**Tiền xử lý dữ liệu**

Trong bài viết này mình sẽ sử dụng 1 mô hình Deeplearning đơn giản để mô hình hóa bài toán.

Như chúng ta đã biết, các mô hình Deeplearning có khả năng end-to-end khá tốt(Từ dữ liệu thô tới mô hình, không cần tiền xử lý trước). Tuy nhiên, Nếu ta kết hợp vừa tiền xử lý vừa sử dụng mô hình end-to-end thì kết quả sẽ tốt hơn nhiều.

Ở đây, mình chỉ sử dụng một số tiền xử lý cơ bản như:

* Chuẩn hóa về chữ thường
* Thay thế các url trong dữ liệu bởi nhãn *link\_spam*
* Tách từ (Sử dụng [underthesea](https://github.com/undertheseanlp/underthesea) của tác giả Vũ Anh)
* Loại bỏ dấu câu và các ký tự đặc biệt
* Xử lý các trường hợp người dùng láy láy âm tiết(Ví dụ: Ngooon quááááá điiiiiiii !!!!!)
* Chuẩn hóa các từ viết tắt cơ bản(Ví dụ: k, ko, k0 --> không, bt --> bình thường,...)
* Loại bỏ số và các từ chỉ có 1 ký tự

Sau khi thực hiện tuần tự và đầy đủ theo quy trình trên, ta thu được bộ dữ liệu sạch cho pha tiếp theo của mô hình. Chia dữ liệu theo tỉ lệ 80:20 để có được dữ liệu train và validation. Tỷ lệ này đều cho mỗi nhãn.

**Vector hóa dữ liệu**

Kỹ thuật được mình sử dụng ở đây là word2vec sau đó thực hiện comment2matrix. Bạn có thể tìm hiểu thêm 1 số phương pháp mô hình hóa vector khác cho từ hoặc hiểu kĩ hơn về word2vec qua bài viết trước đó của mình ở [đây](https://viblo.asia/p/xay-dung-mo-hinh-khong-gian-vector-cho-tieng-viet-GrLZDXr2Zk0).

Trước tiên, ta tiến hành tạo pretrained word embedding bằng thư viện gensim:

Import các thư viện cần thiết.

# -\*- coding: utf-8 -\*-

from gensim.models import Word2Vec

import os

import pandas as pd

Tiến hành đọc dữ liệu đã tiền xử lý ở các file. Ở đây mình lưu mỗi nhãn là 1 file csv nên mình sẽ đọc bằng pandas.

path = './data/'

def readdata(path):

list\_file = os.listdir(path)

data = pd.DataFrame()

for filename in list\_file:

data = pd.concat([data, pd.read\_csv(os.path.join(path, filename), sep = ',')])

return data.Review, data.Label

Đưa dữ liệu về định dạng input của gensim và tiến hành train mô hình. Ở đây mình vector hóa từ thành vector 128 chiều.

reviews, labels = readdata(path)

input\_gensim = []

for review in reviews:

input\_gensim.append(review.split())

model = Word2Vec(input\_gensim, size=128, window=5, min\_count=0, workers=4, sg=1)

model.wv.save("word.model")

Sau khi có được pretrained word embedding, ta tiến hành comment2matrix. Ở đây mình sẽ giải thích tại sao lại là comment2matrix. Lý do là mình muốn sử dụng mô hình CNN cho bài toán của mình.

Tuy nhiên, chúng ta nghe tới Convolutional neural network, chúng ta thường nghĩ ngay tới Computer Vision. Ở một khía cạnh khác, khi sử dụng cho các bài toán NLP, CNN lại cho ra những kết quả hết sức thú vị.

Mô hình CNN sử dụng các bộ lọc xoắn để nắm bắt mối quan hệ địa phương. Nhờ khả năng nắm bắt các mối quan hệ địa phương, CNN có khả năng lọc được các ngữ cảnh gần giữa các từ trong câu, có thể áp dụng rất tốt trong bài toán này.

Vậy, dữ liệu text và dữ liệu ảnh có điểm gì tương đồng mà ta có thể sử dụng một mô hình xử lý ảnh cho dữ liệu text? Có cách nào để chúng ta nhìn nhận một đoạn text như một bức ảnh không?

