**实验一 电影评论情感分析**

1. 实验目的

本实验主要基于TensorFlow深度学习框架建立Text-CNN模型，实现对中文电影评论的情感分类。

1. 实验要求
2. 理解文本分类模型Text-CNN的架构和原理。

算法具体原理可阅读：Kim Y .2014--[《Convolutional Neural Networks for Sentence Classification》](https://arxiv.org/pdf/1408.5882.pdf)一文。

1. 编码实现模型并完成模型训练。
2. 能够使用训练所得模型完成对电影评论的分类。
3. 实验原理

Text-CNN和传统的CNN结构类似，具有词嵌入层、卷积层、池化层和全连接层的四层结构。

其中，Text-CNN的词嵌入层使用二维矩阵来表示长文本。词嵌入将输入文本的每个词语通过空间映射，将独热表示（One-Hot Representation）转换成分布式表示（Distributed Representation），进而可以使用低维的词向量来表示每一个词语。经过词嵌入，每个单词具有相同长度的词向量表示。将各个词语的向量表示连起来便可以得到二维矩阵。得到词向量的方式有多种，常用的是Word2vec方法。若使用预训练好的词向量，在训练模型的时候可以选择更新或不更新词向量，分别对应嵌入层状态为Non-static和Static。

Text-CNN的卷积层是主要部分，卷积核的宽度等于词向量的维度，经卷积后可以提取文本的特征向量。和在图像领域应用类似，Text-CNN可以设置多个卷积核以提取文本的多层特征，长度为N的卷积核可以提取文本中的N-gram特征。

Text-CNN的池化层一般采取Max-over-time pooling，输出最大值，从而判断词嵌入中是否含N-gram。

Text-CNN的全连接层采用了Dropout算法以防过拟合，并使用Softmax函数输出各个类别的概率。

1. 实验所用工具及数据集
2. 主要工具

Python-3.5+、TensorFlow-1.3.0、Numpy-1.13.1、jieba-0.39

1. 数据集
2. 训练集。包含2W条左右中文电影评论，其中正负向评论各1W条左右。
3. 验证集。包含6K条左右中文电影评论，其中正负向评论各3K条左右。
4. 测试集。包含360条左右中文电影评论，其中正负向评论各180条左右。
5. 预训练词向量。中文维基百科词向量word2vec。
6. 实验步骤与方法
7. 加载本实验所有函数库

|  |
| --- |
| import os  import time  import numpy as np  import tensorflow as tf  from datetime import timedelta  from collections import Counter  import tensorflow.contrib.keras as kr  import jieba as jb  from sklearn import metrics  import matplotlib.pyplot as plt  # 若使用GPU训练时可使用下面语句指定编号  # os.environ["CUDA\_VISIBLE\_DEVICES"]="7" |

1. 数据预处理
2. cat\_to\_id(): 分类类别以及id对应词典{pos:0, neg:1};

|  |
| --- |
| def cat\_to\_id(classes=None):  """  :param classes: 分类标签；默认为0--pos 1--neg  :return: {分类标签：id}  """  if not classes:  classes = ['0', '1']  cat2id = {cat: idx for (idx, cat) in enumerate(classes)}  return classes, cat2id |

1. build\_word2id(): 构建词汇表并存储，形如{word: id};

|  |
| --- |
| # only one start  def build\_word2id(file):  """  :param file: word2id保存地址  :return: None  """  word2id = {'\_PAD\_': 0}  path = ['./data/train.txt', './data/validation.txt']  print(path)  for \_path in path:  with open(\_path, encoding='utf-8') as f:  for line in f.readlines():  sp = line.strip().split()  for word in sp[1:]:  if word not in word2id.keys():  word2id[word] = len(word2id)  with open(file, 'w', encoding='utf-8') as f:  for w in word2id:  f.write(w+'\t')  f.write(str(word2id[w]))  f.write('\n') |

1. load\_word2id(): 加载上述构建的词汇表;

|  |
| --- |
| def load\_word2id(path):  """  :param path: word\_to\_id词汇表路径  :return: word\_to\_id:{word: id}  """  word\_to\_id = {}  with open(path, encoding='utf-8') as f:  for line in f.readlines():  sp = line.strip().split()  word = sp[0]  idx = int(sp[1])  if word not in word\_to\_id:  word\_to\_id[word] = idx  return word\_to\_id |

