Xử lý và phân loại bình luận tiếng Việt bằng Spark và PyTorch

Từ Quốc Huy[1], Phạm Phong Phú[1], và Đặng Việt Dũng[1]

Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh
{huytq.15,phupp.15,dungdd.15}@grad.uit.edu.vn

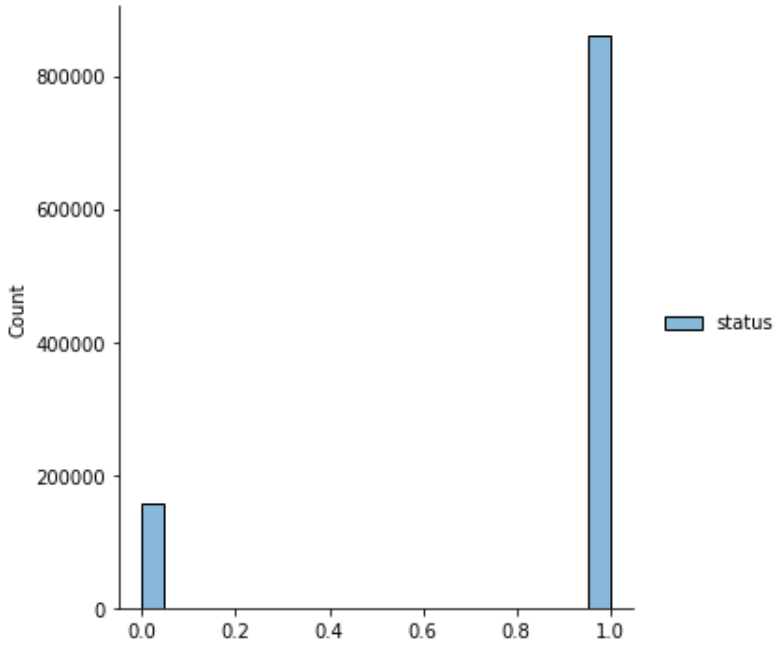
Tóm tắt nội dung Các mạng xã hội nơi người dùng có thể đăng tải bình luận sẽ gặp vấn đề nhiều người dùng đăng tải các bình luận tiêu cực và công kích cá nhân. Để xử lý những bình luận này, bao gồm việc ẩn các bình luận tiêu cực và hạn chế các người dùng thường hay đăng các nội dung tiêu cực tốn khá nhiều chi phí và công sức. Vấn đề đặt ra là cần một công cụ có thể nhận diện và tự đông xử lý các bình luận tiêu cực. Hiện nay thì xử lý ngôn ngữ tự nhiên bằng Spark và PyTorch là hai hướng tiếp cận để có thể giải quyết bài toán phân loại bình luận. Sau khi xử lý dữ liệu và huấn luyện mô hình bằng thuận toán Logistic Regression (LR) trên Spark và Recurrent Neural Network (RNN) bằng PyTorch thì kết quả là hai mô hình có độ chính xác tương đương nhau và đủ để sử dụng trong thực tế.

Từ khóa: Xử lý ngôn ngữ tự nhiên · Bình luận · Spark · PyTorch

1 Giới thiệu

Bài toán đặt ra dựa trên tình huống thực tế đối với công việc hàng ngày của một thành viên trong đội ngũ kiểm duyệt bình luận bằng tiếng Việt trên một trang web mạng xã hội. Người sử dụng trang web có thể đăng tải các bình luận, mà nội dung bình luận có thể là nội dung công kích cá nhân hoặc tiêu cực. Để giải quyết vấn đề này thì thường phải có một đội kiểm duyệt nội dung để ẩn các bình luận này. Vấn đề đặt ra là số lượng bình luận cần kiểm duyệt càng ngày càng nhiều hơn. Để giải quyết yêu cầu này thì vấn đề xây dựng một hệ thống tự động kiểm duyệt bình luận được đặt ra. Hệ thống tự dộng kiểm duyệt ngoài hỗ trợ vận hành hệ thống hiện tại thì có thể áp dụng cho các bài toán ví dụ như kiểm duyệt cho hệ thống thời gian thực trong tương lai. Tóm lại là cần có một giải pháp kiểm duyệt bình luận dựa trên lịch sử kiểm duyệt sẵn có.