Như một cách để giúp chúng ta dễ tưởng tượng hơn, giả sử ta đã sử dụng word embedding như đã trình bày ở trên để biến mỗi từ trong một đoạn văn bản thành các vector có số chiều là **n**. Từ đó ta có thể coi môt câu văn(đoạn văn) như một ma trận **mxn**, trong đó m là kích thước hay số từ có trong văn bản đó. Ma trận này về mặt biểu diễn trông cũng có vẻ tương đồng với ma trận biểu diễn cho một bức ảnh đa mức xám(1 channel) với kích thước **mxn**.

Vậy, về mặt hình thức, ta có thể dễ dàng thấy được việc sử dụng CNN cho bài toán NLP là hoàn toàn có thể. Ta tiến hành chuyển mỗi comment của người dùng thành 1 matrix ngay thôi.

import gensim.models.keyedvectors as word2vec

model\_embedding = word2vec.KeyedVectors.load('./word.model')

word\_labels = []

max\_seq = 200

embedding\_size = 128

for word in model\_embedding.vocab.keys():

word\_labels.append(word)

def comment\_embedding(comment):

matrix = np.zeros((max\_seq, embedding\_size))

words = comment.split()

lencmt = len(words)

for i in range(max\_seq):

indexword = i % lencmt

if (max\_seq - i < lencmt):

break

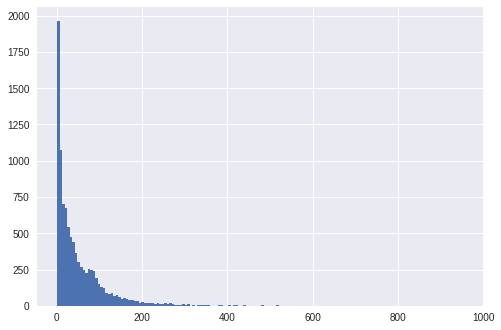
if(words[indexword] in words\_label):

matrix[i] = model\_embedding[words[indexword]]

matrix = np.array(matrix)

return matrix

Ở đây mình xử lý vấn đề các câu dài ngắn khác nhau bằng cách đệ quy các câu ngắn sao cho độ dài các câu là gần như tương đồng và có độ dài tối đa là 200 chiều. Để chọn được con số 200, chúng ta cùng nhìn lại dữ liệu của chúng ta một chút.



Như nhìn thấy ở trên, độ dài của các câu bình luận tập trung nhiều ở ngưỡng dưới 200, trên 200 từ không đáng kể. Với những bình luận trên 200 từ thì ta chỉ tiến hành phân tích trên 200 từ đầu tiên.

**Xây dựng và huấn luyện mô hình**

Về chi tiết mô hình CNN mình sẽ không đề cập tới chi tiết ở đây. Chỉ có một vài lưu ý nhỏ khi áp dụng mô hình CNN cho bài toán NLP.

* Thay vì đầu vào là pixels của hình ảnh, ta có một ma trận **mxn** là biểu diễn cho một câu hay một đoạn văn bản. Mỗi hàng của ma trận là vector đại diện cho một từ.
* Đối với bài toán ứng dụng CNN cho lĩnh vực NLP, bộ lọc này có một điều đặc biệt. Đó là kích thước theo chiều ngang của bộ lọc thường chính bằng số cột của ma trận biểu diễn cho văn bản. Để khi trượt, ta có thể trượt được trên toàn bộ từ, hay nói cách khác là ta có thể nắm bắt được ngữ nghĩa của toàn bộ từ. Khái niệm "trượt" ở đây sẽ được hiểu đơn giản chỉ là trượt từ trên xuống dưới, cho tới khi hết văn bản.
* Chúng ta lại thường bỏ qua khái niệm cửa sổ trong pooling và thực hiện pooling trên toàn bộ ma trận để lấy được một đặc trưng nổi trội duy nhất.

Tiến hành cài đặt một mô hình đơn giản bằng Keras(*Đây là mô hình cực kì đơn giản, chỉ nhằm mục đích giới thiệu*).

Trước tiên, tiến hành số hóa tất cả dữ liệu đầu vào, các câu comment được chuyển về ma trận số, các label được chuyển về dạng one-hot.

