文本分类 | 我爱自然语言处理

笔记本: 自然语言处理
创建时间: 2018/7/30 9:48
URL: http://www.52nlp.cn/category/text-classification

分类目录归档：文本分类

本文首先会介绍一些预备知识，比如softmax、ngram等，然后简单介绍word2vec原理，之后来讲解fastText的原理，并着手使用keras搭建一个简单的fastText分类器，最后，我们会介绍fastText在达观数据的应用。

NO.1

预备知识

1 Softmax回归

Softmax回归（Softmax Regression）又被称作多项逻辑回归（multinomial logistic regression），它是逻辑回归在处理多类别任务上的推广。

在逻辑回归中， 我们有m个被标注的样本：$\left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\right\}$，其中 $x^{(i)} \in R^n$ 。因为类标是二元的，所以我们有 $y^{(i)} \in \{0, 1\}$ 。我们的假设（hypothesis）有如下形式：

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

代价函数（cost function）如下：

$$J(\theta) = -\left[\sum_{i=1}^{m} y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log\ (1 - h_\theta(x^{(i)}))\right]$$

在Softmax回归中，类标是大于2的，因此在我们的训练集 $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$ 中， $y^{(i)} \in \{1, 2, \ldots, K\}$ 中， $y^{(i)} \in \{1, 2, \ldots, K\}$ 。给定一个测试输入x，我们的假设应该输出一个K维的向量，向量内每个元素的值表示x属于当前类别的概率。具体地，假设 $h_\theta(x)$ 形式如下：

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} e^{\theta^{(j)T} x}} \begin{bmatrix} e^{\theta^{(1)T} x} \\ e^{\theta^{(2)T} x} \\ \vdots \\ e^{\theta^{(K)T} x} \end{bmatrix}$$

代价函数如下：

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\}\log\frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}}\right]$$

其中1{·}是指示函数，即1=1,1=0

既然我们说Softmax回归是逻辑回归的推广，那我们是否能够在代价函数上推导出它们的一致性呢？当然可以，于是：

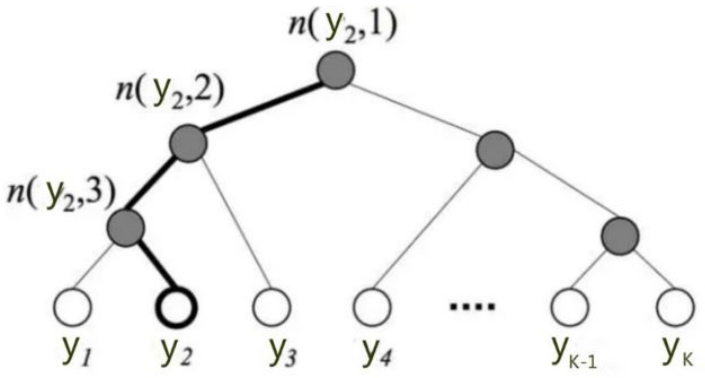
$$\begin{aligned} J(\theta) &= -\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \\ &= -\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\}\log P\left(y^{(i)} = k\middle|x^{(i)};\theta\right) \\ &= -\sum_{i=1}^{m}\sum_{k=0}^{1} 1\left\{y^{(i)} = k\right\}\log\frac{e^{\theta^{(k)T}x^{(i)}}}{\sum_{j=1}^{K} e^{\theta^{(j)T}x^{(i)}}} \end{aligned}$$

可以看到，逻辑回归是softmax回归在K=2时的特例。

2 分层Softmax

你可能也发现了，标准的Softmax回归中，要计算y=j时的Softmax概率：$P(y=j)$，我们需要对所有的K个概率做归一化，这在|y|很大时非常耗时。于是，分层Softmax诞生了，它的基本思想是使用树的层级结构替代扁平化的标准Softmax，使得在计算 $P(y=j)$ 时，只需计算一条路径上的所有节点的概率值，无需在意其它的节点。

