# 负面评论识别

提到负面评论，我们首先要聊下水军。水军顾名思义，是指在论坛大量灌水的人员 ，他们受雇于[网络公关公司](http://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%85%AC%E5%85%B3%E5%85%AC%E5%8F%B8" \t "_blank)、以发帖回帖为主要手段、为雇主进行网络造势的网络人员，有专职和兼职之分。一般来讲，发帖回帖造势常常需要成百上千个人共同完成，那些临时在网上征集来的发帖的人在行内被叫做“网络水军”。2009年12月央视报道网络水军这一新兴现象之后，受到社会广泛关注，不少长期在线的网虫们纷纷加入网络水军一族。水军的大行其道，严重影响了社会舆论，尤其是有组织的针对公众人物、企业的诽谤、攻击行为，造成严重的社会影响。大型的社交媒体以及知名论坛，早期都需要雇佣大量的运营支撑人员来人工鉴别处理。是否可以使用机器学习的技术来达到一定程度的自动化识别负面评论呢？我们利用垃圾邮件识别的技术积累，尝试来完成这一任务。



1. 网络水军

本章主要以IMDB数据集为例子介绍负面评论的识别技术。

本章1.1介绍验证码识别使用的数据集。

本章1.2介绍使用的特征提取方法，包括一维向量和二维向量。

本章1.3介绍使用的模型以及对应的验证结果，包括K近邻、支持向量机和深度学习。

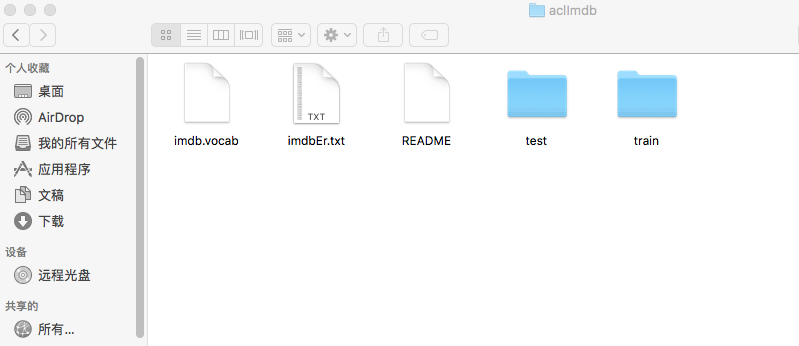
本章演示代码请参考本书Github上的review.py文件。

## 数据集

测试数据来自互联网[电影](http://baike.baidu.com/item/%E7%94%B5%E5%BD%B1/31689" \t "_blank)资料库（Internet Movie Database，IMDB），IMDB是一个关于电影演员、电影、电视节目、电视明星和[电影制作](http://baike.baidu.com/item/%E7%94%B5%E5%BD%B1%E5%88%B6%E4%BD%9C" \t "_blank)的[在线数据库](http://baike.baidu.com/item/%E5%9C%A8%E7%BA%BF%E6%95%B0%E6%8D%AE%E5%BA%93" \t "_blank)。IMDb另一受欢迎的特色是其对应每个数据库条目，并且有47个主要板块的留言板系统。注册用户可以在这些留言板上分享和讨论关于电影，演员，导演的消息。至今已有超过6百万注册用户使用过留言板。我们使用标注为正面评论和负面评论的留言板数据。

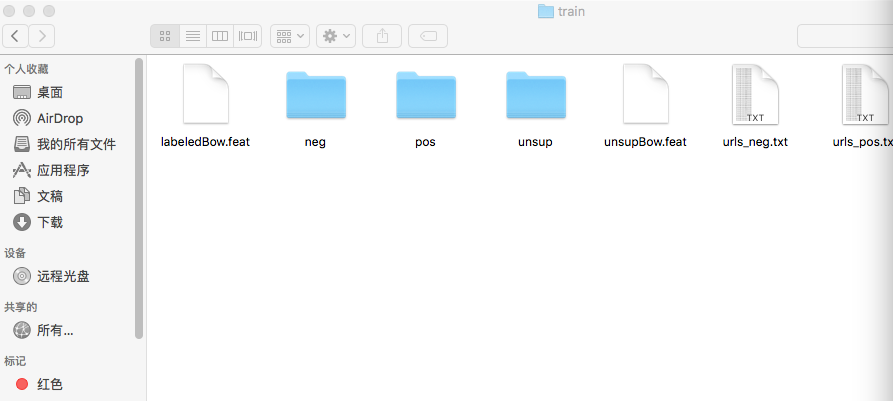
整个数据集一共10万条记录，5万做了标记，5万没有做标记。5万做了标记的数据集合被随机分配成了训练数据集和测试数据集。

