나이브 베이즈 스팸필터

부산대학교 정보·의생명공학대학
정보컴퓨터공학부

나이브 베이즈를 이용한 스팸필터 예제 코드

Overview

- colab notebook link: https://colab.research.google.com/drive/1OfHPUkeAGZetOA-cZ0f8F4vYecknleYY?usp=sharing
- Supporting functions and classes
- Naïve Bayes Classifier
- Model training
- Model Evaluation

Supporting functions: tokenize()

```
from typing import Set
import re

def tokenize(text: str) -> Set[str]:
    text = text.lower()                          # Convert to lowercase,
    all_words = re.findall("[a-z0-9']+", text)   # extract the words, and
    return set(all_words)                        # remove duplicates.

assert tokenize("Data Science is science") == {"data", "science", "is"}
# assert tokenize("Data Science is science") == {"data", "science"} # AssertionError
```

Supporting class: Message

```
from typing import NamedTuple

class Message(NamedTuple):
    text: str
    is_spam: bool
```

```
messages = [Message("spam rules", is_spam=True),
            Message("ham rules", is_spam=False),
            Message("hello ham", is_spam=False)]
```

Class: NaïveBayesClassifier

```
class NaiveBayesClassifier:
    def __init__(self, k: float = 0.5) -> None:

    def train(self, messages: Iterable[Message]) -> None:

    def _probabilities(self, token: str) -> Tuple[float, float]:

    def predict(self, text: str) -> float:
```

```
    def __init__(self, k: float = 0.5) -> None:
        self.k = k  # smoothing factor
        self.tokens: Set[str] = set()
        self.token_spam_counts: Dict[str, int] = defaultdict(int)
        self.token_ham_counts: Dict[str, int] = defaultdict(int)
        self.spam_messages = self.ham_messages = 0
```

Method: train

```
    def train(self, messages: Iterable[Message]) -> None:
        for message in messages:
            # Increment message counts
            if message.is_spam:
                self.spam_messages += 1
            else:
                self.ham_messages += 1

            # Increment word counts
            for token in tokenize(message.text):
                self.tokens.add(token)
                if message.is_spam:
                    self.token_spam_counts[token] += 1
                else:
                    self.token_ham_counts[token] += 1
```

Method: _probabilities

```
    def _probabilities(self, token: str) -> Tuple[float, float]:
        """returns P(token | spam) and P(token | not spam)"""
        spam = self.token_spam_counts[token]
        ham = self.token_ham_counts[token]

        p_token_spam = (spam + self.k) / (self.spam_messages + 2 * self.k)
        p_token_ham = (ham + self.k) / (self.ham_messages + 2 * self.k)

        return p_token_spam, p_token_ham
```

Method: predict

```
    def predict(self, text: str) -> float:
        text_tokens = tokenize(text)
        log_prob_if_spam = log_prob_if_ham = 0.0

        # Iterate through each word in our vocabulary.
        for token in self.tokens:
            prob_if_spam, prob_if_ham = self._probabilities(token)

            # If *token* appears in the message,
            # add the log probability of seeing it;
            if token in text_tokens:
                log_prob_if_spam += math.log(prob_if_spam)
                log_prob_if_ham += math.log(prob_if_ham)

            # otherwise add the log probability of _not_ seeing it
            # which is log(1 - probability of seeing it)
            else:
                log_prob_if_spam += math.log(1.0 - prob_if_spam)
                log_prob_if_ham += math.log(1.0 - prob_if_ham)
```

Method: predict (cont.)

```
        prob_if_spam = math.exp(log_prob_if_spam)
        prob_if_ham = math.exp(log_prob_if_ham)
        return prob_if_spam / (prob_if_spam + prob_if_ham)
```

Unit Test

```
messages = [Message("spam rules", is_spam=True),
            Message("ham rules", is_spam=False),
            Message("hello ham", is_spam=False)]

model = NaiveBayesClassifier(k=0.5)
model.train(messages)

assert model.tokens == {"spam", "ham", "rules", "hello"}
assert model.spam_messages == 1
assert model.ham_messages == 2
assert model.token_spam_counts == {"spam": 1, "rules": 1}
assert model.token_ham_counts == {"ham": 2, "rules": 1, "hello": 1}

text = "hello spam"
```

