AI VIET NAM – COURSE 2023

Project: Text Classification using Neural Network

Ngày 29 tháng 10 năm 2023

Phần 1. Giới thiệu

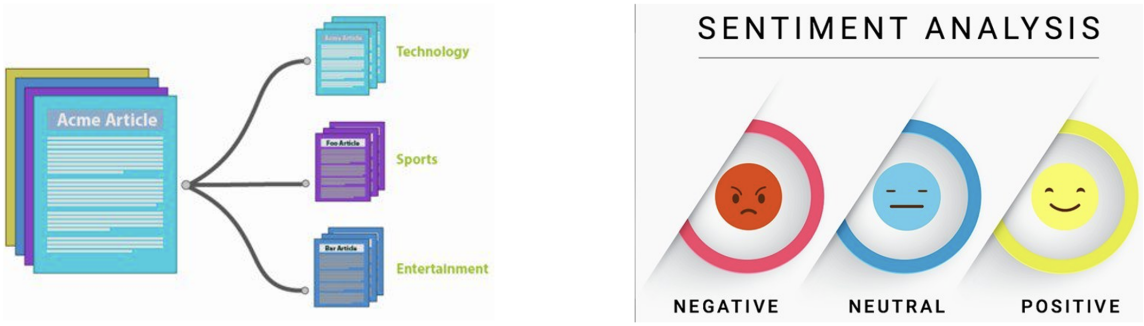
Phân loại văn bản (Text Classification) là một trong những bài toán cơ bản trong xử lý ngôn ngữ tự nhiên. Với yêu cầu của bài toán gán nhãn cho một đơn vị văn bản (văn bản, đoạn văn bản, câu, từ,...) với nhãn được xem xét từ một tập nhãn cho trước.

Phân loại các bài toán phân loại dựa vào đơn vị văn bản cần phân loại có:

- Token-level Classification: gán nhãn cho các đơn vị văn bản là các từ (token). Các bài toán điển hình trong nhóm này là: gán nhãn từ loại (Part-of-Speech Tagging) gán mỗi từ trong câu với các loại từ: danh từ, động từ,... và nhận dạng thực thể (Named Entity Recognition) gán nhãn cho các từ thuộc một thực thể nhất định như người, địa điểm,...

Hình 1: Bài toán gán nhãn từ loại (POS Tagging) và nhận dạng thực thể (NER).

- Document-level Classification: gán nhãn cho các đơn vị văn bản là các cụm từ, câu, đoạn văn. Các bài toán điển hình gồm có phân loại văn bản vào kinh tế, chính trị,... Phân tích cảm xúc (Sentiment Analysis): phân loại văn bản thành nhãn tích cực, tiêu cực hoặc trung tính.



Hình 2: Phân loại văn bản: kinh tế, chính trị,... và phân tích cảm xúc (Sentiment Analysis).

Phân loại các bài toán phân loại dựa vào số nhãn:
- Binary Classification: phân loại với số nhãn là 2. Ví dụ bài toán phân tích cảm xúc
- Multiclass Classification: phân loại với số nhãn nhiều hơn 2. Ví dụ: kinh tế, chính trị, văn hoá,...
- Multilabel Classification: phân loại một đơn vị văn bản có thể thuộc vào một hoặc nhiều nhãn.

Bài toán phân loại có nhiều ứng dụng quan trọng. Ứng dụng trong phân tích cảm xúc bình luận khách hàng với các sản phâm. Phân loại các thông tin giả mạo,...

