情感分析案例

实验报告

姓名:王雨静

班级 ： 1期NLP训练营

日期:2021年3月23日

目录

一、案例简介

情感分析旨在挖掘文本中的主观信息，它是自然语言处理中的经典任务。在本次任务中，我们将在影评文本数据集（Rotten Tomato）上进行情感分析，通过实现课堂讲授的模型方法，深刻体会自然语言处理技术在生活中的应用。

同学们需要实现自己的情感分析器，包括特征提取器（可以选择词袋模型、n-gram模型或词向量模型）、简单的线性分类器以及梯度下降函数。随后在数据集上进行训练和验证。我们提供了代码框架，同学们只需补全model.py中的两个函数。

1.1 数据说明

我们使用来自Rotten Tomato的影评文本数据。其中训练集data_rt.train和测试集data_rt.test均包含了3554条影评，每条影评包含了文本和情感标签。示例如下：

+1 visually , 'santa clause 2' is wondrously creative .

其中，+1 表示这条影评蕴涵了正面感情，后面是影评的具体内容。

1.2 数据特征提取

TODO：补全featureExtractor函数

在这个步骤中，同学们需要读取给定的训练和测试数据集，并提取出文本中的特征，输出特征向量。

同学们可以选择选择词袋模型、n-gram模型或词向量模型中的一种，也可以对比三者的表现有何差异。

1.3 训练分类器

TODO：补全learnPredictor函数

我们提供的训练数据集中，每句话的标签在文本之前，其中+1表示这句话蕴涵了正面感情，-1表示这句话蕴涵了负面感情。因此情感分析问题就成为一个分类问题。

我们采用最小化hinge loss的方法训练分类器，假设我们把每条影评文本

x

映射为对应的特征向量

$\phi(x)$

，hinge loss的定义为

$$L(x,y;\mathbf{w})=\max(0,1-\mathbf{w}\cdot\phi(x)y)$$

同学们需要实现一个简单的线性分类器，并推导出相应的梯度下降函数。

1.4 实验与结果分析

在训练集上完成训练后，同学们需要在测试集上测试分类器性能。本小节要求同学们画出训练集上的损失函数下降曲线和测试集的最终结果，并对结果进行分析。

1.5 评分要求

同学们需要提交源代码和实验报告。实验报告中应包含两部分内容：

- 对hinge loss反向传播的理论推导，请写出参数的更新公式。
- 对实验结果的分析，请描述采用的模型结构、模型在训练集上的损失函数下降曲线和测试集的最终结果，并对结果进行分析。分析可以从模型的泛化能力、参数对模型性能的影响以及不同特征的影响等方面进行。

二、Hinge Loss

2.1 hinge loss 求导

Hinge loss的定义：

$$L(x,y;\mathbf{w})=\max(0,1-\mathbf{w}\cdot\phi(x)y)$$

if L == 0: dL/dw = 0

Else: dL/dw = $-\mathbf{w}\cdot\phi(x)y$

Hinge loss的定义（含有bias):

$$L(x,y;\mathbf{w})=\max(0,1-(\mathbf{w}\cdot\phi(x) + \text{bias}) * y)$$

if L == 0: dL/dw = 0, dL/dbias = 0

Else: dL/dw = $-\mathbf{w}\cdot\phi(x)y$, dL/dbias = -y

三、基于bert特征的模型

3.1 特征提取

数据清洗:(详细代码见notebook)

调用nltk库，删除文本中的“stopword”, 并进行词干化（stemming）/词元化（lemmatisation)，最后，只取前20个词返回。储存在‘text_clean’列中。

用预训练过的bert模型提取特征：

(完整代码请看notebook)

```
import time
import torch
import transformers as ppb
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased') #加载预训练模型
model1 = BertModel.from_pretrained("bert-base-uncased")
model = model1.to('cuda') #利用GPU进行加速

#提取train集中第一个文本的特征
text = train_df['text_clean'][0]
encoded_input = tokenizer(text, return_tensors='pt').to('cuda')
output = model(**encoded_input)
feature = output[0][:,0,:].cpu().detach().numpy()#取出需要的特征，768维向量
feature.shape
```

