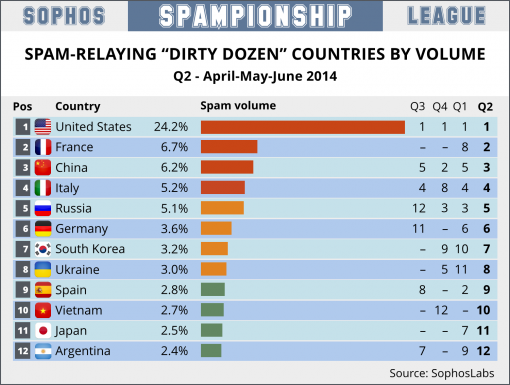
# 第1章 垃圾邮件识别

垃圾邮件作为[因特网](http://baike.baidu.com/item/%E5%9B%A0%E7%89%B9%E7%BD%91)中最具有争议的副产品，对于企业邮箱用户的影响首先就在于给日常办公和邮箱管理者带来额外负担。根据不完全统计，在高效的反垃圾环境下仍然有80%的用户每周需要耗费10分钟左右的时间处理垃圾邮件，而对于中国多数企业邮件应用仍处于低效率反垃圾环境的情况下，这个比例更是呈现数十倍的增长，如图1-1 所示，中国垃圾邮件的总量已经达到全球第三。对于企业邮件服务商而言，垃圾邮件的恶意投送，还会大量占用网络资源，使得[邮件服务器](http://baike.baidu.com/item/%E9%82%AE%E4%BB%B6%E6%9C%8D%E5%8A%A1%E5%99%A8)85%的系统资源在用于处理垃圾邮件的识别，不仅资源浪费极其严重，甚至可能导致网络阻塞瘫痪，影响企业正常业务邮件的沟通。



1. 世界垃圾邮件最多国家排行

更严重的垃圾邮件问题甚至不仅只是影响企业工作效率，甚至会祸及整个服务器。由于企业邮箱平台往往通信质量更好，更容易为国际反垃圾邮件平台白名单所接纳，因此成为垃圾邮件投送者甚至网络黑客的重点攻击目标。通过劫持这些企业邮箱，垃圾邮件投送者就可以大大提高垃圾邮件投送率。这不仅将为企业邮箱用户造成无法预知的影响，同时还可能遭遇反垃圾邮件组织和邮件接收方的壮士断腕。对于这些大量垃圾邮件的投放地址，国际反垃圾组织和接收方服务商将会将其加入国际垃圾邮件数据库中，从而导致该主机不能与其他国家正常通信，严重影响企业海外业务的扩展。

正因为垃圾邮件具备如此严重的危害。如今，在国内外众多企业都在从过去的自主建设服务、借助价格低廉的小型服务商建立邮件系统，转而将263企业邮箱这类大型专业邮件服务商作为唯一选择。显而易见，263企业邮箱这类更大型更专业的邮件服务提供商，不仅在反垃圾邮件技术上更加成熟，同时也更有利于与国内外众多的反垃圾组织与邮件服务商联合起来，组建综合的反垃圾平台，通过共享实时拒收列表来阻挡垃圾邮件的传播。

