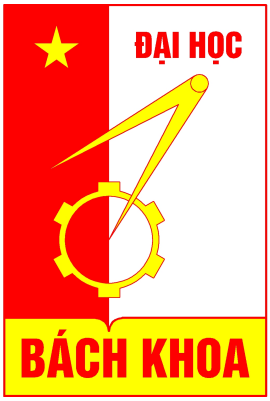
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
———————*******———————



BÁO CÁO
MÔN HỌC
Khai phá dữ liệu

Đề tài:

Phân loại tin nhắn SMS

Sinh viên thực hiện:

Họ và Tên	MSSV	Mã học phần
Đặng Quang Trung	20134145	IT4768
Trịnh Văn Duy	20130614	IT4768
Trần Văn Thành	20133561	IT4768

Giáo viên hướng dẫn: TS. Trịnh Anh Phúc

Hà Nội 14-05-2017

Mục lục

Lời cảm ơn **3**

1 Mô tả bài toán **4**

2 NaiveBayes **5**
2.1 Xử lý dữ liệu 5
2.2 Mô hình naivebayes 5
2.2.1 Định lí naivebayes 5
2.2.2 NaiveBayes cho phân loại spam 5
2.2.3 Ước lượng maximum likelihood 6
2.3 Phân phối gaussian 7
2.3.1 Công thức dư đoán 7

3 Logistic **8**
3.1 Giới thiệu TF-IDF 8
3.2 Giới thiệu về phân loại Logistic regression 9
3.3 Tiền xử lý 9
3.4 Quá trình trainning 10
3.5 Quá trình Test với bộ dữ liệu của Thầy 10

Lời cảm ơn

Chúng em xin chân thành cảm ơn thầy Trịnh Anh Phúc đã cung cấp cho chúng em những kiến thức vô cứng bổ ích khi chúng em bắt đầu tìm hiểu về môn học khai phá dữ liệu cũng như đã giúp đỡ chúng em rất nhiều trong quá trình thực nghiệm và viết bản báo cáo này.

Chương 1

Mô tả bài toán

Bài toán yêu cầu phân loại các tin nhắn chưa được gán nhãn thành 2 loại là tin nhắn rác (spam) và tin nhắn thông thường (ham). Dữ liệu cho là 100 tin nhắn đã được gán nhãn (1 tương ứng với ham và -1 tương ứng với spam). Đây là một bài toán Khai phá dữ liêu (Học máy) trên text với đặc điểm là tin nhắn tương đối ngắn, chứa nhiều ký tự đặc biệt, cần xử lý tốt trước khi tiến hành phân loại.

Chương 2

NaiveBayes

2.1 Xử lý dữ liệu

- Lower hóa tất cả cách word sau khi được tiền xử lý ở hai bước trên.
- Loại bỏ các kí tự gây nhiễu như: [](),"
- Tách các mẫu tin thành các term n-gram (n = 3).
- Xây dựng bộ từ điển theo các term sau khi phân tách n-gram.

2.2 Mô hình naivebayes

2.2.1 Định lí naivebayes

$$P(h|D) = \frac{P(D|h).P(h)}{P(D)}$$

- P(h): Xác suất trước (tiên nghiệm) của giả thiết (phân loại) h
- P(D): Xác suất trước (tiên nghiệm) của việc quan sát được dữ liệu D
- P(D|h): Xác suất (có điều kiện) của việc quan sát được dữ liệu D, nếu biết giả thiết (phân loại) h là đúng
- P(h|D): Xác suất (có điều kiện) của giả thiết (phân loại) h là đúng, nếu quan sát được dữ liệu D

2.2.2 NaiveBayes cho phân loại spam

Cho một tài liệu d(ở đây là một tin nhắn văn bản sms) và một nhãn c(nhẫn c nhận giá trị 1 và - 1 tương ứng với ham và spam).Ta có xác suất để d được phân vào nhãn c theo công thức:

$$P(c|d) = \frac{P(d|c).P(c)}{P(d)}$$

Ta sẽ gắn cho d một nhẫn lớp có khả năng nhất theo:

$$c_{MAP} = \arg\max_{c \in C} P(c|d) = \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)} = \arg\max_{c \in C} P(d|c)P(c)$$

