TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN TP.HCM

KHOA: KHOA HỌC MÁY TÍNH

BÀI TẬP NHÓM

CASE STUDY 3: NEWS HEADLINES DATASET FOR SARCASM DETECTION – HIGH QUALITY DATASET FOR THE TASK OF SARCASM DETECTION

SUBJECT: MACHINE LEARNING

TEACHER: HUỲNH THỊ THANH PHƯƠNG

CLASS: CS114.J22.KHCL

❖ ***Thành viên tham gia:***

1. Huỳnh Minh Tuấn – 17521212
2. Nguyễn Thanh Tú – 17521201
3. Hoàng Ngọc Quân - 17520934
4. Ngô Anh Vũ - 17521272

MỤC LỤC

CHƯƠNG 1: BÀI TOÁN PHÁT HIỆN TIN CHÂM BIẾM

1.1 Giới thiệu.

+ Ngày nay mạng xã hội, trang tin tức,…trên internet xuất hiện ngày càng nhiều và càng phổ biến . Chúng ta có thể dễ dàng truy cập và cập nhật thông tin từ các trang. Chính vì tính phổ biến như vậy mà tin tức cũng được cập nhật liên tục và thông tin lại rất tràn lan không được kiểm soát. Do đó, những thông tin giả mạo xen lẫn với những thông tin đúng đắn, những tin chính thống, làm cho thông tin mang lại không có độ chính xác cho người đọc, gây hoang mang khi người đọc đọc phải.

+ Xuất phát từ vấn đề đó chúng ta cần phải xây dựng môt mô hình dự đoán một Headlines(Tiêu đề) có phải là một tin giả mạo hay không . Có rất nhiều phương pháp dung để phân loại như: Support Vector Machine, K-Nearest Neighbor, Naïve Bayes, Decision Tree…Điểm chung của các phương pháp này điều dựa vào xác suất thống kê . Ở bài toán này ta sẽ dung phương pháp Logistic regression để đánh giá và phân loại những tin châm biếm của một tin . Trong bài toán phân loại và đánh giá này, ta sẽ đánh giá hiệu quả và chính xác của phương pháp này so với các phương pháp khác(1KNN-KNN, Naives Bayes, Decision Tree,…) để xem phương pháp này có phải là phương pháp tối ưu, hiệu quả và chính xác nhất trong các phương pháp của dạng bài toán phân lớp.

1.2. Phát biểu bài toán.

Bài toán " Phát hiện tin giả mạo" có thể được phát biểu như sau:

Cho trước một tập tin tức - vector x = {x1, x2, x3,…,xn}là input và output là một tập giá trị rời rạc y = {y1, y2,y3,…yn} tương ứng với mỗi giá trị xi nhập vào, ta tính được giá trị yi nằm trong vùng giá trị [0, 1]. Dự đoán xi có phải là tin châm biếm hay không.(Nếu yi tiến về 1 là tin châm biếm, tiến về 0 là tin chính thống).

- Hàm sigmod: sigmoid($w_0 + w_1x$) = $\frac{e^{w0+w1x}}{1+e^{w0+w1x}}$
- Hàm mất mát:
  E(w) = - yi ¿log(P(xi)) + (1 – yi) ¿log(1−P(xi))
- Cập nhật cho logistic regression: w = w - $\gamma * \nabla E(w)$

1.3. Mô hình bài toán.

- Mục đích:

  Với mục đích muốn đánh giá một bài toán mô hình tuyến tính giống với Linear Regression và Perceptron Learning Algorithm chúng đều có chung một dạng.

  $$y=f(w^Tx)$$

  Thì cùng với mục đích như vậy thì ta dự đoán đầu ra của Logistic Regression cũng có dạng.

  $$f(x)=\theta(w^Tx)$$

  đối với bài toán hiện tại được áp dụng để xác định sự tối ưu giữa những text với những tin châm biếm hoặc chính thống được đánh dấu (0,1).

- Đầu vào:

  Nhập dữ liệu của một Dataset được thu thập từ 2 trang web
  1) TheOnion toàn đăng những tin châm biếm
  2) HuffPost bao gồm những tin chính thống

  Chỉ thu thập các tiêu đề của các bài viết trong trang web.

```
datastore = pd.read_json("Sarcasm_Headlines_Dataset.json", lines = True).drop('article_link',axis = 1)
print(datastore)
lemmatizer = WordNetLemmatizer()
```

- Đầu ra:
  Xử lý bằng Logistic Regression dể đánh giá tiêu đề bài viết có phải là tin châm biếm hay tin chính thông không?
  - Xử lí tính toán sigmoid

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

  - Xử lí hàm Logistic :

```
class logistic():
    def fit(self, X, Y):
        self.N = X.shape[0]
        self.d = X.shape[1]

        ones = np.ones((self.N, 1))

        self.X = np.concatenate((ones, X), axis = 1)
        self.Y = Y

        self.w = np.random.randn(self.d + 1, 1)

    def predict_prob(self, X):
        one = np.ones((len(X),1))

        X = np.concatenate((one, X),axis = 1)
        a = X.dot(self.w)

        return sigmoid(a)

    def train(self, nepochs, batch_size, lr, lamda, print_every):
        loss = []
        for i in range(nepochs):
            mix_id = np.random.permutation(self.N)
            for j in range(0, self.N, batch_size):
                id_batch = mix_id[j : j + batch_size]
                X_batch = self.X[id_batch]                    # b * d
                Y_batch = self.Y[id_batch].reshape(-1, 1)     # b * 1

                Z_batch = sigmoid(X_batch.dot(self.w)).reshape(-1, 1)   # b * 1

                # dloss = (z - y) * x

                dloss = (np.multiply(Z_batch - Y_batch, X_batch).sum(axis = 0)/len(X_batch)).reshape(-1, 1)

                #dloss = np.ones_like(self.w)
                #for x, y, z in zip(X_batch, Y_batch, Z_batch):
                #  dloss += ((z - y) * x).reshape(-1, 1)

                self.w = self.w - lr * dloss - lamda * self.w    #cập nhật w sau mỗi lần tính dloss

                if j % print_every ==0:
                    loss.append(CELoss(Z_batch, Y_batch, mode = 'mean'))
                    print("{}/{}  [{}/{}]  loss = {}".format(i, nepochs, j, self.N, loss[-1]))

        return loss
```

  - Dự đoán một Headlines có phải là tin châm biếm hay không.
- Đánh giá mô hình:
  - Tính accurary, precision, recall, F1-score

CHƯƠNG 2: XÂY DỰNG MÔ HÌNH TỪ TẬP HUẤN LUYỆN

2.1 Mã giả thuật toán Logistic regession .

- Cho một bảng dữ liệu thể hiện thời gian bỏ ra học của học sinh => được thể hiện qua việc thi đậu (1) hay trượt (0)
- Q: Thời gian ôn thi ảnh hưởng như thế nào đến kết quả vượt qua kì thi ?

