链滴

矩池云 | Tony 老师解读 Kaggle Twitter 情感分析案例

作者：matpool

原文链接：https://ld246.com/article/1586485272929

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<img src="https://ld246.com/images/img-loading.svg" alt="custom120.png" data-src="ttps://b3logfile.com/file/2020/04/custom120-46dd4236.png?imageView2/2/interlace/1/forma/jpg">
今天 Tony 老师给大家带来的案例是 Kaggle 上的 Twitter 的情感分析竞赛。在这个案例中，将用预训练的模型 BERT 来完成对整个竞赛的数据分析。
<h2 id="导入需要的库">导入需要的库</h2>
<pre><code class="highlight-chroma">import numpy as np
import pandas as d
from math import ceil, floor
import tensorflow as tf
import tensorflow keras.layers as L
from tensorflow.keras.initializers import TruncatedNormal
from sklearn impot model_selection
from transformers mport BertConfig, TFBertPreTrainedModel, TFBertMainLayer
from tokenizers iport BertWordPieceTokenizer
</code></pre>
<h2 id="读取并解释数据">读取并解释数据</h2>
在竞赛中，对数据的理解是非常关键的。因此我们首先要做的就是读取数据，然后查看数据的内以及特点。
先用 pandas 来读取 csv 数据，
<pre><code class="highlight-chroma">
train_df = pd.read csv('train.csv')
train_df.dropna(inlace=True)
test_df = pd.read_sv('test.csv')
test_df.loc[:, "selected_text"] = test_df.text.values
submission_df = d.read_csv('sample_submission.csv')

</code></pre>
再查看下我们的数据的数量，我们一共有 27485 条训练数据，3535 条测试数据，
<pre><code class="highlight-chroma">print("train numbers =", train_df.shape)
print("test numbes =", test_df.shape)
</code></pre>
紧接着查看训练数据和测试数据前 10 条表单的字段跟数据，表单中包含了一下几个数据字段：