Xác suất P(d) là như nhau với mọi tin nhắn d.
Tin nhắn d được biểu diễn bởi 1 số thuộc tính nên ta có công thức:

$$c_{MAP} = \arg\max_{c \in C} P(d|c)P(c) = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

Trong đó:

- $x_1, x_2, \ldots, x_n$ là vector các thuộc tính biểu diễn tin nhắn.

Các thuộc tính là độc lập với nhau nên ta có:

$$c_{NB} = \arg\max_{c \in C} P(c_j)\Pi_{x \in X} P(x|c)$$

Một tin nhắc sẽ được biểu diễn bởi 1 số các thuộc tính nên xác suất phân nhãn cho một tin nhắn:

$$c_{NB} = \arg\max_{c_j \in C} P(c_j)\Pi_{x_i \in X_d} P(x_i|c)$$

Trong đó:

- X_d là tập thuộc tính biểu diễn tin nhắn d.
- c_j tương ứng với nhẫn j trong danh sách nhẫn.

2.2.3 Ước lượng maximum likelihood

$$\hat{P}_{c_j} = \frac{doccount(C = c_j)}{N_{doc}}$$

$$\hat{P}_{w_i|c_j} = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w, c_j)}$$

Trong đó:

- $doccount(C = c_j)$ số tin nhắn có nhãn ứng với c_j.
- $count(w_i, c_j)$ số lần xuất hiện w_i(một term, word hay thuộc tính) trong các tin nhắn có nhẫn tương ứng là c_j.

Vấn đề $\hat{P}_{w_i|c_j} = \frac{count(w_i,c_j)}{\sum_{w \in V} count(w,c_j)} = 0$ nếu w_i không có trong tập train dẫn đến dự đoán sai.
Để khăc phục điều này và làm naive bayes mượt hơn dùng công thức sau thay thế:

$$\hat{P}_{(w_i|c)} = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\sum_{w \in V}(count(w, c)) + |V|}$$

2.3 Phân phối gaussian

- Dữ liệu thuộc vào 2 lớp đã biết là Ham và Spam. Mỗi lớp đều có phân bố độ dài tin nhắn normal với trung bình và độ lệch chuẩn μ_{spam} và σ_{spam} cho phân lớp spam và μ_{ham} và σ_{ham} cho phân lớp ham
- Chúng ta có thể định nghĩa một mô hình bằng cách lấy mẫu từ các phân bố này, sử dụng phân lớp Spam với xác suất P_{spam} và phân lớp B với xác suất P_{ham} (trong đó $P_{spam} + P_{ham} = 1$)

Các chuẩn được tính theo công thức với $n = n_{spam}$ hoặc $n = n_{ham}$.

$$\mu = \frac{X_1 + X_2 + X_3 + X_4 + \ldots + X_{n-1} + X_n}{n}$$

$$\sigma^2 = \frac{(X_1 - \mu)^2 + (X_2 - \mu)^2 + \ldots + (X_n - \mu)^2}{n - 1}$$

Cho một tin nhắc sms x_i xác suất để thuộc phân lớp là A là spam hoăc ham là:

$$P(A|x_i) = \frac{P(x_i|A).P(A)}{P(x_i)} = \frac{N(x_i, \mu_A, \sigma_A)P_A}{N(x_i, \mu_A, \sigma_A)P_A + N(x_i, \mu_B, \sigma_B)P_B}$$

Ở đây N() là chuẩn của phân phối Gaussian công thức của N():

$$N(X, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(X-\mu)^2}{2\sigma^2}}$$

2.3.1 Công thức dư đoán

$$c_{NB} = \arg\max_{c_j \in C} P(c_j)P(c_j|len(d))\Pi_{x_i \in X_d} P(x_i|c_j)$$

