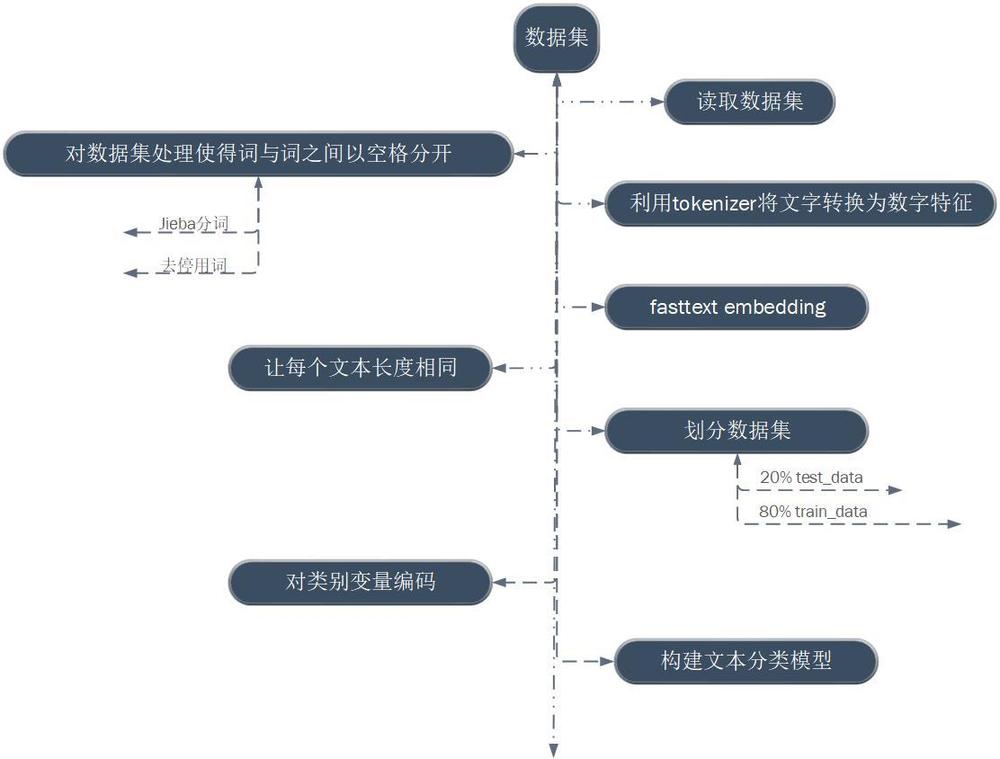
* 卷积运算的定义、动机（稀疏权重、参数共享、等变表示）。一维卷积运算和二维卷积运算。
* 池化运算的定义、种类（最大池化、平均池化等）、动机。
* Text-CNN的原理。
* 利用Text-CNN模型来进行文本分类。

文本分类数据集处理流程如下：



jiebe分词，去停用词

读数据：

def read\_file(path):

with open(path, 'r', encoding="UTF-8") as f:

data = []

labels = []

for line in f:

if line.split('\t')[1] == '':

continue

data.append(line.split('\t')[0])

labels.append(line.split('\t')[1])

return data, labels

# 读文件

data, labels = read\_file('E:/task6/merge.txt')

1. 利用tokenizer将文字转换为数字特征

* 将文字转换为数字特征要将整个数据集放入，这样保证训练集、验证集、测试集文字编码一致
* 其中同时返回对应单词和数字的映射关系，以便在转化词向量过程中获取词向量列表

def get\_tokenizer(data):

tokenizer = Tokenizer(num\_words=None)

tokenizer.fit\_on\_texts(data)

text\_seq = tokenizer.texts\_to\_sequences(data)

# 对应的单词和数字的映射关系

word\_index = tokenizer.word\_index

index\_word = tokenizer.index\_word

return word\_index, index\_word, text\_seq

# 利用tokenizer将文字转换为数字特征

word\_index, index\_word, X\_train\_text\_seq = get\_tokenizer(data)

3. FastText Embedding

def get\_fasttext\_voc(data, word\_index):

'''