特征提取完毕后生成的是一个sample size * 768的numpy array。

3.2 读取数据和特征

给出x样本的index，是否是测试集，返回其对应的特征向量。

```
def extractFeatures(x, test = True):
    if (test == True): return bert_test[x]
    else: return bert_train[x]
```

从Dataframe中读取数据，返回的每一个example是其对应的（index，label）。

```
def readExamples(df):
    examples = []
    for i in range(len(df)):
      y = df.loc[i, 'sentiment']
      examples.append((i, int(y)))
    print('Read %d examples' % (len(examples)))
    return examples
```

为了适配extractFeatures这个函数，对给定的一些函数也进行了修改（具体见notebook）。

3.3 训练分类器（梯度下降)

累计对随机BATCH_SIZE个样本求loss、梯度后计算平均，乘以学习率对weight和bias进行更新。

可选：每500个epoch梯度自动除以2。（参数可调）

```
def learnPredictor(trainExamples, testExamples, featureExtractor,
numIters, eta):
    losses = []
    weights = None
    # BEGIN_YOUR_CODE
    bias = np.random.rand(1, 1)
    weights = (np.random.rand(1, 768) - 0.5) * 2
    for i in range(0, numIters + 1):
```

```
    #   if i > 1 and i % 500 == 0: #每500个epoch减小一次学习率
    #     eta = eta / 2
      gradient_total = np.zeros((1, 768))
      bias_g_total = np.zeros((1, 1))

      loss_total = 0

      for j in range(BATCH_SIZE):
        x, y = random.choice(trainExamples)
        f_x = featureExtractor(x, test = False)
        loss = max(0, 1 - (np.matmul(weights, f_x) + bias) * y)
        loss_total += loss
        if loss == 0:
          gradient = np.zeros((1, 768))
          bias_g = np.zeros((1, 1))
        else:
          gradient = -y * f_x
          bias_g = -y
        gradient_total += gradient
        bias_g_total += bias_g

      gradient_total /= BATCH_SIZE
      bias_g_total /=BATCH_SIZE
      if i % plot_every == 0:
        losses.append(loss_total / BATCH_SIZE)

      if i % 500 == 0:
        print("iter {}/{}, loss: {}".format(i, numIters, loss_total /
BATCH_SIZE))
      weights = weights - eta * gradient_total
      bias = bias - eta*bias_g_total
```

```
    # END_YOUR_CODE
    losses = np.array(losses).reshape(-1)
    return weights, bias, losses
```

3.4 实验结果

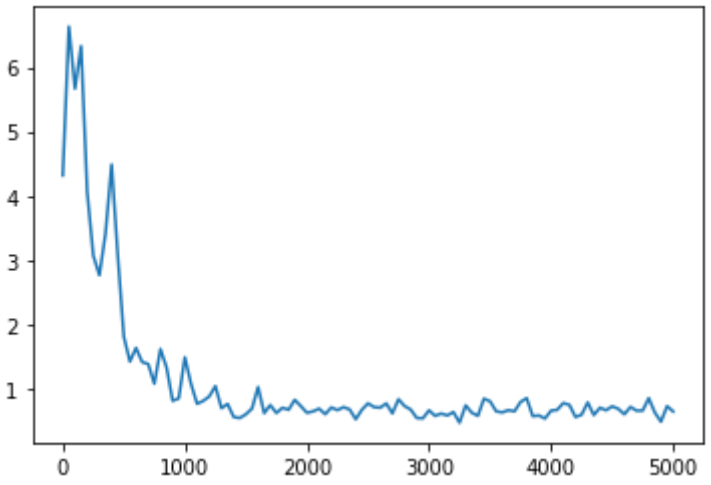

Fig1. 训练集上的损失函数下降曲线

```
train error = 0.23044456949915587, test error = 0.28334271243669107
train auc = 0.8453342327278256, test auc = 0.7920247141875666
```