Chương 3

Logistic

3.1 Giới thiệu TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) là 1 kĩ thuật khai phá dữ liệu từ text sử dụng để phân loại văn bản . Trọng số này sử dụng để đánh giá tầm quan trọng của một từ trong một văn bản , 1 collection hoặc một corpus . Độ quan trọng tăng dần dựa vào số lần từ xuất hiện trong văn bản nhưng bù lại bởi tần suất của từ đó trong corpus . Một vài biến thể của tf-idf thường xuyên được sử dụng trong các hệ thống tìm kiếm như một công cụ chính để đánh giá và sắp xếp văn bản dựa vào user query. Tf-idf có thể sử dụng để lọc những từ stop-words trong một số bài toán như tóm tắt văn bản và phân loại văn bản .

TF : Term Frequency , là số lần term xuất hiện trong văn bản . Vì các văn bản có thể có độ dài ngắn khác nhau nên mọt số term có thể xuất hiện nhiều lần trong một văn bản dài hơn là một văn bản ngắn . Như thế , term frequency thường được chia cho độ dài văn bản (tổng số term trong một văn bản) như một các normalization .

IDF: Inverse Document Frequency, đánh giá tầm quan trọng của một term . Khi tính toán TF , tất cả các terms được coi như có độ quan trọng như nhau . Nhưng một số terms, như "is" , "of" và "that" thường xuất hiện rất nhiều lần nhưng độ quan trọng là không cao . Như thế chúng ta cần giảm wegh xuống

IDF(t) = log(Tổng số văn bản/ Số văn bản chứa term t).

Ví dụ một văn bản chưa 100 từ mà từ "cat" xuất hiện 3 lần . Term frequency cho từ cat này là (3/100) = 0.03 . Giả sử , chúng ta có 10 triệu văn bản và từ "cat" xuất hiện trong một nghìn văn bản . Như thế , idf được dính là log(10,000,000 / 1,000) = 4. Như thế , tf-idf của từ "cat" trong văn bản này sẽ là : 0.03 * 4 = 0.12.

3.2 Giới thiệu về phân loại Logistic regression

Là một bộ phân loại với khá mạnh mẽ, có thể áp dụng cho những bài toán phân loại phi tuyến, phù hợp với trường hợp dữ liệu để huấn luyện ít, trong bài toán của chúng ta, bộ phân loại này là thích hợp cùng với bộ phân loại văn bản Naive Bayes Hàm dự đoán của bộ phân loại này có dạng:

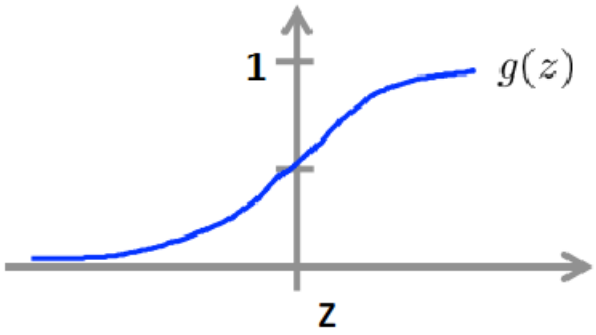
$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

Trong đó:

- θ là trọng số tương ứng với đặc trưng dữ liệu x-i.
- Hàm g(z) ở đây được gọi là hàm sigmoid hay hàm logistic.

Áp dụng với bài toán của chúng ta, bộ phân loại sẽ phân loại tin nhắn là spam nếu $h_\theta(x) \geq 0.5$ và sẽ phân loại tin nhắn là ham nếu ngược lại. Đồ thị của hàm phân loại có dạng:



Hình 3.1: Hình minh họa hàm g(z)

Hàm mất mát (loss function) được định nghĩa theo công thức:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-y^{(i)} log(h_\theta(x^{(i)})) - (1 - y^{(i)}) log(1 - h_\theta x^{(i)}))]$$

Hàm này sẽ được tối ưu theo thuật toán gradient decent hay bất kỳ thuật toán tối ưu không ràng buộc nào khác. Đối với bài toán của chúng ta, sau khi vector hóa tin nhắn bằng trọng số TF-IDF, ta sẽ fit dữ liệu vào bộ phân loại này, kết quả tốt hơn so với khi sử dụng SVM, bởi vì dữ liệu huấn luyện ở đây tương đối ít.