利用fasttext获取词向量

'''

fasttext\_model = FastText([data],

size=FASTEXT\_SIZE, # 需要学习的嵌入大小(默认为100)

window=3, # 上下文窗口大小(默认5)

min\_count=1, # 忽略出现次数低于此值的单词(默认值5)

iter=10, # epoch(默认5)

min\_n = 3, # char ngram的最小长度(默认值3)

max\_n = 6, # char ngram的最大长度(默认值6)

word\_ngrams = 0) # 如果为1，使用子单词(n-grams)信息丰富单词向量。如果是0，这就相当于Word2Vec

# 获取词向量列表

wordEmbedding = np.zeros((len(word\_index) + 1, FASTEXT\_SIZE))

for word, i in word\_index.items():

if word in fasttext\_model:

wordEmbedding[i] = fasttext\_model[word]

return wordEmbedding

# fasttext embedding

wordEmbedding = get\_fasttext\_voc(data, word\_index)

# 让每个文本长度相同

X\_train\_text\_seq = pad\_sequences(X\_train\_text\_seq, maxlen=LEN\_WORDS, padding='post', truncating='post')

# 划分数据集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train\_text\_seq,

labels,

test\_size = 0.2,

random\_state=33)

对类别变量编码

def get\_label\_num(data):

data = [i.replace('\n', '') for i in data]

y\_labels = list(set(data))

le = preprocessing.LabelEncoder()

le.fit(y\_labels)

num\_labels = len(y\_labels)

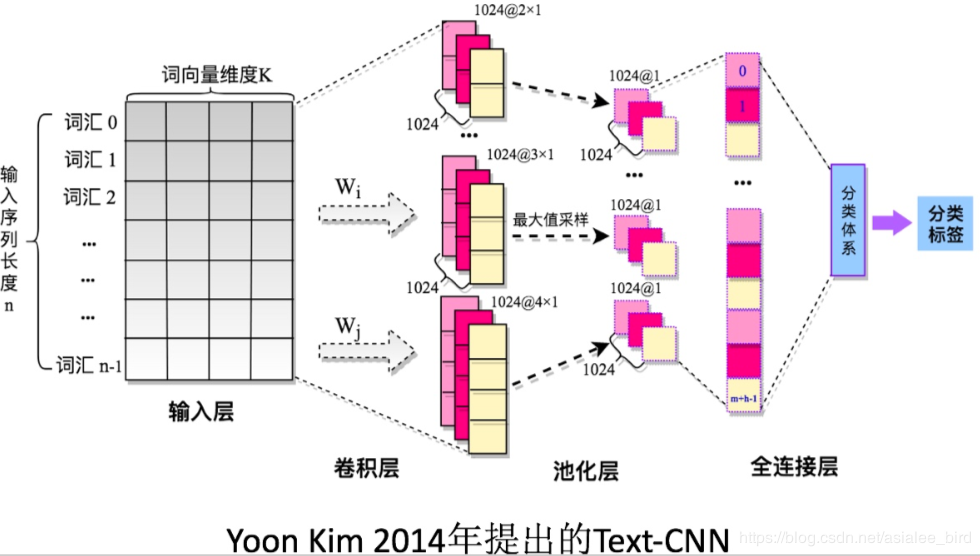
data\_labels = to\_categorical([le.transform([x])[0] for x in data], num\_labels)

return data\_labels

# 对类别变量编码

y\_train\_label = get\_label\_num(y\_train)

y\_test\_label = get\_label\_num(y\_test)



假设我们有一些句子需要对其进行分类。句子中每个词是由n维词向量组成的，也就是说输入矩阵大小为m\*n，其中m为句子长度。CNN需要对输入样本进行卷积操作，对于文本数据，filter不再横向滑动，仅仅是向下移动，有点类似于N-gram在提取词与词间的局部相关性。图中共有三种步长策略，分别是2,3,4，每个步长都有两个filter（实际训练时filter数量会很多）。在不同词窗上应用不同filter，最终得到6个卷积后的向量。然后对每一个向量进行最大化池化操作并拼接各个池化值，最终得到这个句子的特征表示，将这个句子向量丢给分类器进行分类，至此完成整个流程。

（1）嵌入层（Embedding Layer）

通过一个隐藏层, 将 one-hot 编码的词投影到一个低维空间中，本质上是特征提取器，在指定维度中编码语义特征。 这样, 语义相近的词, 它们的欧氏距离或余弦距离也比较近。（作者使用的单词向量是预训练的，方法为fasttext得到的单词向量，当然也可以使用word2vec和GloVe方法训练得到的单词向量）。