训练过程中，模型在训练集的loss从6左右下降到0.7左右。前1000个epoch下降较快。

模型在训练集上的准确率为76.96%，测试集的准确率为71.67%。

样本数量均为3554，95%的置信区间约为+-0.7%，所以测试集上的准确率显著得低于训练集上的准确率，这说明模型在训练集上出现了过拟合现象。

3.5 模型参数与过拟合现象

调整学习率eta和BATCH SIZE，对模型的学习情况和过拟合现象进行研究。

为保证学习率统一，这里的实验中没有使用自动减小的学习率。

由于每次初始化的参数具有随机性，抽中的样本质量也有随机性，为减小这些对实验结果的影响，重复实验20次，取平均值。

超参数：

```
BATCH_SIZEs = [1, 4, 16, 64, 256, 1024]

etas = [0.00001, 0.00003, 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3]

numIters = 2000
```

Heatmap: `vmin = 0.27, vmax =0.52`

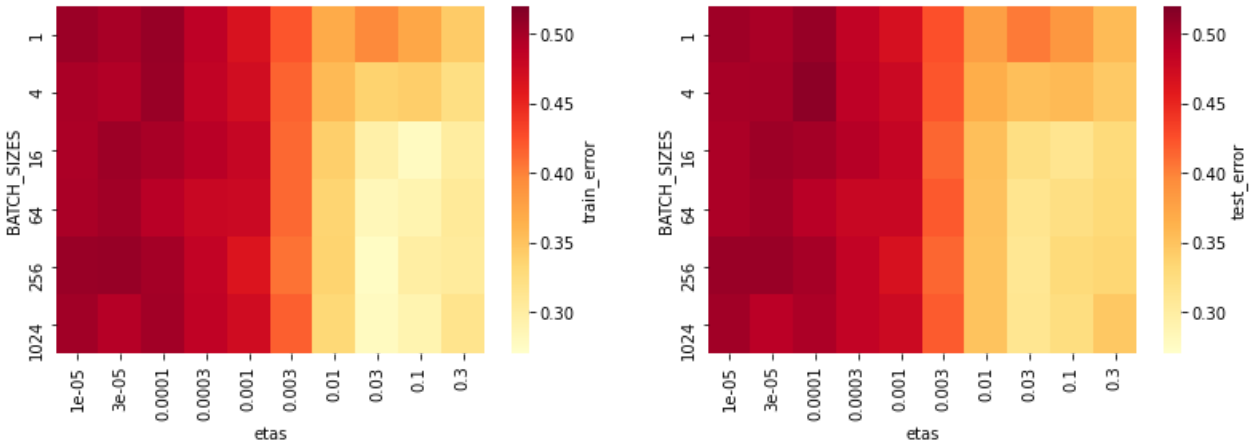

Fig2. 训练集和测试集的平均error

当学习率较小时，模型的loss可能被困在局部极小值里，无法跳出，故无法走向全局最小值；学习率较小时，模型的收敛速度也会比较慢。

因此学习率非常小时，模型的效果可能非常不好，从Fig2中可以发现，当学习率小于0.001时，错误率在0.5左右，和随机猜的效果差不多。

随着学习率的提升，在同样2000个epoch下，模型效果逐渐提升。

但是当学习率太大时，学习速度较快，但也因为步长较大，在全局最优附近，但可能总是跨过最优，无法达到。上图中，eta = 0.3时，模型收敛得没有0.1好。

为了验证上面的猜想，我们取eta = 0.00001, 0.1 和0.3，画一下他们的loss图。

BATCH SIZE选择16。

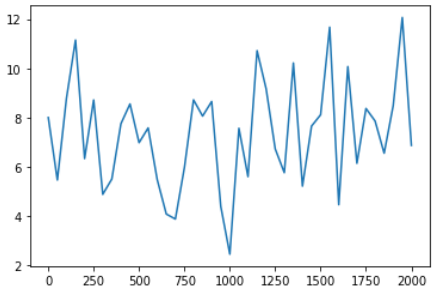

Fig3. eta = 1e-5

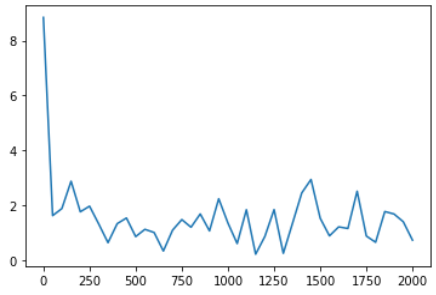

Fig4. eta = 0.1

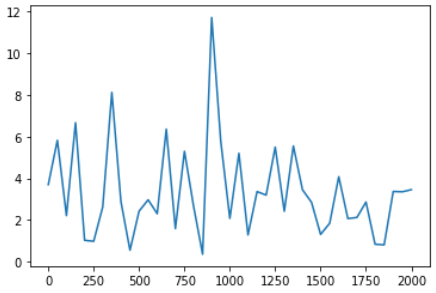

Fig5. eta = 0.3

从Fig3中可以发现，当eta = 0.00001时，loss在6-12范围波动，没有收敛。

从Fig4中可以发现，当eta = 0.1时，模型的loss从8左右下降到1左右，虽然有抖动，但是还是保持在较小的范围内，收敛效果比较好。（但存在过拟合训练集的可能性）

从Fig5中可以发现，当eta = 0.3时，模型的loss收敛到0-6之间后，仍有较大的抖动。

这佐证了我们上面的猜想。

算出train_error和test_error的差值，来考察模型的过拟合现象：

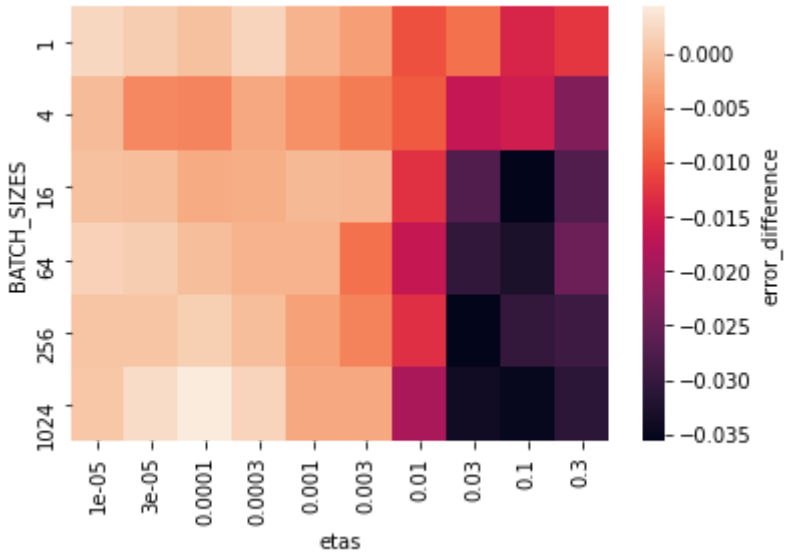

Fig6. 差值heatmap

从上图可以发现，总体来说，batch size越大，eta越大，error difference越大即训练集上的error比测试集上的error小得更多，过拟合现象越严重。（泛化能力越差？）

四、词袋模型

这部分的代码大部分沿用了给定的代码。

4.1 部分代码

```
# 词袋
class bag:
  def __init__(self, name):
    self.name = name          #词袋名
    self.word2index = {}      #词转index
    self.word2count = {}      #每个词的出现次数
    self.index2word = {} #index转词
    self.n_words = 0 # Count SOS and EOS

  def index_words(self, sentence):
    for word in sentence.split(' '):
        self.index_word(word)

  def index_word(self, word):
    if word not in self.word2index:
        self.word2index[word] = self.n_words #该词的index