3.3 Tiền xử lý

- Mã hóa các xâu có pattern phù hợp, ví dụ các từ trong tin nhắn khớp với pattern date sẽ được thay thế bằng từ 'date', các từ khớp

với phone_pattern sẽ được thay thế bằng từ 'phone', tương tự với link_pattern, currency_pattern, emotion_pattern, number_pattern và sex_pattern. Quá trình này được thực hiện trên cả tập Train và Test.

- Thay thế các ký tự đặc biệt, quan trọng, giữ lại dấu ? và thay thế bằng 'qMark', còn lại các ký tự đặc biệt khác không khớp với pattern_emotion sẽ được thay thế bằng dấu cách.
- Lower hóa tất cả cách word sau khi được tiền xử lý ở hai bước trên
- Với những bước tiền xử lý trên, ta sẽ dùng bộ đánh trọng số TF-IDF với những tin nhắn trong tập train, cùng với đó là fit bộ đánh trong số đó cho tập test.

3.4 Quá trình trainning

- Dữ liệu gồm 100 tin nhắn sau khi được tiền xử lý
- Sẽ được chia thành tập train gồm 75 tin nhắn, tập validate để tuning tham số gồm 25 tin nhắn còn lại.
- Kết quả được đánh giá dựa trên các ước lượng
- Sử dụng bộ phân loại Logistic Regression (Vì bộ phân loại này thích hợp hơn so với SVM do tập train quá ít dữ liệu)
- Kết quả tương đối tốt (F1= 0.85 -> 0.97)

3.5 Quá trình Test với bộ dữ liệu của Thầy

- Sử dụng toàn bộ 100 tin nhắn để train như trước
- Và chạy test trên 300 tin nhắn thầy gửi.

