Computer Security

Project (1)
Spam Mail Analysis

소프트웨어 학과

김혜미 (201533640)

유수화 (201533654)

홍건수 (201433718)

1. Preparing the text data

We have two mail data folders. There are four subdirectories of ‘lingspam_public’ directory. And we have two subdirectories of ‘ling-spam’ directory. When we analyzed the data inside the directory, some data was preprocessed but some data was not. Therefore, we preprocessed the data that requires preprocessing.

Our objective is to remove those words from the document which may not contribute to the information we want to extract. Emails may contain a lot of undesirable characters like punctuation marks, stop words, digits, etc which may not be helpful in detecting the spam email. Our data preprocessing process is as follows.

1. **Stop words removal**

Stop words like “and”, “the”, “of”, etc are very common in all English sentences and are not very meaningful in deciding spam or legitimate status, so these words have been removed from the emails.

2. **Lemmatization**

We grouped the word together which have different inflected forms of a word so they can be analysed as a single item. For example, “contain”, “contains,” and “contained” would all be represented as “contain”.

3. **Removal of non-words**

We removed the non-words like punctuation marks or special characters from the mail documents, after creating a dictionary.

2. Creating word dictionary

As a first step, we need to create a dictionary of words and their frequency.

```
def make_Dictionary(train_dir):
    emails = [os.path.join(train_dir, f) for f in
os.listdir(train_dir)]
    all_words = []
    for mail in emails:
        with open(mail) as m:
            for i, line in enumerate(m):
                if i == 2:  # Body of email is only 3rd line
of text file
                    words = line.split()
                    all_words += words

    dictionary = Counter(all_words)
```

Once the dictionary is created, we added just a few lines of code written below to the above function to remove non-words. We also removed single characters in the dictionary which are useless.

```
list_to_remove = list(dictionary)

for item in list_to_remove:
    if item.isalpha() == False:
        del dictionary[item]
    elif len(item) == 1:
        del dictionary[item]
dictionary = dictionary.most_common(3000)

print(dictionary)

return dictionary
```

Through this process, we have chosen 3000 most frequently used words in the dictionary. The results are shown in the following page.

```
/Users/irene/PycharmProjects/forKAKAO/venv/bin/python /Users/irene/PycharmProjects/forKAKAO/forKaKao/solution.py
[('order', 1414), ('address', 1293), ('report', 1216), ('mail', 1127), ('send', 1079), ('language', 1072), ('email', 1051),
```

```
print(dictionary) :
```