textID: 文本数据记录的唯一 ID；

```
</li>
<li>
<p>text: 原始语句；</p>
</li>
<li>
<p>selected_text: 表示情感的语句；</p>
</li>
<li>
<p>sentiment: 情感类型, neutral 中立, positive 积极, negative 消极；</p>
</li>
</ol>
<p>从数据中我们可以得出，目标就是根据现有的情感从原本是的语句中选出能代表这个情感的语句
分。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">train_df.head(10)
</span></span><span class="highlight-line"><span class="highlight-cl">test_df.head(10)
</span></span></code></pre>
<h2 id="定义常量">定义常量</h2>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl"># bert预训练权重跟数据存放的目录
</span></span><span class="highlight-line"><span class="highlight-cl">PATH = "./bert-ba
e-uncased/"
</span></span><span class="highlight-line"><span class="highlight-cl"># 语句最大长度
</span></span><span class="highlight-line"><span class="highlight-cl">MAX_SEQUENCE_
ENGTH = 128
</span></span></code></pre>
<h2 id="载入词向量">载入词向量</h2>
<p>BERT 是依据一个固定的词向量来进行训练的。因此在竞赛中需要先使用 BertWordPieceTokeni
er 来加载这些词向量，其中的 lowercase=True 表示所有的词向量都是小写。设置大小写不敏感可
减少模型对资源的占用。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">TOKENIZER = BertWordPieceTokenizer(f"{PATH}/vocab.txt", lowercase=True)
</span></span></code></pre>
<h2 id="定义数据加载器">定义数据加载器</h2>
<p>定义数据预处理函数</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">
</span></span><span class="highlight-line"><span class="highlight-cl">def preprocess(tw
et, selected_text, sentiment):
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"># 将被转成byte str
ng的原始字符串转成utf-8的字符串
</span></span><span class="highlight-line"><span class="highlight-cl">    tweet = tweet.
ecode('utf-8')
</span></span><span class="highlight-line"><span class="highlight-cl">    selected_text =
elected_text.decode('utf-8')
</span></span><span class="highlight-line"><span class="highlight-cl">    sentiment = se
timent.decode('utf-8')
</span></span><span class="highlight-line"><span class="highlight-cl">    tweet = " ".join(
tr(tweet).split())
</span></span><span class="highlight-line"><span class="highlight-cl">    selected_text =
 ".join(str(selected_text).split())
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl"># 标记出selected t
```

```
xt和text共有的单词
    idx_start, idx_e
d = None, None
    for index in (i fo
 i, c in enumerate(tweet) if c == selected_text[0]):
        if tweet[index
index+len(selected_text)] == selected_text:
            idx_start =
ndex
            idx_end =
ndex + len(selected_text)
            break
    intersection = [
] * len(tweet)
    if idx_start != N
ne and idx_end != None:
        for char_idx i
 range(idx_start, idx_end):
            intersectio
[char_idx] = 1

# 对原始数据用词
量进行编码, 这里会返回原始数据中的词在词向量中的下标
# 和原始数据中每
词向量的单词在文中的起始位置跟结束位置
    enc = TOKENIZ
R.encode(tweet)
    input_ids_orig,
ffsets = enc.ids, enc.offsets
    target_idx = []
    for i, (o1, o2) in
enumerate(offsets):
        if sum(inters
ction[o1: o2]) &gt; 0:
            target_idx.
ppend(i)
    target_start = t
rget_idx[0]
    target_end = ta
get_idx[-1]

    sentiment_map
= {
        'positive': 38
3,
        'negative': 49
7,
        'neutral': 869
,
    }

# 将情感标签和原
的语句的词向量组合在一起组成我们新的数据
    input_ids = [101
```

```
 + [sentiment_map[sentiment]] + [102] + input_ids_orig + [102]
    input_type_ids
 [0] * (len(input_ids_orig) + 4)
    attention_mask
= [1] * (len(input_ids_orig) + 4)
    offsets = [(0, 0),
0, 0), (0, 0)] + offsets + [(0, 0)]
    target_start +=
3
    target_end +=

# 计算需要paddni
g的长度, BERT是以固定长度进行输入的，因此对于不足的我们需要做pandding
    padding_length
= MAX_SEQUENCE_LENGTH - len(input_ids)
    if padding_leng
h &gt; 0:
        input_ids = i
put_ids + ([0] * padding_length)
        attention_ma
k = attention_mask + ([0] * padding_length)
        input_type_id
 = input_type_ids + ([0] * padding_length)
        offsets = offs
ts + ([(0, 0)] * padding_length)
    elif padding_le
gth &lt; 0:
        pass

    return (
        input_ids, att
ntion_mask, input_type_ids, offsets,
        target_start,
arget_end, tweet, selected_text, sentiment,
    )
```

定义数据加载器

```
class TweetDatase
(tf.data.Dataset):

    outputTypes = (
        tf.dtypes.int3
,  tf.dtypes.int32,   tf.dtypes.int32,
        tf.dtypes.int3
,  tf.dtypes.float32, tf.dtypes.float32,
        tf.dtypes.stri
g, tf.dtypes.string,  tf.dtypes.string,
    )

    outputShapes =
(
```

```
        (128,),    (128,)
 (128,),
        (128, 2), (),
),
        (),        (),      (),
    )

    def _generator(
weet, selected_text, sentiment):
        for tw, st, se
n zip(tweet, selected_text, sentiment):
            yield prep
ocess(tw, st, se)

    def __new__(cls,
weet, selected_text, sentiment):
        return tf.data
Dataset.from_generator(
            cls._genera
or,
            output_ty
es=cls.outputTypes,
            output_sh
pes=cls.outputShapes,
            args=(twe
t, selected_text, sentiment)
        )

    @staticmethod
    def create(dataf
ame, batch_size, shuffle_buffer_size=-1):
        dataset = Tw
etDataset(
            dataframe.
ext.values,
            dataframe.
elected_text.values,
            dataframe.
entiment.values
        )

        dataset = da
aset.cache()
        if shuffle_buf
er_size != -1:
            dataset =
ataset.shuffle(shuffle_buffer_size)
        dataset = da
aset.batch(batch_size)
        dataset = da
aset.prefetch(tf.data.experimental.AUTOTUNE)

        return datase
```

