

ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



BÁO CÁO
CASE STUDY 3
LOGISTIC REGRESSION
NEWS HEADLINES FOR SARCASM DETECTION

Môn: Máy học

Lớp: KHTN2017

Mã lớp: CS114.J21.KHTN

GVHD: Huỳnh Thị Thanh Thương

Lê Quốc Thịnh

17521087

Hà Quốc Tiến

17521122

TP. HCM, ngày 12 tháng 05 năm 2019

Mục lục

1. PHÁT BIỂU BÀI TOÁN	3
1.1. Mục đích	3
1.2. Đầu vào	3
1.3. Đầu ra.....	3
1.4. Cách đánh giá mô hình	3
2. XÂY DỰNG MÔ HÌNH TỪ TẬP HUẤN LUYỆN.....	3
2.1. Tiền xử lí.....	3
2.2. Xây dựng mô hình Logistic Regression.....	4
3. THỰC NGHIỆM ĐÁNH GIÁ	4
3.1. Các cách thức đánh giá	4
3.2. Kết quả đánh giá	5
3.3. Nhận xét	5
4. LẬP TRÌNH CÀI ĐẶT	5
4.1. Tool và thư viện đã dùng	5
4.1.1. Anaconda	5
4.1.2. Sklearn	6
4.1.3. NumPy	6
4.1.4. Numba.....	6
4.1.5. Pandas	6
4.2. Cách cài đặt tool và chạy chương trình.....	6
4.2.1. Cài đặt tool.....	6
4.2.2. Chạy chương trình	6
4.3. Mã nguồn	7
4.4. Kết quả chạy.....	9
5. TÀI LIỆU THAM KHẢO	9

1. PHÁT BIỂU BÀI TOÁN

1.1. Mục đích

Xây dựng các mô hình dự đoán, phát hiện tin bài châm biếm dựa trên phương pháp Logistic Regression và đánh giá, so sánh với các phương pháp máy học đã biết.

1.2. Đầu vào

Dataset gồm 26.709 điểm dữ liệu là các bài viết với những thông tin sau: đường dẫn liên kết đến bài viết (“article_link”), tiêu đề (“Headline”) và đặc trưng mang giá trị nhị phân trả lời cho câu hỏi liệu đây có phải là tin châm biếm hay không (“is_sarcastic”).

1.3. Đầu ra

Mô hình phỏng đoán liệu một headline x có phải là tin châm biếm hay không (bài toán Binary Classification).

1.4. Cách đánh giá mô hình

Đánh giá dựa trên accuracy, precision, recall và F1-score.

2. XÂY DỰNG MÔ HÌNH TỪ TẬP HUẤN LUYỆN

2.1. Tiền xử lí

- Dùng thư viện pandas để đọc file json, có thể tiếp tục loại bỏ các ký tự đặc biệt và các stopwords khỏi “Headline” vì chúng dường như không mang ý nghĩa đối với bộ dữ liệu.
- Chia Dataset thành 2 tập trainData và testData, trong đó 20.000 records đầu tiên dùng để huấn luyện mô hình và 6.709 records còn lại dùng để kiểm tra.
- Tiếp tục chia 2 tập này thành các tập: trainX, trainY, testX, testY. Trong đó, trainX, testX là các điểm dữ liệu chỉ chứa “Headline” còn trainY, testY là giá trị nhị phân “is_sarcastic”.
- Vector hóa các tiêu đề trên trainX, testX bằng cách biến đổi tf-idf. Độ dài của mỗi vector ứng với tổng số term khác nhau trong trainX và mỗi phần tử của vector được tính toán dựa trên công thức:

$$w(t, d) = tf(t, d).idf(t, D)$$

Trong đó,

$$tf(t, d) = \frac{n(t)}{n(d)}$$

$$idf(t, D) = \log \left(\frac{|D|}{|d: d \in D, t \in D|} \right)$$

Với:

$n(t)$: số lần lặp lại của term t trong document d

$n(d)$: tổng số term trong document d

$|D|$: tổng số document

$|d: d \in D, t \in D|$: tổng số document d có chứa term t

2.2. Xây dựng mô hình Logistic Regression

- Mô hình Logistic Regression có dạng:

$$\hat{y} = P(x) = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

Trong đó,

$$z = \ln(\text{odds}) = w_0 + w_1x_1 + \dots + w_nx_n = X \cdot W$$

- Việc xây dựng mô hình đồng nghĩa với việc tìm W sao cho:

$$W = \text{argmin } E(W)$$

Trong đó,

$$E(W) = \sum_{i=1}^k E(W, x_i, y_i)$$

$$E(W, x_i, y_i) = -[y_i \log(P(x_i)) + (1 - y_i) \log(1 - P(x_i))]$$

Với:

$E(W)$: độ lỗi của mô hình, hay còn gọi là hàm mất mát

$E(W, x_i, y_i)$: độ lỗi tại điểm dữ liệu (x_i, y_i)

- Sử dụng Gradient Descent với momentum để cập nhật cho W :

$$W = W - v_t$$

Trong đó,

$$v_t = \gamma \cdot v_{t-1} + \frac{\alpha}{1 - \gamma} \cdot \nabla L(W)$$

$$\nabla L(W) = X^T \cdot (P(X) - Y)$$

Với:

γ : độ lưu trữ vận tốc trước đó

v_{t-1} : vận tốc trước đó

α : learning rate

$\nabla L(W)$: gradient

3. THỰC NGHIỆM ĐÁNH GIÁ

3.1. Các cách thức đánh giá

- Accuracy: số điểm được dự đoán chính xác trên tổng số điểm dữ liệu:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Precision: số điểm dương tính (positive) được dự đoán chính xác trên tổng số điểm được dự đoán là dương tính:

$$Precision = \frac{TP}{TP + FP}$$

- Recall: số điểm dương tính được dự đoán chính xác trên tổng số điểm dương tính trên thực tế:

$$Recall = \frac{TP}{TP + FN}$$

- F1-score: hàm trung bình điều hòa của Precision và Recall:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

Với:

TP (True Positive): mẫu dương tính được dự đoán chính xác

TN (True Negative): mẫu âm tính (negative) được dự đoán chính xác

FP (False Positive): mẫu âm tính được dự đoán là dương tính

FN (False Negative): mẫu dương tính được dự đoán là âm tính

3.2. Kết quả đánh giá

```
In [1]: runfile('C:/Users/tien/.spyder-py3/ml case study #3/source.py', wdir='C:/Users/tien/.spyder-py3/ml case study #3')
SkLearn model:
{'accuracy': 0.8424504397078552, 'precision': 0.8303350970017637, 'recall': 0.8034129692832764, 'f1': 0.816652211621856}
BDG Logistic Regression:
{'accuracy': 0.8309733194216724, 'precision': 0.7902391725921137, 'recall': 0.8344709897610921, 'f1': 0.8117529880478087}
KNN model:
{'accuracy': 0.7545088686838575, 'precision': 0.799067599067599, 'recall': 0.5849829351535836, 'f1': 0.6754679802955664}
Decision Tree model:
{'accuracy': 0.7366224474586377, 'precision': 0.6967851099830795, 'recall': 0.7027303754266212, 'f1': 0.6997451146983857}
Naïve Bayes model:
{'accuracy': 0.840512744075123, 'precision': 0.8652788688138257, 'recall': 0.7518771331058021, 'f1': 0.8046018991964939}
```

3.3. Nhận xét

Nhìn chung, phương pháp KNN và Decision Tree cho kết quả còn hạn chế so với các phương pháp khác. Trong khi đó, phương pháp Logistic Regression được sinh viên tự xây dựng dù không phải là mô hình hiệu quả nhất nhưng vẫn ở mức chấp nhận được.

4. LẬP TRÌNH CÀI ĐẶT

4.1. Tool và thư viện đã dùng

4.1.1. Anaconda

Anaconda là một bản phân phối miễn phí và mã nguồn mở của Python. Nó bao gồm nhiều thư viện phổ biến của Python và một số công cụ như Spyder IDE, Jupyter notebook,... Anaconda hướng đến việc quản lý các package một cách đơn giản, phù hợp với mọi người. Hệ thống quản lý package của Anaconda là Conda. Bản phân phối Anaconda tính đến nay được sử dụng hơn 6 triệu người, và đi kèm hơn 250 gói package khoa học dữ liệu phù hợp cho Windows, Linux và MacOS.

4.1.2. Sklearn

Scikit-learn là thư viện mã nguồn mở mang tính đơn giản và hiệu quả cho việc khai thác dữ liệu và phân tích dữ liệu, đặc biệt thư viện cung cấp rất nhiều công cụ cho máy học. Nó được xây dựng dựa trên numpy, scipy và matplotlib nên rất tương thích với các thư viện này.

4.1.3. NumPy

Numpy là thư viện quan trọng cho việc tính toán khoa học trong Python. Nó cung cấp mảng đa chiều hiệu năng cao và những công cụ để làm việc với mảng.