[1, 0, 0] : neutral

[0, 1, 0] : positive

[0, 0, 1] : negative

train\_data = []

label\_data = []

for x in tqdm(pre\_reviews):

train\_data.append(comment\_embedding(x))

train\_data = np.array(train\_data)

for y in tqdm(labels):

label\_ = np.zeros(3)

try:

label\_[int(y)] = 1

except:

label\_[0] = 1

label\_data.append(label\_)

Import các thư viện cần thiết và định nghĩa các tham số cho mô hình:

import numpy as np

from tensorflow.keras import layers

from tensorflow import keras

import tensorflow as tf

from keras.preprocessing import sequence

sequence\_length = 200

embedding\_size = 128

num\_classes = 3

filter\_sizes = 3

num\_filters = 150

epochs = 50

batch\_size = 30

learning\_rate = 0.01

dropout\_rate = 0.5

Định nghĩa kiến trúc mô hình:

x\_train = train\_data.reshape(train\_data.shape[0], sequence\_length, embedding\_size, 1).astype('float32')

y\_train = np.array(label\_data)

# Define model

model = keras.Sequential()

model.add(layers.Convolution2D(num\_filters, (filter\_sizes, embedding\_size),

padding='valid',

input\_shape=(sequence\_length, embedding\_size, 1), activation='relu'))

model.add(layers.MaxPooling2D(pool\_size=(198, 1)))

model.add(layers.Dropout(dropout\_rate))

model.add(layers.Flatten())

model.add(layers.Dense(128, activation='relu'))

model.add(layers.Dense(3, activation='softmax'))

# Train model

adam = tf.train.AdamOptimizer()

model.compile(loss='categorical\_crossentropy',

optimizer=adam,

metrics=['accuracy'])

print(model.summary())

Tiến hành train mô hình, sử dụng 7000 sample đầu để train, còn lại cho validaion.

model.fit(x = x\_train[:7000], y = y\_train[:7000], batch\_size = batch\_size, verbose=1, epochs=epochs, validation\_data=(x\_train[:3000], y\_train[:3000]))

model.save('models.h5')

Kêt quả thu được sau vào epoch đầu:

Epoch 1/50

7000/7000 [==============================] - 4s 549us/step - loss: 0.4499 - acc: 0.8257 - val\_loss: 0.6694 - val\_acc: 0.6894

Epoch 2/50

7000/7000 [==============================] - 4s 521us/step - loss: 0.4048 - acc: 0.8413 - val\_loss: 0.5958 - val\_acc: 0.7220

Epoch 3/50

7000/7000 [==============================] - 4s 508us/step - loss: 0.3789 - acc: 0.8496 - val\_loss: 0.6041 - val\_acc: 0.7118

**Mô hình dự đoán**

Đến thời điểm hiện tại, chúng ta đã có mô hình đã được huấn luyện cho bài toán dưới dạng **h5** file. Giờ đây, để test lại độ chính xác của mô hình với các dữ liệu thực tế(dữ liệu thực tế chưa có nhãn, ta dùng model này để gán nhãn cho nó), ta tiến hành load lại model:

from keras.models import load\_model

model\_sentiment = load\_model("models.h5")

Quá trình tiền xử lý dữ liệu test phải giống hệt với quá trình tiền xử lý với dữ liệu train, do đó, ta tái sử dụng lại các hàm này. Các hàm này là hàm **pre\_process** trong phần tiền xử lý dữ liệu(hàm này mình chỉ nó những yêu cầu cần xử lý ở trên chứ chưa có code vì cái này cho bạn tự xử lí phù hợp với dữ liệu của bạn) và hàm **comment\_embedding** trong phần vector hóa dữ liệu.

def pre\_process(text):

# các bạn tự thêm các tiền xử lý của mình vào đây nhá

return text

text = "đồ ăn ở đây vừa nhiều vừa ngon"

text = pre\_process(text)

maxtrix\_embedding = np.expand\_dims(comment\_embedding(text), axis=0)

maxtrix\_embedding = np.expand\_dims(maxtrix\_embedding, axis=3)

result = model.predict(maxtrix\_embedding)

result = np.argmax(result)

print("Label predict: ", result)

Kết quả label mà model dự đoán được lưu trong biến result, đấy sẽ là một trong các giá trị **0**, **1** hoặc **2**. Như trong định nghĩa ngay từ đầu của mình (-1: Negative, 0: Neutral, 1: Positive) thì label chuẩn của câu **đồ ăn ở đây vừa nhiều vừa ngon** là nhãn **Positive**(lưu ý: nhãn -1 trong python là index của vị trí cuối cùng trong mảng, tức là ứng với giá trị là 2).