本章1.1介绍垃圾邮件识别使用的数据集。

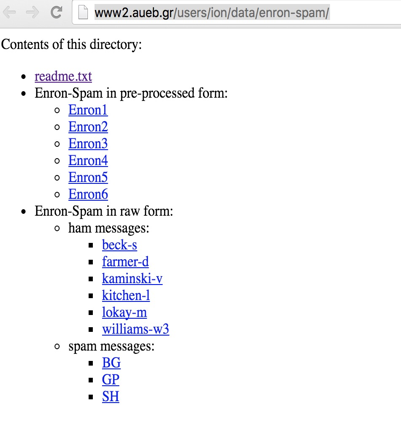
本章1.2介绍使用的特征提取方法，包括词袋模型和TF-IDF模型。

本章1.3介绍使用的模型以及对应的验证结果，包括朴素贝叶斯、支持向量基和深度学习。

## 数据集

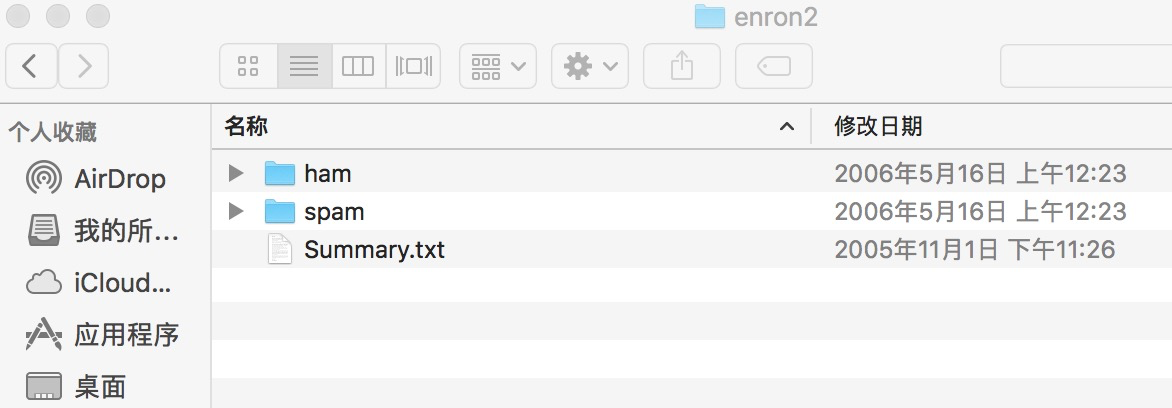
垃圾邮件识别使用的数据集为Enron-Spam数据集，Enron-Spam数据集是目前在电子邮件相关研究中使用最多的公开数据集，其邮件数据是安然公司（Enron Corporation, 原是世界上最大的综合性天然气和电力公司之一，在北美地区是头号天然气和电力批发销售商）150位高级管理人员的往来邮件。这些邮件在安然公司接受美国联邦能源监管委员会调查时被其公布到网上。

机器学习领域使用Enron-Spam数据集来研究文档分类、词性标注、垃圾邮件识别等，由于Enron-Spam数据集都是真实环境下的真实邮件，非常具有实际意义。



1. Enron-Spam数据集主页

Enron-Spam数据集合如图1-3 所示使用不同文件夹区分正常邮件和垃圾邮件。



1. Enron-Spam数据集文件夹结构

正常邮件内容举例如下：

Subject: christmas baskets

the christmas baskets have been ordered .

we have ordered several baskets .

individual earth - sat freeze - notis

smith barney group baskets

rodney keys matt rodgers charlie

notis jon davis move

team

phillip randle chris hyde

harvey

freese

faclities

垃圾邮件内容举例如下：

Subject: fw : this is the solution i mentioned lsc

oo

thank you ,

your email address was obtained from a purchased list ,

reference # 2020 mid = 3300 . if you wish to unsubscribe

from this list , please click here and enter

your name into the remove box . if you have previously unsubscribed

and are still receiving this message , you may email our abuse

control center , or call 1 - 888 - 763 - 2497 , or write us at : nospam ,

6484 coral way , miami , fl , 33155 " . 2002

web credit inc . all rights reserved .

Enron-Spam数据集对应的网址为：<http://www2.aueb.gr/users/ion/data/enron-spam/>

## 特征提取

### 方法一：词袋模型

文本特征提取有两个非常重要的模型：

* 词集模型：单词构成的集合，集合自然每个元素都只有一个，也即词集中的每个单词都只有一个
* 词袋模型：如果一个单词在文档中出现不止一次，并统计其出现的次数（频数）

两者本质上的区别，词袋是在词集的基础上增加了频率的纬度，词集只关注有和没有，词袋还要关注有几个。

假设我们要对一篇文章进行特征化，最常见的方式就是词袋。

导入相关的函数库

**>>> from** **sklearn.feature\_extraction.text** **import** CountVectorizer

实例化分词对象

**>>>** vectorizer = CountVectorizer(min\_df=1)

**>>>** vectorizer

CountVectorizer(analyzer=...'word', binary=False, decode\_error=...'strict',

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

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

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

strip\_accents=None, token\_pattern=...'(?u)\\b\\w\\w+\\b',

tokenizer=None, vocabulary=None)

将文本进行词袋处理

**>>>** corpus = [

**...**  'This is the first document.',

**...**  'This is the second second document.',

**...**  'And the third one.',

**...**  'Is this the first document?',

**...** ]

**>>>** X = vectorizer.fit\_transform(corpus)

**>>>** X

<4x9 sparse matrix of type '<... 'numpy.int64'>'

with 19 stored elements in Compressed Sparse ... format>

获取对应的特征名称

**>>>** vectorizer.get\_feature\_names() == (

**...**  ['and', 'document', 'first', 'is', 'one',

**...**  'second', 'the', 'third', 'this'])

True

获取词袋数据，至此我们已经完成了词袋化。但是对于程序中的其他文本，如何可以使用现有的词袋的特征进行向量化呢？

**>>>** X.toarray()

array([[0, 1, 1, 1, 0, 0, 1, 0, 1],

[0, 1, 0, 1, 0, 2, 1, 0, 1],

[1, 0, 0, 0, 1, 0, 1, 1, 0],

[0, 1, 1, 1, 0, 0, 1, 0, 1]]...)