</code></pre>
<h2 id="定义模型">定义模型</h2>
我们使用 BERT 模型来进行这次竞赛，这里对 BERT 模型做一些简单的介绍。
BERT 的全称是 Bidirectional Encoder Representation from Transformers，即双向 Transfor
er 的 Encoder，因为 decoder 是不能获要预测的信息的。
模型的主要创新点都在 pre-train 方法上，即用了 Masked LM 和 Next Sentence Prediction
种方法分别捕捉词语和句子级别 representation。
BERT 主要特点如下：

使用了 Transformer 作为算法的主要框架，Trabsformer 能更彻底的捕捉语句中的双向关系；<
p>

使用了 Mask Language Model 和 Next Sentence Prediction 的多任务训练目标；

使用更强大的机器训练更大规模的数据，Google 开源了 BERT 模型，我们可以直接使用 BERT
为 Word2Vec 的转换矩阵并高效的将其应用到自己的任务中。

BERT 的本质是在海量的语料基础上，运行自监督学习方法让单词学习得到一个较好的特征表示

在之后特定任务中，可以直接使用 BERT 的特征表示作为该任务的词嵌入特征。所以 BERT 提供
是一个供其它任务迁移学习的模型，该模型可以根据任务微调或者固定之后作为特征提取器。
在竞赛中，我们定义了一个 BertModel 类，里面使用 TFBertPreTrainedModel 来进行推理。<
p>
BERT 的输出我们保存在 hidden_states 中，然后将这个得到的 hidden_states 结果在加入到 D
nse Layer，最后输出我们需要提取的表示情感的文字的起始位置跟结束位置。
这两个位置信息就是我们需要从原文中提取的词向量的位置。
<pre><code class="highlight-chroma"><span class="highlight
cl">
class BertModel(T
BertPreTrainedModel):

drop out rate,
止过拟合
    dr = 0.1
# hidden state数量
    hs = 2

    def __init__(self,
config, *inputs, **kwargs):
        super().__init_
(config, *inputs, **kwargs)

        self.bert = TF
ertMainLayer(config, name="bert")
        self.concat =
.Concatenate()
        self.dropout
 L.Dropout(self.dr)
        self.qa_outpu
s = L.Dense(

```
            config.nu
_labels,
            kernel_initi
lizer=TruncatedNormal(stddev=config.initializer_range),
            dtype='flo
t32',
            name="qa
outputs")

    @tf.function
    def call(self, inp
ts, **kwargs):
        _, _, hidden_s
ates = self.bert(inputs, **kwargs)

        hidden_states
= self.concat([
            hidden_sta
es[-i] for i in range(1, self.hs+1)
        ])

        hidden_states
= self.dropout(hidden_states, training=kwargs.get("training", False))
        logits = self.
a_outputs(hidden_states)
        start_logits,
nd_logits = tf.split(logits, 2, axis=-1)
        start_logits =
tf.squeeze(start_logits, axis=-1)
        end_logits =
f.squeeze(end_logits, axis=-1)

        return start_l
gits, end_logits
```

定义训练函数

```
def train(model, d
taset, loss_fn, optimizer):

    @tf.function
    def train_step(
odel, inputs, y_true, loss_fn, optimizer):
        with tf.Gradi
ntTape() as tape:
            y_pred =
odel(inputs, training=True)
            loss  = loss
fn(y_true[0], y_pred[0])
            loss += los
_fn(y_true[1], y_pred[1])
            scaled_loss
= optimizer.get_scaled_loss(loss)
```

```
        scaled_gradients = tape.gradient(scaled_loss, model.trainable_variables)
        gradients = optimizer.get_unscaled_gradients(scaled_gradients)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        return loss, y_pred

    epoch_loss = 0.
    for batch_num, sample in enumerate(dataset):
        loss, y_pred = train_step(model, sample[:3], sample[4:6], loss_fn, optimizer)

        epoch_loss = loss

        print(
            f"training .. batch {batch_num+1:03d} : "
            f"train loss {epoch_loss/(batch_num+1):.3f} ",
            end='\r')
```