Từ dữ liệu lịch sử kiểm duyệt, có thể thu thập được một bộ dữ liệu gồm nội dung bình luận và trạng thái kiểm duyệt. Dữ liệu gồm 1 triệu dòng được trích xuất trong 6 tháng đầu năm 2021.



Hình 1. Phân phố trạng thái trên tập bình luận

2 Xử lý dữ liệu

2.1 Tiền xử lý dữ liệu

Dữ liệu đầu vào là các bình luận bằng tiếng Việt. Các bình luận này do người dùng nhập nên tiềm ẩn một số vấn đề như từ viết tắt, viết sai chính tả, chứa email, hashtag, các emoji,... Để có thể phân tích dữ liệu, cần làm sạch dữ liệu đầu vào.

Loại bỏ Hashtag bỏ kí tự # trong #hashtag để đưa về dạng chuỗi không có kí tự #

Loại bỏ URL bỏ các chuỗi được nhận diện là URL

Loại bỏ email bỏ các chuỗi nhận diện là email

Unescape các kí tự trong URL phải unescape các kí tự đặc biệt nếu dữ liệu là dạng HTML hoặc XML

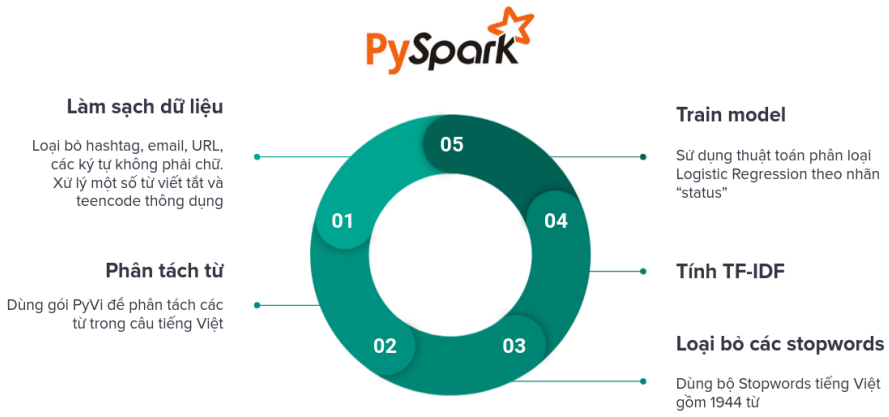
Loại bỏ các kí tự không phải chữ chỉ giữ lại các kí tự a-z và A-Z, đồng thời các kí tự có dấu tiếng Việt.

Loại bỏ các chữ mà người dùng nhập đúp loại bỏ các chữ người dùng nhập đúp, ví dụ người dùng có thể nhập "anhhh", trong khi đúng thì phải là "anh", trường hợp này phải bỏ đi cụm "hh".

Đồng bộ một số từ đưa một số từ đồng nghĩa về đòng dạng, ví dụ từ "haha" sẽ được dùng chung cho tiếng cười, mà người dùng có thể nhập là "hahaha", "hoho", "hihi", "kkk",...

```
df
  .withColumn("content", regexp_replace(col("content"), hashtag\_regex,
      "$1"))
  .withColumn("content", regexp_replace(col("content"), url\_regex, ""))
  .withColumn("content", regexp_replace(col("content"), email\_regex,
      ""))
  .withColumn("content", html_unescape("content"))
  .withColumn("content", regexp_replace(col("content"),
      not\_char\_regex, " "))
  .withColumn("content", regexp_replace(col("content"),
      r"([^oplen])\1+", "$1"))
  .withColumn("content", regexp_replace(col("content"),
      r"([oplen])\1\1+", "$1"))
  .withColumn("content", trim(col("content")))
```

