ICT607: Artificial Intelligence for Cybersecurity

April 16, 2023

1 Spam fighting: an iterative approach

Reference book: Chio, C., & Freeman, D. (2018). *Machine learning and security: Protecting systems with data and algorithms* (First edition). O'Reilly Media. (page 18)

Reference programs: https://github.com/oreilly-mlsec/book-resources/tree/master/chapter1

1.1 Email processing

We first define a set of functions to help with loading and preprocessing the data and labels, as demonstrated in the following code.

[1]:

```python
import email
```

[2]:

```python
# Combine the different parts of the email into a flat list of strings
def flatten_to_string(parts):
  ret = []
  if type(parts) == str:
    ret.append(parts)
  elif type(parts) == list:
    for part in parts:
      ret += flatten_to_string(part)
  elif parts.get_content_type == 'text/plain':
    ret += parts.get_payload()
  return ret
```

The `flatten_to_string` function takes a nested list of strings and flattens it into a flat list of strings. If the input `parts` is a string, it is directly appended to the `ret` list. If the input `parts` is a list, the function recursively calls itself on each element of the list and concatenates the resulting lists. If the input parts is an email message object and has a content type of `text/plain`, the payload (i.e., the actual text) is appended to the ret list.

In the context of the email library in Python, `get_payload` is a method of the `email.message`.Message class that returns the payload (i.e., the content) of an email message object. The payload can be any object, but it is typically a string that represents the body of the email message.

[3]:

```python
# Extract subject and body text from a signle email file
def extract_email_text(path):
  # Load a single email file from an input file
  with open(path, errors='ignore') as f:
    msg = email.message_from_file(f)
  if not msg:
    return ""

  # Read the email subject
  subject = msg['Subject']
  if not subject:
    subject = ''

  # Read the email body
  body = ' '.join(m for m in flatten_to_string(msg.get_payload())
  if type(m) == str)

  if not body:
    body = ""

  return subject + ' ' + body
```

The `extract_email_text` function extracts the subject and body text from an email file specified by its path. It first loads the file using `email.message_from_file` function from the `email` library. The `Subject` field of the message is extracted and assigned to the subject variable. The function then calls the `flatten_to_string` function to extract the payload (text) of the message and concatenate it into a single string. The resulting string is assigned to the `body` variable. If either the `subject` or `body` variables are empty, they are assigned empty strings. Finally, the `subject` and `body` strings are concatenated with a space character and returned.

1.2 Access dataset

We will use the 2007 TREC Public Spam Corpus dataset that to explore spam versus ham (not spam) classification problem. This is a lightly cleaned raw email message corpus containing 75,419 messages collected from an email server over a three-month period in 2007. One-third of the dataset is made up of spam examples, and the rest is ham.

1.3 Downloading 2007 TREC Public Spam Corpus

1. Read the “Agreement for use” https://plg.uwaterloo.ca/~gvcormac/treccorpus07/
2. Download 255 MB Corpus (trec07p.tgz) and keep it on your Google drive folder (e.g., Colab Notebooks/AICS/ folder)

[4]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[5]:

```
!tar xzf /content/drive/MyDrive/Colab\ Notebooks/AICS/trec07p.tgz
```

The above command can be read as ”extract the contents of the `trec07p.tgz` archive file located in the directory `/content/drive/MyDrive/Colab Notebooks/AICS`.

- tar is a command-line utility in Unix-based operating systems used for working with TAR archives.
- x is a command-line option used to extract the contents of the archive.
- z is a command-line option used to decompress the archive using the gzip utility before extracting its contents.
- f is a command-line option used to specify the name of the archive file to work with.

[6]:

```
DATA_DIR = './trec07p/data/'
LABELS_FILE = './trec07p/full/index'
TRAINING_SET_RATIO = 0.7
```

[7]:

```
labels = {}

# Read the labels
with open(LABELS_FILE) as f:
  for line in f:
    line = line.strip()
    label, key = line.split()
    labels[key.split('/')[-1]] = 1 if label.lower() == 'ham' else 0
```

The above code is reading a set of labels from a file and storing them in a Python dictionary called labels. The keys of the dictionary are file names, and the values are binary labels indicating whether each file is classified as “ham” or “spam”.