IMDB文件夹下一级目录主要包含训练数据集和测试数据集两个文件夹。



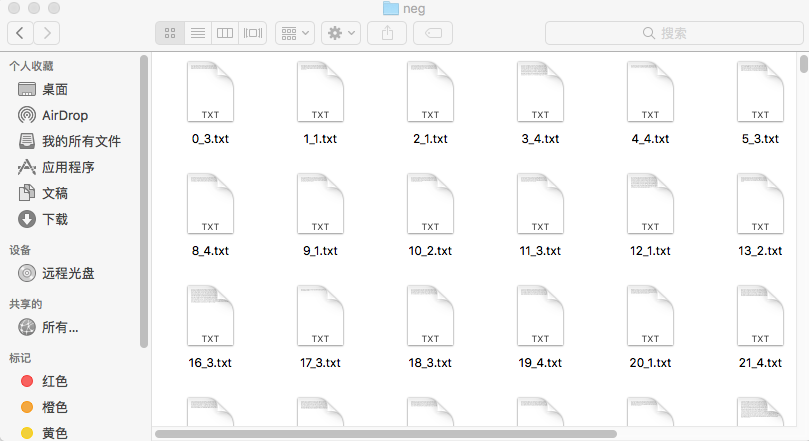
1. IMDB一级文件夹结构

以训练数据集为例，IMDB二级目录下主要包含三个目录，分别为正面评论、负面评论和未标识。



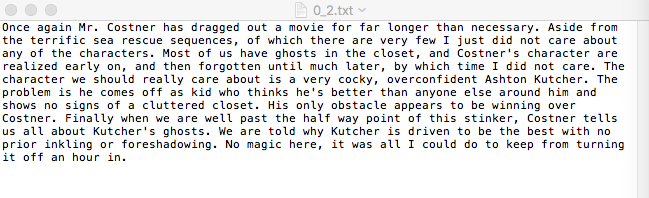
1. IMDB二级文件夹结构

以正面评论为例，IMDB三级目录下为评论文件，每个评论以单独文件形式保存。



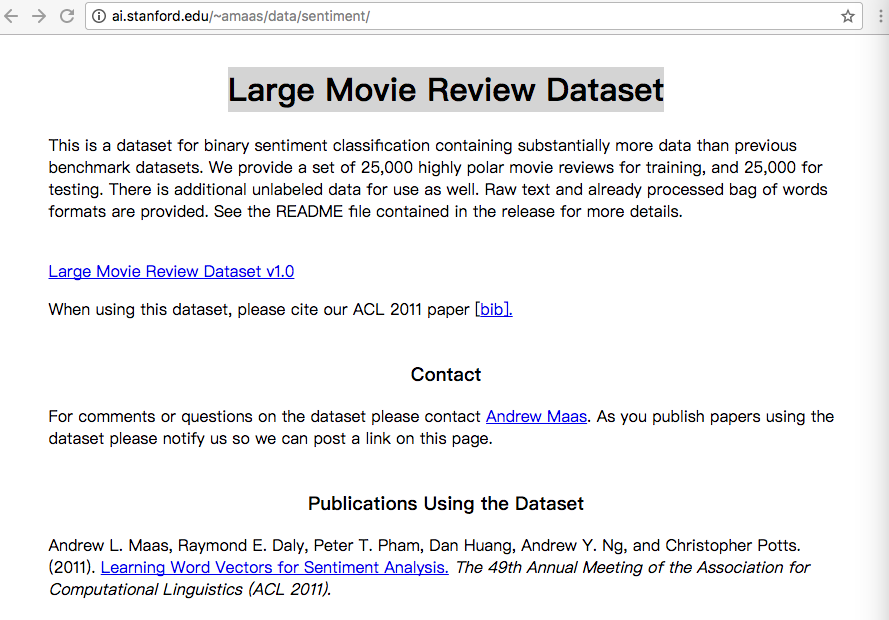
1. IMDB三级文件夹结构

评论文件记载了原始的评论内容。



1. IMDB评论文件内容

IMDB数据的下载地址为：http://ai.stanford.edu/~amaas/data/sentiment/



1. IMDB数据集下载主页

## 特征提取

### 方法一：词袋&TF-IDF模型

我们使用最常见的词袋模型提取文件特征。

把一个评论文件作为一个完整的字符串处理，定义函数load\_one\_file加载文件到一个字符串变量中返回。

def load\_one\_file(filename):  
 x=""  
 with open(filename) as f:  
 for line in f:  
 line=line.strip('\n')  
 line = line.strip('\r')  
 x+=line  
 return x

遍历目录，加载目录下全部文件，以字符串集合的形式返回。

def load\_files\_from\_dir(rootdir):  
 x=[]  
 list = os.listdir(rootdir)  
 for i in range(0, len(list)):  
 path = os.path.join(rootdir, list[i])  
 if os.path.isfile(path):  
 v=load\_one\_file(path)  
 x.append(v)  
 return x

分别加载训练数据集目录下的正面评论和负面评论目录下全部文件，同时进行标记，正面评论为0，负面评论为1。

path="../data/review/aclImdb/train/pos/"  
print "Load %s" % path  
x\_train=load\_files\_from\_dir(path)  
y\_train=[0]\*len(x\_train)  
path="../data/review/aclImdb/train/neg/"  
print "Load %s" % path  
tmp=load\_files\_from\_dir(path)  
y\_train+=[1]\*len(tmp)

x\_train+=tmp

分别加载训练数据集目录下的正面评论和负面评论目录下全部文件，同时进行标记，正面评论为0，负面评论为1。