下图是一个分层Softmax示例：



树的结构是根据类标的频数构造的霍夫曼树。K个不同的类标组成所有的叶子节点，K-1个内部节点作为内部参数，从根节点到某个叶子节点经过的节点和边形成一条路径，路径长度被表示为 $L(y_j)$ 。于是， $P(y_j)$ 就可以被写成：

$$P(y_j) = \prod_{l=1}^{L(y_j)-1} \sigma([\![n(y_j, l+1) = LC\left(n(y_j, l)\right)]\!] \cdot \theta_{n(y_j,l)}{}^T X)$$

其中：

$\sigma(\cdot)$ 表示sigmoid函数；

$LC(n)$ 表示n节点的左孩子；

$[\![x]\!]$ 是一个特殊的函数，被定义为：

$$[\![x]\!] = \begin{cases} 1 & \text{if x == true} \\ -1 & \text{otherwise} \end{cases}$$

$\theta_{n(y_j,l)}$ 是中间节点 $n(y_j, l)$ 的参数；X是Softmax层的输入。

上图中，高亮的节点和边是从根节点到 y_2 的路径，路径长度 $L(y_2) = 4, P(y_2)$

可以被表示为：

$$\begin{aligned} P(y_2) &= P(n(y_2, 1), \text{left}) \cdot P(n(y_2, 2), \text{left}) \cdot P(n(y_2, 3), \text{right}) \\ &= \sigma(\theta_{n(y_2,1)}{}^T X) \cdot \sigma(\theta_{n(y_2,2)}{}^T X) \cdot \sigma(-\theta_{n(y_2,3)}{}^T X) \end{aligned}$$

于是，从根节点走到叶子节点 y_2 ，实际上是在做了3次二分类的逻辑回归。

通过分层的Softmax，计算复杂度一下从|K|降低到log|K|。

---

3 n-gram特征

在文本特征提取中，常常能看到n-gram的身影。它是一种基于语言模型的算法，基本思想是将文本内容按照字节顺序进行大小为N的滑动窗口操作，最终形成长度为N的字节片段序列。看下面的例子：
我来到达观数据参观

相应的bigram特征为：我来 来到 到达 达观 观数 数据 据参 参观

相应的trigram特征为：我来到 来到达 到达观 达观数 观数据 数据参 据参观

注意一点：n-gram中的gram根据粒度不同，有不同的含义。它可以是字粒度，也可以是词粒度的。上面所举的例子属于字粒度的n-gram，词粒度的n-gram看下面例子：

我 来到 达观数据 参观

相应的bigram特征为：我/来到 来到/达观数据 达观数据/参观

相应的trigram特征为：我/来到/达观数据 来到/达观数据/参观
n-gram产生的特征只是作为文本特征的候选集，你后面可能会采用信息熵、卡方统计、IDF等文本特征选择方式筛选出比较重要特征。

NO.2

Word2vec
你可能要问，这篇文章不是介绍fastText的么，怎么开始介绍起了word2vec?