        self.word2count[word] = 1    #初始化count为1
        self.index2word[self.n_words] = word #更新index2word
        self.n_words += 1
    else:
        self.word2count[word] += 1



bag_all = bag('all')
bag_pos = bag('pos')
bag_neg = bag('neg')
```

```
for i in range(len(train_df)):
  label = train_df.loc[i, 'sentiment']
  sentence = train_df.loc[i, 'text_clean']
  bag_all.index_words(sentence)
  if label == 1: bag_pos.index_words(sentence)
  elif label == -1: bag_neg.index_words(sentence)
  else: print('error')

def extractFeatures(x):
    vector = np.zeros(bag_all.n_words + 1)
    vector[bag_all.n_words] = 1 #常数项
    for word in sentence.split(' '):
      try:
        index = bag_all.word2index[word]
      except:
        continue
      # vector[index] = 1 #+=1
      vector[index] += 1

    return vector
def learnPredictor(trainExamples, testExamples, featureExtractor,
numIters, eta):
    weights = None
    losses = []

    weights = (np.random.rand(1, bag_all.n_words + 1) - 0.5) * 2
    for i in range(numIters):
      if i > 1 and i % 500 == 0: #每500个epoch减小一次学习率
        eta = eta / 2
      gradient_total = np.zeros((1, bag_all.n_words + 1))
```

```
        loss_total = 0

        for j in range(BATCH_SIZE):
          x, y = random.choice(trainExamples)
          f_x = featureExtractor(x)
          loss = max(0, 1 - np.matmul(weights, f_x)* y)
          loss_total += loss
          if loss == 0:
            gradient = np.zeros((1, bag_all.n_words + 1))
          else:
            gradient = -y * f_x
          gradient_total += gradient

        gradient_total /= BATCH_SIZE
        losses.append(loss_total / BATCH_SIZE)

        if i % 50 == 0:
          print("iter {}/{}, loss: {}".format(i, numIters, loss_total /
BATCH_SIZE))
        weights = weights - eta * gradient_total

    losses = np.array(losses).reshape(-1)
    # END_YOUR_CODE
    return weights, losses
```