我们定义词袋的特征空间叫做词汇表vocabulary。

vocabulary=vectorizer.vocabulary\_

针对其他文本进行词袋处理时，可以直接使用现有的词汇表。

**>>> new\_**vectorizer = CountVectorizer(min\_df=1, vocabulary=vocabulary)

在本例中，将整个邮件包括正文当成一个字符串处理，其中回车和换行需要过滤掉。

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**

遍历指定文件夹下全部文件，加载数据。

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

Enron-Spam数据集的数据分散在6个文件夹Enron1到Enron6，正常文件在ham文件夹下，垃圾邮件在spam文件夹下，依次记载全部数据。

def load\_all\_files():  
 ham=[]  
 spam=[]  
 for i in range(1,7):  
 path="../data/mail/enron%d/ham/" % i  
 print "Load %s" % path  
 ham+=load\_files\_from\_dir(path)  
 path="../data/mail/enron%d/spam/" % i  
 print "Load %s" % path  
 spam+=load\_files\_from\_dir(path)  
 return ham,spam

使用词袋模型，向量化正常邮件和垃圾邮件样本，其中ham文件夹下的样本标记为0，标记为正常邮件，spam文件夹下的样本标记为1，标记为垃圾邮件。

def get\_features\_by\_wordbag():  
 ham, spam=load\_all\_files()  
 x=ham+spam  
 y=[0]\*len(ham)+[1]\*len(spam)  
 vectorizer = CountVectorizer(  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 max\_features=max\_features,  
 stop\_words='english',  
 max\_df=1,  
 min\_df=1 )  
 print vectorizer  
 x=vectorizer.fit\_transform(x)  
 x=x.toarray()  
 return x,y

* CountVectorize函数比较重要的几个参数为：
* 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。

### 方法二：TF-IDF模型

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)，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)中出现的频率成反比下降。TF-IDF加权的各种形式常被[搜索引擎](http://baike.baidu.com/item/%E6%90%9C%E7%B4%A2%E5%BC%95%E6%93%8E)应用，作为文件与用户查询之间相关程度的度量或评级。

TF-IDF的主要思想是：如果某个词或短语在一篇文章中出现的频率TF(Term Frequency，词频)，词频高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。TF-IDF实际上是：TF \* IDF。TF表示词条在文档d中出现的频率。IDF（inverse document frequency，逆向文件频率）的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大，则说明词条t具有很好的类别区分能力。如果某一类文档C中包含词条t的文档数为m，而其它类包含t的文档总数为k，显然所有包含t的文档数n=m+k，当m大的时候，n也大，按照IDF公式得到的IDF的值会小，就说明该词条t类别区分能力不强。但是实际上，如果一个词条在一个类的文档中频繁出现，则说明该词条能够很好代表这个类的文本的特征，这样的词条应该给它们赋予较高的权重，并选来作为该类文本的特征词以区别与其它类文档。

在Scikit-Learn中实现了TF-IDF算法，实例化TfidfTransformer即可。

**>>> from** **sklearn.feature\_extraction.text** **import** TfidfTransformer

**>>>** transformer = TfidfTransformer(smooth\_idf=**False**)

**>>>** transformer TfidfTransformer(norm=...'l2', smooth\_idf=False, sublinear\_tf=False, use\_idf=True)

TF-IDF模型通常和词袋模型配合使用，对词袋模型生成的数组进一步处理。

**>>>** counts = [[3, 0, 1],

**...**  [2, 0, 0],

**...**  [3, 0, 0],

**...**  [4, 0, 0],

**...**  [3, 2, 0],

**...**  [3, 0, 2]]

**...**

**>>>** tfidf = transformer.fit\_transform(counts)

**>>>** tfidf

<6x3 sparse matrix of type '<... 'numpy.float64'>' with 9 stored elements in Compressed Sparse ... format>

**>>>** tfidf.toarray()

array([[ 0.81940995, 0. , 0.57320793],

[ 1. , 0. , 0. ],

[ 1. , 0. , 0. ],

[ 1. , 0. , 0. ],

[ 0.47330339, 0.88089948, 0. ],

[ 0.58149261, 0. , 0.81355169]])

在本例中，获取完ham和spam数据后，使用词袋模型CountVectorizer进行词袋化，其中binary参数需要设置为True，然后再使用TfidfTransformer计算TF-IDF。

**def get\_features\_by\_wordbag\_tdjfd():  
 ham, spam=load\_all\_files()  
 x=ham+spam  
 y=[0]\*len(ham)+[1]\*len(spam)  
 vectorizer = CountVectorizer(binary=True,  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 max\_features=max\_features,  
 stop\_words='english',  
 max\_df=1.0,  
 min\_df=1 )  
 x=vectorizer.fit\_transform(x)  
 x=x.toarray()  
 transformer = TfidfTransformer(smooth\_idf=False)  
 tfidf = transformer.fit\_transform(x)  
 x = tfidf.toarray()  
 return x,y**

### 方法三：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。TensorFlow默认支持了这种模型。

tf.contrib.learn.preprocessing.VocabularyProcessor (max\_document\_length, min\_frequency=0,

vocabulary=None,

tokenizer\_fn=None)

其中各个参数的含义为：

* max\_document\_length:，文档的最大长度。如果文本的长度大于最大长度，那么它会被剪切，反之则用0填充
* min\_frequency，词频的最小值，出现次数小于最小词频则不会被收录到词表中
* vocabulary，CategoricalVocabulary 对象
* tokenizer\_fn，分词函数

