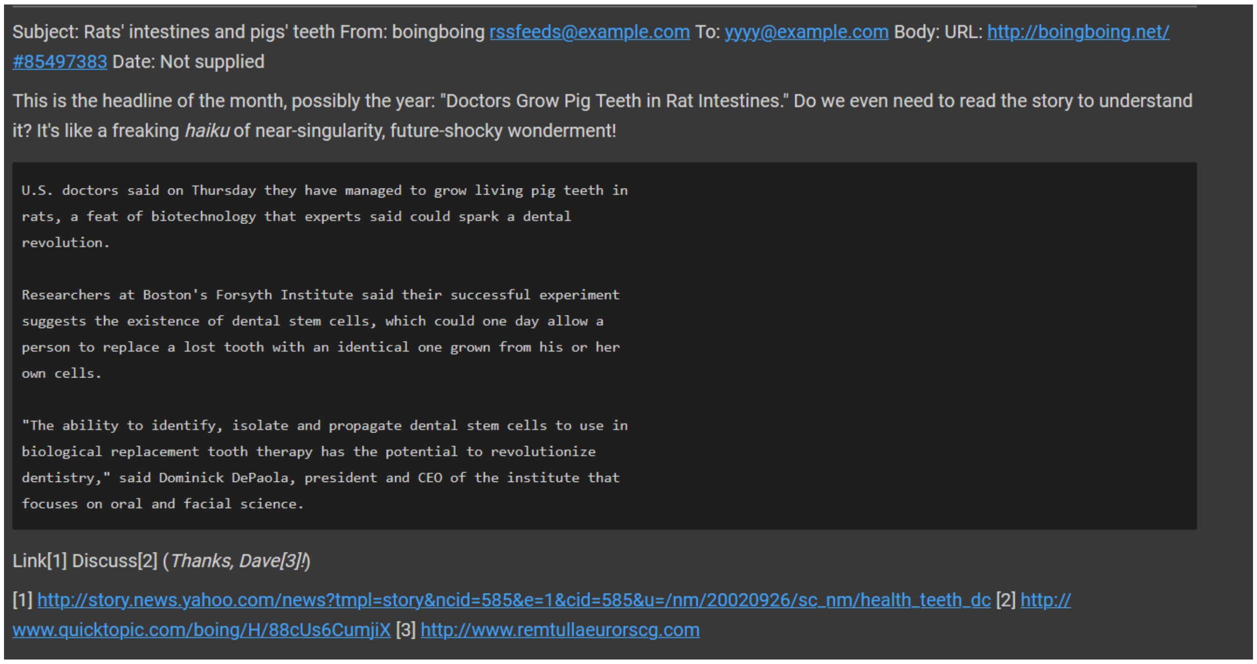
We extracted the files from their bz2 format and now we are going to export the emails to induvidual text files to be able to analyze thier content and so one can supply an anwser to question A.

[25]:
```
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")
```

2 Problem 1: Question A

Starting by looking at the readme.html provided by spamassassin its stated that the easy_ham emails dont 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 serveral aspects close to spam. 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.

Example of email in text file se below:



Subject: Rats' intestines and pigs' teeth From: boingboing rssfeeds@example.com To: yyyy@example.com Body: URL: http://boingboing.net/#85497383 Date: Not supplied

This is the headline of the month, possibly the year: "Doctors Grow Pig Teeth in Rat Intestines." Do we even need to read the story to understand it? It's like a freaking *haiku* of near-singularity, future-shocky wonderment!

```
U.S. doctors said on Thursday they have managed to grow living pig teeth in
rats, a feat of biotechnology that experts said could spark a dental
revolution.

Researchers at Boston's Forsyth Institute said their successful experiment
suggests the existence of dental stem cells, which could one day allow a
person to replace a lost tooth with an identical one grown from his or her
own cells.

"The ability to identify, isolate and propagate dental stem cells to use in
biological replacement tooth therapy has the potential to revolutionize
dentistry," said Dominick DePaola, president and CEO of the institute that
focuses on oral and facial science.
```

Link[1] Discuss[2] (*Thanks, Dave[3]!*)

[1] http://story.news.yahoo.com/news?tmpl=story&ncid=585&e=1&cid=585&u=/nm/20020926/sc_nm/health_teeth_dc [2] http://www.quicktopic.com/boing/H/88cUs6CumjiX [3] http://www.remtullaeurorscg.com

3 Problem 1: Question B

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 resamblens to the code seen in "Problem 1" but for more structured code this was the aproch used. After the mails are put into the list the data spliting is perfomed. As mentioned in the lecture the trainset consist of 75% of the mails and only 25% for the test. Opening the files as bute streams and decodeing them with the help of a list that contains diffrent encodings that the files have

[26]:
```
#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)
```

Labels are important for the model to differentiate between ham and spam based on the content in the training dataset.

4 Problem 2: Preprocessing

[27]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
↪precision_score, recall_score, confusion_matrix
```

[28]:
```
#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)
```

Here we do some preprocessing. We create labels - 1 for ham and 0 for spam. We also create and training and test split with the the help of train_test_split(). We set the random state value to 64 so that when we run the program again we get the same split so that we can reproduce results.

5 Problem 3: Easy Ham

Here we perform the actual training of the model and then predict. We first use fit_transform(train_files) to train the model on the training data. We then transform the model, using transform(test_files) (not fit_transform as that would fit the model to the test data thus making the test useless) and finally make a prediction and observe the metrics.

The fit method that comes with the library sklearn is used to train the model on the dataset we provide. It loks at the variables used and try to learn the connections between them. The transform method as its name says, transfrom the input data so it can be used in the right format so to speak. And the predict method makes predictions on the new data that is provided by the transfrom method.

[33]:
```
from sklearn.naive_bayes import MultinomialNB

vectorizer = CountVectorizer()
x_ez_train = vectorizer.fit_transform(X_ez_train)
x_ez_test = vectorizer.transform(X_ez_test)

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)
```

```
The Multinomial Naive Bayesian Classifier
Accuracy: 96.85%
```

```
Precision: 98.10%
Recall: 82.40%
Confusion Matrix:
[[636   2]
 [ 22 103]]

The Bernoulli Naive Bayesian Classifier
Accuracy: 90.69%
Precision: 100.00%
Recall: 43.20%
Confusion Matrix:
[[638   0]
 [ 71  54]]
```

6 Problem 4: Hard Ham

Same as done in problem 3. Only diffrence is that the hard data is used instead

[35]:
```
vectorizer = CountVectorizer()
x_hd_train = vectorizer.fit_transform(X_hd_train)
x_hd_test = vectorizer.transform(X_hd_test)

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)
```

```
The Multinomial Naive Bayesian Classifier
Accuracy: 92.55%
Precision: 93.02%
Recall: 96.00%
Confusion Matrix:
[[ 54   9]
 [  5 120]]

The Bernoulli Naive Bayesian Classifier
Accuracy: 84.04%
Precision: 81.46%
Recall: 98.40%
Confusion Matrix:
[[ 35  28]
 [  2 123]]
```

Both classifiers perform better on easy ham in terms of precision vs the hard ham. The Bernoulli Naive Bayes classifier achieves good precision but with a lower recall. But the Multinomial Naive Bayes 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 perfromed 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.