1. build\_word2vec(): 基于预训练好的word2vec构建训练语料中所含词语的word2vec;

|  |
| --- |
| def build\_word2vec(fname, word2id, save\_to\_path=None):  """  :param fname: 预训练的word2vec.  :param word2id: 语料文本中包含的词汇集.  :param save\_to\_path: 保存训练语料库中的词组对应的word2vec到本地  :return: 语料文本中词汇集对应的word2vec向量{id: word2vec}.  """  import gensim  n\_words = max(word2id.values()) + 1  model = gensim.models.KeyedVectors.load\_word2vec\_format(fname, binary=True)  word\_vecs = np.array(np.random.uniform(-1., 1., [n\_words, model.vector\_size]))  for word in word2id.keys():  try:  word\_vecs[word2id[word]] = model[word]  except KeyError:  pass  if save\_to\_path:  with open(save\_to\_path, 'w', encoding='utf-8') as f:  for vec in word\_vecs:  vec = [str(w) for w in vec]  f.write(' '.join(vec))  f.write('\n')  return word\_vecs |

1. load\_corpus\_word2vec(): 加载上述构建的word2ve;

|  |
| --- |
| def load\_corpus\_word2vec(path):  """加载语料库word2vec词向量,相对wiki词向量相对较小"""  word2vec = []  with open(path, encoding='utf-8') as f:  for line in f.readlines():  sp = [float(w) for w in line.strip().split()]  word2vec.append(sp)  return np.asarray(word2vec) |

1. load\_corpus(): 加载语料库：train/dev/test;

|  |
| --- |
| def load\_corpus(path, word2id, max\_sen\_len=70):  """  :param path: 样本语料库的文件  :return: 文本内容contents，以及分类标签labels(onehot形式)  """  \_, cat2id = cat\_to\_id()  contents, labels = [], []  with open(path, encoding='utf-8') as f:  for line in f.readlines():  sp = line.strip().split()  label = sp[0]  content = [word2id.get(w, 0) for w in sp[1:]]  content = content[:max\_sen\_len]  if len(content) < max\_sen\_len:  content += [word2id['\_PAD\_']] \* (max\_sen\_len - len(content))  labels.append(label)  contents.append(content)  counter = Counter(labels)  print('总样本数为：%d' % (len(labels)))  print('各个类别样本数如下：')  for w in counter:  print(w, counter[w])  contents = np.asarray(contents)  labels = [cat2id[l] for l in labels]  labels = kr.utils.to\_categorical(labels, len(cat2id))  return contents, labels |

1. batch\_index(): 生成批处理id序列。

|  |
| --- |
| def batch\_index(length, batch\_size, is\_shuffle=True):  """  生成批处理样本序列id.  :param length: 样本总数  :param batch\_size: 批处理大小  :param is\_shuffle: 是否打乱样本顺序  :return:  """  index = [idx for idx in range(length)]  if is\_shuffle:  np.random.shuffle(index)  for i in range(int(np.ceil(length / batch\_size))):  yield index[i \* batch\_size:(i + 1) \* batch\_size] |

经过数据预处理，数据的格式如下：

x: [1434, 5454, 2323, ..., 0, 0, 0]

y: [0, 1]

x为构成一条评论的词所对应的id。 y为onehot编码: pos-[1, 0], neg-[0, 1]