Sau đây là code python (3.5) thể hiện ý tưởng trên. Để chạy được code này cần cài đặt các thư viện numpy, sklearn, và codecs. Các file test.txt và train.txt được đặt ở cùng một thư mục với file source code.

```
1  import numpy as np
2  import re
3  from sklearn.linear_model.logistic import LogisticRegression
4  from sklearn.cross_validation import train_test_split, cross_val_score
5  from sklearn.feature_extraction.text import TfidfVectorizer
6  from sklearn.metrics import accuracy_score
7  from sklearn.metrics import confusion_matrix
8  from builtins import str
9  import codecs
10 date_pattern = []
11 phone_pattern = []
12 link_pattern = []
13 currency_pattern = []
14 emotion_pattern = []
15 number_pattern=[]
```

```
sex_pattern=[]

date_string = ' date '
phone_string = ' phone '
link_string = ' link '
currency_string = ' currency '
emotion_string = ' emotion '
number_string=' number '
sex_string=' sex'

date_pattern.append(r'\d{1,2}/\d{1,2}/\d{2,4}')
date_pattern.append(r'\d{1,2}-\d{1,2}-\d{2,4}')
date_pattern.append(r'\+\d{1,2}:\d{1,4}')
date_pattern.append(r'\d{1,2}/\d{1,4}')
date_pattern.append(r'\d{1,2}-\d{1,4}')
date_pattern.append(r'\d{1,2}h\d{0,2}')
date_pattern.append(r'phut')
date_pattern.append(r'giay')

phone_pattern.append(r'\+\d{10,12}')
phone_pattern.append(r'\d{3,5}\.\d{3,4}\.\d{3,5}')
phone_pattern.append(r'\d{8,12}')
phone_pattern.append(r'1800\d{2,4}')
phone_pattern.append(r'1900\d{2,4}')
phone_pattern.append(r'\d{4}')
phone_pattern.append(r'[0]\d{2,3}')
phone_pattern.append(r'195')
phone_pattern.append(r'900')
phone_pattern.append(r'999')
phone_pattern.append(r'1342')
phone_pattern.append(r'191')
phone_pattern.append(r'888')
phone_pattern.append(r'333')
phone_pattern.append(r'1414')
phone_pattern.append(r'1576')
phone_pattern.append(r'8170')
phone_pattern.append(r'9123')
phone_pattern.append(r'9118')
phone_pattern.append(r'266')
phone_pattern.append(r'153')
phone_pattern.append(r'199')
phone_pattern.append(r'9029')
phone_pattern.append(r'8049')
phone_pattern.append(r'1560')
phone_pattern.append(r'9191')
phone_pattern.append(r'8709')
phone_pattern.append(r'9241')
phone_pattern.append(r'7393')
phone_pattern.append(r'7719')

link_pattern.append(r'www\..*')
link_pattern.append(r'http://.*')

currency_pattern.append(r'[0-9|\,\.]{3,}VND')
currency_pattern.append(r'[0-9|\.]{3,}VND')
currency_pattern.append(r'[0-9|\.]{3,}d')
currency_pattern.append(r'[0-9|\.]{3,}d')
currency_pattern.append(r'[0-9|\.]{3,}tr')
currency_pattern.append(r'[0-9|\.]{3,}Tr')
currency_pattern.append(r'[0-9|\.]{3,}TR')
currency_pattern.append(r'\d{1,3}trieu')
```

```
currency_pattern.append(r'\d{1,3}tr')
currency_pattern.append(r'\d{1,1000}ti')
currency_pattern.append(r'\d{1,3}k')
currency_pattern.append(r'\d{1,3}nghin')
currency_pattern.append(r'\d{1,3}tram')

emotion_pattern.append(r'o.O')
emotion_pattern.append(r'O.o')
emotion_pattern.append(r'\(y\)')
emotion_pattern.append(r'\(Y\)')
emotion_pattern.append(r':v')
emotion_pattern.append(r':V')
emotion_pattern.append(r':3')
emotion_pattern.append(r'-_-')
emotion_pattern.append(r'\^_\^')
emotion_pattern.append(r'<3')
emotion_pattern.append(r':-\*')
emotion_pattern.append(r':\*')
emotion_pattern.append(r":'\(")
emotion_pattern.append(r':p␣')
emotion_pattern.append(r':P')
emotion_pattern.append(r':d')
emotion_pattern.append(r':D')
emotion_pattern.append(r':-\?')
emotion_pattern.append(r'>\.<')
emotion_pattern.append(r'><')
emotion_pattern.append(r':-\w␣')
emotion_pattern.append(r':\)\)')
emotion_pattern.append(r';\)\)')
emotion_pattern.append(r'=\)\)')
emotion_pattern.append(r':-\)')
emotion_pattern.append(r':\)')
emotion_pattern.append(r':\]')
emotion_pattern.append(r'=\)')
emotion_pattern.append(r':-\(')
emotion_pattern.append(r':\(')
emotion_pattern.append(r':\[')
emotion_pattern.append(r'=\(')
emotion_pattern.append(r'sock')
emotion_pattern.append(r'haizz')


sex_pattern.append(r'xxx')
sex_pattern.append(r'sexy')
sex_pattern.append(r'9x')

number_pattern.append(r'\d{1,}')
corpus=list()
labels=list()
test_corpus=list()
test_corpus_to_print=list()
file_train=codecs.open('train.txt','r','utf-8')
file_test=codecs.open('test.txt','r')
for line in file_train:
    if line[0]=='1':
        labels.append(1)
        corpus.append(line[1:].strip().lower())
    else:
        labels.append(-1)
        corpus.append(line[2:].strip().lower())