Phần 2. Thực hành

Huấn luyện mô hình phân loại văn bản sử dụng mạng nơ-ron gồm các bước:
- Tải bộ dữ liệu NTC-CSV
- Tiền xử lý dữ liệu văn bản
- Biểu diễn dữ liệu văn bản thành các vectors
- Xây dựng mô hình phân loại
- Huấn luyện mô hình
- Dự đoán, đánh giá mô hình

1. Tải về bộ dữ liệu NTC-SCV

Bộ dữ liệu NTC-SCV gồm 50.000 mẫu trong đó có 30.000 mẫu cho training, 10.000 mẫu cho validation và 10.000 mẫu cho testing. Số nhãn là 2: positive (tích cực) và negative (tiêu cực).

```
# Download data from github
!git clone https://github.com/congnghia0609/ntc-scv.git
!unzip ./ntc-scv/data/data_test.zip -d ./data
!unzip ./ntc-scv/data/data_train.zip -d ./data
!rm -rf ./ntc-scv

# Read data from file
import os
import pandas as pd

def load_data_from_path(folder_path):
    examples = []
    for label in os.listdir(folder_path):
        full_path = os.path.join(folder_path, label)
        for file_name in os.listdir(full_path):
            file_path = os.path.join(full_path, file_name)
            with open(file_path, "r", encoding="utf-8") as f:
                lines = f.readlines()
            sentence = " ".join(lines)
            if label == "neg":
                label = 0
            if label == "pos":
                label = 1
            data = {
                'sentence': sentence,
                'label': label
            }
            examples.append(data)
    return pd.DataFrame(examples)

folder_paths = {
    'train': './data/data_train/train',
    'valid': './data/data_train/test',
    'test': './data/data_test/test'
}

train_df = load_data_from_path(folder_paths['train'])
valid_df = load_data_from_path(folder_paths['valid'])
test_df = load_data_from_path(folder_paths['test'])
```

2. Tiền xử lý dữ liệu

Với bộ dữ liệu NTC-SCV có một số bình luận viết bằng tiếng anh hoặc tiếng pháp và chứ các thẻ HTML, đường dẫn URLs. Tiền xử lý dữ liệu gồm 2 bước:

- Xoá bỏ những bình luận không phải tiếng việt

  Sử dụng thư viện langid được mô tả như hình sau:

Hình 3: Phát hiện ngôn ngữ sử dụng thư việ langid.

```
# Install library
!pip install langid

from langid.langid import LanguageIdentifier, model
def identify_vn(df):
    identifier = LanguageIdentifier.from_modelstring(model, norm_probs=True)
    not_vi_idx = set()
    THRESHOLD = 0.9
    for idx, row in df.iterrows():
        score = identifier.classify(row["sentence"])
        if score[0] != "vi" or (score[0] == "vi" and score[1] <= THRESHOLD):
            not_vi_idx.add(idx)
    vi_df = df[~df.index.isin(not_vi_idx)]
    not_vi_df = df[df.index.isin(not_vi_idx)]
    return vi_df, not_vi_df

train_df_vi, train_df_other = identify_vn(train_df)
```

- Làm sạch dữ liệu

  Các bước tiền làm sạch liệu:

  – Xoá bỏ thẻ HTML, đường dẫn URL
  – Xoá bỏ dấu câu, số
  – Xoá bỏ các ký tự đặc biệt, emoticons,...
  – Chuẩn hoá khoảng trắng
  – Chuyển sang viết thường

```
import re
import string

def preprocess_text(text):

    url_pattern = re.compile(r'https?://\s+\wwww\.\s+')
    text = url_pattern.sub(r" ", text)
```

```
    html_pattern = re.compile(r'<[^<>]+>')
    text = html_pattern.sub(" ", text)

    replace_chars = list(string.punctuation + string.digits)
    for char in replace_chars:
        text = text.replace(char, " ")

    emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U0001F1F2-\U0001F1F4"  # Macau flag
        u"\U0001F1E6-\U0001F1FF"  # flags
        u"\U0001F600-\U0001F64F"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U0001F1F2"
        u"\U0001F1F4"
        u"\U0001F620"
        u"\u200d"
        u"\u2640-\u2642"
        "]+", flags=re.UNICODE)
    text = emoji_pattern.sub(r" ", text)

    text = " ".join(text.split())

    return text.lower()

train_df_vi['preprocess_sentence'] = [
    preprocess_text(row['sentence']) for index, row in train_df_vi.iterrows()
]
valid_df['preprocess_sentence'] = [
    preprocess_text(row['sentence']) for index, row in valid_df.iterrows()
]
test_df['preprocess_sentence'] = [
    preprocess_text(row['sentence']) for index, row in test_df.iterrows()
]
```