1. 建立Text-CNN模型
2. 使用TensorFlow框架完成Text-CNN模型的建立
3. 配置模型相关参数，在COINFIG类中完成

|  |
| --- |
| class CONFIG():  update\_w2v = True # 是否在训练中更新w2v  vocab\_size = 59290 # 词汇量，与word2id中的词汇量一致  n\_class = 2 # 分类数：分别为pos和neg  max\_sen\_len = 75 # 句子最大长度  embedding\_dim = 50 # 词向量维度  batch\_size = 100 # 批处理尺寸  n\_hidden = 256 # 隐藏层节点数  n\_epoch = 10 # 训练迭代周期，即遍历整个训练样本的次数  opt = 'adam' # 训练优化器：adam或者adadelta  learning\_rate = 0.001 # 学习率；若opt=‘adadelta'，则不需要定义学习率  drop\_keep\_prob = 0.5 # dropout层，参数keep的比例  num\_filters = 256 # 卷积层filter的数量  kernel\_size = 4 # 卷积核的尺寸；nlp任务中通常选择2,3,4,5  print\_per\_batch = 100 # 训练过程中,每100词batch迭代，打印训练信息  save\_dir = './checkpoints/' # 训练模型保存的地址  train\_path = './data/train.txt'  dev\_path = './data/validation.txt'  test\_path = './data/test.txt'  word2id\_path = './data/word\_to\_id.txt'  pre\_word2vec\_path = './data/wiki\_word2vec\_50.bin'  corpus\_word2vec\_path = './data/corpus\_word2vec.txt'  # 定义时间函数，供计算模型迭代时间使用  def time\_diff(start\_time):  """当前距初始时间已花费的时间"""  end\_time = time.time()  diff = end\_time - start\_time  return timedelta(seconds=int(round(diff)))  # 建立Text-CNN模型  class TextCNN(object):  def \_\_init\_\_(self, config, embeddings=None):  self.update\_w2v = config.update\_w2v  self.vocab\_size = config.vocab\_size  self.n\_class = config.n\_class  self.max\_sen\_len= config.max\_sen\_len  self.embedding\_dim = config.embedding\_dim  self.batch\_size = config.batch\_size  self.num\_filters = config.num\_filters  self.kernel\_size = config.kernel\_size  self.n\_hidden = config.n\_hidden  self.n\_epoch = config.n\_epoch  self.opt = config.opt  self.learning\_rate = config.learning\_rate  self.drop\_keep\_prob = config.drop\_keep\_prob  self.x = tf.placeholder(tf.int32, [None, self.max\_sen\_len], name='x')  self.y = tf.placeholder(tf.int32, [None, self.n\_class], name='y')  if embeddings is not None:  self.word\_embeddings = tf.Variable(embeddings, dtype=tf.float32, trainable=self.update\_w2v)  else:  self.word\_embeddings = tf.Variable(  tf.zeros([self.vocab\_size, self.embedding\_dim]),  dtype=tf.float32,  trainable=self.update\_w2v)  self.build()  def cnn(self):  """  :param mode:默认为None，主要调节dropout操作对训练和预测带来的差异。  :return: 未经softmax变换的fully-connected输出结果  """  inputs = self.add\_embeddings()  with tf.name\_scope("cnn"):  # CNN layer  conv = tf.layers.conv1d(inputs, self.num\_filters, self.kernel\_size, name='conv')  # global max pooling layer  gmp = tf.reduce\_max(conv, reduction\_indices=[1], name='gmp')  # dropout 卷积层后加dropout效果太差  # gmp = tf.contrib.layers.dropout(gmp, self.drop\_keep\_prob)  with tf.name\_scope("score"):  # fully-connected  fc = tf.layers.dense(gmp, self.n\_hidden, name='fc1')  # dropout  fc = tf.contrib.layers.dropout(fc, self.drop\_keep\_prob)  # nonlinear  fc = tf.nn.relu(fc)  # fully-connected  pred = tf.layers.dense(fc, self.n\_class, name='fc2')  return pred  def add\_embeddings(self):  inputs = tf.nn.embedding\_lookup(self.word\_embeddings, self.x)  return inputs  def add\_loss(self, pred):  cost = tf.nn.softmax\_cross\_entropy\_with\_logits(logits=pred, labels=self.y)  cost = tf.reduce\_mean(cost)  return cost  def add\_optimizer(self, loss):  if self.opt == 'adadelta':  optimizer = tf.train.AdadeltaOptimizer(learning\_rate=1.0, rho=0.95, epsilon=1e-6)  else:  optimizer = tf.train.AdamOptimizer(self.learning\_rate)  opt = optimizer.minimize(loss)  return opt  def