Hours	0.5	0.75	1.00	1.25	1.50	1.75	1.75	2.00	2.25	2.50	2.75	3.00	3.25	3.50	4.00	4.25	4.50	4.75	5.00	5.50
Pass	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1

2.2 Minh họa thuật toán Logistic regession .

	A	B	C	D	E	F	G	H	I	J
1	i	xi	yi	log odds	e^L	Ŷ=P(x)	E(w)	W_0 new	W_1 new	
2	1	0.5	0	0	1	0.5	0.693147	-0.02	-0.01	
3	2	0.75	0	-0.0275	0.972875	0.493125	0.679492	-0.03973	-0.02479	
4	3	1	0	-0.06452	0.937519	0.483876	0.661408	-0.05908	-0.04415	
5	4	1.25	0	-0.11427	0.892021	0.471465	0.637645	-0.07794	-0.06772	
6	5	1.5	0	-0.17952	0.83567	0.45524	0.607409	-0.09615	-0.09504	
7	6	1.75	0	-0.26246	0.769156	0.434759	0.570502	-0.11354	-0.12547	
8	7	1.75	1	-0.33311	0.716691	0.417484	0.873509	-0.09024	-0.08469	
9	8	2	0	-0.25962	0.771341	0.435456	0.571737	-0.10766	-0.11953	
10	9	2.25	1	-0.3766	0.686192	0.406948	0.899071	-0.08393	-0.06616	
11	10	2.5	0	-0.24932	0.779329	0.43799	0.576236	-0.10145	-0.10995	
12	11	2.75	1	-0.40383	0.667759	0.400393	0.915309	-0.07747	-0.044	
13	12	3	0	-0.20946	0.811021	0.447825	0.593891	-0.09538	-0.09774	
14	13	3.25	1	-0.41303	0.661645	0.398187	0.920834	-0.07131	-0.0195	
15	14	3.5	0	-0.13956	0.869739	0.465166	0.625799	-0.08992	-0.08462	
16	15	4	1	-0.42841	0.651543	0.394506	0.930122	-0.0657	0.012255	
17	16	4.25	1	-0.01361	0.98648	0.496597	0.699977	-0.04556	0.097834	
18	17	4.5	1	0.394691	1.483925	0.597411	0.515149	-0.02946	0.1703	
19	18	4.75	1	0.779466	2.180308	0.685565	0.377512	-0.01688	0.230042	
20	19	5	1	1.133332	3.105987	0.756453	0.279115	-0.00714	0.278752	
21	20	5.5	1	1.525996	4.599723	0.82142	0.196721	5.4E-06	0.318039	
22										solver

Batch Gradient Descent | **Stochastic Gradient Descent**

CHƯƠNG 3: THỰC NGHIỆM ĐÁNH GIÁ

3.1 Xử lí dữ liệu từ Dataset .

1. Xử lý dữ liệu :

- Trước hết, ta cần gọi các thư viện hỗ trợ cho việc tính toán cũng như xử lý dữ liệu

```
[3]  import numpy as np
     import re
     import nltk
     from nltk.stem import WordNetLemmatizer
     import pandas as pd
     import matplotlib.pyplot as plt
```

- Tuy nhiên, ta cần download 1 số thư viện hỗ trợ cho việc xử lý dữ liệu trong bài này là 'wordnet' và 'stopword' từ thư viện nltk

```
[4]  nltk.download('wordnet')
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

- Mở file dataset bằng lệnh pd.read_json trong Python(vì dataset định dạng là json và lệnh đọc file json trong thư viện pandas), đồng thời bỏ đi phần article link vì không quan trọng trong việc huấn luyện mô hình.
- Khởi tạo hàm lemmatizer từ nltk.stem trong thư viện nltk

```
datastore = pd.read_json("Sarcasm_Headlines_Dataset.json", lines = True).drop('article_link',axis = 1)
print(datastore)
```

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

- Kết quả khi đọc file dataset:

	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snape happy birthday in th...	0
5	advancing the world's women	0
6	the fascinating case for eating lab-grown meat	0
7	this ceo will send your kids to school, if you...	0
8	top snake handler leaves sinking huckabee camp...	1
9	friday's morning email: inside trump's presser...	0
10	airline passengers tackle man who rushes cockp...	0
11	facebook reportedly working on healthcare feat...	0
12	north korea praises trump and urges us voters ...	0
13	actually, cnn's jeffrey lord has been 'indefen...	0
14	barcelona holds huge protest in support of ref...	0
15	nuclear bomb detonates during rehearsal for 's...	1
16	cosby lawyer asks why accusers didn't come for...	1
17	stock analysts confused, frightened by boar ma...	1
18	bloomberg's program to build better cities jus...	0
19	craig hicks indicted	0
20	courtroom sketch artist has clear manga influe...	1

Có 26709 dòng.

- Thực hiện tokenize các câu trong headline (loại bỏ các kí tự đặc biệt trong câu , kể cả dấu khoảng trắng).

```
[ ]  def tokenize(arr_s):
         "Return all words in sentence, removed all special characters"

         arr = []
         for s in arr_s:
             s = "something " + s + " something"
             s = re.sub(r"[^a-zA-Z]+", ' ', s)
             s = re.split(r"\s+", s)
             s = s[1:-1]
             for j in range(len(s)):
                 s[j] = lemmatizer.lemmatize(s[j])
             arr.append(s)
         arr = np.array(arr)

         return arr
```

- Tạo bộ từ điển

```
[ ]  def GetVocabulary(str_arr):
         vocabulary = []
         sentences = tokenize(str_arr)

         for tokens in sentences:
             for token in tokens:
                 vocabulary.append(token)

         vocabulary = set(vocabulary)
         vocabulary = np.array(list(vocabulary))
         return vocabulary
     ####################################################
     vocabulary = GetVocabulary(datastore['headline'])
     print(vocabulary.shape)
```

- Sau khi tạo bộ từ điển từ headline, ta thu được 21868 dòng.

```
vocabulary = GetVocabulary(datastore['headline'])
print(vocabulary.shape)
```

```
(21868,)
```

- Đánh số cho các từ trong bộ từ điển với số lượng từ vựng đánh từ 0

```
[13]  word2idx = {w : idx for idx, w in enumerate(vocabulary)}
      print(word2idx)
```

```
{'deets': 0, 'lehrer': 1, 'coveted': 2, 'bluth': 3, 'bear': 4,
```

- Chuyển int thành one-hot, là 1 vector có toàn bộ giá trị là 0 trừ tại một vị trí đặc biệt nào đó thì giá trị sẽ là 1, ví dụ chúng ta có 4 ký tự: ABCD, chúng ta sẽ có các one-hot vector tương ứng với từng ký tự như sau:
  A: [1,0,0,0]
  B: [0,1,0,0]
  C: [0,0,1,0]
  D: [0,0,0,1]
  Số chiều của one-hot vector sẽ phụ thuộc vào số lượng vào số lượng phần tử có trong tập hợp mà chúng ta cần biểu diễn. Trong ví dụ trên vì tập hợp chúng ta chỉ có 4 phần tử (‘A’,’B’,’C’,’D’) nên vector của chúng ta là 4 chiều

Quay trở lại bài case study, mục tiêu của chúng ta là biến các câu headline đã được chuyển về dạng integer thành one-hot với hàm Int2Onehot:

```
[14] def Int2Onehot(idx):
         """
         Vi du:
             idx = [1, 2, 0]   len(vocabulary) = 4

             --> [[0, 1, 0, 0],
                  [0, 0, 1, 0],
                  [1, 0, 0, 0]]
         """
         dim = len(vocabulary)
         onehot = np.zeros((len(idx), dim))

         onehot[range(len(idx)), idx] = 1

         return onehot
     ###
     Int2Onehot([1,2, 3])
```

```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

- Tạo túi từ với hàm words2bag để khi trả về kết quả là các vector one-hot như trên, sau đó khởi tạo hàm docs2bag với việc sử dụng hàm tokenize và words2bag nhằm đưa các câu vào bộ từ điển

```
[15]  def words2bag(words):
          """
          Vi du : ["tuan", "huy"]   vocabulary = ["huy", "tuan", "cogiao"]
          --> idx = [1, 0]
          --> Int2Onehot(idx)
          """
          idx = np.zeros((len(words)),dtype = np.int)

          for i, word in enumerate(words):
              idx[i] = word2idx[word]
          return Int2Onehot(idx)

      def docs2bag(docs):
          """
          vi du:
          ["tuan huy","tuan cogiao tuan"] vocabulary = ["huy", "tuan", "cogiao"]
          --> [[1,1,0],
               [0,2,1]]
          """
          sentences = tokenize(docs)

          bags = []

          for sentence in sentences:
              onehots = words2bag(sentence)
              bags.append(onehots.sum(axis = 0))
          return np.array(bags)
```