（2）卷积层（Convolution Laye）

在处理图像数据时，CNN使用的卷积核的宽度和高度的一样的，但是在text-CNN中，卷积核的宽度是与词向量的维度一致！这是因为我们输入的每一行向量代表一个词，在抽取特征的过程中，词做为文本的最小粒度。而高度和CNN一样，可以自行设置（通常取值2,3,4,5），高度就类似于n-gram了。由于我们的输入是一个句子，句子中相邻的词之间关联性很高，因此，当我们用卷积核进行卷积时，不仅考虑了词义而且考虑了词序及其上下文（类似于skip-gram和CBOW模型的思想）。

（3）池化层（Pooling Layer）

因为在卷积层过程中我们使用了不同高度的卷积核，使得我们通过卷积层后得到的向量维度会不一致，所以在池化层中，我们使用1-Max-pooling对每个特征向量池化成一个值，即抽取每个特征向量的最大值表示该特征，而且认为这个最大值表示的是最重要的特征。当我们对所有特征向量进行1-Max-Pooling之后，还需要将每个值给拼接起来。得到池化层最终的特征向量。在池化层到全连接层之前可以加上dropout防止过拟合。

（4）全连接层（Fully connected layer）

全连接层跟其他模型一样，假设有两层全连接层，第一层可以加上’relu’作为激活函数，第二层则使用softmax激活函数得到属于每个类的概率。

（5）TextCNN的小变种

在词向量构造方面可以有以下不同的方式： CNN-rand: 随机初始化每个单词的词向量通过后续的训练去调整。 CNN-static: 使用预先训练好的词向量，如word2vec训练出来的词向量，在训练过程中不再调整该词向量。 CNN-non-static: 使用预先训练好的词向量，并在训练过程进一步进行调整。 CNN-multichannel: 将static与non-static作为两通道的词向量。

（6）参数与超参数

sequence\_length （Q: 对于CNN, 输入与输出都是固定的，可每个句子长短不一, 怎么处理? A: 需要做定长处理, 比如定为n, 超过的截断, 不足的补0. 注意补充的0对后面的结果没有影响，因为后面的max-pooling只会输出最大值，补零的项会被过滤掉）