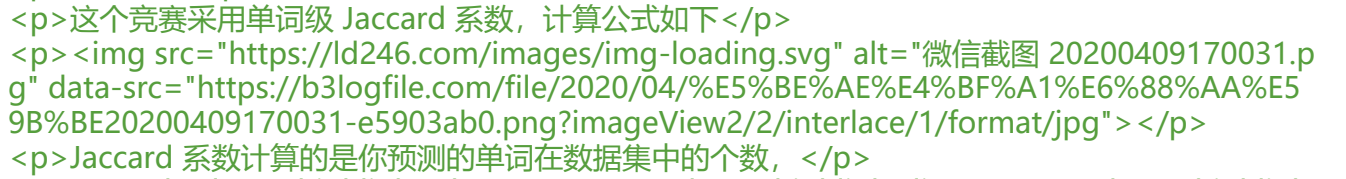
定义预制函数

```
def predict(model, dataset, loss_fn, optimizer):

    @tf.function
    def predict_step(model, inputs):
        return model(inputs)

    def to_numpy(*args):
        out = []
        for arg in arg:
            if arg.dtype == tf.string:
                arg = [s.decode('utf-8') for s in arg.numpy()]
                out.append(arg)
            else:
                arg = arg.numpy()
                out.append(arg)
        return out

```

```
    offset = tf.zeros([0, 128, 2], dtype=tf.dtypes.int32)
    text = tf.zeros([,], dtype=tf.dtypes.string)
    selected_text = f.zeros([0,], dtype=tf.dtypes.string)
    sentiment = tf.eros([0,], dtype=tf.dtypes.string)
    pred_start = tf.eros([0, 128], dtype=tf.dtypes.float32)
    pred_end = tf.zros([0, 128], dtype=tf.dtypes.float32)

    for batch_num, ample in enumerate(dataset):

        print(f"predicing ... batch {batch_num+1:03d}"+" "*20, end='\r')

        y_pred = predict_step(model, sample[:3])

        # add batcho accumulators
        pred_start = t.concat((pred_start, y_pred[0]), axis=0)
        pred_end = tfconcat((pred_end, y_pred[1]), axis=0)
        offset = tf.cocat((offset, sample[3]), axis=0)
        text = tf.conct((text, sample[6]), axis=0)
        selected_text = tf.concat((selected_text, sample[7]), axis=0)
        sentiment = t.concat((sentiment, sample[8]), axis=0)

    pred_start, predend, text, selected_text, sentiment, offset = \
        to_numpy(prd_start, pred_end, text, selected_text, sentiment, offset)

    return pred_star, pred_end, text, selected_text, sentiment, offset
```

判断函数

这个竞赛采用单词级 Jaccard 系数，计算公式如下



Jaccard 系数计算的是你预测的单词在数据集中的个数，

```
def jaccard(str1, str2):
    a = set(str1.low
```

```
r().split())
    b = set(str2.lower().split())
    c = a.intersection(b)
    return float(len(c)) / (len(a) + len(b) - len(c))
```

定义预测结果解码函数

解码函数通过模型预测拿到的 start 和 end 的 index 位置信息，然后和之前拿到的词向量在样句子中的位置进行比较，将这个区间内的所有的单词都提取出来作为我们的预测结果。

```
def decode_prediction(pred_start, pred_end, text, offset, sentiment):

    def decode(pred_start, pred_end, text, offset):

        decoded_text = ""
        for i in range(pred_start, pred_end+1):
            decoded_text += text[offset[i][0]:offset[i][1]]
            if (i+1) &lt; len(offset) and offset[i][1] &lt; offset[i+1][0]:
                decoded_text += " "
        return decoded_text

    decoded_predictions = []
    for i in range(len(text)):
        if sentiment[i] == "neutral" or len(text[i].split()) &lt; 2:
            decoded_text = text[i]
        else:
            idx_start = np.argmax(pred_start[i])
            idx_end = np.argmax(pred_end[i])
            if idx_start &gt; idx_end:
                idx_end = idx_start
            decoded_text = str(decode(idx_start, idx_end, text[i], offset[i]))
            if len(decoded_text) == 0:
                decoded_text
```

```
_text = text[i]
        decoded_predictions.append(decoded_text)

    return decoded_predictions
```