2.2 Huẩn luyện mô hình bằng Logistic Regression trong Spark



Hình 2. Quá trình xây dựng mô hình

Chuẩn bị dữ liệu Trước khi bắt đầu xử lý thì cần chia nhỏ dữ liệu ra thành các file nhỏ. Các file sẽ được phân loại theo trường *status* và được chia thành 20 file nhỏ.

```
# Read raw data
rawData = spark.read.csv('./input/train.csv', inferSchema=True,
    header=True)
rawValidData = spark.read.csv('./input/validation.csv',
    inferSchema=True, header=True)
rawTestData = spark.read.csv('./input/test.csv', inferSchema=True,
    header=True)

# Split raw data
rawData.repartition(20).write.partitionBy("status").csv('./splitted/train',
    mode="overwrite")
rawValidData.repartition(20).write.partitionBy("status").csv('./splitted/valid',
    mode="overwrite")
rawTestData.repartition(20).write.partitionBy("status").csv('./splitted/test',
    mode="overwrite")
```

Do đây là dữ liệu tiếng Việt cho nên cần một bộ từ dừng cho tiếng Việt. Trong bài báo cáo này thì một bộ từ dừng gồm 1944 từ và cụm từ tiếng Việt và các dấu cách sẽ được thay bằng dấu gạch dưới (_)

```
a_ha
ai
ai_ai
amen
anh
ba
ba_ba
```

Xây dựng mô hình

Chuỗi hóa sử dụng gói PyVi để tách các câu tiếng Việt thành các từ và cụm từ.

```
tokenize_text_udf = udf(lambda s:
    ViPosTagger.postagging(ViTokenizer.tokenize(s))[0],
    ArrayType(StringType()))
def lower_arr_str(x):
    res = []
    for x_ in x: res.append(x_.lower())
    return res
to_lower_str_arr = udf(lower_arr_str, ArrayType(StringType()))
def tokenizeDf(df : DataFrame) -> DataFrame:
    return df.withColumn("content1",
        to_lower_str_arr(tokenize_text_udf(col("content"))))
```

Loại bỏ từ dừng từ dừng là các từ được sử dụng để hoàn chỉnh câu, tuy nhiên không đóng góp vào ý nghĩa cuối cùng của câu. Do đó hoàn toàn có thể bỏ qua sự tồn tại của các từ dừng trong câu. Từ dừng sẽ được dùng trong bộ từ dừng tiếng Việt được đề cập ở phần trước.

```
stopwords_file = open("./vietnamese-stopwords.txt", "r")
stopwords_list = stopwords_file.read().split('\n')
stopwords_remover = StopWordsRemover(
    inputCol="content1",
    outputCol="content2",
    stopWords=stopwords_list,
)
```

Băm TF và tính IDF TF-IDF là một cách để biểu diễn dữ liệu chuỗi dưới dạng ma trận bằng cách sử dụng các số mà các số đó có thể biểu diễn được giá trị của các từ trong câu.

```
hashing_tf = HashingTF(
    inputCol="content2",
    outputCol="term_frequency",
)
idf = IDF(
    inputCol="term_frequency",
    outputCol="features",
    minDocFreq=5
)
```

Huấn luyện mô hình Đưa các bước chuẩn bị dữ liệu ở trên thành Spark Pipeline. Spark Pipeline này sẽ nhận tập training và cho ra được mô hình nhận diện bình luận.

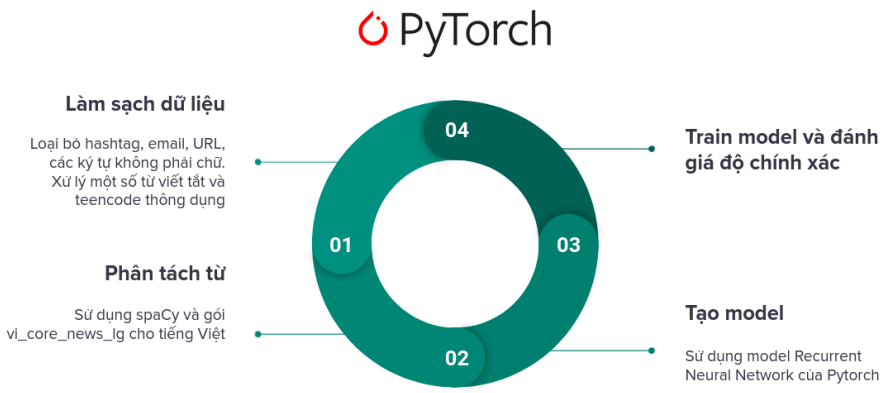
```
lr = LogisticRegression(labelCol="status")
sematic_analysis_pipeline = Pipeline(
    stages=[stopwords_remover, hashing_tf, idf, lr]
)
model = sematic_analysis_pipeline.fit(trainingData)
```