num\_classes （多分类, 分为几类）

vocabulary\_size （语料库的词典大小, 记为|D|）

embedding\_size （将词向量的维度, 由原始的 |D| 降维到 embedding\_size）

filter\_size\_arr （多个不同size的filter）

|  |
| --- |
| # -\*- coding: utf-8 -\*- |
|  | """ |
|  | Created on Wed May 22 22:06:27 2019 |
|  |  |
|  | @author: pc |
|  | """ |
|  | from keras.preprocessing.text import Tokenizer |
|  | import numpy as np |
|  | from gensim.models.fasttext import FastText |
|  | from sklearn.model\_selection import train\_test\_split |
|  | from sklearn import preprocessing |
|  | from keras.utils.np\_utils import to\_categorical |
|  | from keras.preprocessing.sequence import pad\_sequences |
|  | from keras.models import Sequential, Model |
|  | from keras.layers import Embedding, Conv1D, MaxPooling1D, Dropout, BatchNormalization, Dense, Flatten |
|  | from keras.engine.input\_layer import Input |
|  | from keras.layers.merge import concatenate |
|  | from keras.models import load\_model |
|  |  |
|  | from keras.utils import plot\_model |
|  | #import os |
|  | #os.environ['PATH'] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/' |
|  |  |
|  |  |
|  | FASTEXT\_SIZE = 200 |
|  | LEN\_WORDS = 20 |
|  |  |
|  | def read\_file(path): |
|  | with open(path, 'r', encoding="UTF-8") as f: |
|  | data = [] |
|  | labels = [] |
|  | for line in f: |
|  | if line.split('\t')[1] == '': |
|  | continue |
|  | data.append(line.split('\t')[0]) |
|  | labels.append(line.split('\t')[1]) |
|  | return data, labels |
|  |  |
|  |  |
|  | def get\_tokenizer(data): |
|  | tokenizer = Tokenizer(num\_words=None) |
|  | tokenizer.fit\_on\_texts(data) |
|  | text\_seq = tokenizer.texts\_to\_sequences(data) |
|  | # 对应的单词和数字的映射关系 |
|  | word\_index = tokenizer.word\_index |
|  | index\_word = tokenizer.index\_word |
|  |  |
|  | return word\_index, index\_word, text\_seq |
|  |  |
|  |  |
|  | def get\_fasttext\_voc(data, word\_index): |
|  | ''' |
|  | 利用fasttext获取词向量 |
|  | ''' |
|  | fasttext\_model = FastText([data], |
|  | size=FASTEXT\_SIZE, # 需要学习的嵌入大小(默认为100) |
|  | window=3, # 上下文窗口大小(默认5) |
|  | min\_count=1, # 忽略出现次数低于此值的单词(默认值5) |
|  | iter=10, # epoch(默认5) |
|  | min\_n = 3, # char ngram的最小长度(默认值3) |
|  | max\_n = 6, # char ngram的最大长度(默认值6) |
|  | word\_ngrams = 0) # 如果为1，使用子单词(n-grams)信息丰富单词向量。如果是0，这就相当于Word2Vec |
|  | # 获取词向量列表 |
|  | wordEmbedding = np.zeros((len(word\_index) + 1, FASTEXT\_SIZE)) |
|  | for word, i in word\_index.items(): |
|  | if word in fasttext\_model: |
|  | wordEmbedding[i] = fasttext\_model[word] |
|  |  |
|  | return wordEmbedding |
|  |  |
|  |  |
|  | def get\_label\_num(data): |
|  | data = [i.replace('\n', '') for i in data] |
|  | y\_labels = list(set(data)) |
|  | le = preprocessing.LabelEncoder() |
|  | le.fit(y\_labels) |
|  | num\_labels = len(y\_labels) |
|  | data\_labels = to\_categorical([le.transform([x])[0] for x in data], num\_labels) |
|  | return data\_labels |
|  |  |
|  | def model\_CNN(X\_train, X\_test, y\_train, y\_test, index\_word, embedding\_matrix): |
|  | ''' |
|  | 模型结构： |
|  | 嵌入层：将词编码数据转换为固定尺寸的稠密向量，同时把词向量矩阵加载到Embedding层 |
|  | 卷积池化层：256 \* 3 \* 3 |
|  | 卷积池化层：128 \* 3 \* 3 |
|  | Dropout：0.1 |
|  | BatchNormalization: 批量标准化层，在每一个批次的数据中标准化前一层的激活项 |
|  | 全连接：256,'relu' |
|  | 分类器：2， 'softmax' |
|  | ''' |
|  | model = Sequential() |
|  | model.add(Embedding(len(index\_word) + 1, # imput\_dim: 词汇表大小，即最大整数index+1 |
|  | FASTEXT\_SIZE, # output\_dim: 词向量的维度 |
|  | weights=[embedding\_matrix], # 加载词向量矩阵 |
|  | input\_length=LEN\_WORDS, # input\_lenth: 输入序列的长度 |
|  | trainable=False)) # 设置trainable=False使得这个编码层不可再训练 |
|  | # filters:输出空间的维度，kernel\_size: 1D 卷积窗口的长度，padding:"same" 表示填充输入以使输出具有与原始输入相同的长度 |
|  | model.add(Conv1D(256, 3, padding='same')) |
|  | # pool\_size:最大池化的窗口大小, strides:作为缩小比例的因数 |