- `labels = {}`: This initializes an empty dictionary called labels that will be used to store the file labels.
- `with open(LABELS_FILE) as f`: This opens a file specified by the LABELS_FILE variable and creates a file object f to read its contents. The with statement is used to ensure that the file is properly closed after it is read.
- `for line in f`: This iterates over each line in the file, where line is a string containing the text on that line.
- `line = line.strip()`: This removes any leading or trailing whitespace from the line.
- `label, key = line.split()`: This splits the line into two parts, separated by whitespace. The first part is assigned to the label variable, and the second part is assigned to the key variable.
- `labels[key.split('/')[-1]] = 1 if label.lower() == 'ham' else 0`: This extracts the filename from the key by splitting it on the forward slash (/) character and taking the last element of the resulting list (i.e., the filename without any directories). It then checks if the label is “ham” (case-insensitive) and assigns a value of 1 if it is, or 0 if it is not. The resulting value is stored in the labels dictionary under the filename key.

[8]:

```
import os

def read_email_files():
  X = []
  y = []
  for i in range(len(labels)):
    filename = 'inmail.' + str(i+1)
    email_str = extract_email_text(os.path.join(DATA_DIR, filename))
    X.append(email_str)
    y.append(labels[filename])
  return X, y
```

The above code defines a function called read_email_files() that reads a set of email files, extracts their contents, and returns the contents and corresponding labels.

- `X = []`: This initializes an empty list called X that will be used to store the email contents.
- `y = []`: This initializes an empty list called y that will be used to store the corresponding labels.
- `for i in range(len(labels))`: This iterates over the indices of the labels dictionary (which contains the file labels).
- `filename = 'inmail.' + str(i+1)`: This constructs the filename of the current email file by concatenating the string 'inmail.' with the current index plus one (to account for 0-based indexing).
- `email_str = extract_email_text(os.path.join(DATA_DIR, filename))`: This extracts the contents of the email file with the given filename by calling the extract_email_text() function, passing in the path to the email file (constructed by joining the DATA_DIR path and the filename).
- `X.append(email_str)`: This adds the email contents to the X list.
- `y.append(labels[filename])`: This adds the corresponding label to the y list by looking up the label in the labels dictionary using the current filename as the key.
- `return X, y`: This returns the X and y lists as a tuple.

[9]:

```
from sklearn.model_selection import train_test_split

X, y = read_email_files()

X_train, X_test, y_train, y_test, idx_train, idx_test = train_test_split(X, y,␣
 ↪range(len(y)), train_size=TRAINING_SET_RATIO, random_state=2)
```

Now that you have prepared the raw data, you need to do some further processing of the tokens to convert each email to a vector representation that a machine learning algorithm accepts as input.

[10]:

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X_train_vector = vectorizer.fit_transform(X_train)
X_test_vector = vectorizer.transform(X_test)
```

The above code is using the scikit-learn library's CountVectorizer class to convert the raw text data of email messages into a numerical representation that can be used as input to machine learning algorithms.

- `vectorizer = CountVectorizer()`: This creates an instance of the CountVectorizer class, which is used to convert text data into a bag-of-words representation, where each word in the text is treated as a separate feature and the frequency of occurrence of each word is counted.

- `X_train_vector = vectorizer.fit_transform(X_train)`: This applies the fit_transform() method of the CountVectorizer object to the training data X_train, which learns the vocabulary of the training data (i.e., the set of unique words that appear in the emails) and returns a sparse matrix X_train_vector where each row corresponds to an email and each column corresponds to a word in the vocabulary, with the values indicating the frequency of occurrence of the corresponding word in the corresponding email. The fit_transform() method is a combination of two steps: first, it fits the vectorizer to the training data (i.e., it learns the vocabulary of the training data), and second, it transforms the training data into a bag-of-words representation using the learned vocabulary.
- `X_test_vector = vectorizer.transform(X_test)`: This applies the transform() method of the CountVectorizer object to the test data X_test, which uses the vocabulary learned from the training data to convert the test data into a bag-of-words representation, returning a sparse matrix X_test_vector. The transform() method only transforms the test data into a bag-of-words representation using the vocabulary learned from the training data. It does not update the vocabulary or learn new features.

1.4 Classifier – Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem, which describes the probability of a hypothesis given evidence. In the context of classification, the hypothesis corresponds to the class label and the evidence corresponds to the input features.

Multinomial Naive Bayes (`MultinomialNB`): Assumes that the input features are discrete counts, such as word frequencies in text classification.

[11]:

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Initialize the classifier and make label predictions
mnb = MultinomialNB()
mnb.fit(X_train_vector, y_train)
y_pred = mnb.predict(X_test_vector)

# Print results
print('Accuracy {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
Accuracy 0.956
```

1.5 Practice task

Try different ML algorithms for spam vs ham classification and improve the classification accuracy. In addition to classification accuracy, use confusion matrix to report performance.