4.1.4. Numba

Numba dịch code hàm Python thành mã máy ở runtime, dùng thư viện compiler LLVM. Thuật toán số học viết trong Python được compile bởi Numba có thể tiệm cận tốc độ của C hay FORTRAN. Numba được thiết kế để làm việc với mảng NumPy một cách hiệu quả.

4.1.5. Pandas

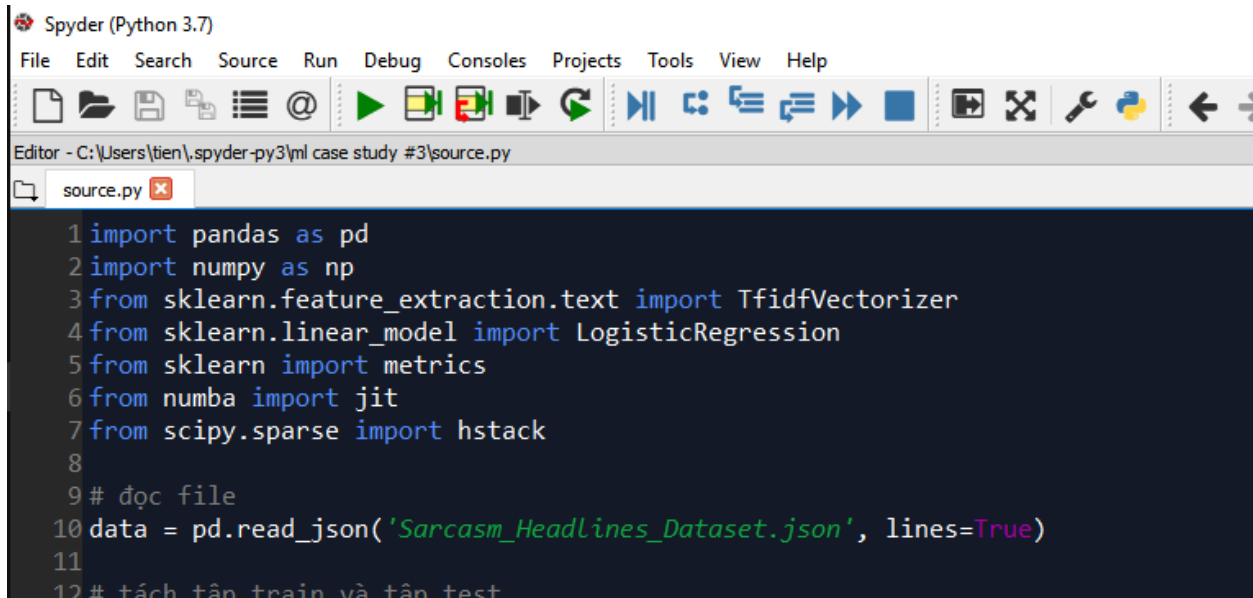
Pandas là một thư viện mã nguồn mở cung cấp cấu trúc dữ liệu hiệu năng cao, dễ sử dụng và các công cụ phân tích dữ liệu cho ngôn ngữ Python.

4.2. Cách cài đặt tool và chạy chương trình

4.2.1. Cài đặt tool

- Cài đặt Anaconda: <https://docs.anaconda.com/anaconda/install/>
- Thư viện Sklearn, NumPy, Numba và Pandas đều đã được bao gồm trong Anaconda.

4.2.2. Chạy chương trình



```
1 import pandas as pd
2 import numpy as np
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 from sklearn.linear_model import LogisticRegression
5 from sklearn import metrics
6 from numba import jit
7 from scipy.sparse import hstack
8
9 # đọc file
10 data = pd.read_json('Sarcasm_Headlines_Dataset.json', lines=True)
11
12 # tách tập train và tập test
```

Soạn thảo mã nguồn trên Spyder IDE và chọn nút  để chạy.

4.3. Mã nguồn

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from numba import jit
from scipy.sparse import hstack

# đọc file
data = pd.read_json('Sarcasm_Headlines_Dataset.json', lines=True)

# tách tập train và tập test
trainData = data[:20000]
trainX = np.array(trainData['headline'])
trainY = np.array(trainData['is_sarcastic'])
testData = data[20000:]
testX = np.array(testData['headline'])
testY = np.array(testData['is_sarcastic'])

# vector hóa tiêu đề
vectorizer = TfidfVectorizer().fit(trainX)
trainXvectorized = vectorizer.transform(trainX)
testXvectorized = vectorizer.transform(testX)