4.2 实验结果

超参数:

```
numIters = 500
eta = 0.3
BATCH_SIZE = 256
```

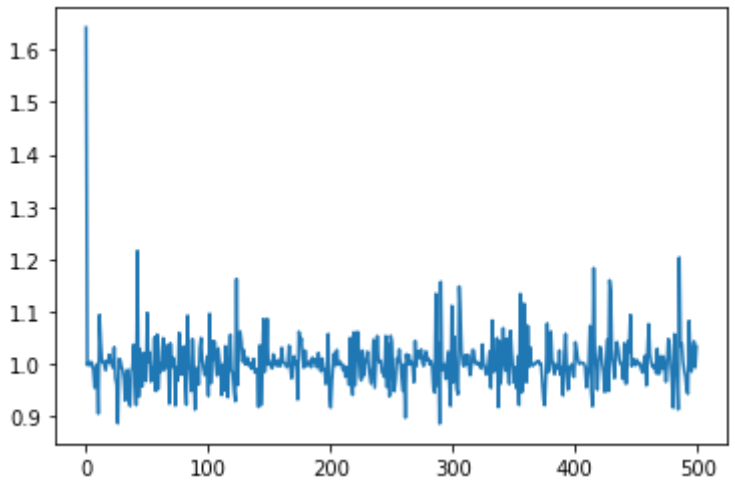

Fig7. 词袋模型loss

```
train error = 0.501969611705121, test error = 0.5073157006190209
train auc = 0.5, test auc = 0.5
```

虽然loss下降了，但是模型预测的准确率并没有提升。

可能是特征向量太稀疏，且特征向量对句子的表示并不好造成的。

五、基于N_gram (tfidf)的模型

调用了sklearn库的TfidfVectorizer提取特征，并使用卡方检验选出最好的200个特征。

ngram_range = (1, 3)

模型部分和bert类似。

5.1 寻找最佳学习率

```
if i > 1 and i % 200 == 0: #每200个epoch增加一次学习率 #寻找最佳学习率
  eta = eta  * 5 #从0.00001开始；或者eta = eta + 0.1从0.1开始
```