path="../data/review/aclImdb/test/pos/"  
print "Load %s" % path  
x\_test=load\_files\_from\_dir(path)  
y\_test=[0]\*len(x\_test)  
path="../data/review/aclImdb/test/neg/"  
print "Load %s" % path  
tmp=load\_files\_from\_dir(path)  
y\_test+=[1]\*len(tmp)

x\_test+=tmp

使用Scikit-Learn的CountVectorizer函数进行词袋化处理，同时将抽取的词汇表保存，用于词袋化测试数据集。其中非常重要的几个参数的含义为：

* decode\_error，处理解码失败的方式，分为‘strict’、‘ignore’、‘replace’三种方式
* strip\_accents，在预处理步骤中移除重音的方式
* max\_features，词袋特征个数的最大值
* stop\_words，判断word结束的方式
* max\_df，df最大值
* min\_df，df最小值
* binary，默认为False，当与TF-IDF结合使用时需要设置为True

本例中处理的数据集均为英文，所以针对解码失败直接忽略，使用ignore方式，stop\_words的方式使用english，strip\_accents方式为ascii方式。

x\_train, y\_train, x\_test, y\_test=load\_all\_files()  
vectorizer = CountVectorizer(  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 max\_features=max\_features,  
 stop\_words='english',  
 max\_df=1.0,  
 min\_df=1 )  
print vectorizer  
x\_train=vectorizer.fit\_transform(x\_train)  
x\_train=x\_train.toarray()  
vocabulary=vectorizer.vocabulary\_

复用vocabulary对测试数据集进行词袋化处理，其中由于设置了vocabulary，max\_features的值会被忽略。

vectorizer = CountVectorizer(  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 vocabulary=vocabulary,  
 stop\_words='english',  
 max\_df=1.0,  
 min\_df=1 )  
print vectorizer  
x\_test=vectorizer.fit\_transform(x\_test)  
x\_test=x\_test.toarray()