# mô hình từ thư viện
model = LogisticRegression(solver='saga').fit(trainXvectorized, trainY)
y_pred_sklearn = model.predict(testXvectorized)
model.score(testXvectorized, testY)

# hàm tính accuracy, precision, recall, f1-score
def score(y_true, y_pred):
    accu = metrics.accuracy_score(y_true, y_pred)
    prec = metrics.precision_score(y_true, y_pred)
    reca = metrics.recall_score(y_true, y_pred)
    f1score = metrics.f1_score(y_true, y_pred)
    return {'accuracy': accu, 'precision': prec, 'recall': reca, 'f1': f1score}

print('SkLearn model: ')
print(score(testY, y_pred_sklearn))

# hàm tính sigmoid
@jit
```

```

def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))

# hàm tính độ lỗi
@jit
def loss(p, y):
    return np.sum(-y * np.log(p) - (1 - y) * np.log(1 - p))

@jit
# hàm thêm cột 1 vào ma trận examples
def add1Col(x):
    intercept = np.ones((x.shape[0], 1))
    return hstack((intercept, x))

# hàm tính momentum
@jit
def getMomentum(gradient, vPrev, alpha, gamma=0.9):
    v = gamma*vPrev + alpha/(1-gamma)*gradient
    return v

# hàm chạy batch gradient descent
@jit
def BGD(x, y, lr, numIter=10000, epsilon=1e-10):
    x = add1Col(x)
    theta = np.zeros(x.shape[1])
    v = np.zeros(x.shape[1])
    E = -float('inf')
    for i in range(numIter):
        z = x.dot(theta)
        p = sigmoid(z)
        gradient = x.T.dot(p - y)
        v = getMomentum(gradient, v, lr)
        theta = theta - v
        Enext = loss(p, y)
        if abs(E-Enext) < epsilon:
            break
        E = Enext
    return theta

@jit
def predict_probs(X, theta):
    return sigmoid(X.dot(theta))

```



```

@jit
def predict(X, theta, threshold=0.5):
    return np.array(predict_probs(X, theta) >= threshold, dtype=int)

w = BGD(trainXvectorized, trainY, 0.015, 200)
y_pred = predict(add1Col(testXvectorized), w)
print('BDG Logistic Regression: ')
print(score(testY, y_pred))

# model KNN
from sklearn.neighbors import KNeighborsClassifier
KNNmodel = KNeighborsClassifier(n_neighbors = 7,
metric='euclidean').fit(trainXvectorized, trainY)
knn_y_pred = KNNmodel.predict(testXvectorized)
print('KNN model: ')
print(score(testY, knn_y_pred))

# model Decision Tree
from sklearn.tree import DecisionTreeClassifier
DTmodel = DecisionTreeClassifier().fit(trainXvectorized, trainY)
dt_y_pred = DTmodel.predict(testXvectorized)
print('Decision Tree model: ')
print(score(testY, dt_y_pred))

# naive bayes model
from sklearn.naive_bayes import BernoulliNB
NBmodel = BernoulliNB().fit(trainXvectorized, trainY)
nb_y_pred = NBmodel.predict(testXvectorized)
print('Naive Bayes model: ')
print(score(testY, nb_y_pred))

```

4.4. Kết quả chạy

```

In [1]: runfile('C:/Users/tien/.spyder-py3/ml case study #3/source.py', wdir='C:/Users/tien/.spyder-py3/ml case study #3')
SkLearn model:
{'accuracy': 0.8424504397078552, 'precision': 0.8303350970017637, 'recall': 0.8034129692832764, 'f1': 0.816652211621856}
BDG Logistic Regression:
{'accuracy': 0.8309733194216724, 'precision': 0.7902391725921137, 'recall': 0.8344709897610921, 'f1': 0.8117529880478087}
KNN model:
{'accuracy': 0.7545088686838575, 'precision': 0.799067599067599, 'recall': 0.5849829351535836, 'f1': 0.6754679802955664}
Decision Tree model:
{'accuracy': 0.7366224474586377, 'precision': 0.6967851099830795, 'recall': 0.7027303754266212, 'f1': 0.6997451146983857}
Naive Bayes model:
{'accuracy': 0.840512744075123, 'precision': 0.8652788688138257, 'recall': 0.7518771331058021, 'f1': 0.8046018991964939}

```

5. TÀI LIỆU THAM KHẢO

- [Preprocessing with sklearn: a complete and comprehensive guide](#)
- [Pandas Documentation](#)

- [SkLearn Documentation](#)
- [NumPy Documentation](#)
- [Machine learning Cơ bản](#)