[('order', 1414), ('address', 1293), ('report', 1216), ('mail', 1127), ('send', 1079), ('language', 1072), ('email', 1051), ('program', 1001), ('our', 987), ('list', 935), ('one', 917), ('name', 878), ('receive', 826), ('money', 788), ('free', 762), ('work', 755), ('information', 677), ('business', 654), ('please', 652), ('university', 595), ('us', 564), ('day', 556), ('follow', 544), ('internet', 520), ('over', 511), ('http', 479), ('check', 472), ('call', 469), ('each', 466), ('include', 452), ('com', 448), ('linguistic', 442), ('number', 423), ('want', 420), ('letter', 419), ('need', 418), ('many', 412), ('here', 397), ('market', 395), ('start', 390), ('even', 386), ('fax', 383), ('form', 380), ('most', 377), ('first', 373), ('web', 366), ('service', 363), ('interest', 362), ('software', 352), ('remove', 349), ('read', 347), ('those', 345), ('week', 344), ('every', 332), ('credit', 329), ('ll', 326), ('site', 320), ('edu', 318), ('much', 318), ('english', 318), ('product', 317), ('bulk', 312), ('phone', 311), ('must', 299), ('two', 298), ('offer', 297), ('cost', 294), ('best', 291), ('www', 290), ('computer', 289), ('link', 283), ('state', 279), ('card', 278), ('de', 272), ('home', 268), ('available', 266), ('system', 264), ('message', 260), ('own', 260), ('after', 258), ('opportunity', 257), ('question', 257), ('place', 256), ('pay', 256), ('within', 254), ('million', 254), ('hour', 253), ('world', 253), ('write', 253), ('sell', 251), ('copy', 248), ('show', 248), ('help', 248), ('below', 247), ('same', 242), ('conference', 241), ('file', 238), ('through', 235), ('directory', 234), ('before', 232), ('easy', 229), ('where', 229), ('try', 226), ('income', 226), ('book', 226), ('thank', 225), ('tell', 222), ('today', 220), ('company', 219), ('member', 216), ('example', 215), ('word', 215), ('back', 214), ('advertise', 213), ('different', 213), ('month', 210), ('subject', 209), ('etc', 207), ('contact', 206), ('ask', 206), ('cash', 204), ('response', 204), ('little', 203), ('learn', 203), ('add', 202), ('cd', 200), ('per', 198), ('research', 198), ('linguist', 198), ('type', 197), ('anyone', 196), ('again', 196), ('down', 196), ('process', 196), ('right', 196), ('ve', 195), ('least', 195), ('is', 195), ('special', 194), ('price', 193), ('save', 192), ('provide', 192), ('level', 192), ('re', 190), ('off', 190), ('live', 190), ('require', 188), ('line', 186), ('financial', 186), ('sure', 186), ('few', 186), ('change', 186), ('post', 185), ('box', 185), ('search', 185), ('let', 184), ('ffa', 183), ('never', 181), ('paper', 179), ('four', 177), ('thing', 177), ('teach', 175), ('case', 175), ('life', 175), ('page', 174), ('participate', 173), ('great', 172), ('text', 172), ('really', 171), ('between', 170), ('both', 170), ('net', 170), ('buy', 169), ('plan', 167), ('success', 167), ('part', 166), ('simply', 164), ('instruction', 164), ('move', 164), ('request', 164), ('click', 164), ('package', 162), ('reply', 161), ('ever', 160), ('less', 160), ('point', 160), ('seem', 159), ('discourse', 158), ('several', 158), ('dollar', 158), ('step', 157), ('method', 157), ('win', 157), ('why', 157), ('theory', 157), ('student', 156), ('simple', 156), ('thousand', 155), ('put', 155), ('friend', 154), ('next', 154), ('important', 154), ('ship', 154), ('guarantee', 153), ('rate', 153), ('linguistics', 153), ('legal', 152), ('note', 150), ('fact', 150), ('course', 150), ('floodgate', 150), ('under', 148), ('account', 147), ('talk', 147), ('allow', 146), ('return', 146), ('since', 146), ('american', 145), ('keep', 145), ('until', 145), ('position', 145), ('school', 145), ('result', 144), ('ad', 143), ('analysis', 143), ('experience', 143), ('present', 143), ('remember', 142), ('study', 142), ('reference', 142), ('another', 141), ('sale', 141), ('better', 141), ('speaker', 139), ('wish', 139), ('problem', 139), ('person', 139), ('believe', 139), ('city', 139), ('group', 139), ('bank', 138), ('vium', 136), ('leave', 136), ('once', 136), ('full', 136), ('print', 135), ('lot', 135), ('section', 135), ('answer', 135), ('family', 135), ('complete', 134), ('area', 134), ('international', 134), ('plus', 134), ('total', 134), ('georgetown', 134), ('department', 133), ('too', 133), ('grammar', 133), ('date', 133), ('lose', 132), ('office', 132), ('second', 132), ('issue', 131), ('bill', 131), ('future', 131), ('earn', 131), ('above', 130), ('while', 130), ('further', 129), ('long', 129), ('amount', 127), ('mean', 127), ('ca', 126), ('set', 126), ('join', 126), ('purchase', 125), ('run', 125), ('publish', 124), ('server', 124), ('nothing', 123), ('without', 123), ('possible', 123), ('become', 123), ('adult', 123), ('video', 122), ('still', 121), ('real', 121), ('base', 120), ('accept', 120), ('already', 120), ('something', 120), ('code', 119), ('registration', 118), ('sign', 118), ('everyone', 118), ('uk', 118), ('game', 118), ('personal', 117), ('major', 117), ('support', 117), ('fee', 116), ('understand', 116), ('someone', 115), ('however', 115), ('record', 114), ('speech', 114), ('small', 113), ('detail', 113), ('visit', 113), ('minute', 112), ('mailing', 112),

3. Feature extraction process

Now, we can extract word count vector (our feature here) of 3000 dimensions for each email of training set. Each **word count vector** contains the frequency of 3000 words in the training file. Most of them will be zero. So initialize as ‘np.zeros’. Each word count vector contains the frequency of 3000 dictionary words in the training file.

The below python code will generate a feature vector matrix of training set and denote 3000 words of dictionary.

```
# extract feature
ID = {word: i for i, (word, _) in enumerate(dictionary)}

def extract(words, dictionary):
feature = np.zeros(len(dictionary), dtype=np.int32) for word
in filter(lambda w: w in ID, set(words)):

feature[ID[word]] = words.count(word) return feature

train_features = np.array([extract(read(str(file)),
dictionary) for file in trainset])
train_labels = np.array([file.name.startswith('spm') for file
in trainset], dtype=np.bool)
```

At first, extract the top 3000 words of the training set dictionary and generate the IDs in order of frequency. To eliminate duplication, group words into ‘sets’. And filter by ID to characterize them. This will be **train_featrues** and spam, ie spm in the filename. Included files are classified a**s train_labels.**

Creates vector as dictionary size called feature, then check and extracts it for each file.