假设有如下句子需要处理：

x\_text =[

'i love you',

'me too'

]

基于以上句子生成词汇表，并对'i me too'这句话进行编码。

vocab\_processor = learn.preprocessing.VocabularyProcessor(max\_document\_length)

vocab\_processor.fit(x\_text)

print next(vocab\_processor.transform(['i me too'])).tolist()

x = np.array(list(vocab\_processor.fit\_transform(x\_text)))

print x

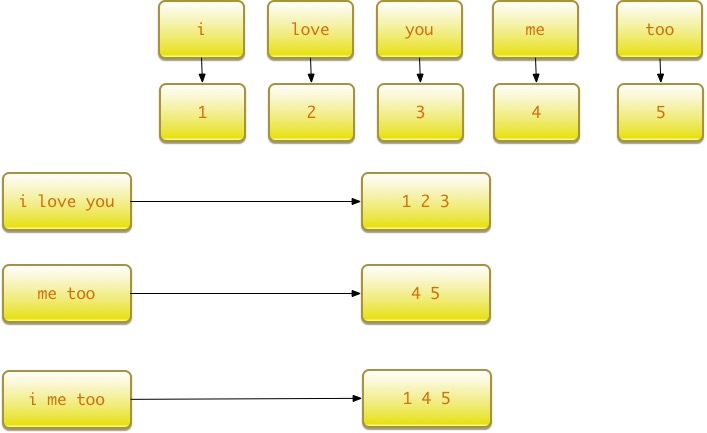
运行结果为：

[1, 4, 5, 0]

[[1 2 3 0]

[4 5 0 0]]

整个过程如图1-4 所示。



1. 使用词汇表模型进行编码

在本例中，获取完ham和spam数据后，通过VocabularyProcessor函数对数据集进行处理，获取词汇表，并按照定义的最大文本长度进行截断处理，没有达到最大文本长度的使用0填充。

global max\_document\_length  
x=[]  
y=[]  
ham, spam=load\_all\_files()  
x=ham+spam  
y=[0]\*len(ham)+[1]\*len(spam)  
vp=tflearn.data\_utils.VocabularyProcessor(max\_document\_length=max\_document\_length,  
 min\_frequency=0,  
 vocabulary=None,  
 tokenizer\_fn=None)  
x=vp.fit\_transform(x, unused\_y=None)  
x=np.array(list(x))  
return x,y

## 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，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)  
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，整个系统准确度为94.33%，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 | 5937 | 632 |
| F | 133 | 6875 |

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=5000, 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, vocabulary=None)

0.943278712835

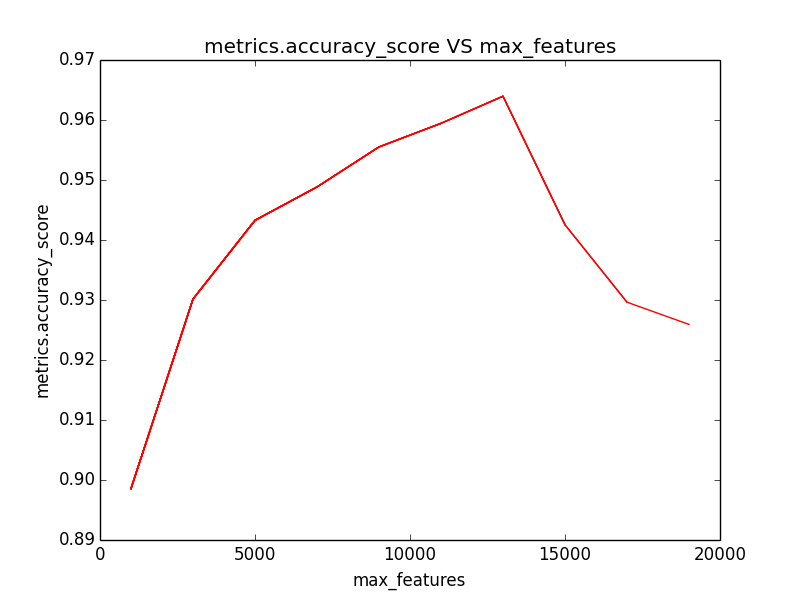
[[5937 632]

[ 133 6785]]

从调优的角度，我们试图分析词袋最大特征数max\_features对结果的影响，我们分别计算max\_features从1000到10000对评估准确度的影响。

global max\_features  
a=[]  
b=[]  
for i in range(1000,20000,2000):  
 max\_features=i  
 print "max\_features=%d" % i  
 x, y = get\_features\_by\_wordbag()  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.4, random\_state=0)  
 gnb = GaussianNB()  
 gnb.fit(x\_train, y\_train)  
 y\_pred = gnb.predict(x\_test)  
 score=metrics.accuracy\_score(y\_test, y\_pred)  
 a.append(max\_features)  
 b.append(score)  
 plt.plot(a, b, 'r')