3. Biểu diễn dữ liệu văn bản thành các vectors

Để biểu diễn dữ liệu văn bản thành các đặc trưng (vectors), chúng ta sử dụng thư viện torchtext.

```
!pip install -q torchtext==0.16.0

# word-based tokenizer
from torchtext.data.utils import get_tokenizer
tokenizer = get_tokenizer("basic_english")

# create iter dataset
def yield_tokens(sentences, tokenizer):
    for sentence in sentences:
        yield tokenizer(sentence)

# build vocabulary
from torchtext.vocab import build_vocab_from_iterator

vocab_size = 10000
vocabulary = build_vocab_from_iterator(
```

```
    yield_tokens(train_df_vi['preprocess_sentence'], tokenizer),
    max_tokens=vocab_size,
    specials=["<unk>"]
)
vocabulary.set_default_index(vocabulary["<unk>"])

# convert iter into torchtext dataset
from torchtext.data.functional import to_map_style_dataset
def prepare_dataset(df):
    for index, row in df.iterrows():
        sentence = row['preprocess_sentence']
        encoded_sentence = vocabulary(tokenizer(sentence))
        label = row['label']
        yield encoded_sentence, label

train_dataset = prepare_dataset(train_df_vi)
train_dataset = to_map_style_dataset(train_dataset)

valid_dataset = prepare_dataset(valid_df)
valid_dataset = to_map_style_dataset(valid_dataset)
```

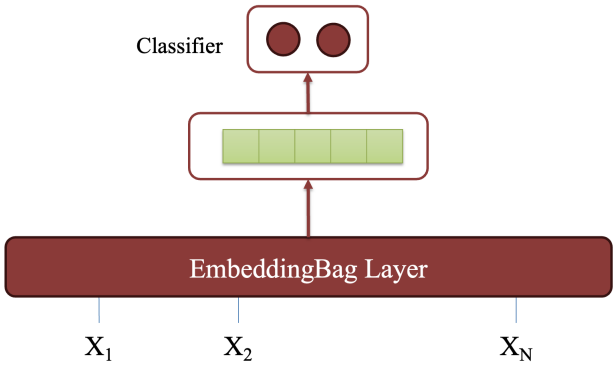
```
import torch

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def collate_batch(batch):
    encoded_sentences, labels, offsets = [], [], [0]
    for encoded_sentence, label in batch:
        labels.append(label)
        encoded_sentence = torch.tensor(encoded_sentence, dtype=torch.int64)
        encoded_sentences.append(encoded_sentence)
        offsets.append(encoded_sentence.size(0))

    labels = torch.tensor(labels, dtype=torch.int64)
    offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
    encoded_sentences = torch.cat(encoded_sentences)
    return encoded_sentences.to(device), offsets.to(device), labels.to(device)

from torch.utils.data import DataLoader

batch_size = 128
train_dataloader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=True,
    collate_fn=collate_batch
)
valid_dataloader = DataLoader(
    valid_dataset,
    batch_size=batch_size,
    shuffle=False,
    collate_fn=collate_batch
)
```

4. Xây dựng mô hình phân loại

- Xây dựng mô hình gồm các lớp sau:

1. EmbeddingBag: chuyển các từ với index tương ứng sang biểu diễn vector có chiều embedding_dim, sau đó tính giá trị trung bình các vector trong câu tạo thành vector đại diện cho mỗi văn bản.