文本处理领域还有一种特征提取方法，叫做TF-IDF模型（term frequency–inverse document frequency，词频与逆向文件频率）。TF-IDF是一种[统计方法](http://baike.baidu.com/item/%E7%BB%9F%E8%AE%A1%E6%96%B9%E6%B3%95" \t "_blank)，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在[语料库](http://baike.baidu.com/item/%E8%AF%AD%E6%96%99%E5%BA%93" \t "_blank)中出现的频率成反比下降。TF-IDF与词袋模型结合后可以提高检测能力，我们使用TF-IDF将词袋模型生成的数据做进一步处理。

transformer = TfidfTransformer(smooth\_idf=False)  
x\_train=transformer.fit\_transform(x\_train)  
x\_train=x\_train.toarray()  
x\_test=transformer.transform(x\_test)  
x\_test=x\_test.toarray()

### 方法二：词汇表模型

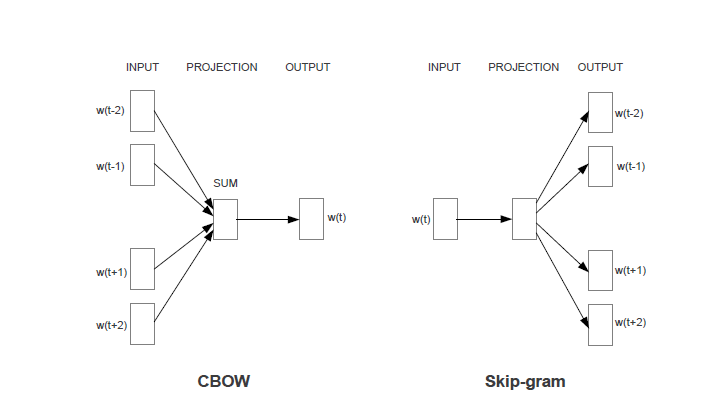
词袋模型可以很好的表现文本由哪些单词组成，但是却无法表达出单词之间的前后关系，于是人们借鉴了词袋模型的思想，使用生成的词汇表对原有句子按照单词逐个进行编码。在本例中，使用TensorFlow的tflearn.data\_utils.VocabularyProcessor函数即可。

def get\_features\_by\_tf():  
 global max\_document\_length  
 x\_train, x\_test, y\_train, y\_test=load\_all\_files()  
 vp=tflearn.data\_utils.VocabularyProcessor(

max\_document\_length=max\_document\_length,  
 min\_frequency=0,  
 vocabulary=None,  
 tokenizer\_fn=None)  
 x\_train=vp.fit\_transform(x\_train, unused\_y=None)  
 x\_train=np.array(list(x\_train))  
 x\_test=vp.transform(x\_test)  
 x\_test=np.array(list(x\_test))  
 return x\_train, x\_test, y\_train, y\_test

### 方法三：word2vec模型

word2vec是Google在2013年年中开源的一款将词表征为实数值向量的高效工具，采用的模型有CBOW(Continuous Bag-Of-Words，即连续的词袋模型)和Skip-Gram 两种。word2vec通过训练，可以把对文本内容的处理简化为K维向量空间中的向量运算，而向量空间上的相似度可以用来表示文本语义上的相似度。因此，word2vec 输出的词向量可以被用来做很多NLP相关的工作，比如聚类、找同义词、词性分析等等。



1. CBOW和Skip-gram原理图

 CBOW模型能够根据输入周围n-1个词来预测出这个词本身，而skip-gram模型能够根据词本身来预测周围有哪些词。也就是说，CBOW模型的输入是某个词A周围的n个单词的词向量之和，输出是词A本身的词向量；而skip-gram模型的输入是词A本身，输出是词A周围的n个单词的词向量。

word2vec最常用的开源实现之一就是gensim，网址为<http://radimrehurek.com/gensim/>

gensim的安装非常简单。

pip install --upgrade gensim

gensim的使用非常简洁，加载数据和训练数据可以合并，训练好模型后就可以按照单词获取对应的向量表示。

sentences = [['first', 'sentence'], ['second', 'sentence']]

model = gensim.models.Word2Vec(sentences, min\_count=1)

print model['first']

其中word2vec有很多可以影响训练速度和质量的参数。第一个参数可以对字典做截断. 少于min\_count次数的单词会被丢弃掉, 默认值为5。

model = Word2Vec(sentences, min\_count=10)

另外一个是神经网络的隐藏层的单元数，推荐值为几十到几百。

model = Word2Vec(sentences, size=200)