Sau khi có được mô hình, ta có thể đánh giá độ chính xác của mô hình dựa trên tập dữ liệu validation và test. Để đánh giá độ chính xác của mô hình thì có thể sử dụng lớp MulticlassClassificationEvaluator của PySpark.

```
val_df = model.transform(validationData)
test_df = model.transform(testData)
evaluator = MulticlassClassificationEvaluator(labelCol="status",
    metricName="accuracy")
accuracy_val = evaluator.evaluate(val_df)
```

```
accuracy_test = evaluator.evaluate(test_df)
```

2.3 Huấn luyện mô hình bằng Recurrent Neural Network trong PyTorch



Hình 3. Quá trình xây dựng mô hình

Chuẩn bị dữ liệu Một trong những khái niệm chính của TorchText là Field. Chúng xác định cách dữ liệu sẽ được xử lý. Trong phần phân loại bình luận này thì dữ liệu bao gồm cả chuỗi thô của bình luận và trạng thái, "duyệt" hoặc "ẩn".

Các tham số của một Field chỉ định cách dữ liệu sẽ được xử lý. Field TEXT sẽ được sử dụng để xác định cách bình luận sẽ được đánh giá và Field LABEL để xử lý trạng thái bình luận.

Field TEXT: ta có tham số tokenize="spacy". Điều này xác định rằng "tokenization" (hành động tách chuỗi thành các "token" rời rạc) được thực hiện bằng cách sử dụng trình mã hóa spaCy. Nếu không có đối số mã hóa nào được truyền, thì mặc định chỉ đơn giản là tách chuỗi trên khoảng trắng. Cần chỉ định tokenizer_language="vi_core_news_lg" để cho TorchText biết mô hình spaCy xử lý tiếng Việt sẽ sử dụng.

Field LABEL được định nghĩa bởi một LabelField, một lớp con của lớp Field được sử dụng để xử lý các nhãn.

```
CONTENT = data.Field(tokenize = 'spacy',
                     tokenizer_language = 'vi_core_news_lg')
STATUS = data.LabelField(dtype = torch.float)
```

```
fields = [('content', CONTENT), ('status', STATUS)]
train_data, valid_data, test_data = data.TabularDataset.splits(
                                    path = './input',
                                    train = 'train_filtered.csv',
                                    validation =
                                        'validation_filtered.csv',
                                    test = 'test_filtered.csv',
                                    format = 'csv',
                                    fields = fields,
                                    skip_header = True
)
```

Dữ liệu được đặt trong 3 tập tin, các dữ liệu đều đã được làm sạch từ phần trước.

word	index	one-hot vector
I	0	[1, 0, 0, 0]
hate	1	[0, 1, 0, 0]
this	2	[0, 0, 1, 0]
film	3	[0, 0, 0, 1]