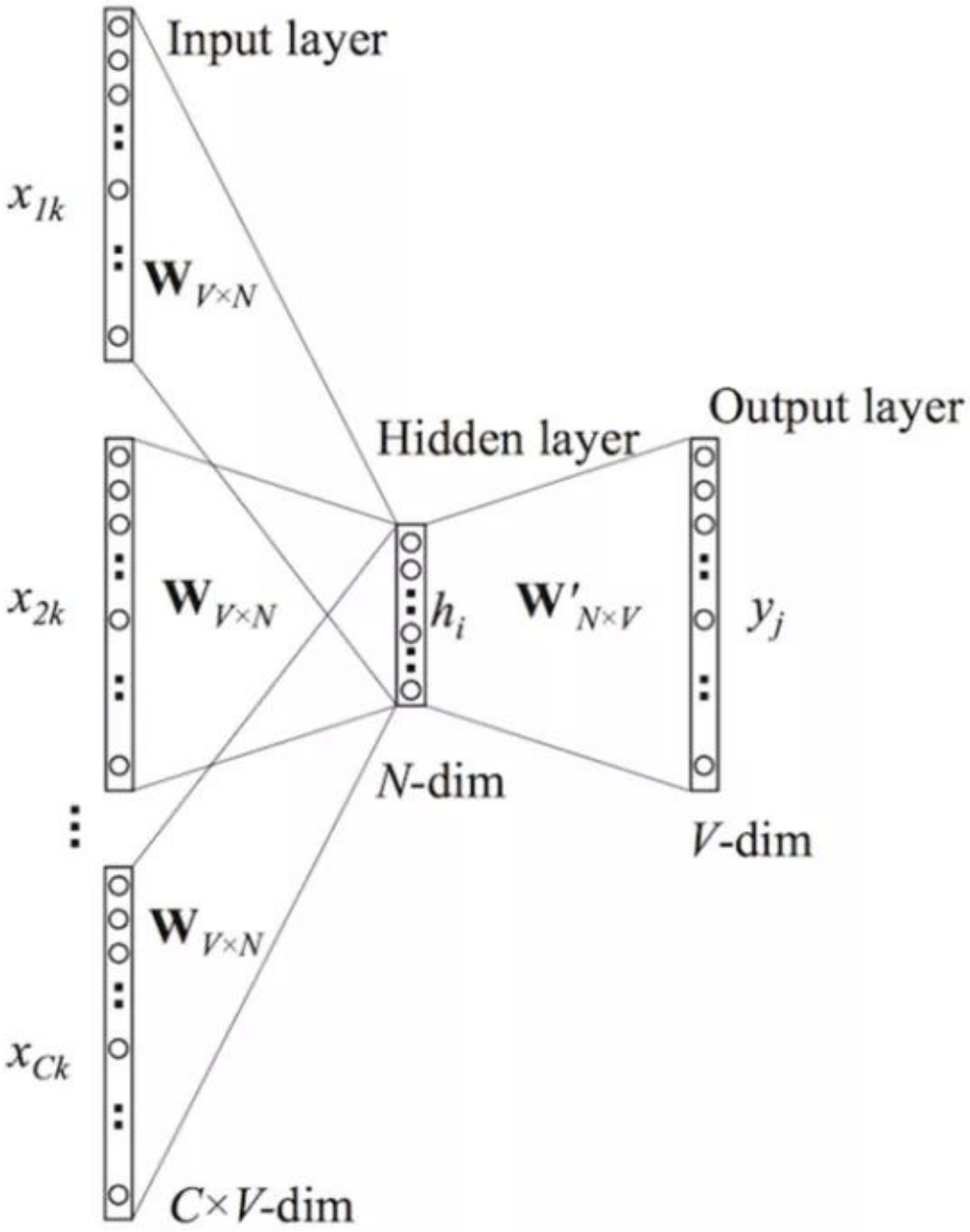
最主要的原因是word2vec的CBOW模型架构和fastText模型非常相似。于是，你看到facebook开源的fastText工具不仅实现了fastText文本分类工具，还实现了快速词向量训练工具。word2vec主要有两种模型：skip-gram 模型和CBOW模型，这里只介绍CBOW模型，有关skip-gram模型的内容请参考达观另一篇技术文章：

漫谈Word2vec之skip-gram模型

---

1 模型架构

CBOW模型的基本思路是：用上下文预测目标词汇。架构图如下所示：



输入层由目标词汇y的上下文单词 $\{x_1, \ldots, x_c\}$ 组成， x_i 是被onehot编码过的V维向量，其中V是词汇量；隐含层是N维向量h；输出层是被onehot编码过的目标词y。输入向量通过 $V * N$ 维的权重矩阵W连接到隐含层；隐含层通过 $N * V$ 维的权重矩阵 W' 连接到输出层。因为词库V往往非常大，使用标准的softmax计算相当耗时，于是CBOW的输出层采用的正是上文提到过的分层Softmax。