4. Training the Classifiers

I will be using **scikit-learn ML library** for **training classifiers**. Because in order to build an e-mail spam filter, I have to train a math model that learns decision boundaries in the feature space between two classes.

I have trained two models here namely **Naive Bayes classifier** and **Support Vector Machines (SVM)**.

Naive Bayes classifier is a supervised probabilistic classifier based on Bayes theorem assuming independence between every pair of features. Naive Bayes classifier is especially important, so let's learn more about it.

In machine learning, naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

The following is a description of the **Support Vector Machines.**

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

SVMs are supervised binary classifiers which are very effective when you have higher number of features. The goal of SVM is to separate some subset of training data from rest called the support vectors (boundary of separating hyper-plane).

Once you have trained your classifier, you can see **the performance of the model** in the test set. A word number vector is extracted for each mail by a test set, and a class (ham or spam) is predicted by a trained NB classifier and an SVM model.

Checking Performance

So I will check the performance of built email spam filter. I can see the performance by creating and printing the captions_matrix for Multinomial NB and SVM. Test-set contains 130 spam emails and 130 non-spam emails.

The diagonal elements represents the correctly identified(a.k.a. true identification) mails where as non-diagonal elements represents wrong classification (false identification) of mails.

```
result1 = model1.predict(test_matrix)
result2 = model2.predict(test_matrix)
print (confusion_matrix(test_labels, result1))
print (confusion_matrix(test_labels, result2))
```

Result :

```
/Users/irene/PycharmP
[[129   1]
 [  9 121]]
[[126   4]
 [  6 124]]
FINISH
```

And The following results indicate:

Multinomial NB	Ham	Spam
Ham	129	1
Spam	9	121
Accuracy	0.961538461538462	

SVM(Linear)	Ham	Spam
Ham	126	4
Spam	6	124
Accuracy	0.961538461538462	

Both models achieved similar performance on the test set, except that the SVM was slightly balanced for false identification.

Apply to 'lingspam_public.tar'

Until now, it has applied to **ling-spam**. We have applied these codes to the **lingspam_public** as well.

• bare

```
# make Dictionary
sequences = Path('/Users/irene/PycharmProjects/forKAKAO/
forKaKao/lingspam_public/bare')
```

```
[[716  14]
 [  3 135]]
[[722   8]
 [  7 131]]
Accuracy 1: 0.9804147465437788
Accuracy 1: 0.9827188940092166
```

• lemm

```
# make Dictionary
sequences = Path('/Users/irene/PycharmProjects/forKAKAO/
forKaKao/lingspam_public/lemm')
```

```
[[703   8]
 [  4 153]]
[[705   6]
 [  7 150]]
Accuracy 1: 0.9861751152073732
Accuracy 1: 0.9850230414746544
```

• stop

```
# make Dictionary
sequences = Path('/Users/irene/PycharmProjects/forKAKAO/
forKaKao/lingspam_public/stop')
```

```
[[731   8]
 [  1 128]]
[[734   5]
 [  5 124]]
Accuracy 1: 0.9896313364055299
Accuracy 1: 0.988479262672811
```

CODE

- Using data "ling-spam"

```
import os
import numpy as np
from collections import Counter
from sklearn.naive_bayes import MultinomialNB, GaussianNB,
BernoulliNB
from sklearn.svm import SVC, NuSVC, LinearSVC
from sklearn.metrics import confusion_matrix

def extract_features(mail_dir):
    files = [os.path.join(mail_dir,fi) for fi in
sorted(os.listdir(mail_dir))]

    features_matrix = np.zeros((len(files),3000))
    docID = 0;
    for fil in files:
      with open(fil) as fi:
        for i,line in enumerate(fi):
          if i == 2:
            words = line.split()
            for word in words:
              wordID = 0
              for i,d in enumerate(dictionary):
                if d[0] == word:
                  wordID = i
                  features_matrix[docID,wordID] = words.count(word)
        docID = docID + 1
    return features_matrix


def make_Dictionary(train_dir):
    emails = [os.path.join(train_dir, f) for f in
os.listdir(train_dir)]
    all_words = []
    for mail in emails:
        with open(mail) as m:
            for i, line in enumerate(m):
                if i == 2:  # Body of email is only 3rd line of text
file
                    words = line.split()
                    all_words += words

    dictionary = Counter(all_words)

    list_to_remove = list(dictionary)

    for item in list_to_remove:
```

```
        if item.isalpha() == False:
            del dictionary[item]
        elif len(item) == 1:
            del dictionary[item]
    dictionary = dictionary.most_common(3000)

    return dictionary



# Create a dictionary of words with its frequency

train_dir = '/Users/irene/PycharmProjects/forKAKAO/forKaKao/train-
mails'
dictionary = make_Dictionary(train_dir)

# Prepare feature vectors per training mail and its labels

train_labels = np.zeros(702)
train_labels[351:701] = 1
train_matrix = extract_features(train_dir)

# Training SVM and Naive bayes classifier

model1 = MultinomialNB()
model2 = LinearSVC()
model1.fit(train_matrix, train_labels)
model2.fit(train_matrix, train_labels)

# Test the unseen mails for Spam
test_dir = '/Users/irene/PycharmProjects/forKAKAO/forKaKao/test-
mails'
test_matrix = extract_features(test_dir)
test_labels = np.zeros(260)
test_labels[130:260] = 1
result1 = model1.predict(test_matrix)
result2 = model2.predict(test_matrix)
print (confusion_matrix(test_labels, result1))
print (confusion_matrix(test_labels, result2))
print('FINISH')
```