Hình 4. Ví dụ one-hot vector

Tiếp theo là bước xây dựng bộ từ vựng (vocabulary). Các từ vựng được sắp xếp theo thứ tự giảm dần của tần suất xuất hiện và được đại diện bằng một chỉ số. Mỗi chỉ số sẽ được dùng để dựng one-hot vector (Xem hình 4). One-hot vector này là một vector có số chiều bằng số từ trong bộ từ vựng và với từ tương ứng chỉ só thì chiều chỉ số bằng 1, còn lại sẽ bằng 0. Do số lượng của bộ từ vựng quá lớn nên chỉ chọn giữ lại 25.000 từ vựng đầu tiên theo thứ tự giảm dần của độ phổ biến. Đối với những từ đã bị cắt đi thì sẽ được đánh dấu bằng kí tự <unk>. Kích thước của bộ từ vựng sẽ là 25.002, trong đó gồm 25.000 từ phổ biến nhất và hai từ <pad> và <unk>.

```
MAX_VOCAB_SIZE = 25_000
CONTENT.build_vocab(train_data, max_size = MAX_VOCAB_SIZE)
STATUS.build_vocab(train_data)
```

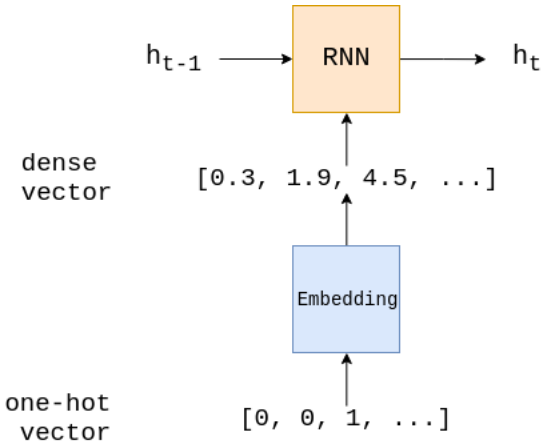
Bước cuối cùng của việc chuẩn bị dữ liệu là tạo các iterator. Cần phải duyệt nhiều lần qua iterator trong vòng lặp huấn luyện/đánh giá và chúng trả về một loạt các mẫu (được lập chỉ mục và chuyển đổi thành tensor) ở mỗi lần lặp.

Sử dụng BucketIterator là một loại iterator đặc biệt sẽ trả về một loạt các mẫu trong đó mỗi mẫu có độ dài tương tự nhau, giảm thiểu số lượng <pad> của mỗi mẫu.

```
train_iterator, valid_iterator, test_iterator =
    data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = 64,
    sort_key = lambda x: len(x.content),
    sort_within_batch = False,
    device = device)
```

Xây dựng mô hình Lớp mô hình RNN được tạo ra trong PyTorch là một lớp con của nn.Module.

```
class RNN(nn.Module):
    def __init__(self, input_dim, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding(input_dim, embedding_dim)
        self.rnn = nn.RNN(embedding_dim, hidden_dim)
        self.fc = nn.Linear(hidden_dim, output_dim)
    def forward(self, text):
        embedded = self.embedding(text)
        output, hidden = self.rnn(embedded)
        assert torch.equal(output[-1,:,:], hidden.squeeze(0))
        return self.fc(hidden.squeeze(0))
```



Hình 5. Mô tả quá trình chuyển từ one-hot vector thành dense vector

Hàm __init__ thì phải định nghĩa các tầng embedding, RNN và linear. Đa số các tầng sẽ có các tham số mặc định trừ một số trường hợp đặc biệt. Tầng

embedding được dùng để chuyển one-hot vector thành một dense embedding vector (Xem hình 5). Tầng embedding này chỉ đơn giản là một tầng kết nối đầy đủ. Ngoài việc giảm số chiều của đầu vào cho RNN, các từ có tác động tương tự đến cảm xúc của bài đánh giá cũng được ánh xạ gần nhau trong không gian vector dày đặc này. Tầng RNN sẽ nhận dense vector và trạng thái ẩn h_{t-1} để tính ra trạng thái ẩn h_t. Cuối cùng tầng linear lấy trạng thái ẩn cuối cùng và đăng trạng thái này vào tầng kết nối đầy đủ $f(h_T)$, và chuyển nó thành chiều đầu ra tương ứng.

Hàm forward được gọi khi đăng các mẫu vào mô hình.