- Thực hiện tạo bộ dữ liệu và nhãn

```
[16]  data = docs2bag(datastore['headline'])
      label = np.array(datastore['is_sarcastic'])
      print(data.shape, label.shape)
```

```
(26709, 21868) (26709,)
```

- Khởi tạo hàm sigmoid và cross entropy loss (hàm mất mát):

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

def CELoss(output, target, mode = 'sum'):
    a = - (target * np.log(output) + (1 - target) * np.log(1 - output))
    if (mode == 'sum'):
        return a.sum()
    if (mode == 'mean'):
        return a.sum() / len(target)
```

- Tạo 1 lớp Logistic lưu các thông tin về fit mô hình, dự đoán mô hình, và huấn luyện nhằm phục vụ cho việc huấn luyện và đánh giá mô hình Logistic Regression

```
class logistic():
    def fit(self, X, Y):
        self.N = X.shape[0]
        self.d = X.shape[1]

        ones = np.ones((self.N, 1))

        self.X = np.concatenate((ones, X), axis = 1)
        self.Y = Y

        self.w = np.random.randn(self.d + 1, 1)

    def predict_prob(self, X):
        one = np.ones((len(X),1))

        X = np.concatenate((one, X),axis = 1)
        a = X.dot(self.w)

        return sigmoid(a)
```

```
def train(self, nepochs, batch_size, lr, lamda, print_every):
    loss = []
    for i in range(nepochs):
        mix_id = np.random.permutation(self.N)
        for j in range(0, self.N, batch_size):
            id_batch = mix_id[j : j + batch_size]
            X_batch = self.X[id_batch]                  # b * d
            Y_batch = self.Y[id_batch].reshape(-1, 1)     # b * 1

            Z_batch = sigmoid(X_batch.dot(self.w)).reshape(-1, 1)   # b * 1

            # dloss = (z - y) * x

            dloss = (np.multiply(Z_batch - Y_batch, X_batch).sum(axis = 0)/len(X_batch)).reshape(-1, 1)

            #dloss = np.ones_like(self.w)
            #for x, y, z in zip(X_batch, Y_batch, Z_batch):
            #  dloss += ((z - y) * x).reshape(-1, 1)

            self.w = self.w - lr * dloss - lamda * self.w

            if j % print_every ==0:
                loss.append(CELoss(Z_batch, Y_batch, mode = 'mean'))
                print("{}/{}  [{}/{}]  loss = {}".format(i, nepochs, j, self.N, loss[-1]))

    return loss
```

Lưu ý: hàm train ở trong class Logistic, train ở đây chúng ta sẽ train theo batch để hạn chế tốn chi phí tài nguyên bộ nhớ Ram, vì nếu thực hiện theo dòng sẽ bị tràn Ram và không thể cho ra kết quả được

- Chia dữ liệu để train: lấy 20000 dòng đầu để huấn luyện mô hình và 6709 dòng còn lại để test

```
[13]  model = logistic()
      model.fit(data[:20000], label[:20000])
```

- Tính độ mất mát theo 100 epoch với 100 batch với learning rate = 0.05(5e-2), lambda = 0.00001 (1e-5) và trong mỗi lần thực hiện tính độ lỗi thì in ra 1000 dòng trong bộ training. Huấn luyện với bộ dữ liệu này nhiều lần.

```
loss = model.train(100, 100, 5e-2, 1e-5, 1000)
```

- Kết quả sau khi thực hiện lần đầu:

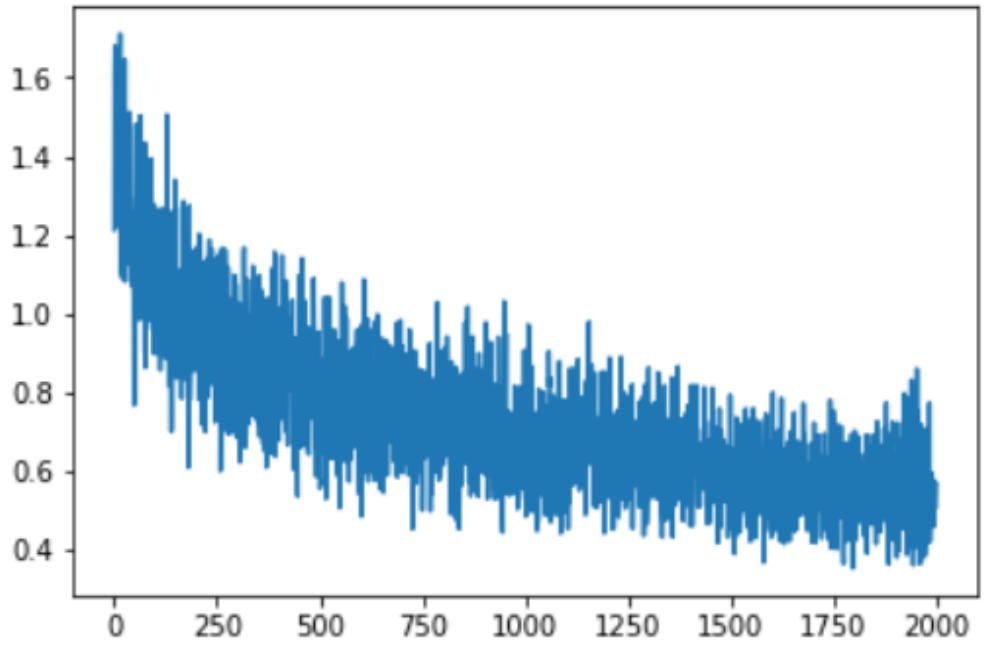
```
0/100  [0/20000]  loss = 1.2133449484906214
0/100  [1000/20000]  loss = 1.6061506534914367
0/100  [2000/20000]  loss = 1.6799491455561242
0/100  [3000/20000]  loss = 1.6111471139386717
0/100  [4000/20000]  loss = 1.6228136293085758
0/100  [5000/20000]  loss = 1.2979915967165818
0/100  [6000/20000]  loss = 1.4471725381722818
0/100  [7000/20000]  loss = 1.5053746773942749
0/100  [8000/20000]  loss = 1.405055942100391
0/100  [9000/20000]  loss = 1.4292202491569004
0/100  [10000/20000]  loss = 1.2213216898084966
0/100  [11000/20000]  loss = 1.218697812639915
0/100  [12000/20000]  loss = 1.3618292451092526
0/100  [13000/20000]  loss = 1.6534965066346814
0/100  [14000/20000]  loss = 1.711153671959828
0/100  [15000/20000]  loss = 1.5494823292982707
0/100  [16000/20000]  loss = 1.2414171803131333
0/100  [17000/20000]  loss = 1.11407839219301
0/100  [18000/20000]  loss = 1.102192478926923
0/100  [19000/20000]  loss = 1.2148116335332437
1/100  [0/20000]  loss = 1.089202921648526
1/100  [1000/20000]  loss = 1.555528391992649
1/100  [2000/20000]  loss = 1.402023335296174
1/100  [3000/20000]  loss = 1.188551442184724
```