|  | model.add(MaxPooling1D(3, 3, padding='same')) |
|  | model.add(Conv1D(128, 3, padding='same')) |
|  | model.add(MaxPooling1D(3, 3, padding='same')) |
|  | model.add(Conv1D(64, 3, padding='same')) |
|  |  |
|  | model.add(Flatten()) |
|  | # rate: 在 0 和 1 之间浮动。需要丢弃的输入比例 |
|  | model.add(Dropout(0.1)) |
|  | model.add(BatchNormalization()) |
|  | # units: 正整数，输出空间维度 |
|  | model.add(Dense(256, activation='relu')) |
|  | model.add(Dropout(0.1)) |
|  | model.add(Dense(2, activation='softmax')) |
|  | # 配置训练模型 |
|  | model.compile(loss='categorical\_crossentropy', # 表示目标应该是分类格式的 |
|  | optimizer='adam', # 随机优化的一种方法 |
|  | metrics=['accuracy'] # 模型评估标准 |
|  | ) |
|  | # 给定数量的迭代训练模型 |
|  | model.summary() |
|  | model.fit(X\_train, y\_train, |
|  | batch\_size=32, |
|  | epochs=15, |
|  | validation\_data=(X\_test, y\_test)) |
|  | model.save('E:/task7/cnn\_model.h5') |
|  | #生成一个模型图，第一个参数为模型，第二个参数为要生成图片的路径及文件名，还可以指定两个参数： |
|  | #show\_shapes:指定是否显示输出数据的形状，默认为False |
|  | #show\_layer\_names:指定是否显示层名称，默认为True |
|  | plot\_model(model,to\_file='E:/task7/cnn.png',show\_shapes=True,show\_layer\_names=False) |
|  |  |
|  |  |
|  |  |
|  |  |
|  | def model\_TextCNN(X\_train, X\_test, y\_train, y\_test, index\_word, embedding\_matrix): |
|  | ''' |
|  | 模型结构： |
|  | 词嵌入， |
|  | 卷积池化 \* 3：256 \* 3 \* 4 |
|  | 拼接三个模型的输出向量， |
|  | 全连接， |
|  | Dropout， |
|  | 全连接 |
|  | ''' |
|  | # shape: 一个尺寸元组（整数）表明期望的输入是按批次的LEN\_WORDS维向量 |
|  | main\_input = Input(shape=(LEN\_WORDS, ), dtype='float32') |
|  | embed = Embedding(len(index\_word) + 1, |
|  | FASTEXT\_SIZE, |
|  | weights=[embedding\_matrix], |
|  | input\_length=LEN\_WORDS, |
|  | trainable=False)(main\_input) |
|  | # 词窗大小分别为3，4，5 |
|  | # strides指明卷积的步长 |
|  | cnn1 = Conv1D(256, 3, padding='same', strides=1, activation='relu')(embed) |
|  | cnn1 = MaxPooling1D(pool\_size=4)(cnn1) |
|  | cnn2 = Conv1D(256, 4, padding='same', strides=1, activation='relu')(embed) |
|  | cnn2 = MaxPooling1D(pool\_size=4)(cnn2) |
|  | cnn3 = Conv1D(256, 5, padding='same', strides=1, activation='relu')(embed) |
|  | cnn3 = MaxPooling1D(pool\_size=4)(cnn3) |
|  | # 合并三个模型的输出向量 |
|  | cnn = concatenate([cnn1,cnn2,cnn3], axis=-1) |
|  | flat = Flatten()(cnn) |
|  | drop = Dropout(0.1)(flat) |
|  | main\_output = Dense(2, activation='softmax')(drop) |
|  | model = Model(inputs=main\_input, output=main\_output) |
|  | # 配置训练模型 |
|  | model.compile(loss='categorical\_crossentropy', # 表示目标应该是分类格式的 |
|  | optimizer='adam', # 随机优化的一种方法 |
|  | metrics=['accuracy'] # 模型评估标准 |
|  | ) |
|  | # 给定数量的迭代训练模型 |
|  | model.summary() |
|  | model.fit(X\_train, y\_train, |
|  | batch\_size=32, |
|  | epochs=15, |
|  | validation\_data=(X\_test, y\_test)) |
|  | plot\_model(model, to\_file='E:/task7/textcnn.png', show\_shapes=True, show\_layer\_names=False) |
|  | model.save('E:/task7/textcnn\_model.h5') |
|  |  |
|  |  |
|  | # 读文件 |
|  | data, labels = read\_file('E:/task6/merge.txt') |
|  |  |
|  | # 利用tokenizer将文字转换为数字特征 |
|  | word\_index, index\_word, X\_train\_text\_seq = get\_tokenizer(data) |
|  | # fasttext embedding |
|  | wordEmbedding = get\_fasttext\_voc(data, word\_index) |
|  |  |
|  | # 让每个文本长度相同 |
|  | X\_train\_text\_seq = pad\_sequences(X\_train\_text\_seq, maxlen=LEN\_WORDS, padding='post', truncating='post') |
|  |  |
|  | # 划分数据集 |
|  | X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train\_text\_seq, |
|  | labels, |
|  | test\_size = 0.2, |
|  | random\_state=33) |
|  | # 对类别变量编码 |
|  | y\_train\_label = get\_label\_num(y\_train) |
|  | y\_test\_label = get\_label\_num(y\_test) |
|  | # 构建CNN分类模型 |
|  | model\_CNN(X\_train, X\_test, y\_train\_label, y\_test\_label, index\_word, wordEmbedding) |
|  | model\_TextCNN(X\_train, X\_test, y\_train\_label, y\_test\_label, index\_word, wordEmbedding) |
|  |  |