2 前向传播

输入是如何计算而获得输出呢？先假设我们已经获得了权重矩阵 W 和 W' （具体的推导见第3节），隐含层h的输出的计算公式：

$$\mathrm{h} = \frac{1}{C} W \cdot (\sum_{i=1}^{C} x_i)$$

即：隐含层的输出是C个上下文单词向量的加权平均，权重为 W。

接着我们计算输出层的每个节点：

$$u_j = {v'}_{w_j}^T \cdot h$$

这里 ${v'}_{w_j}$ 是矩阵 W' 的第j列，最后，将 u_j 作为softmax函数的输入，得到 y_j ：

$$y_j = p\left(w_{y_j} \middle| w_1, \ldots, w_C\right) = \frac{\exp\left(u_j\right)}{\sum_{j'=1}^{V} \exp\left(u_j'\right)}$$

---

3 反向传播学习权重矩阵

在学习权重矩阵和过程中，我们首先随机产生初始值，然后feed训练样本到我们的模型，并观测我们期望输出和真实输出的误差。接着，我们计算误差关于权重矩阵的梯度，并在梯度的方向纠正它们。

首先定义损失函数，objective是最大化给定输入上下文，target单词的条件概率。因此，损失函数为：

$$\begin{aligned} \mathrm{E} &= -\log p(w_O \mid w_I) \\ &= -u_{j^*} - \log \sum_{j'=1}^{V} \exp\left(u_{j'}\right) \\ &= -{v_{w_O}}^T \cdot h - \log \sum_{j'=1}^{V} \exp\left({v_{w_{j'}}}^T \cdot h\right) \end{aligned}$$

这里， j^* 表示目标单词在词库V中的索引。

如何更新权重 W' ?

如何更新权重 W'_{ij}？

我们先对E关于 W'_{ij} 求导：

$$\begin{aligned}\frac{\partial E}{\partial w'_{ij}} &= \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} \\ &= (y_j - \mathbb{I}j = j^*\mathbb{I}) \cdot h_i\end{aligned}$$

$\mathbb{I} j = j^* \mathbb{I}$ 函数表示：

$$\mathbb{I}j = j^*\mathbb{I} = \begin{cases} 1 & \text{if j == j}^* \\ -1 & \text{otherwise} \end{cases}$$

于是，W'_{ij} 的更新公式：

$$w_{ij}^{\prime(new)} = w_{ij}^{\prime(old)} - \eta \cdot (y_j - \mathbb{I}j = j^*\mathbb{I}) \cdot h$$

如何更新权重 *W*?

我们首先计算E关于隐含层节点的导数：

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} (y_j - \mathbb{I}j = j^*\mathbb{I}) \cdot w'_{ij}$$

然后，E关于权重的导数为：

$$\begin{aligned}\frac{\partial E}{\partial w_{ki}} &= \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} \\ &= \sum_{j=1}^{V} \left(y_j - \mathbb{I}j = j^*\mathbb{I}\right) \cdot w'_{ij} \cdot \frac{1}{C} \cdot x_k\end{aligned}$$

于是，w_{ki} 的更新公式：

$$w_{ki}^{(new)} = w_{ki}^{(old)} - \eta \cdot \sum^{V} \left(y_j - \mathbb{I}j = j^*\mathbb{I}\right) \cdot w'_{ij} \cdot \frac{1}{C} \cdot x_k$$

$j=1$

NO.3

fastText分类
终于到我们的fastText出场了。这里有一点需要特别注意，一般情况下，使用fastText进行文本分类的同时也会产生词的embedding，即embedding是fastText分类的产物。除非你决定使用预训练的embedding来训练fastText分类模型，这另当别论。

1 字符级别的n-gram
word2vec把语料库中的每个单词当成原子的，它会为每个单词生成一个向量。这忽略了单词内部的形态特征，比如：“apple” 和“apples”，“达观数据”和“达观”，这两个例子中，两个单词都有较多公共字符，即它们的内部形态类似，但是在传统的word2vec中，这种单词内部形态信息因为它们被转换成不同的id丢失了。

为了克服这个问题，fastText使用了字符级别的n-grams来表示一个单词。对于单词“apple”，假设n的取值为3，则它的trigram有:

“<ap”，“app”，“ppl”，“ple”，“le>”