```
98/100  [16000/20000]  loss = 0.4256791107630508
98/100  [17000/20000]  loss = 0.6799525027798999
98/100  [18000/20000]  loss = 0.5167557960010005
98/100  [19000/20000]  loss = 0.5789825003420668
99/100  [0/20000]  loss = 0.4663029335175025
99/100  [1000/20000]  loss = 0.4340310453586309
99/100  [2000/20000]  loss = 0.4349254005098171
99/100  [3000/20000]  loss = 0.7737979535507759
99/100  [4000/20000]  loss = 0.4199154111267365
99/100  [5000/20000]  loss = 0.4601959291630962
99/100  [6000/20000]  loss = 0.4639196162684505
99/100  [7000/20000]  loss = 0.5433166067444519
99/100  [8000/20000]  loss = 0.5959357733506454
99/100  [9000/20000]  loss = 0.5713495281784585
99/100  [10000/20000]  loss = 0.5700961962503367
99/100  [11000/20000]  loss = 0.4597356779342459
99/100  [12000/20000]  loss = 0.48692755944092064
99/100  [13000/20000]  loss = 0.5772678338624861
99/100  [14000/20000]  loss = 0.46207678624818027
99/100  [15000/20000]  loss = 0.522107710693939
99/100  [16000/20000]  loss = 0.5729718115799953
99/100  [17000/20000]  loss = 0.5043442074320257
99/100  [18000/20000]  loss = 0.5395030109137011
99/100  [19000/20000]  loss = 0.5685204831001209
```

- Biểu diễn kết quả trên đồ thị, ta thấy:

```
[15] plt.plot(loss)
```

```
[<matplotlib.lines.Line2D at 0x7f15aa848f98>]
```



- Thực hiện đánh giá độ chính xác của mô hình trên 20000 dòng để training và 6709 để testing ở lần đầu, ta thu được kết quả:

```
[16] ((model.predict_prob(data[:20000]) > 0.5).reshape(-1) == label[:20000]).sum() / len(data[:20000])
```

```
0.75695
```

```
[17] ((model.predict_prob(data[20000:]) > 0.5).reshape(-1) == label[20000:]).sum() / len(data[20000:])
```

```
0.7446713370099866
```

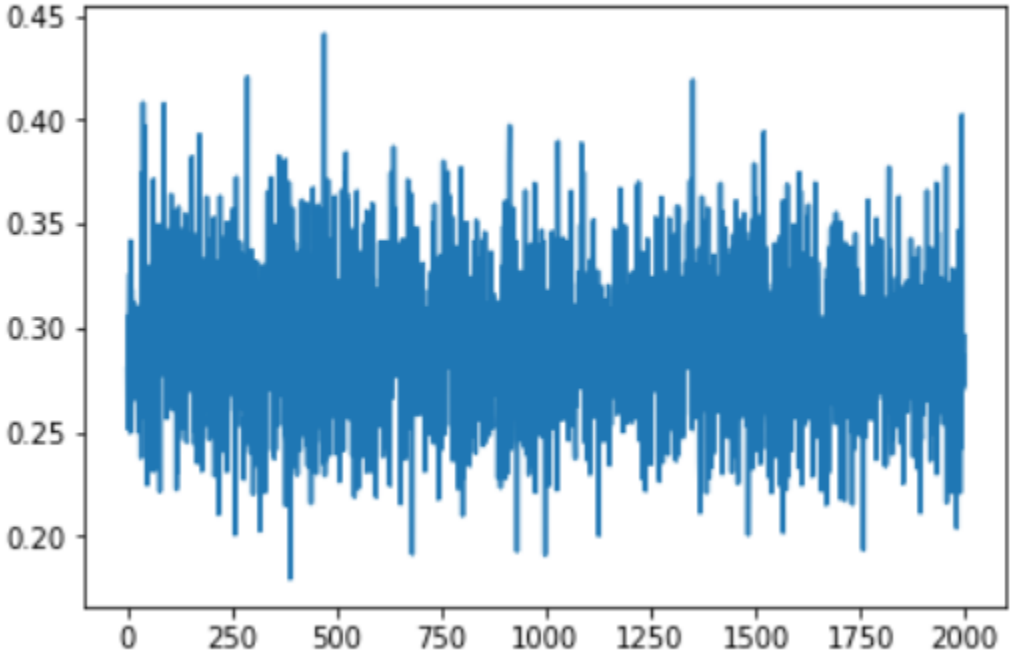
- Sau vài lần huấn luyện, ta được kết quả:

```
0/100  [0/20000]  loss = 0.280518070898688 35
0/100  [1000/20000]  loss = 0.2515924538406629
0/100  [2000/20000]  loss = 0.27361645378430854
0/100  [3000/20000]  loss = 0.3059058557350321
0/100  [4000/20000]  loss = 0.2808547704422307
0/100  [5000/20000]  loss = 0.3022923807503264
0/100  [6000/20000]  loss = 0.3260559872894011
0/100  [7000/20000]  loss = 0.24962189234443485
0/100  [8000/20000]  loss = 0.3419438987567741
0/100  [9000/20000]  loss = 0.2676494098031099
0/100  [10000/20000]  loss = 0.27884944185140553
0/100  [11000/20000]  loss = 0.31849304427270897
0/100  [12000/20000]  loss = 0.3007216135968735
0/100  [13000/20000]  loss = 0.3130652344942685
0/100  [14000/20000]  loss = 0.2652718503026782
0/100  [15000/20000]  loss = 0.2977348223044448
0/100  [16000/20000]  loss = 0.26971932080048755
0/100  [17000/20000]  loss = 0.3065575472724562
0/100  [18000/20000]  loss = 0.2782544372372719
0/100  [19000/20000]  loss = 0.27818368040536506
1/100  [0/20000]  loss = 0.30857221873900 42
1/100  [1000/20000]  loss = 0.308971475896174 74
1/100  [2000/20000]  loss = 0.28278510186240 01
```

```
98/100  [17000/20000]  loss = 0.280522166145441 23
98/100  [18000/20000]  loss = 0.237645521148169 23
98/100  [19000/20000]  loss = 0.27791668968257 54
99/100  [0/20000]  loss = 0.25613068537642 79
99/100  [1000/20000]  loss = 0.203853416092175 08
99/100  [2000/20000]  loss = 0.28847629835105 55
99/100  [3000/20000]  loss = 0.30737165058321 62
99/100  [4000/20000]  loss = 0.27125612114241 04
99/100  [5000/20000]  loss = 0.34685393776950 92
99/100  [6000/20000]  loss = 0.34566278229964 35
99/100  [7000/20000]  loss = 0.32751017240550 373
99/100  [8000/20000]  loss = 0.26391356047359 45
99/100  [9000/20000]  loss = 0.22084593471375 719
99/100  [10000/20000]  loss = 0.26886980886706 64
99/100  [11000/20000]  loss = 0.24207744070335 147
99/100  [12000/20000]  loss = 0.25170859674298 623
99/100  [13000/20000]  loss = 0.27107155676073 09
99/100  [14000/20000]  loss = 0.40253339479009 15
99/100  [15000/20000]  loss = 0.282007265085520 43
99/100  [16000/20000]  loss = 0.270364291636257 25
99/100  [17000/20000]  loss = 0.28683426483377 04
99/100  [18000/20000]  loss = 0.272683168979177 44
99/100  [19000/20000]  loss = 0.296329386507873 94
```

- Vẽ biểu đồ ra, ta thấy:

```
[<matplotlib.lines.Line2D at 0x7f15ac472710>]
```



Ta thấy kết quả có lúc tăng lúc giảm nhưng nhìn chung đồ thị vẫn có xu hướng giảm dần

- Thực hiện đánh giá độ chính xác của mô hình trên 20000 dòng để training và 6709 để testing sau khi huấn luyện mô hình nhiều lần, ta thu được kết quả:

```
[46] ((model.predict_prob(data[:20000]) > 0.5).reshape(-1) == label[:20000]).sum() / len(data[:20000])
```

```
0.90375
```

```
((model.predict_prob(data[20000:]) > 0.5).reshape(-1) == label[20000:]).sum() / len(data[20000:])
```

```
0.8314204799523028
```

Ta thấy tỉ lệ chính xác của mô hình khi huấn luyện là 90.375% và trên tập testing là khoảng 83%, nghĩa là mô hình này khá ổn khi nhóm mong muốn mô hình đạt khoảng 85% khi testing.

- Tính accuracy, precision call, f1-score:

3.2. Thuật giải Logistic regression.

1. Xử lý dữ liệu:

- Trước hết, ta cần gọi các thư viện hỗ trợ cho việc tính toán cũng như xử lý dữ liệu

```
[3]  import numpy as np
     import re
     import nltk
     from nltk.stem import WordNetLemmatizer
     import pandas as pd
     import matplotlib.pyplot as plt
```

- Tuy nhiên, ta cần download 1 số thư viện hỗ trợ cho việc xử lý dữ liệu trong bài này là 'wordnet' và 'stopword' từ thư viện nltk

```
[4]  nltk.download('wordnet')
     nltk.download('stopwords')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

- Mở file dataset bằng lệnh pd.read_json trong Python(vì dataset định dạng là json và lệnh đọc file json trong thư viện pandas), đồng thời bỏ đi phần article link vì không quan trọng trong việc huấn luyện mô hình.
- Khởi tạo hàm lemmatizer từ nltk.stem trong thư viện nltk

```
datastore = pd.read_json("Sarcasm_Headlines_Dataset.json", lines = True).drop('article_link',axis = 1)
print(datastore)
```

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
```

- Kết quả khi đọc file dataset:

	headline	is_sarcastic
0	former versace store clerk sues over secret 'b...	0
1	the 'roseanne' revival catches up to our thorn...	0
2	mom starting to fear son's web series closest ...	1
3	boehner just wants wife to listen, not come up...	1
4	j.k. rowling wishes snape happy birthday in th...	0
5	advancing the world's women	0
6	the fascinating case for eating lab-grown meat	0
7	this ceo will send your kids to school, if you...	0
8	top snake handler leaves sinking huckabee camp...	1
9	friday's morning email: inside trump's presser...	0
10	airline passengers tackle man who rushes cockp...	0
11	facebook reportedly working on healthcare feat...	0
12	north korea praises trump and urges us voters ...	0
13	actually, cnn's jeffrey lord has been 'indefen...	0
14	barcelona holds huge protest in support of ref...	0
15	nuclear bomb detonates during rehearsal for 's...	1
16	cosby lawyer asks why accusers didn't come for...	1
17	stock analysts confused, frightened by boar ma...	1
18	bloomberg's program to build better cities jus...	0
19	craig hicks indicted	0
20	courtroom sketch artist has clear manga influe...	1

Có 26709 dòng.

- Thực hiện tokenize các câu trong headline (loại bỏ các kí tự đặc biệt trong câu , kể cả dấu khoảng trắng).

```
[ ]  def tokenize(arr_s):
         "Return all words in sentence, removed all special characters"

         arr = []
         for s in arr_s:
             s = "something " + s + " something"
             s = re.sub(r"[^a-zA-Z]+", ' ', s)
             s = re.split(r"\s+", s)
             s = s[1:-1]
             for j in range(len(s)):
                 s[j] = lemmatizer.lemmatize(s[j])
             arr.append(s)
         arr = np.array(arr)

         return arr
```

- Tạo bộ từ điển

```
[ ]  def GetVocabulary(str_arr):
         vocabulary = []
         sentences = tokenize(str_arr)

         for tokens in sentences:
             for token in tokens:
                 vocabulary.append(token)

         vocabulary = set(vocabulary)
         vocabulary = np.array(list(vocabulary))
         return vocabulary
     ####################################################
     vocabulary = GetVocabulary(datastore['headline'])
     print(vocabulary.shape)
```

- Sau khi tạo bộ từ điển từ headline, ta thu được 21868 dòng.

```
vocabulary = GetVocabulary(datastore['headline'])
print(vocabulary.shape)
```

```
(21868,)
```

- Đánh số cho các từ trong bộ từ điển với số lượng từ vựng đánh từ 0

```
[13]  word2idx = {w : idx for idx, w in enumerate(vocabulary)}
      print(word2idx)
```

```
{'deets': 0, 'lehrer': 1, 'coveted': 2, 'bluth': 3, 'bear': 4,
```

- Chuyển int thành one-hot, là 1 vector có toàn bộ giá trị là 0 trừ tại một vị trí đặc biệt nào đó thì giá trị sẽ là 1, ví dụ chúng ta có 4 ký tự: ABCD, chúng ta sẽ có các one-hot vector tương ứng với từng ký tự như sau:
  A: [1,0,0,0]
  B: [0,1,0,0]
  C: [0,0,1,0]
  D: [0,0,0,1]
  Số chiều của one-hot vector sẽ phụ thuộc vào số lượng vào số lượng phần tử có trong tập hợp mà chúng ta cần biểu diễn. Trong ví dụ trên vì tập hợp chúng ta chỉ có 4 phần tử (‘A’,’B’,’C’,’D’) nên vector của chúng ta là 4 chiều

Quay trở lại bài case study, mục tiêu của chúng ta là biến các câu headline đã được chuyển về dạng integer thành one-hot với hàm Int2Onehot:

```
[14] def Int2Onehot(idx):
         """
         Vi du:
             idx = [1, 2, 0]   len(vocabulary) = 4

             --> [[0, 1, 0, 0],
                  [0, 0, 1, 0],
                  [1, 0, 0, 0]]
         """
         dim = len(vocabulary)
         onehot = np.zeros((len(idx), dim))

         onehot[range(len(idx)), idx] = 1

         return onehot
     ###
     Int2Onehot([1,2, 3])
```

```
array([[0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

- Tạo túi từ với hàm words2bag để khi trả về kết quả là các vector one-hot như trên, sau đó khởi tạo hàm docs2bag với việc sử dụng hàm tokenize và words2bag nhằm đưa các câu vào bộ từ điển

```
[15]  def words2bag(words):
          """
          Vi du : ["tuan", "huy"]   vocabulary = ["huy", "tuan", "cogiao"]
          --> idx = [1, 0]
          --> Int2Onehot(idx)
          """
          idx = np.zeros((len(words)),dtype = np.int)

          for i, word in enumerate(words):
              idx[i] = word2idx[word]
          return Int2Onehot(idx)

      def docs2bag(docs):
          """
          vi du:
          ["tuan huy","tuan cogiao tuan"] vocabulary = ["huy", "tuan", "cogiao"]
          --> [[1,1,0],
               [0,2,1]]
          """
          sentences = tokenize(docs)

          bags = []

          for sentence in sentences:
              onehots = words2bag(sentence)
              bags.append(onehots.sum(axis = 0))
          return np.array(bags)
```