Unit Test (cont.)

```
probs_if_spam = [
    (1 + 0.5) / (1 + 2 * 0.5),      # "spam"  (present)
    1 - (0 + 0.5) / (1 + 2 * 0.5),  # "ham"   (not present)
    1 - (1 + 0.5) / (1 + 2 * 0.5),  # "rules" (not present)
    (0 + 0.5) / (1 + 2 * 0.5)       # "hello" (present)
]

probs_if_ham = [
    (0 + 0.5) / (2 + 2 * 0.5),      # "spam"  (present)
    1 - (2 + 0.5) / (2 + 2 * 0.5),  # "ham"   (not present)
    1 - (1 + 0.5) / (2 + 2 * 0.5),  # "rules" (not present)
    (1 + 0.5) / (2 + 2 * 0.5),      # "hello" (present)
]

p_if_spam = math.exp(sum(math.log(p) for p in probs_if_spam))
p_if_ham = math.exp(sum(math.log(p) for p in probs_if_ham))

# Should be about 0.83
assert model.predict(text) == p_if_spam / (p_if_spam + p_if_ham)

print(p_if_spam / (p_if_spam + p_if_ham))
```

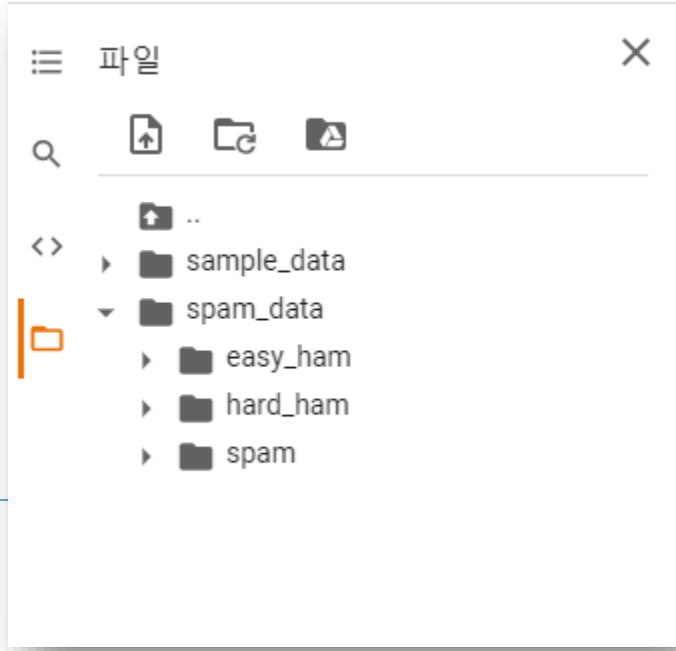
Model Training – Download training data and extract all

```
from io import BytesIO  # 바이트를 파일로 다루기 위해 필요
import requests         # 파일을 내려받기 위해 필요
import tarfile          # 파일이 .tar.bz 형식이기 때문에 필요

BASE_URL = "https://spamassassin.apache.org/old/publiccorpus"
FILES = ["20021010_easy_ham.tar.bz2",
         "20021010_hard_ham.tar.bz2",
         "20030228_spam.tar.bz2"]

OUTPUT_DIR = 'spam_data'

for filename in FILES:
  content = requests.get(f"{BASE_URL}/{filename}").content
  fin = BytesIO(content)
  with tarfile.open(fileobj=fin, mode='r:bz2') as tf:
    tf.extractall(OUTPUT_DIR)
```



Model training

```
# modify the path to wherever you've put the files
path = '/content/spam_data/*/*'
data: List[Message] = []
# glob.glob returns every filename that matches the wildcarded path
for filename in glob.glob(path):
    is_spam = "ham" not in filename
    # There are some garbage characters in the emails, the errors='ignore'
    # skips them instead of raising an exception.
    with open(filename, errors='ignore') as email_file:
        for line in email_file:
            if line.startswith("Subject:"):
                subject = line.lstrip("Subject: ")
                data.append(Message(subject, is_spam))
                break  # done with this file
random.seed(0)      # just so you get the same answers as me
train_messages, test_messages = split_data(data, 0.75)
model = NaiveBayesClassifier()
model.train(train_messages)
```

Model Evaluation

```
from collections import Counter

predictions = [(message, model.predict(message.text))
               for message in test_messages]

# Assume that spam_probability > 0.5 corresponds to spam prediction
# and count the combinations of (actual is_spam, predicted is_spam)
confusion_matrix = Counter((message.is_spam, spam_probability > 0.5)
                           for message, spam_probability in predictions)

print(confusion_matrix)
```

```
Counter({(False, False): 660, (True, True): 86, (True, False): 46, (False, True): 33})
```