add\_accuracy(self, pred):  correct\_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(self.y, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))  return accuracy  def get\_batches(self, x, y=None, batch\_size=100, is\_shuffle=True):  for index in batch\_index(len(x), batch\_size, is\_shuffle=is\_shuffle):  n = len(index)  feed\_dict = {  self.x: x[index]  }  if y is not None:  feed\_dict[self.y] = y[index]  yield feed\_dict, n  def build(self):  self.pred = self.cnn()  self.loss = self.add\_loss(self.pred)  self.accuracy = self.add\_accuracy(self.pred)  self.optimizer = self.add\_optimizer(self.loss)  def train\_on\_batch(self, sess, feed):  \_, \_loss, \_acc = sess.run([self.optimizer, self.loss, self.accuracy], feed\_dict=feed)  return \_loss, \_acc  def test\_on\_batch(self, sess, feed):  \_loss, \_acc = sess.run([self.loss, self.accuracy], feed\_dict=feed)  return \_loss, \_acc  def predict\_on\_batch(self, sess, feed, prob=True):  result = tf.argmax(self.pred, 1)  if prob:  result = tf.nn.softmax(logits=self.pred, dim=1)  res = sess.run(result, feed\_dict=feed)  return res  def predict(self, sess, x, prob=False):  yhat = []  for \_feed, \_ in self.get\_batches(x, batch\_size=self.batch\_size, is\_shuffle=False):  \_yhat = self.predict\_on\_batch(sess, \_feed, prob)  yhat += \_yhat.tolist()  # yhat.append(\_yhat)  return np.array(yhat)  def evaluate(self, sess, x, y):  """评估在某一数据集上的准确率和损失"""  num = len(x)  total\_loss, total\_acc = 0., 0.  for \_feed, \_n in self.get\_batches(x, y, batch\_size=self.batch\_size):  loss, acc = self.test\_on\_batch(sess, \_feed)  total\_loss += loss \* \_n  total\_acc += acc \* \_n  return total\_loss / num, total\_acc / num  def fit(self, sess, x\_train, y\_train, x\_dev, y\_dev, save\_dir=None, print\_per\_batch=100):  saver = tf.train.Saver()  if save\_dir:  if not os.path.exists(save\_dir):  os.makedirs(save\_dir)  sess.run(tf.global\_variables\_initializer())  print('Training and evaluating...')  # 存储准确率  cnn\_train\_accuracy = []  cnn\_val\_accuracy = []  start\_time = time.time()  total\_batch = 0 # 总批次  best\_acc\_dev = 0.0 # 最佳验证集准确率  last\_improved = 0 # 记录上次提升批次  require\_improvement = 500 # 如果超过500轮模型效果未提升，提前结束训练  flags = False  for epoch in range(self.n\_epoch):  print('Epoch:', epoch + 1)  for train\_feed, train\_n in self.get\_batches(x\_train, y\_train, batch\_size=self.batch\_size):  loss\_train, acc\_train = self.train\_on\_batch(sess, train\_feed)  loss\_dev, acc\_dev = self.evaluate(sess, x\_dev, y\_dev)  if total\_batch % print\_per\_batch == 0:  if acc\_dev > best\_acc\_dev:  # 保存在验证集上性能最好的模型  best\_acc\_dev = acc\_dev  last\_improved = total\_batch  if save\_dir:  saver.save(sess=sess, save\_path=os.path.join(save\_dir, 'sa-model'))  improved\_str = '\*'  else:  improved\_str = ''  time\_dif = time\_diff(start\_time)  msg = 'Iter: {0:>6}, Train Loss: {1:>6.2}, Train Acc: {2:>7.2%},' + \  ' Val Loss: {3:>6.2}, Val Acc: {4:>7.2%}, Time: {5} {6}'  print(msg.format(total\_batch, loss\_train, acc\_train, loss\_dev, acc\_dev, time\_dif, improved\_str))  total\_batch += 1  if total\_batch - last\_improved > require\_improvement:  print('No optimization for a long time, auto-stopping...')  flags = True  break  if flags:  break  plt.plot(cnn\_train\_accuracy)  plt.plot(cnn\_val\_accuracy)  plt.ylim(ymin=0.5, ymax=1.01)  plt.title("The accuracy of CNN model")  plt.legend(["train", "val"]) |