2. Linear: chuyển các vector đại diện cho mỗi văn bản thành số node đầu ra là 2 node cho bài toán phân loại 2 lớp.



Hình 4: Mô hình phân loại văn bản sử dụng mạng nơ ron.

```python
from torch import nn

class TextClassificationModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class):
        super(TextClassificationModel, self).__init__()
        self.embedding = nn.EmbeddingBag(vocab_size, embed_dim, sparse=False)
        self.fc = nn.Linear(embed_dim, num_class)
        self.init_weights()

    def init_weights(self):
        initrange = 0.5
        self.embedding.weight.data.uniform_(-initrange, initrange)
        self.fc.weight.data.uniform_(-initrange, initrange)
        self.fc.bias.data.zero_()

    def forward(self, inputs, offsets):
        embedded = self.embedding(inputs, offsets)
        return self.fc(embedded)

num_class = len(set(train_df_vi['label']))
vocab_size = len(vocabulary)
embed_dim = 256
model = TextClassificationModel(vocab_size, embed_dim, num_class).to(device)
```

- Định nghĩa hàm loss và optimizer

```python
learning_rate = 5

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

5. Huấn luyện mô hình

- Định nghĩa hàm huấn luyện mô hình:

```python
import time

def train(model, optimizer, criterion, train_dataloader, epoch=0, log_interval=25):
    model.train()
    total_acc, total_count = 0, 0
    losses = []
    start_time = time.time()
```

```
    for idx, (inputs, offsets, labels) in enumerate(train_dataloader):
        optimizer.zero_grad()
        predictions = model(inputs, offsets)

        # compute loss
        loss = criterion(predictions, labels)
        losses.append(loss.item())

        # backward
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 0.1)
        optimizer.step()
        total_acc += (predictions.argmax(1) == labels).sum().item()
        total_count += labels.size(0)
        if idx % log_interval == 0 and idx > 0:
            elapsed = time.time() - start_time
            print(
                "| epoch {:3d} | {:5d}/{:5d} batches "
                "| accuracy {:8.3f}".format(
                    epoch, idx, len(train_dataloader), total_acc / total_count
                )
            )
            total_acc, total_count = 0, 0
            start_time = time.time()

    epoch_acc = total_acc / total_count
    epoch_loss = sum(losses) / len(losses)
    return epoch_acc, epoch_loss
```

- Định nghĩa hàm đánh giá

```
def evaluate(model, criterion, valid_dataloader):
    model.eval()
    total_acc, total_count = 0, 0
    losses = []

    with torch.no_grad():
        for idx, (inputs, offsets, labels) in enumerate(valid_dataloader):
            predictions = model(inputs, offsets)
            loss = criterion(predictions, labels)
            losses.append(loss)
            total_acc += (predictions.argmax(1) == labels).sum().item()
            total_count += labels.size(0)

    epoch_acc = total_acc / total_count
    epoch_loss = sum(losses) / len(losses)
    return epoch_acc, epoch_loss
```

- Huấn luyện mô hình:

```
num_class = len(set(train_df_vi['label']))
vocab_size = len(vocabulary)
embed_dim = 100
model = TextClassificationModel(vocab_size, embed_dim, num_class).to(device)

learning_rate = 5
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

num_epochs = 5
for epoch in range(1, num_epochs+1):
```

```
    epoch_start_time = time.time()
    train_acc, train_loss = train(model, optimizer, criterion, train_dataloader, epoch)
    eval_acc, eval_loss = evaluate(model, criterion, valid_dataloader)
    print("-" * 59)
    print(
        "| End of epoch {:3d} | Time: {:5.2f}s | Train Accuracy {:8.3f} | Train Loss {:8.3f} "
        "| Valid Accuracy {:8.3f} | Valid Loss {:8.3f} ".format(
            epoch, time.time() - epoch_start_time, train_acc, train_loss, eval_acc, eval_loss
        )
    )
    print("-" * 59)
```

3. Dự đoán và đánh giá mô hình

- Định nghĩa hàm dự đoán:

```
def predict(text):
    with torch.no_grad():
        encoded = torch.tensor(vocabulary(tokenizer(text)))
        output = model(encoded, torch.tensor([0]))
        return output.argmax(1).item()
```

- Đánh giá độ chính xác trên tập test

