Data Literacy Exercise 01

Matthias Gamperl, University of Passau, Winter Semester 2022

Theoretical part of assignment 01

1. EXAMple question

given:

$P(\text{overdrafted}) = 0.80$
$P(\text{late} \mid \text{overdrafted}) = 0.60$
$P(\text{late}) = 0.60$

find:

$P(\text{overdrafted} \mid \text{late}) = \frac{P(\text{late} \mid \text{overdrafted}) \cdot P(\text{overdrafted})}{P(\text{late})}$

$P(\text{overdrafted} \mid \text{late}) = P(\text{late} \mid \text{overdrafted}) \cdot P(\text{overdrafted}) \cdot \frac{1}{P(\text{late})} = 0.60 \cdot 0.80 \cdot \frac{1}{0.60} = 0.80$

$0.80 = 0.80 \rightarrow$ true

You can conclude that

$P(\text{overdrafted}) = P(\text{late}) = 0.80$

the probability of an overdraft, given that somebody is late is 0.80!!!

2. Theory Question - Pooled Testing

a) The probability of at least one being infected, let call it $P(I \geq 1)$, can be calculated as 1 minus the probability that none of the people are infected, let call it $P(I^0)$.

b) The probability, given one test is infected is calculated as $P(I^1 \mid I^1)$, $P(I^2 \mid I^1)$

where $P(I \mid I) = P(I^1 \mid I^1) \cdot P(I^2 \mid I^1)$

and that will be calculated $P(I^1 \mid I^1)$

c) Pooling is better if the probability of at least one being infected is greater than 0.5.

Introduction

We have probably encountered such a number of spam messages, even once you started using digital forms of communication. In this exercise, we will use a Bayesian approach to judge the probability of a message being spam based on the words in the message.

To start you will need to download the dataset from

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset which contains a number of messages labelled as spam or non-spam. Use the file to move ahead in the dataset.

You will now need the packages `pandas`, `nltk` and `sklearn` installed

```
pip install pandas
pip install nltk
pip install scikit-learn
```

Loading and preparing the data

We start by loading and preparing the data. We will create a `pandas` `DataFrame` with columns `text` containing the corresponding whole a message is judged spam or ham and the cleaned version of the message respectively.

If you are unfamiliar with `Pandas`, we suggest you to take a look here:

https://www.kaggle.com/code/pythonkevin/pandas-tutorial-for-beginners and the package will likely be used throughout the course.

Since we would like to judge ham and spam (free stuff) messages, we first import the data file and run a function to clean and save the text. We use the `nltk` package for cleaning the spam files. The text will only be used to calculate the algorithm at the end of the notebook.

```
import pandas as pd
import nltk, re, os, sys, urllib.request, zipfile, io, ssl
from nltk.stem import WordNetLemmatizer

# We will use the stopwords to remove any message that contains
# only stopwords after cleaning and lemmatization.
nltk.download('stopwords', quiet=True)

def read_spam(path, encoding):
    rows = []
    with open(path, encoding=encoding):
        pass
    # Reading the file line by line.
    with open(path, encoding=encoding) as f:
        for line in f:
            parts = line.rstrip('\n').split('\t', maxsplit=1)
            if len(parts) == 2:
                rows.append(parts)
    df = pd.DataFrame(rows, columns=['label', 'text'])
    df['text_length'] = df['text'].str.len()
    return df

def clean_text(df, text_col='text', out_col='cleaned_text'):
    lemmatizer = WordNetLemmatizer()
    stopwords = set(nltk.corpus.stopwords.words('english'))
    def _clean(text):
        text = str(text).lower()
        text = re.sub(r'http\S+|www\.\S+', ' ', text)
        text = re.sub(r'[^a-z0-9\s]', ' ', text)
        tokens = [lemmatizer.lemmatize(t) for t in text.split() if t not in stopwords]
        return ' '.join(tokens)
    df[out_col] = df[text_col].apply(_clean)
    return df

df = read_spam('./data/SMSSpamCollection', 'utf-8')
df = clean_text(df)
df.shape
df.head()
```

```
(5572, 3)
```

	label	text	text_length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fin...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

```
df['label'].value_counts()
```

```
label
ham     4825
spam    747
Name: count, dtype: int64
```

Calculate prior probabilities

```
# Calculate the prior probabilities of a message being spam or ham, based on their occurrence.
# YOUR CODE HERE
# raise NotImplemented()
prior = df['label'].value_counts()/len(df)
prior_ham = prior['ham']
prior_spam = prior['spam']
print(f"The probability of a message being ham is {prior_ham:.4f}")
print(f"The probability of a message being spam is {prior_spam:.4f}")
```

```
The probability of a message being ham is 0.8659
The probability of a message being spam is 0.1341
```