CODE

- Using data "lingspam_public"

```
from pathlib import Path
from collections import Counter
import numpy as np
from random import shuffle
from sklearn.naive_bayes import MultinomialNB, GaussianNB,
BernoulliNB
from sklearn.svm import SVC, NuSVC, LinearSVC
from sklearn.metrics import confusion_matrix


# make Dictionary
sequences = Path('/Users/irene/PycharmProjects/forKAKAO/forKaKao/
lingspam_public/stop')
counter = Counter()

folder = sorted(sequences.iterdir())
dataset = [file for sequence in folder for file in
sorted(sequence.glob('*.txt'))]

shuffle(dataset)

trainset = dataset[:int(len(dataset)*.7)]
testset = dataset[int(len(dataset)*.7):]

def read(file):
    with open(file) as f:
        for i, line in enumerate(f):
            if i == 2 :
                return line.split()

for file in trainset:
    counter.update(read(str(file)))

# remove unused
list_in_remove = list(filter(lambda w: not w.isalpha() or len(w) ==
1, counter))
for word in list_in_remove:
    del counter[word]

# extract feature
feature_size = 3000
dictionary = counter.most_common(feature_size)
ID = {word: i for i, (word, _) in enumerate(dictionary)}
```

```
def extract(words, dictionary):
    feature = np.zeros(len(dictionary), dtype=np.int32)
    for word in filter(lambda w: w in ID, set(words)):
        feature[ID[word]] = words.count(word)
    return feature

train_features = np.array([extract(read(str(file)), dictionary) for
file in trainset])
train_labels = np.array([file.name.startswith('spm') for file in
trainset], dtype=np.bool)

# training
model1 = MultinomialNB()
model2 = LinearSVC()

model1.fit(train_features, train_labels)
model2.fit(train_features, train_labels)


# testing
test_features = np.array([extract(read(str(file)), dictionary) for
file in testset])
test_labels = np.array([file.name.startswith('spm') for file in
testset], dtype=np.bool)

prediction1 = model1.predict(test_features)
prediction2 = model2.predict(test_features)

acc1 = (prediction1 == test_labels).sum() / test_labels.size
acc2 = (prediction2 == test_labels).sum() / test_labels.size

print(confusion_matrix(test_labels, prediction1))
print(confusion_matrix(test_labels, prediction2))

print(f'Accuracy 1: {acc1}')
print(f'Accuracy 1: {acc2}')
```

References

1. Androutsopoulos, J. Koutsias, K.V. Chandrinos, George Paliouras, and C.D. Spyropoulos, "An Evaluation of Naive Bayesian Anti-Spam Filtering". In Potamias, G., Moustakis, V. and van Someren, M. (Eds.), Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), Barcelona, Spain, pp. 9-17, 2000.

2. https://www.kdnuggets.com/2017/03/email-spam-filtering-an-implementation-with -python-and-scikit-learn.html

3. Introduction to Machine Learning with Python, O'REILLY 4. Foundations for Analytics with Python, O'REILLY

4. https://www.datasciencecentral.com/profiles/blogs/email-spam-filtering-a-python-implementation-with-scikit-learn

5. https://github.com/tristaneljed/Spamector

6. https://github.com/abhijeet3922/Mail-Spam-Filtering/blob/master/enron-spamfilter.py

7. https://appliedmachinelearning.blog/2017/12/21/predict-the-happiness-on-tripadvisor-reviews-using-dense-neural-network-with-keras-hackerearth-challenge/