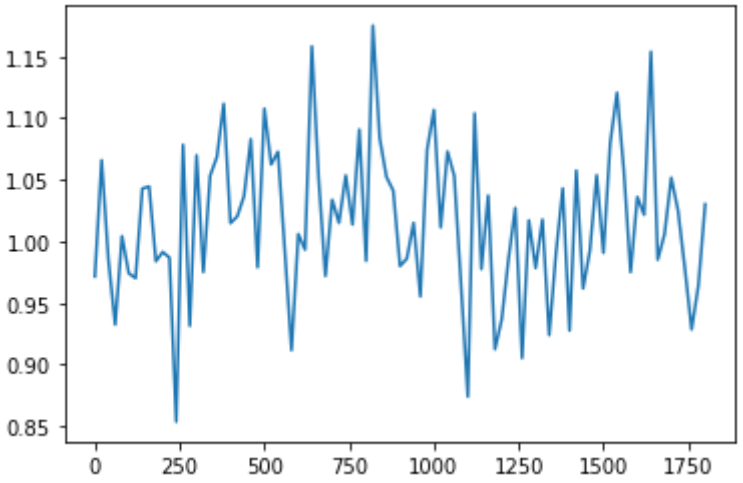

Fig8. 寻找最佳学习率

然而不论怎么改变学习率，loss都在1左右抖动，并没有收敛

寻找最佳学习率失败orz

5.2 模型效果

```
BATCH_SIZE = 64

numIters = 2000

eta = 0.8
```

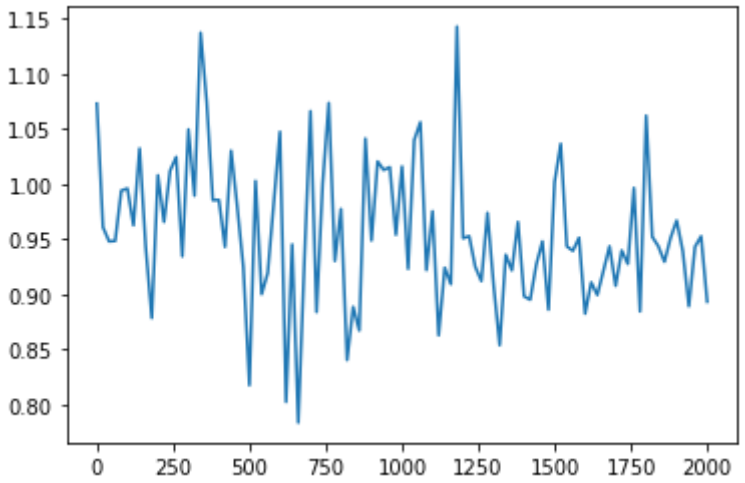

Fig9. tfidf loss

```
train error = 0.3719752391671356, test error = 0.4048958919527293

train auc = 0.6759901256618783, test auc = 0.6227424753401353
```

虽然loss并没有明显的下降趋势，但是模型有60%左右的准确率，AUC值也达到了0.6以上，说明**模型效果**还是比随机猜要好一些。

5.3 特征分析

将weight最大和最小的10个找出来:

```
performance        1.9325477352597422

thoughtful   1.9391378826622891

heart        1.9898153752508092

best   2.0983735105807773

moving       2.203389553488226

solid        2.22608579015358

fun    2.32905075658139

life   2.592618665204932
```

```
work   2.984103849043693
film   3.6028638247385896


bad    -3.0762600898761243
really      -2.3100782901552104
title       -2.215058765667623
dull   -2.205218108925349
tedious     -1.855107678072646
flat   -1.8315368326233137
cliche      -1.7621734855476483
pretentious       -1.664090040406095
boring      -1.6005579018104779
silly       -1.5360737011653947
```

可以发现，weight比较大的词（使得句子最终得分更高），例如“thoughtful”、“best” “solid”等，大多代表着对电影的积极评价。

weight比较小（负值，绝对值大）的词（使得句子最终得分更低），例如“bad”、“tedious” “boring”等，大多代表着对电影的负面评价。

这与直觉/模型的原理是相符的。

六、其他模型

6.1 MLP Classifier

使用bert提取出的特征：

```
train score: 0.765053460889139
test score: 0.7352279122115926
```

使用tfidf特征：

```
train score: 0.6339335959482274
test score: 0.5996060776589758
```

6.2 SGD Classifier

使用bert提取出的特征：

```
train score: 0.8320202588632527
test score: 0.7031513787281936
```

使用tfidf特征：

```
train score: 0.5576814856499719
test score: 0.5841305571187394
```

在没有精确调参的情况下，所得到的的结果差距不是很大。和我们实验的结果也类似。

这可能是因为都是线性模型，且提取出的特征的质量限制了模型效果的上限。

（结论：还是特征更重要！）

七、小结

在这次实验中实践了用不同方法提取文本特征，并手写了随机梯度下降的函数，探索了参数、特征等对模型效果的影响。

不意外的，bert优于tfidf优于词袋。

疑惑点：

同一模型，为什么类似的loss，但模型的效果可能有很大差距？

Loss不收敛，但模型效果却可能在提升，这是为什么？

怎么寻找合适的学习率？（从很小如e-5开始试，可以试到1）

模型改进：

可能可以加入正则化项来改善过拟合问题。