- Thực hiện tạo bộ dữ liệu và nhãn

```
[16]  data = docs2bag(datastore['headline'])
      label = np.array(datastore['is_sarcastic'])
      print(data.shape, label.shape)
```

```
(26709, 21868) (26709,)
```

2. Level 1: Code tay thuật toán

- Khởi tạo hàm sigmoid và cross entropy loss (hàm mất mát):

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

def CELoss(output, target, mode = 'sum'):
    a = - (target * np.log(output) + (1 - target) * np.log(1 - output))
    if (mode == 'sum'):
        return a.sum()
    if (mode == 'mean'):
        return a.sum() / len(target)
```

- Tạo 1 lớp Logistic lưu các thông tin về fit mô hình, dự đoán mô hình, và huấn luyện nhằm phục vụ cho việc huấn luyện và đánh giá mô hình Logistic Regression

```
class logistic():
    def fit(self, X, Y):
        self.N = X.shape[0]
        self.d = X.shape[1]

        ones = np.ones((self.N, 1))

        self.X = np.concatenate((ones, X), axis = 1)
        self.Y = Y

        self.w = np.random.randn(self.d + 1, 1)

    def predict_prob(self, X):
        one = np.ones((len(X),1))

        X = np.concatenate((one, X),axis = 1)
        a = X.dot(self.w)

        return sigmoid(a)
```

```
def train(self, nepochs, batch_size, lr, lamda, print_every):
    loss = []
    for i in range(nepochs):
        mix_id = np.random.permutation(self.N)
        for j in range(0, self.N, batch_size):
            id_batch = mix_id[j : j + batch_size]
            X_batch = self.X[id_batch]                    # b * d
            Y_batch = self.Y[id_batch].reshape(-1, 1)     # b * 1

            Z_batch = sigmoid(X_batch.dot(self.w)).reshape(-1, 1)   # b * 1

            # dloss = (z - y) * x

            dloss = (np.multiply(Z_batch - Y_batch, X_batch).sum(axis = 0)/len(X_batch)).reshape(-1, 1)

            #dloss = np.ones_like(self.w)
            #for x, y, z in zip(X_batch, Y_batch, Z_batch):
            #  dloss += ((z - y) * x).reshape(-1, 1)

            self.w = self.w - lr * dloss - lamda * self.w

            if j % print_every ==0:
                loss.append(CELoss(Z_batch, Y_batch, mode = 'mean'))
                print("{}/{}  [{}/{}]  loss = {}".format(i, nepochs, j, self.N, loss[-1]))

    return loss
```

Lưu ý: hàm train ở trong class Logistic, train ở đây chúng ta sẽ train theo batch để hạn chế tốn chi phí tài nguyên bộ nhớ Ram, vì nếu thực hiện theo dòng sẽ bị tràn Ram và không thể cho ra kết quả được

- Chia dữ liệu để train: lấy 20000 dòng đầu để huấn luyện mô hình và 6709 dòng còn lại để test

```
[13]  model = logistic()
      model.fit(data[:20000], label[:20000])
```

- Tính độ mất mát theo 100 epoch với 100 batch với learning rate = 0.05(5e-2), lambda = 0.00001 (1e-5) và trong mỗi lần thực hiện tính độ lỗi thì in ra 1000 dòng trong bộ training. Huấn luyện với bộ dữ liệu này nhiều lần.

```
      loss = model.train(100, 100, 5e-2, 1e-5, 1000)
```

- Kết quả sau khi thực hiện lần đầu:

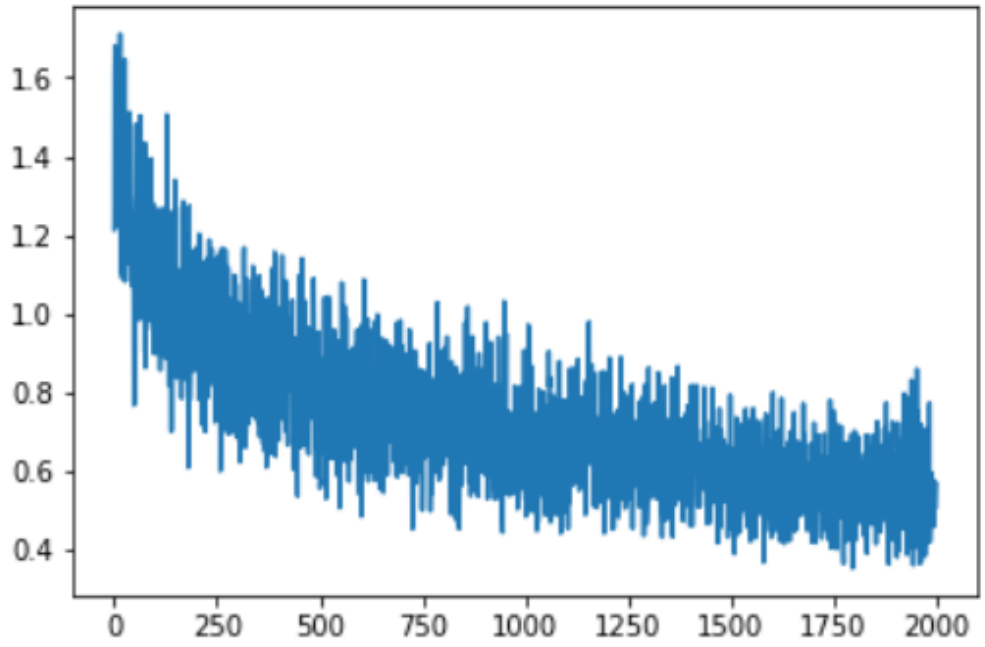
```
0/100  [0/20000]  loss = 1.2133449484906214
0/100  [1000/20000]  loss = 1.6061506534914367
0/100  [2000/20000]  loss = 1.6799491455561242
0/100  [3000/20000]  loss = 1.6111471139386717
0/100  [4000/20000]  loss = 1.6228136293085758
0/100  [5000/20000]  loss = 1.2979915967165818
0/100  [6000/20000]  loss = 1.4471725381722818
0/100  [7000/20000]  loss = 1.5053746773942749
0/100  [8000/20000]  loss = 1.405055942100391
0/100  [9000/20000]  loss = 1.4292202491569004
0/100  [10000/20000]  loss = 1.2213216898084966
0/100  [11000/20000]  loss = 1.218697812639915
0/100  [12000/20000]  loss = 1.3618292451092526
0/100  [13000/20000]  loss = 1.6534965066346814
0/100  [14000/20000]  loss = 1.711153671959828
0/100  [15000/20000]  loss = 1.5494823292982707
0/100  [16000/20000]  loss = 1.2414171803131333
0/100  [17000/20000]  loss = 1.114078392193O1
0/100  [18000/20000]  loss = 1.102192478926923
0/100  [19000/20000]  loss = 1.2148116335332437
1/100  [0/20000]  loss = 1.089202921648526
1/100  [1000/20000]  loss = 1.555528391992649
1/100  [2000/20000]  loss = 1.402023335296174
1/100  [3000/20000]  loss = 1.188551442184724
```