## 模型训练与验证

### 方法一：朴素贝叶斯算法

使用朴素贝叶斯算法，特征提取使用词袋模型。

gnb = GaussianNB()  
gnb.fit(x\_train,y\_train)  
y\_pred=gnb.predict(x\_test)

评估结果的准确度和TT、FF、TF、FT四个值。

print metrics.accuracy\_score(y\_test, y\_pred)  
print metrics.confusion\_matrix(y\_test, y\_pred)

在词袋最大特征数为5000的情况下，仅使用词袋模型时，整个系统准确度为68.64%，TT、FF、TF、FT矩阵如表1-1 所示。

1. 基于词袋模型的朴素贝叶斯验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 6255 | 6245 |
| F | 1595 | 10905 |

完整输出结果为：

CountVectorizer(analyzer=u'word', binary=False, decode\_error='ignore',

        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'content',

        lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

        ngram\_range=(1, 1), preprocessor=None, stop\_words='english',

        strip\_accents='ascii', token\_pattern=u'(?u)\\b\\w\\w+\\b',

        tokenizer=None,

NB and wordbag

0.6864

[[ 6255  6245]

 [ 1595 10905]]

在词袋最大特征数为5000的情况下，同时使用词袋和TF-IDF模型时，整个系统准确度约为75.89%，TT、FF、TF、FT矩阵如表1-2 所示。

1. 基于词袋和TF-IDF模型的朴素贝叶斯验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 8663 | 3837 |
| F | 2191 | 10309 |

完整输出结果为：

CountVectorizer(analyzer=u'word', binary=False, decode\_error='ignore',

        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'content',

        lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

        ngram\_range=(1, 1), preprocessor=None, stop\_words='english',

        strip\_accents='ascii', token\_pattern=u'(?u)\\b\\w\\w+\\b',

        tokenizer=None,

NB and wordbag&TF-IDF

0.75888

[[ 8663  3837]

 [ 2191 10309]]

### 方法二：支持向量机算法

使用支持向量机算法，特征提取使用词袋模型。

clf = svm.SVC()  
clf.fit(x\_train, y\_train)  
y\_pred = clf.predict(x\_test)

评估结果的准确度和TT、FF、TF、FT四个值。

print metrics.accuracy\_score(y\_test, y\_pred)  
print metrics.confusion\_matrix(y\_test, y\_pred)

在词袋最大特征数为5000且同时使用TF-IDF模型的情况下，整个系统准确度为79.70%，TT、FF、TF、FT矩阵如表1-3 所示。

1. 基于词袋和TF-IDF模型的支持向量机验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 9697 | 2803 |
| F | 2271 | 10229 |

完整输出结果为：

CountVectorizer(analyzer=u'word', binary=False, decode\_error='ignore',

        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'content',

        lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

        ngram\_range=(1, 1), preprocessor=None, stop\_words='english',

        strip\_accents='ascii', token\_pattern=u'(?u)\\b\\w\\w+\\b',

        tokenizer=None,

SVM and wordbag&TF-IDF

0.79704

[[ 9697 2803]

[ 2271 10229]]

### 方法三：深度学习算法之MLP

使用MLP算法，隐含层设计为2层，每次节点数分别为5和2，特征提取使用词袋和TF-IDF模型。

clf = MLPClassifier(solver='lbfgs',  
 alpha=1e-5,  
 hidden\_layer\_sizes = (5, 2),  
 random\_state = 1)  
print clf  
clf.fit(x\_train, y\_train)  
y\_pred = clf.predict(x\_test)

评估结果的准确度和TT、FF、TF、FT四个值。

print metrics.accuracy\_score(y\_test, y\_pred)  
print metrics.confusion\_matrix(y\_test, y\_pred)

在词袋最大特征数为5000且同时使用TF-IDF模型的情况下，整个系统准确度为87.06%，TT、FF、TF、FT矩阵如表1-4 所示。

1. 基于词袋和TF-IDF模型的MLP验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 10908 | 1592 |
| F | 1643 | 10857 |