可视化结果成图标，结果如图1-4 所示，可见max\_features值越大，模型评估准确度越高，同时整个系统运算时间也增长，当max\_features超过约13000以后，系统准确率反而下降，所以max\_features设置为13000左右，系统准确度达到最大，接近96.4%，但是通过实验，当max\_features超过5000时计算时间明显过长且对准确率提升不明显，所以折中角度max\_features取5000也满足实验要求。



1. 词袋最大特征树对朴素贝叶斯算法预测结果的影响

当max\_features设置为13000时系统运行结果为：

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=13000, 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, vocabulary=None)

0.963965299918

[[6369 200]

[ 286 6632]]

使用朴素贝叶斯算法，特征提取使用TF-IDF模型，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)  
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，整个系统准确度为95.91%，TT、FF、TF、FT矩阵如表1-2 所示，同等条件下准确率比词袋模型提升。

1. 基于TF-IDF模型的朴素贝叶斯验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 6471 | 98 |
| F | 453 | 6465 |

完整输出结果为：

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

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

lowercase=True, max\_df=1.0, max\_features=5000, 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, vocabulary=None)

TfidfTransformer(norm=u'l2', smooth\_idf=False, sublinear\_tf=False,

use\_idf=True)

NB and wordbag

0.959145844146

[[6471 98]

[ 453 6465]]

### **方法二：支持向量基算法**

使用支持向量基算法，特征提取使用词袋模型，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)  
clf = svm.SVC()  
clf.fit(x\_train, y\_train)  
y\_pred = clf.predict(x\_test)

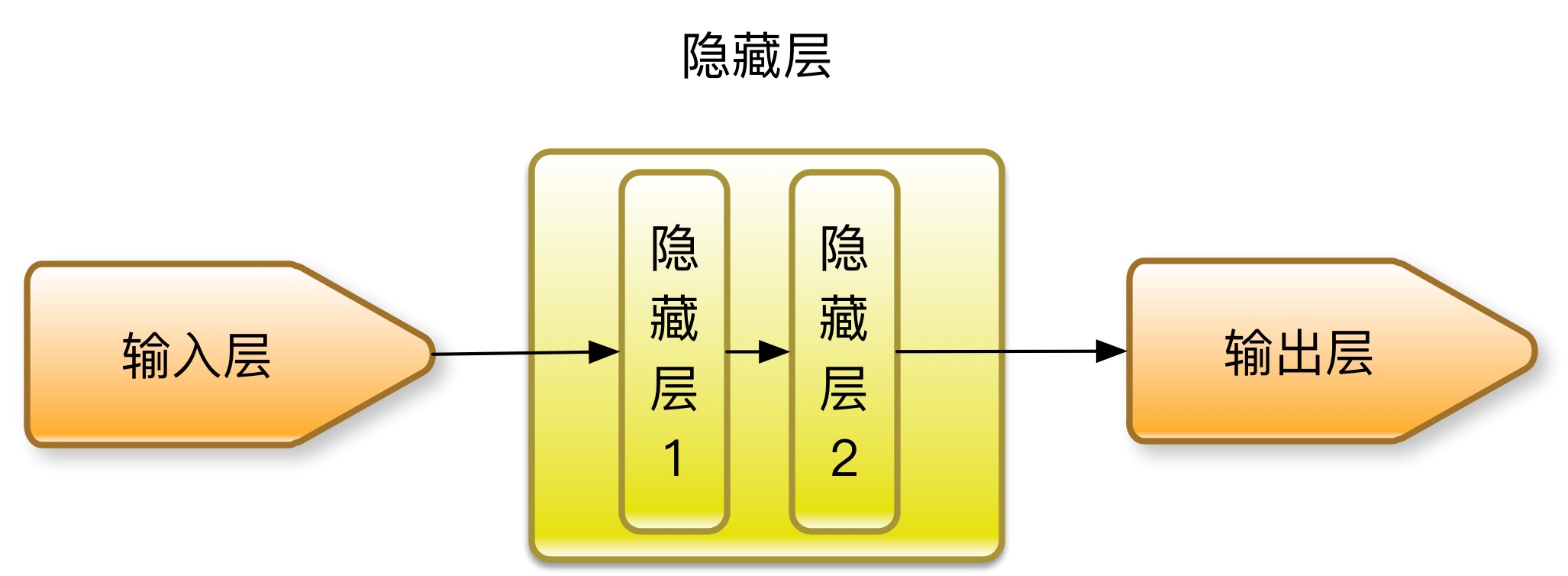
评估结果的准确度和TT、FF、TF、FT四个值。

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

### **方法三：深度学习算法之MLP**

近几年有学者尝试使用深度学习算法提高垃圾邮件识别率。

我们先使用深度学习算法最简单的一种MLP（Multi-layer Perceptron，多层感知机），我们构造包括两层隐藏层的MLP，每层节点数分别为5和2，结构如图1-5 所示。