开始训练

将训练数据分成 5 个 folds，每个 folds 训练 5 个 epoch，使用 adam 优化器，learning rate 置成 3e-5，batch size 使用 32。

```
num_folds = 5
num_epochs = 5
batch_size = 32
learning_rate = 3e-5

optimizer =  tf.keras.optimizers.Adam(learning_rate)
optimizer = tf.keras.mixed_precision.experimental.LossScaleOptimizer(
    optimizer, 'dynamic')

config = BertConfig(output_hidden_states=True, num_labels=2)
model = BertModel.from_pretrained(PATH, config=config)

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

kfold = model_selection.KFold(
    n_splits=num_flds, shuffle=True, random_state=42)

test_preds_start = np.zeros((len(test_df), 128), dtype=np.float32)
test_preds_end = np.zeros((len(test_df), 128), dtype=np.float32)

for fold_num, (train_idx, valid_idx) in enumerate(kfold.split(train_df.text)):
    print("\nfold %2d" % (fold_num+1))

# 创建train, valid, test数据集
    train_dataset = tweetDataset.create(
        train_df.iloc[train_idx], batch_size, shuffle_buffer_size=2048)
```

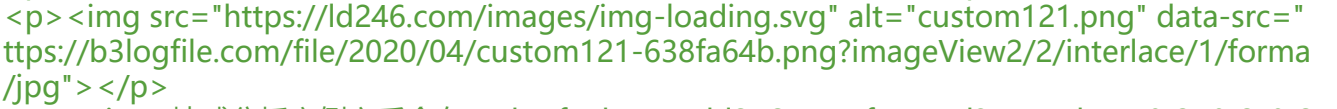
```
    valid_dataset = TweetDataset.create(
        train_df.iloc[valid_idx], batch_size, shuffle_buffer_size=-1)
    test_dataset = TweetDataset.create(
        test_df, batch_size, shuffle_buffer_size=-1)

    best_score = float('-inf')
    for epoch_num in range(num_epochs):
        print("\nepoch %03d" % (epoch_num+1))

        train(model, train_dataset, loss_fn, optimizer)

        pred_start, pred_end, text, selected_text, sentiment, offset = \
            predict(model, valid_dataset, loss_fn, optimizer)

        selected_text_pred = decode_prediction(
            pred_start, pred_end, text, offset, sentiment)
        jaccards = []
        for i in range(len(selected_text)):
            jaccards.append(
                jaccard(selected_text[i], selected_text_pred[i]))

        score = np.mean(jaccards)
        print(f"valid jaccard epoch {epoch_num+1:03d}: {score}"+" "*15)

        if score > best_score:
            best_score = score

    # predict test set
            test_pred_start, test_pred_end, test_text, _, test_sentiment, test_offset = \
                predict(model, test_dataset, loss_fn, optimizer)

    test_preds_start += test_pred_start * 0.2
```

    test_preds_end += test_pred_end * 0.2

# 重置模型，避免OM
    session = tf.compat.v1.get_default_session()
    graph = tf.compat.v1.get_default_graph()
    del session, graph, model
    model = BertModel.from_pretrained(PATH, config=config)
</code></pre>
<h2 id="预测测试数据-并生成提交文件">预测测试数据，并生成提交文件</h2>
<pre><code class="highlight-chroma">selected_text_pred = decode_prediction(
    test_preds_start, test_preds_end, test_text, test_offset, test_sentiment)

def f(selected):
    return " ".join(set(selected.lower().split()))
submission_df.loc[:, 'selected_text'] = selected_text_pred
submission_df['selected_text'] = submission_df['selected_text'].map(f)

submission_df.to_csv("submission.csv", index=False)

</code></pre>
这个方案在提交的时候在 553 个队伍中排名 153 位, 分数为 0.68。

Twitter 情感分析案例之后会在矩池云 Demo 镜像中上线，以直接使用。另矩池云还支持了 Paddle、MindSpore、MegEngine、Jittor 等国产深度学习框架，免安装直接运行。