Calculate likelihoods for words

```
# Calculate the probabilities of a word being in a spam or ham message
# YOUR CODE HERE
# raise NotImplemented()
ham = df[df['label'] == 'ham']
spam = df[df['label'] == 'spam']
ham_words = ham['cleaned_text'].str.split().explode()
spam_words = spam['cleaned_text'].str.split().explode()
ham_counts = ham_words.value_counts()
spam_counts = spam_words.value_counts()
print("Top 5 ham words:")
print(ham_counts.head())
print("\nTop 5 spam words:")
print(spam_counts.head())
```

```
Top 5 ham words:
cleaned_text
u    1094
ok    464
lar   441
gt    293
lt    289
Name: count, dtype: int64

Top 5 spam words:
cleaned_text
call    347
free    224
2       189
txt     150
ur      144
Name: count, dtype: int64
```

Calculate posterior probability

```
# Calculate the posterior probability of a message being spam
# YOUR CODE HERE
# raise NotImplemented()
def posterior(text, ham=ham, spam=spam, prior_ham=prior_ham, prior_spam=prior_spam):
    tokens = text.lower().split()
    n = len(tokens)
    if n == 0:
        return prior_spam
    p = 1.0
    for t in tokens:
        p *= (spam[t].sum() / len(spam)) / (ham[t].sum() / len(ham))
    return prior_spam * p / (prior_spam * p + prior_ham)

print("Test messages:")
for t in ["call me now", "free call", "ok lar", "free call ok"]:
    print(f"  {t!r} -> {posterior(t):.4f}")
```

```
Test messages:
  'call me now' -> 0.9999
  'free call' -> 0.9999
  'ok lar' -> 0.0000
  'free call ok' -> 0.9999
```

Now we can create a classifier

Create the following class in a new file, name it classifier.py. Note that for the exercise, it is okay to ignore words not in the likelihood table. Furthermore, you do not need to calculate evidence $P(w)$ for judging whether a message is more likely ham or spam.

```
class Classifier:
    def __init__(self):
        pass
    def train(self, df, text_col='cleaned_text', label_col='label'):
        prior = df[label_col].value_counts()/len(df)
        self.prior_ham = prior['ham']
        self.prior_spam = prior['spam']
        ham = df[df[label_col] == 'ham']
        spam = df[df[label_col] == 'spam']
        self.ham = ham[text_col].str.split().explode().value_counts()
        self.spam = spam[text_col].str.split().explode().value_counts()
        self.n = len(df)
        return self
    def predict(self, text):
        tokens = str(text).lower().split()
        n = len(tokens)
        if n == 0:
            return self.prior_spam
        p = 1.0
        for t in tokens:
            p *= (self.spam[t].sum() / len(self.spam)) / (self.ham[t].sum() / len(self.ham))
        return self.prior_spam * p / (self.prior_spam * p + self.prior_ham)
    def predict_proba(self, texts):
        return [self.predict(t) for t in texts]
    def evaluate(self, df, text_col='text', label_col='label'):
        preds = self.predict_proba(df[text_col])
        acc = sum(p > 0.5 == (l == 'spam') for p, l in zip(preds, df[label_col]))/len(df)
        print(f"Accuracy: {acc:.4f}")
        return acc
```

```
# YOUR CODE HERE
# raise NotImplemented()
# Train on the full dataset
clf = Classifier().train(df, text_col='cleaned_text', label_col='label')
# Evaluate on the same data
clf.evaluate(df, text_col='text', label_col='label')
```

```
Accuracy: 0.9866
```

```
0.9866
```

Bonus (optional)

Can you increase the classification accuracy of the Naive Bayes method? Write down at the beginning of the notebook the accuracy that you achieve on one of the new tasks. Can you substantiate algorithm the improvement?

We could think about smoothing out the probability distribution of a word over time to ensure we do not overfit the data. The problem here is that we get data sources constantly, where our results are very dependent on the current data.

Suppose that we have a spam only, which contains a word, which has never been written in our training dataset. Our new filter will pounce always the probability of the word occurring in it, which is our likelihood. Our posterior will be 1 and our filter will be very sure that the message is spam. Our filter will be very sure, because we have only seen the data once at the accuracy for the last set is the same as the training set. Using a very small dataset probability is very sure or something not in the probabilities exists.

Copyright © 2022 Matthias Gamperl. Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.