1. 用于垃圾邮件检测的MLP结构图

在Scikit-Learn中可以使用MLPClassifier实现MLP，使用支持向量基算法，特征提取使用词袋模型，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)  
clf = MLPClassifier(solver=**'lbfgs'**,  
 alpha=1e-5,  
 hidden\_layer\_sizes = (5, 2),  
 random\_state = 1)  
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的情况下，整个系统准确度为98.01%，TT、FF、TF、FT矩阵如表1-3 所示，同等条件下准确率比词袋模型提升。

1. 基于词袋模型的MLP验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 6406 | 163 |
| F | 105 | 6813 |

完整输出结果为：

DNN and wordbag

maxlen=5000

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=(20, 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)

0.980129013124

[[6406 163]

[ 105 6813]]

### **方法四：深度学习算法之CNN**

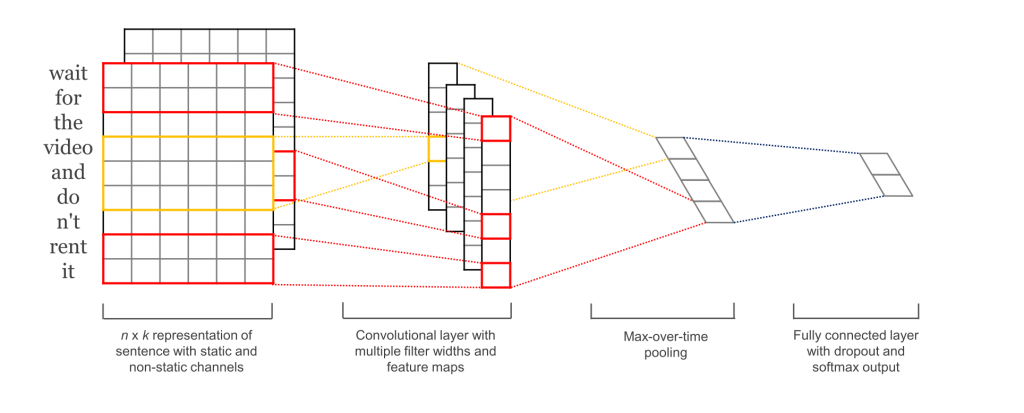
CNN的诞生是为了解决图像处理领域计算量巨大而无法进行深度学习的问题，CNN通过卷积计算、池化等大大降低了计算量，同时识别效果还满足需求。图像通常是二维数组，文字通常都是一维数据，是否可以通过某种转换后，也使用CNN对文字进行处理呢？答案是肯定的。

我们回顾下在图像处理时，CNN是如何处理二维数据的。如图1-7 所示，CNN使用二维卷积函数处理小块图像，提炼高级特征进一步分析。典型的二维卷积函数处理图片的大小为3\*3、4\*4等。



1. CNN处理图像数据的过程

同样的原理，我们可以使用一维的卷积函数处理文字片段，提炼高级特征进一步分析。典型的一维卷积函数处理文字片段的大小为3、4、5等。



1. CNN处理文本数据的过程

这个要感谢[Yoon Kim](https://arxiv.org/find/cs/1/au:+Kim_Y/0/1/0/all/0/1)的经典论文Convolutional Neural Networks for Sentence Classification

言归正传，使用词汇表编码后，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x,y=get\_features\_by\_tf()  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)

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')