Huấn luyện mô hình Đầu tiên phải tạo một optimizer. Optimizer được dùng để cập nhật các tham số cho mô hình. Thuật toán stochastic gradient descent (SGD) sẽ được sử dụng. Tiếp theo phải định nghĩa loss function, hay được gọi là criterion trong PyTorch.

```
N_EPOCHS = 5
best_valid_loss = float('inf')
for epoch in range(N_EPOCHS):
    start_time = time.time()
    train_loss, train_acc = train(model, train_iterator, optimizer,
        criterion)
    valid_loss, valid_acc = evaluate(model, valid_iterator, criterion)
    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)
    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        torch.save(model.state_dict(), './model_rnn/model.pt')
    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m
        {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc:
        {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {valid_loss:.3f} | Val. Acc:
        {valid_acc*100:.2f}%')
```

Cần phải huấn luyện mô hình qua nhiều lần lặp. Mỗi lần lặp thì mô hình sẽ tính lại và chọn ra mô hình cho độ mất mát trên tập validate tốt nhất.

3 Kết quả

3.1 Mô hình bằng Logistic Regression trong Spark

Mô hình LR đạt độ chính xác 87.3% trên tập validation và đạt 87.2% trên tập test.

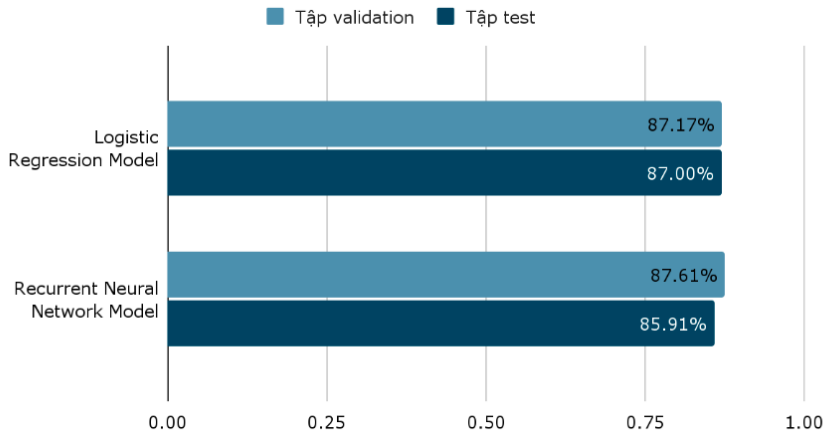
```
Validation accuracy: 87.31911%
Test accuracy: 87.18459%
```

3.2 Mô hình bằng Recurrent Neural Network trong PyTorch

Sau 5 lần huấn luyện thì mô hình RNN đạt độ chính xác 87.6% trên tập validation và đạt 85.9% trên tập test.

```
Validation accuracy: 87.61%
Test accuracy: 85.91%
```

4 Kết luận



Hình 6. So sánh độ chính xác giữa hai mô hình

Cả hai mô hình đều cho đô chính xác cao (87%) đổi với tập dữ liệu validation, mô hình RNN có độ chính xác cao hơn 0.3%. Riêng đối với tập test thì mô hình LR cho độ chính xác cao hơn 1.2% so với RNN.

Thời gian huấn luyện mô hình LR tốn khoảng 15 phút, trong khi đó việc huấn luyện mô hình RNN phải tốn 5 lần lặp (có thể nhiều hơn) với mỗi lần chạy 15 phút thì thời gian huấn luyện mô hình RNN tốn nhiều hơn so với LR.

Trong điều kiện thực tế, có thể chạy kết hợp cả hai mô hình. Nếu hai mô hình cho ra kết quả khác nhau với 1 bình luận thì bình luận đó sẽ đưa vào trạng thái đợi người kiểm duyệt. Nếu kết quả giống nhau thì có thẻ sử dụng trực tiếp kết quả đó.

Có thể nghiên cứu thêm hướng phát triển về bước tiền xử lý dữ liệu. Các phương pháp tiền xử lý bình luận được đề cập ở trên vẫn chưa đầy đủ và còn phải đối mặt với các bài toán như teen code, sai chính tả tiếng Việt, từ viết tắt,...