```
predictions, labels = [], []
for index, row in test_df.iterrows():
    sentence = row['preprocess_sentence']
    label = row['label']
    prediction = predict(sentence)
    predictions.append(prediction)
    labels.append(label)

sum(torch.tensor(predictions) == torch.tensor(labels))/len(labels)
```

Kết quả độ chính xác trên tập test là 87

Phần 3. Câu hỏi trắc nghiệm

Câu hỏi 1 Độ đo đánh giá hiệu suất mô hình phân loại văn bản là?
a) Accuracy
b) BLEU
c) ROUGE
d) MSE

Câu hỏi 2 Bộ dữ liệu được sử dụng cho bài toán phân loại văn bản cho tiếng việt được sử dụng trong project là?
a) NTC-SCV
b) IMDB-Review
c) SQuAD
d) WikiSum

Câu hỏi 3 Tỉ lệ bộ dữ liệu dùng cho training, validation và testing là?
a) 0.7 : 0.2 : 0.1
b) 0.6 : 0.2 : 0.2
c) 0.7 : 0.1 : 0.2
d) 0.8 : 0.1 : 0.1

Câu hỏi 4 Bước tiền xử lý dữ liệu không được sử dụng trong phần lập trình là?
a) Xoá bỏ đường dẫn URL
b) Xoá bỏ các thẻ HTML
c) Xoá bỏ các từ dừng
d) Chuyển sang viết thường

Câu hỏi 5 Thư viện langid được sử dụng để làm gì?
a) Phát hiện ngôn ngữ cho đoạn văn bản
b) Dịch máy
c) Tóm tắt văn bản
d) Phân loại văn bản

Câu hỏi 6 Phương pháp tách từ được sử dụng trong phần lập trình là?
a) Word-based Tokenization
b) Character-based Tokenization
c) Subword Tokenization
d) Sentence Tokenization

Câu hỏi 7 Kích thược bộ từ điển được sử dụng trong phần lập trình là?
a) 20.000

b) 10.000
c) 15.000
d) 25.000

Câu hỏi 8 Phương pháp biểu diễn văn bản được sử dụng trong phần lập trình là?
a) TF-IDF
b) Dense Representation
c) BoW
d) One-hot Encoding

Câu hỏi 9 Kích thước của biến output trong đoạn code sau là?

```
import torch.nn as nn

vocab_size = 7
embedding_dim = 4
embedding_sum = nn.EmbeddingBag(vocab_size, embedding_dim, mode='sum')
inputs = torch.tensor([1, 2, 4, 5, 4, 3], dtype=torch.long)
offsets = torch.tensor([0, 3], dtype=torch.long)
outputs = embedding_sum(inputs, offsets)
outputs.shape
```

a) 2 x 4
b) 7 x 4
c) 2 x 7
d) 4 x 7

Câu hỏi 10 Tham số 'mode' trong EmbeddingBag Layer sử dụng trong phần lập trình là?
a) max
b) min
c) mean
d) sum

Câu hỏi 11 Số node đầu ra của mô hình phân loại được định nghĩa trong phần lập trình là?
a) 5
b) 4
c) 3
d) 2

Câu hỏi 12 Phương pháp khởi tạo giá trị trọng số bias trong Linear Layer được sử dụng trong phần lập trình là?
a) zero_()
b) one_()
c) random
d) uniform()

Câu hỏi 13 Hàm sau được sử dụng để làm gì?

```python
def evaluate(model, criterion, valid_dataloader):
    model.eval()
    total_acc, total_count = 0, 0
    losses = []

    with torch.no_grad():
        for idx, (inputs, offsets, labels) in enumerate(valid_dataloader):
            predictions = model(inputs, offsets)
            loss = criterion(predictions, labels)
            losses.append(loss)
            total_acc += (predictions.argmax(1) == labels).sum().item()
            total_count += labels.size(0)

    epoch_acc = total_acc / total_count
    epoch_loss = sum(losses) / len(losses)
    return epoch_acc, epoch_loss
```

a) Huấn luyện mô hình trên tập training
b) Đánh giá mô hình trên tập validation
c) Biểu diễn văn bản thành vectors
d) Tiền xử lý văn bản

- Hết -