1. 模型训练与验证

使用训练集和验证集完成模型训练、验证。返回训练、验证损失和准确率。

|  |
| --- |
| # 模型训练与验证  def train():  config = CONFIG()  print('加载word2id===========================')  word2id = load\_word2id(config.word2id\_path)  print(‘加载word2vec==========================')  word2vec = load\_corpus\_word2vec(config.corpus\_word2vec\_path)  print('加载train语料库========================')  x\_tr, y\_tr = load\_corpus(config.train\_path, word2id, max\_sen\_len=config.max\_sen\_len)  print('加载Validation语料库==========================')  x\_val, y\_val = load\_corpus(config.dev\_path, word2id, max\_sen\_len=config.max\_sen\_len)  print('训练模型===============================')  tc = TextCNN(CONFIG, embeddings=word2vec)  with tf.Session() as sess:  init\_op = tf.global\_variables\_initializer()  sess.run(init\_op)  tc.fit(sess, x\_tr, y\_tr, x\_val, y\_val, config.save\_dir, config.print\_per\_batch) |

1. 模型测试

使用测试集完成模型的测试。通过准确率、召回率、F1-分数、混淆矩阵指标来评估模型的性能。

|  |
| --- |
| # 模型测试  def test():  config = CONFIG()  print('加载word2id===========================')  word2id = load\_word2id(config.word2id\_path)  config.vocab\_size = len(word2id)  print('加载test语料库=========================')  x, y = load\_corpus(config.test\_path, word2id, max\_sen\_len=config.max\_sen\_len)  # x, y = x[:10], y[:10]  model = TextCNN(config)  with tf.Session() as sess:  init\_op = tf.global\_variables\_initializer()  sess.run(init\_op)  saver = tf.train.Saver()  ckpt = tf.train.get\_checkpoint\_state(config.save\_dir)  if ckpt and ckpt.model\_checkpoint\_path:  saver.restore(sess, ckpt.model\_checkpoint\_path)  yhat = model.predict(sess, x)  cat, cat2id = cat\_to\_id()  y\_cls = np.argmax(y, 1)  # 评估  print("Precision, Recall and F1-Score...")  print(metrics.classification\_report(y\_cls, yhat, target\_names=cat))  # 混淆矩阵  print("Confusion Matrix...")  cm = metrics.confusion\_matrix(y\_cls, yhat)  print(cm) |

1. 预测

使用predict函数完成电影评论的情感分析。

|  |
| --- |
| '''  预测  '''  def sent\_to\_id(inputs):  """  将语句进行分词，然后将词语转换为word\_to\_id中的id编码  :param inputs: 句子：列表的形式  :return: 用id表征的语句  """  sentences = []  cut\_sents = [jb.cut(w) for w in inputs]  config = CONFIG()  word2id = load\_word2id(config.word2id\_path)  for cut\_sent in cut\_sents:  sentence = [word2id.get(w, 0) for w in cut\_sent]  sentence = sentence[:config.max\_sen\_len]  if len(sentence) < config.max\_sen\_len:  sentence += [word2id['\_PAD\_']] \* (config.max\_sen\_len - len(sentence))  sentences.append(sentence)  return np.asarray(sentences)  def predict(x, label=False, prob=False):  """  :param x: 语句列表  :param label: 是否以分类标签的形式：posn或eg输出。默认为：0/1  :param prob: 是否以概率的形式输出。  :return: 情感预测结果  """  if label and prob:  raise Exception("label和prob两个参数不能同时为True!")  x = sent\_to\_id(x)  config = CONFIG()  model = TextCNN(config)  with tf.Session() as sess:  init\_op = tf.global\_variables\_initializer()  sess.run(init\_op)  saver = tf.train.Saver()  ckpt = tf.train.get\_checkpoint\_state(config.save\_dir)  if ckpt and ckpt.model\_checkpoint\_path:  saver.restore(sess, ckpt.model\_checkpoint\_path)  y = model.predict(sess, x, prob=prob)  if label:  cat, \_ = cat\_to\_id()  y = [cat[w] for w in y.tolist()]  return y  # 使用训练所得模型进行电影评论分析 label :0--pos /1--neg  tf.reset\_default\_graph()  test = ['完成度很高的公路喜剧片，亮点在于人物塑造完整和细节使用精彩，剧作与表演堪称杰出。','爆米花电影，特效不错，可以一看。但也有几处硬伤']  print(predict(test, label=False, prob=True)) |