```
98/100  [16000/20000]  loss = 0.4256791107630508
98/100  [17000/20000]  loss = 0.6799525027798999
98/100  [18000/20000]  loss = 0.5167557960010005
98/100  [19000/20000]  loss = 0.5789825003420668
99/100  [0/20000]  loss = 0.4663029335175025
99/100  [1000/20000]  loss = 0.4340310453586309
99/100  [2000/20000]  loss = 0.4349254005098171
99/100  [3000/20000]  loss = 0.7737979535507759
99/100  [4000/20000]  loss = 0.4199154111267365
99/100  [5000/20000]  loss = 0.4601959291630962
99/100  [6000/20000]  loss = 0.4639196162684505
99/100  [7000/20000]  loss = 0.5433166067444519
99/100  [8000/20000]  loss = 0.5959357733506454
99/100  [9000/20000]  loss = 0.5713495281784585
99/100  [10000/20000]  loss = 0.5700961962503367
99/100  [11000/20000]  loss = 0.4597356779342459
99/100  [12000/20000]  loss = 0.48692755944092064
99/100  [13000/20000]  loss = 0.5772678338624861
99/100  [14000/20000]  loss = 0.46207678624818027
99/100  [15000/20000]  loss = 0.522107710693939
99/100  [16000/20000]  loss = 0.5729718115799953
99/100  [17000/20000]  loss = 0.5043442074320257
99/100  [18000/20000]  loss = 0.5395030109137011
99/100  [19000/20000]  loss = 0.5685204831001209
```

- Biểu diễn kết quả trên đồ thị, ta thấy:

```
[15] plt.plot(loss)
```

```
[<matplotlib.lines.Line2D at 0x7f15aa848f98>]
```



- Thực hiện đánh giá độ chính xác của mô hình trên 20000 dòng để training và 6709 để testing ở lần đầu, ta thu được kết quả:

```
[16] ((model.predict_prob(data[:20000]) > 0.5).reshape(-1) == label[:20000]).sum() / len(data[:20000])
```

```
0.75695
```

```
[17] ((model.predict_prob(data[20000:]) > 0.5).reshape(-1) == label[20000:]).sum() / len(data[20000:])
```

```
0.7446713370099866
```

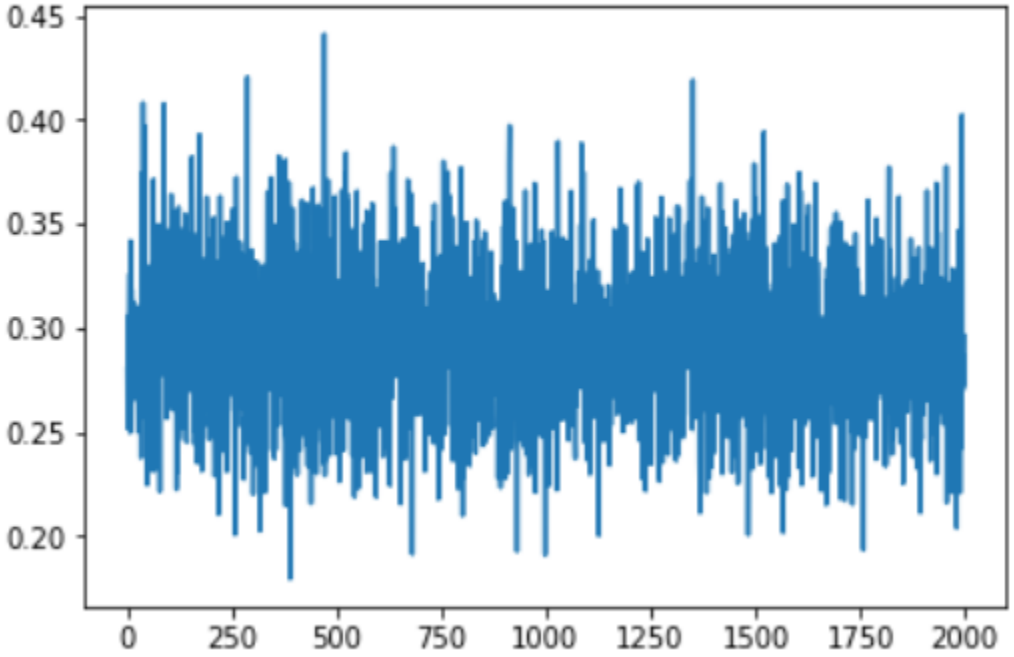
- Sau vài lần huấn luyện, ta được kết quả:

```
0/100  [0/20000]  loss = 0.280518070898688 35
0/100  [1000/20000]  loss = 0.2515924538406629
0/100  [2000/20000]  loss = 0.27361645378430854
0/100  [3000/20000]  loss = 0.3059058557350321
0/100  [4000/20000]  loss = 0.2808547704422307
0/100  [5000/20000]  loss = 0.3022923807503264
0/100  [6000/20000]  loss = 0.3260559872894011
0/100  [7000/20000]  loss = 0.24962189234443485
0/100  [8000/20000]  loss = 0.3419438987567741
0/100  [9000/20000]  loss = 0.2676494098031099
0/100  [10000/20000]  loss = 0.27884944185140553
0/100  [11000/20000]  loss = 0.31849304427270897
0/100  [12000/20000]  loss = 0.3007216135968735
0/100  [13000/20000]  loss = 0.3130652344942685
0/100  [14000/20000]  loss = 0.2652718503026782
0/100  [15000/20000]  loss = 0.2977348223044448
0/100  [16000/20000]  loss = 0.26971932080048755
0/100  [17000/20000]  loss = 0.3065575472724562
0/100  [18000/20000]  loss = 0.2782544372372719
0/100  [19000/20000]  loss = 0.27818368040536506
1/100  [0/20000]  loss = 0.30857221873900 42
1/100  [1000/20000]  loss = 0.308971475896174 74
1/100  [2000/20000]  loss = 0.28278510186240 01
```

```
98/100  [17000/20000]  loss = 0.280522166145441 23
98/100  [18000/20000]  loss = 0.237645521148169 23
98/100  [19000/20000]  loss = 0.27791668968257 54
99/100  [0/20000]  loss = 0.25613068537642 79
99/100  [1000/20000]  loss = 0.203853416092175 08
99/100  [2000/20000]  loss = 0.28847629835105 55
99/100  [3000/20000]  loss = 0.30737165058321 62
99/100  [4000/20000]  loss = 0.27125612114241 04
99/100  [5000/20000]  loss = 0.34685393776950 92
99/100  [6000/20000]  loss = 0.34566278229964 35
99/100  [7000/20000]  loss = 0.32751017240550 373
99/100  [8000/20000]  loss = 0.26391356047359 45
99/100  [9000/20000]  loss = 0.22084593471375 719
99/100  [10000/20000]  loss = 0.26886980886706 64
99/100  [11000/20000]  loss = 0.24207744070335 147
99/100  [12000/20000]  loss = 0.25170859674298 623
99/100  [13000/20000]  loss = 0.27107155676073 09
99/100  [14000/20000]  loss = 0.40253339479009 15
99/100  [15000/20000]  loss = 0.282007265085520 43
99/100  [16000/20000]  loss = 0.270364291636257 25
99/100  [17000/20000]  loss = 0.28683426483377 04
99/100  [18000/20000]  loss = 0.272683168979177 44
99/100  [19000/20000]  loss = 0.296329386507873 94
```

- Vẽ biểu đồ ra, ta thấy:

```
[<matplotlib.lines.Line2D at 0x7f15ac472710>]
```



Ta thấy kết quả có lúc tăng lúc giảm nhưng nhìn chung đồ thị vẫn có xu hướng giảm dần

- Thực hiện đánh giá độ chính xác của mô hình trên 20000 dòng để training và 6709 để testing sau khi huấn luyện mô hình nhiều lần, ta thu được kết quả:

```
[46] ((model.predict_prob(data[:20000]) > 0.5).reshape(-1) == label[:20000]).sum() / len(data[:20000])
```