完整输出结果为：

MLP and wordbag

MLPClassifier(activation='relu', alpha=1e-05, batch\_size='auto', beta\_1=0.9,

beta\_2=0.999, early\_stopping=False, epsilon=1e-08,

hidden\_layer\_sizes=(5, 2), learning\_rate='constant',

learning\_rate\_init=0.001, max\_iter=200, momentum=0.9,

nesterovs\_momentum=True, power\_t=0.5, random\_state=1, shuffle=True,

solver='lbfgs', tol=0.0001, validation\_fraction=0.1, verbose=False,

warm\_start=False)

0.8706

[[10908 1592]

[ 1643 10857]]

### 方法四：深度学习算法之CNN

使用词汇表编码后的数据，获得训练数据集和测试数据集。

x\_train, x\_test, y\_train, y\_test=get\_features\_by\_tf()

将训练和测试数据进行填充和转换，不到最大长度的数据填充0，由于是二分类问题，把标记数据二值化。定义输入参数的最大长度为文档的最大长度。

trainX = pad\_sequences(trainX, maxlen=max\_document\_length, value=0.)  
testX = pad\_sequences(testX, maxlen=max\_document\_length, value=0.)  
# Converting labels to binary vectors  
trainY = to\_categorical(trainY, nb\_classes=2)  
testY = to\_categorical(testY, nb\_classes=2)

network = input\_data(shape=[None,max\_document\_length], name='input')

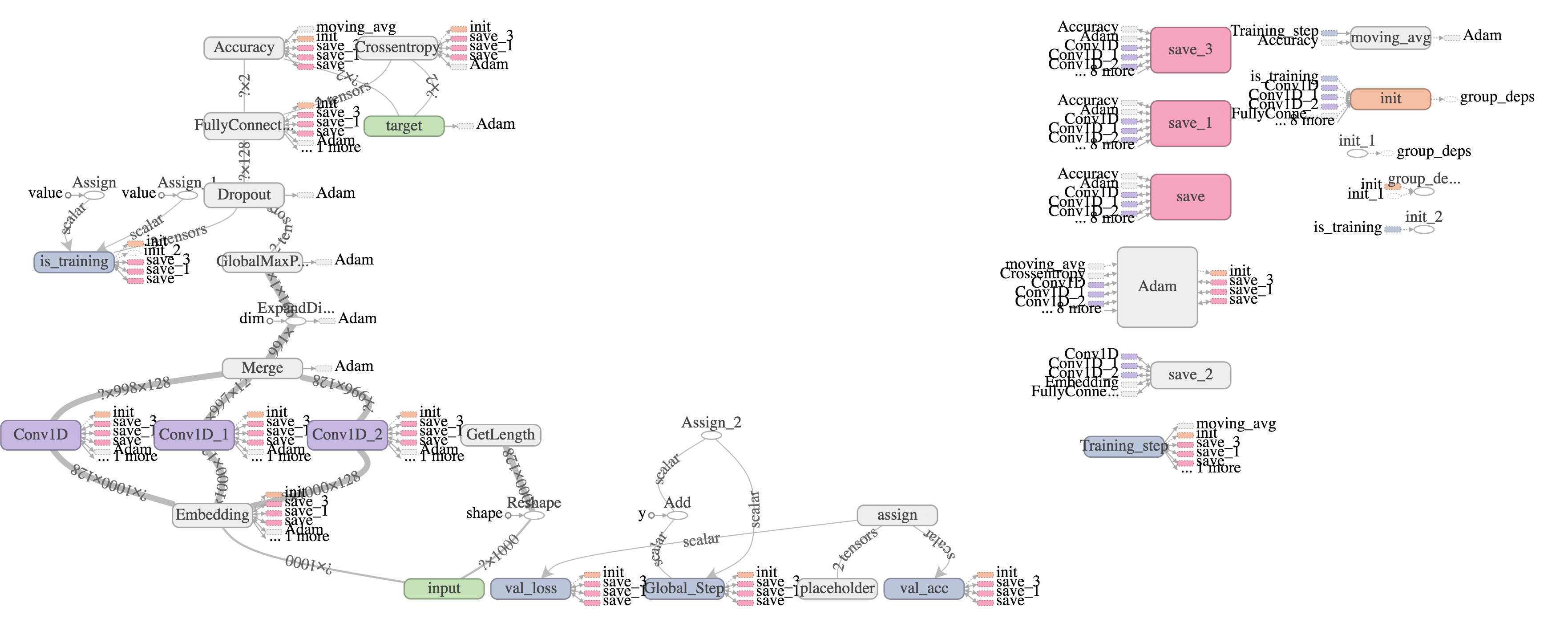
定义CNN模型，其实使用3个数量为128核，长度分别为3、4、5的一维卷积函数处理数据。