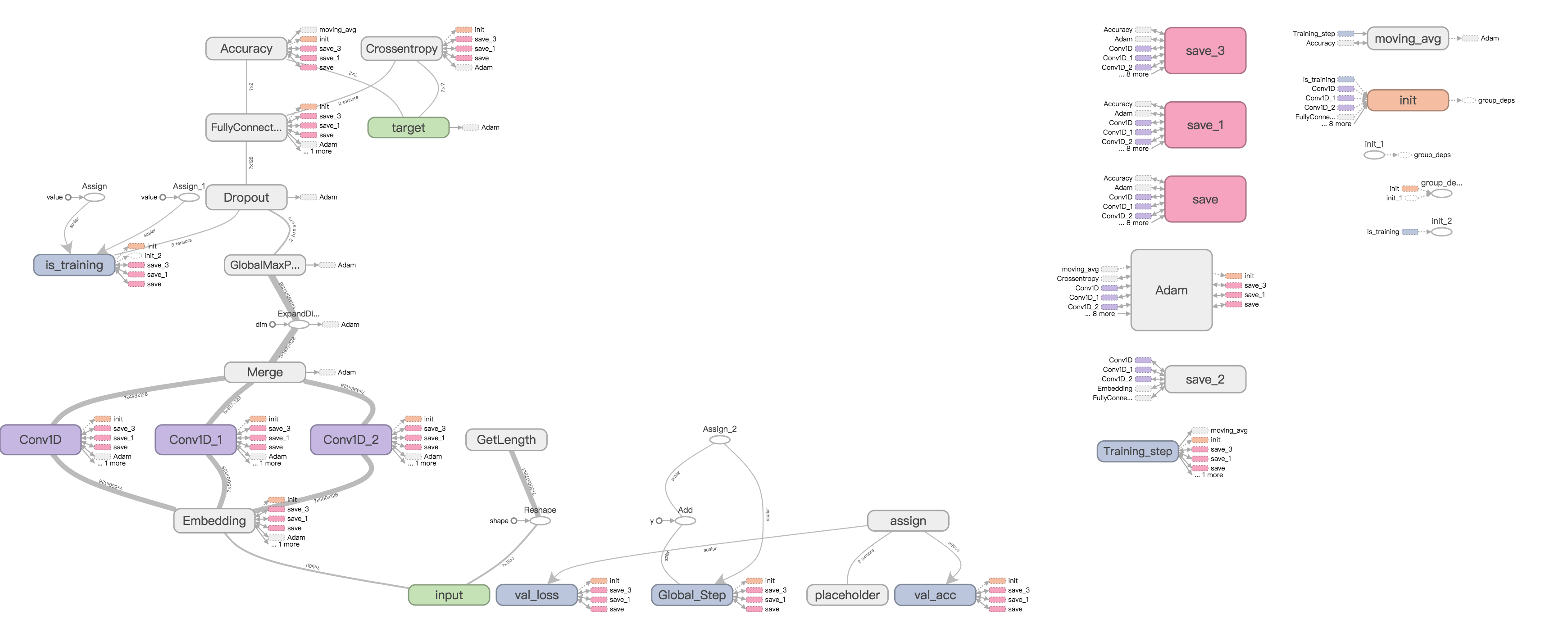
定义CNN模型，其实使用3个数量为128核，长度分别为3、4、5的一维卷积函数处理数据。

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="spam")

CNN的结构如图1-9 所示。



1. 处理垃圾邮件的CNN结构图

在我的mac本上运行了近3个小时后，对测试数据集的识别准确度达到了令人满意的98.30%。

Training Step: 680  | total loss: 0.01691 | time: 2357.838s

| Adam | epoch: 005 | loss: 0.01691 - acc: 0.9992 | val\_loss: 0.05177 - val\_acc: 0.9830 -- iter: 13524/13524

--

### **方法五：深度学习算法之RNN**

RNN由于特殊的结构，可以使用以前的记忆协助分析当前的数据，这点非常适合语言文本相关任务的处理，因此在NLP等领域使用广泛。

我们使用词汇表编码后，将数据集合随机分配成训练集合和测试集合，其中测试集合比例为40%。

x,y=get\_features\_by\_tf()  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.4, random\_state = 0)

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')