```

```
for line in file_test:
    test_corpus.append(line[1:].strip().lower())
    test_corpus_to_print.append(line[1:].strip())


# test_corpus_to_print=test_corpus

i=0
for line in corpus:
    for pattern in date_pattern:
        line=re.sub(pattern, date_string, line)
    for pattern in emotion_pattern:
        line=re.sub(pattern, emotion_string, line)
    for pattern in sex_pattern:
        line=re.sub(pattern, sex_string, line)
    for pattern in currency_pattern:
        line=re.sub(pattern, currency_string, line)
    for pattern in phone_pattern:
        line=re.sub(pattern, phone_string, line)
    for pattern in link_pattern:
        line=re.sub(pattern, link_string, line)
    for pattern in number_pattern:
        line=re.sub(pattern, number_string, line)
    line=line.replace('?',' qMark ')
    line=line.replace('$',' currency ')
    stop_list = ['.', ',', '/', ';', ':', '&', '@', '!', '‘',
                "'", '"', '>', '<', '*', '%', '#', '(', ')', '[',']',
               '-', '_', '=', '+', '{', '}', '~', '^', '*', '|', '\\']
    for item in stop_list:
        line=line.replace(item,' ')
    corpus[i]=line
    i+=1



i=0
for line in test_corpus:
    for pattern in date_pattern:
        line=re.sub(pattern, date_string, line)
    for pattern in emotion_pattern:
        line=re.sub(pattern, emotion_string, line)
    for pattern in sex_pattern:
        line=re.sub(pattern, sex_string, line)
    for pattern in currency_pattern:
        line=re.sub(pattern, currency_string, line)
    for pattern in phone_pattern:
        line=re.sub(pattern, phone_string, line)
    for pattern in link_pattern:
        line=re.sub(pattern, link_string, line)
    for pattern in number_pattern:
        line=re.sub(pattern, number_string, line)
    line=line.replace('?',' qMark ')
    line=line.replace('$',' currency ')
    stop_list = ['.', ',', '/', ';', ':', '&', '@', '!', '‘',
    "'", '"', '>', '<', '*', '%', '#', '(', ')', '[',
     ']', '-', '_', '=', '+', '{', '}', '~', '^', '*', '|', '\\']
    for item in stop_list:
        line=line.replace(item,' ')
    test_corpus[i]=line
    i+=1

```

```
199
200
201 Vectorizer=TfidfVectorizer()
202 X_train_raw=corpus
203 y_train=labels
204 X_test_raw=test_corpus
205
206 # for i,item in enumerate(X_train_raw):
207 #     print(str(i)+"␣"+item)
208 X_train=Vectorizer.fit_transform(X_train_raw)
209 X_test=Vectorizer.transform(X_test_raw)
210 classifier=LogisticRegression()
211 classifier.fit(X_train, y_train)
212 predictions=classifier.predict(X_test)
213
214 for i in range(len(predictions)):
215     print(str(predictions[i]) +"␣"+ test_corpus[i])
216
217 file_result=open('result.txt','w')
218 for i in range(len(predictions)):
219     file_result.write(str(predictions[i]) + "␣"
220                 + test_corpus_to_print[i]+"\n")
221 # for i, prediction in enumerate(predictions[:]):
222 #     print('Ground␣truth␣is␣%s␣␣␣Prediction:␣%s.␣Message:␣%s'
223                                 % (y_test[i],prediction, X_test_raw[i]))
224 # print("Accuracy:␣",accuracy_score(y_test, predictions))
225 # confusion_matrix = confusion_matrix(y_test, predictions)
226 # print(confusion_matrix)
227 # precisions = cross_val_score(classifier, X_train,
228                 y_train, cv=5,scoring='precision')
229 # print('Precision', np.mean(precisions), precisions)
230 # recalls = cross_val_score(classifier, X_train,
231                 y_train, cv=5,scoring='recall')
232 # print('Recalls', np.mean(recalls), recalls)
233 # f1s = cross_val_score(classifier, X_train,
234                 y_train, cv=5,scoring='f1')
235 # print('F1', np.mean(f1s), f1s)
```

Tài liệu tham khảo

[1] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[2] Slide Text Classification and Naïve Bayes Dan Jurafsky

[3]