network = tflearn.embedding(network, input\_dim=1000000, output\_dim=128)  
branch1 = conv\_1d(network, 128, 3, padding='valid', activation='relu', regularizer="L2")  
branch2 = conv\_1d(network, 128, 4, padding='valid', activation='relu', regularizer="L2")  
branch3 = conv\_1d(network, 128, 5, padding='valid', activation='relu', regularizer="L2")  
network = merge([branch1, branch2, branch3], mode='concat', axis=1)  
network = tf.expand\_dims(network, 2)  
network = global\_max\_pool(network)  
network = dropout(network, 0.8)  
network = fully\_connected(network, 2, activation='softmax')  
network = regression(network, optimizer='adam', learning\_rate=0.001,  
 loss='categorical\_crossentropy', name='target')

实例化CNN对象并进行训练数据，一共训练5轮。

model = tflearn.DNN(network, tensorboard\_verbose=0)  
model.fit(trainX, trainY,  
 n\_epoch=5, shuffle=True, validation\_set=(testX, testY),  
 show\_metric=True, batch\_size=100,run\_id="review")

完整的CNN结构如图1-8 所示。



1. 用于识别负面评论的CNN结构图

运行程序，经过5轮训练，准确率达到了%。

## 本章小结

本章基于MNIST数据集介绍了验证码的识别方法。针对MNIST数据集，特征提取方法有一维向量和二维向量两种方法。训练模型介绍了K近邻、支持向量机以及深度学习的DNN和CNN算法，其中DNN算法表现优异，识别率达到了99%以上。

### 参考文献

* <http://china.huanqiu.com/hot/2013-09/4320971.html>
* <http://dl.sohu.com/20130911/n386353688.shtml>
* Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). [Learning Word Vectors for Sentiment Analysis.](http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf) The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).
* C. O. Alm, D. Roth, and R. Sproat. 2005. Emotions from text: machine learning for text-based emotion predic- tion. In Proceedings of HLT/EMNLP, pages 579–586.
* A. Andreevskaia and S. Bergler. 2006. Mining Word- Net for fuzzy sentiment: sentiment tag extraction from WordNet glosses. In Proceedings of the European ACL, pages 209–216.
* Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. 2003. a neural probabilistic language model. Journal of Ma- chine Learning Research, 3:1137–1155, August.
* D. M. Blei, A. Y. Ng, and M. I. Jordan. 2003. Latent dirichlet allocation. Journal of Machine Learning Re- search, 3:993–1022, May.
* J. Boyd-Graber and P. Resnik. 2010. Holistic sentiment analysis across languages: multilingual supervised la- tent Dirichlet allocation. In Proceedings of EMNLP, pages 45–55.
* R. Collobert and J. Weston. 2008. A unified architecture for natural language processing. In Proceedings of the ICML, pages 160–167.
* Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, [Thumbs up? Sentiment Classification using Machine Learning Techniques](http://www.cs.cornell.edu/home/llee/papers/sentiment.home.html), Proceedings of EMNLP 2002.
* Bo Pang and Lillian Lee, [A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts](http://www.cs.cornell.edu/home/llee/papers/cutsent.home.html), Proceedings of ACL 2004.
* Bo Pang and Lillian Lee, [Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales](http://www.cs.cornell.edu/home/llee/papers/pang-lee-stars.home.html), Proceedings of ACL 2005.
* Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.
* Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.
* Optimizing word2vec in gensim, <http://radimrehurek.com/2013/09/word2vec-in-python-part-two-optimizing/>
* <http://blog.csdn.net/mytestmy/article/details/26969149?utm_source=tuicool&utm_medium=referral>
* <http://techblog.youdao.com/?p=915>
* <http://blog.csdn.net/u014595019/article/details/51884529>
* <http://blog.csdn.net/Star_Bob/article/details/47808499>
* http://radimrehurek.com/gensim/models/word2vec.html