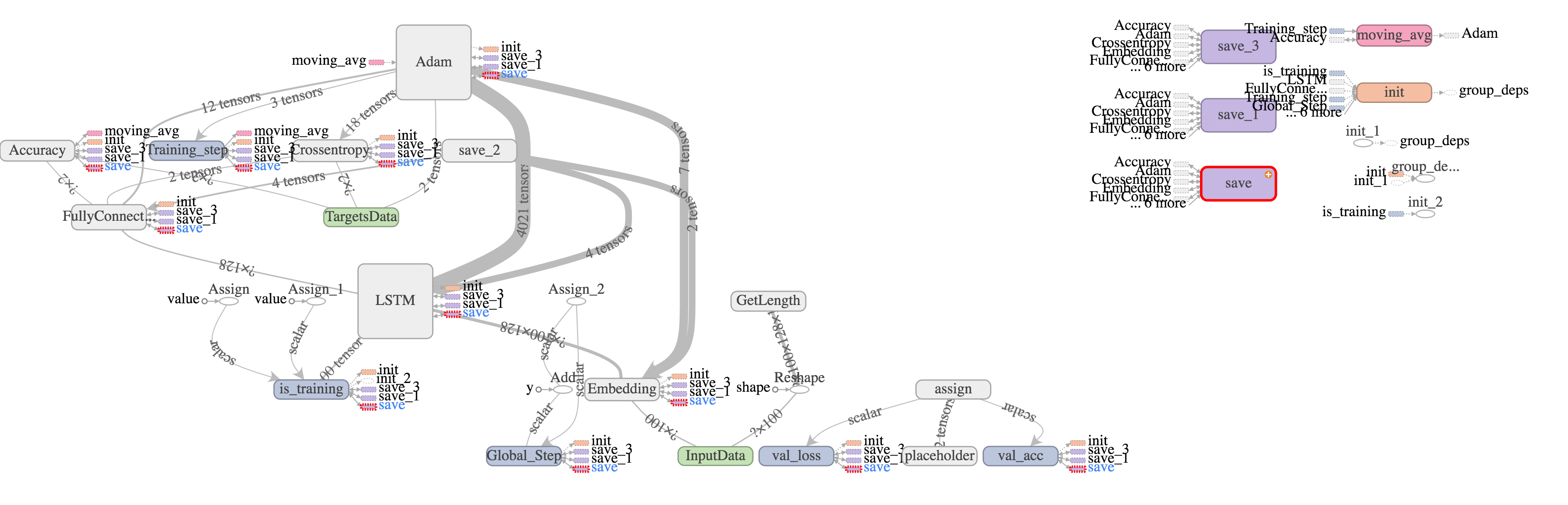
定义RNN结构，使用最简单的单层LSTM结构。

# Network building  
net = tflearn.input\_data([None, max\_document\_length])  
net = tflearn.embedding(net, input\_dim=1024000, output\_dim=128)  
net = tflearn.lstm(net, 128, dropout=0.8)  
net = tflearn.fully\_connected(net, 2, activation='softmax')  
net = tflearn.regression(net, optimizer='adam', learning\_rate=0.001,  
 loss='categorical\_crossentropy')

实例化RNN对象并进行训练数据，一共训练5轮。

# Training  
model = tflearn.DNN(net, tensorboard\_verbose=0)  
model.fit(trainX, trainY, validation\_set=(testX, testY), show\_metric=True,  
 batch\_size=10,run\_id="spm-run",n\_epoch=5)

RNN的结构如图1-10 所示。



1. 处理垃圾邮件的RNN结构图

在我的mac本上运行了近3个小时后，对测试数据集的识别准确度达到了令人满意的。

## 本章小结

### 参考文献

* <https://wenku.baidu.com/view/e41634fdcc22bcd127ff0c6b.html>
* <http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html#sklearn.feature_extraction.text.CountVectorizer>
* <http://www2.aueb.gr/users/ion/data/enron-spam/>
* http://blog.csdn.net/u013713117/article/details/69261769
* C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265.
* McCallum and K. Nigam (1998). [A comparison of event models for Naive Bayes text classification.](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.1529) Proc. AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48.
* V. Metsis, I. Androutsopoulos and G. Paliouras (2006). [Spam filtering with Naive Bayes – Which Naive Bayes?](http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.5542) 3rd Conf. on Email and Anti-Spam (CEAS).
* <http://news.ifeng.com/a/20140725/41314715_0.shtml?f=hao123>
* <http://scikit-learn.org/stable/modules/svm.html#svm>
* Ali Rodan, Hossam Faris, Ja’far Alqatawna，Optimizing Feedforward Neural Networks Using Biogeography Based Optimization for E-Mail Spam Identification，Int. J. Communications, Network and System Sciences, 2016, 9, 19-28
* Guzella, T.S. and Caminhas, W.M. (2009) A Review of Machine Learning Approaches to Spam Filtering. Expert Sys-  tems with Applications, 36, 10206-10222.
* http://dx.doi.org/10.1016/j.eswa.2009.02.037
* Rao, J.M. and Reiley, D.H. (2012) The Economics of Spam. The Journal of Economic Perspectives, 26, 87-110.  http://dx.doi.org/10.1257/jep.26.3.87
* Stern, H., et al. (2008) A Survey of Modern Spam Tools. CiteSeer.
* Su, M.-C., Lo, H.-H. and Hsu, F.-H. (2010) A Neural Tree and Its Application to Spam E-Mail Detection. Expert Sys-  tems with Applications, 37, 7976-7985.
* http://dx.doi.org/10.1016/j.eswa.2010.04.038
* Kanich, C., Weaver, N., McCoy, D. Halvorson, T., Kreibich, C., Levchenko, K., Paxson, V., Voelker, G.M. and Sa-  vage, S. (2011) Show Me the Money: Characterizing Spam-Advertised Revenue. USENIX Security Symposium, 15.
* <http://www.chinaz.com/web/2012/0521/252930.shtml>
* <http://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting>
* <http://scikit-learn.org/dev/modules/neural_networks_supervised.html#multi-layer-perceptron>
* [“Learning representations by back-propagating errors.”](http://www.iro.umontreal.ca/~pift6266/A06/refs/backprop_old.pdf) Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams.
* [“Stochastic Gradient Descent”](http://leon.bottou.org/projects/sgd) L. Bottou - Website, 2010.
* [“Backpropagation”](http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm) Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen - Website, 2011.
* [“Efficient BackProp”](http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf) Y. LeCun, L. Bottou, G. Orr, K. Müller - In Neural Networks: Tricks of the Trade 1998.
* [“Adam: A method for stochastic optimization.”](http://arxiv.org/pdf/1412.6980v8.pdf) Kingma, Diederik, and Jimmy Ba. arXiv preprint arXiv:1412.6980 (2014)
* Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng,and Christopher Potts. (2011).Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)
* <http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/>
* [Yoon Kim](https://arxiv.org/find/cs/1/au:+Kim_Y/0/1/0/all/0/1)，Convolutional Neural Networks for Sentence Classification，EMNLP 2014
* <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>
* https://github.com/clayandgithub/zh\_cnn\_text\_classify