```
0.90375
```

```
((model.predict_prob(data[20000:]) > 0.5).reshape(-1) == label[20000:]).sum() / len(data[20000:])
```

```
0.8314204799523028
```

Ta thấy tỉ lệ chính xác của mô hình khi huấn luyện là 90.375% và trên tập testing là khoảng 83%, nghĩa là mô hình này khá ổn khi nhóm mong muốn mô hình đạt khoảng 85% khi testing.

3. Level 2: Sử dụng thư viện sklearn

- Gọi thư viện hỗ trợ cho Logistic Regression từ thư viện sklearn

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

- Tính precision, recall, f1-score (đánh giá mô hình):

```
## precision = (số cái đúng) / (tổng số cái dự đoán đúng)
## recall = (số cái đúng) / số cái thực sự đúng

predict_label = logreg.predict(data[N:])
true_label = label[N:]

### Số lượng label thực sự là châm biếm trong tất cả những label dự đoán là châm biếm
true_sarcastics = 0
for i in range(len(predict_label)):
  if (predict_label[i] == 1 and predict_label[i] == true_label[i]):
    true_sarcastics += 1

### Số lượng label là châm biếm trong tổng số tất cả label
relevant_sarcastics = true_label.sum()


### Số lượng label dự đoán là đúng
predicted_sarcastics = predict_label.sum()
# for i in predicted:
#     if (i == labelx):
#         predicted_x += 1

precision = true_sarcastics / predicted_sarcastics
recall = true_sarcastics / relevant_sarcastics
print(precision, recall)
```

```
0.8319598710139735 0.7924914675767918
```

```
def recall(predict_label : list, true_label : list, label : int):
  true_positive = 0
  for i in range(len(predict_label)):
    if (predict_label[i] == label and predict_label[i] == true_label[i]):
      true_positive += 1

  relevant_elements = 0
  for i in range(len(predict_label)):
    if (true_label[i] == label):
      relevant_elements += 1

  return true_positive / relevant_elements

def precision(predict_label : list, true_label : list, label : int):
  true_positive = 0
  for i in range(len(predict_label)):
    if (predict_label[i] == label and predict_label[i] == true_label[i]):
      true_positive += 1

  positive = 0
  for i in range (len (predict_label)):
    if (predict_label[i] == label):
      positive += 1

  return true_positive / positive

def f1_score(predict_label : list, true_label : list, label : int):
  precision_score = precision(predict_label, true_label, label)
  recall_score = recall(predict_label, true_label, label)

  f1 = 2 * (precision_score * recall_score) / (precision_score + recall_score)

  return f1
```

Đặt sarcastic = 1, non_sarcastic = 0, tính precision, recall và f1 score dựa trên 2 biến này

```
[ ] sarcastic = 1
    non_sarcastic = 0

    print("Sarcastic..........precision = {}   recall = {}".format(precision(predict_label, true_label, sarcastic), recall(predict_label, true_label, sarcastic)))
    print("Non_Sarcastic.....precision = {}   recall = {}".format(precision(predict_label, true_label, non_sarcastic), recall(predict_label, true_label, non_sarcas

[ ] print("Sarcastic..........f1 score = {}".format(f1_score(predict_label, true_label, sarcastic)))
    print("NoneSarcastic.....f1 score = {}".format(f1_score(predict_label, true_label, non_sarcastic)))
```

Kết quả thu được:

```
Sarcastic..........precision = 0.8319598710139735   recall = 0.7924914675767918
Non_Sarcastic.....precision = 0.844818785094436   recall = 0.8758930934109552
```

```
Sarcastic..........f1 score = 0.8117461982170949
NoneSarcastic.....f1 score = 0.8600753540340392
```

CHƯƠNG 4: LẬP TRÌNH CÀI ĐẶT

4.1. Tool và thư viện đã sử dụng.

- Tool: Jupyter Notebook và Google Colab.
- Ngôn ngữ lập trình: python

1. Hướng dẫnc sử dụng Google Colab và các thư viện trong Jupyter Notebook :

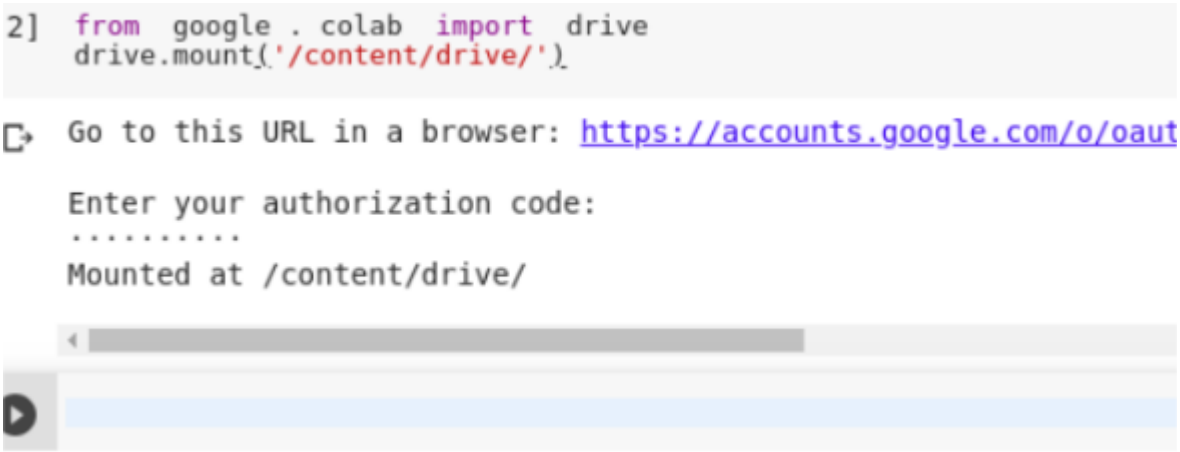
- Làm việc với Google colab:

- Kết nối với drive tiện trong việc lưu file cũng như đọc dữ liệu từ file. Chạy 2 dòng code dưới đây để kết nối Colab với Google drive.

```
from google.colab import drive
drive.mount('/content/drive/')
```

- Sau khi chạy dòng code trên, trên màn hình sẽ xuất hiện như sau:

- Sau đó, click vào đường dẫn và copy mã xác minh vào khung chữ nhật. Sau khi nhập đoạn mã xác minh xong, ta nhận được thông báo trên màn hình như sau:



- Giờ ta có thể truy cập drive một cách đơn giản. Lệnh `!ls "/content/drive/My Drive/"` cho biết các thư mục trong Google Drive.

4.2. Source code báo cáo:

https://colab.research.google.com/drive/16Wcir4rKh4urTnBhtjiZnQBY9egdmxy8

Sarcasm Headlines Dataset:

https://drive.google.com/drive/folders/1J2rE5WcKpbxHL0j3ZBkbprLuUyQMa4dG?fbclid=IwAR3lPVsonvs3XVjMzOxaW7pQyj_5Ec09BmaeV_N-_eR6voDWKPxM9S9n47M

PHỤ LỤC

Tài liệu tham khảo:

1. Hướng dẫn sử dụng Google Colab: https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
2. Logistic Regression: https://en.wikipedia.org/wiki/Logistic_regression
3. Precision_and_recall: https://en.wikipedia.org/wiki/Precision_and_recall

Link video quá trình chạy code:

https://drive.google.com/file/d/1S88XhNNWSbpCcCF8JuJbQGnvXA7vtqjQ/view?fbclid=IwAR3vbX15YPVqs8LXcMrtnhc5iVVexGGPWmQl-KOhnFk7oW2Mv_Zk3I9hHqg

-