其中，<表示前缀，>表示后缀。于是，我们可以用这些trigram来表示“apple”这个单词，进一步，我们可以用这5个trigram的向量叠加来表示“apple”的词向量。

这带来两点好处：

1. 对于低频词生成的词向量效果会更好。因为它们的n-gram可以和其它词共享。

2. 对于训练词库之外的单词，仍然可以构建它们的词向量。我们可以叠加它们的字符级n-gram向量。

2 模型架构
之前提到过，fastText模型架构和word2vec的CBOW模型架构非常相似。下面是fastText模型架构图：

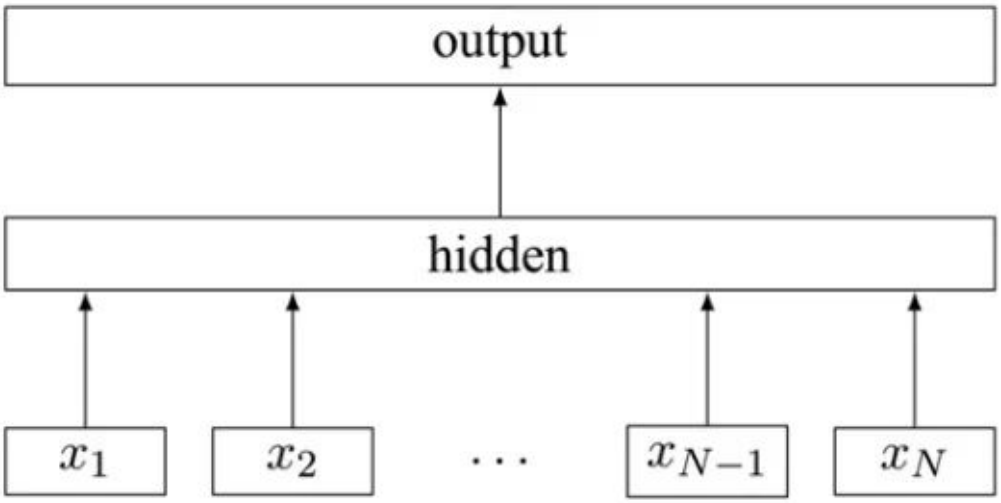


Figure 1: Model architecture of `fastText` for a sentence with N ngram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

注意：此架构图没有展示词向量的训练过程。可以看到，和CBOW一样，fastText模型也只有三层：输入层、隐含层、输出层（Hierarchical Softmax），输入都是多个经向量表示的单词，输出都是一个特定的target，隐含层都是对多个词向量的叠加平均。

不同的是，CBOW的输入是目标单词的上下文，fastText的输入是多个单词及其n-gram特征，这些特征用来表示单个文档；CBOW的输入单词被onehot编码过，fastText的输入特征是被embedding过；CBOW的输出是目标词汇，fastText的输出是文档对应的类标。

值得注意的是，fastText在输入时，将单词的字符级别的n-gram向量作为额外的特征；在输出时，fastText采用了分层Softmax，大大降低了模型训练时间。这两个知识点在前文中已经讲过，这里不再赘述。

fastText相关公式的推导和CBOW非常类似，这里也不展开了。

---

3 核心思想

现在抛开那些不是很讨人喜欢的公式推导，来想一想fastText文本分类的核心思想是什么?

仔细观察模型的后半部分，即从隐含层输出到输出层输出，会发现它就是一个softmax线性多类别分类器，分类器的输入是一个用来表征当前文档的向量；模型的前半部分，即从输入层输入到隐含层输出部分，主要在做一件事情：生成用来表

的那个部分，即从输入层输入到隐含层输出部分，主要做一件事情：生成用来表征文档的向量。那么它是如何做的呢？叠加构成这篇文档的所有词及n-gram的词向量，然后取平均。叠加词向量背后的思想就是传统的词袋法，即将文档看成一个由词构成的集合。

于是fastText的核心思想就是：将整篇文档的词及n-gram向量叠加平均得到文档向量，然后使用文档向量做softmax多分类。这中间涉及到两个技巧：字符级n-gram特征的引入以及分层Softmax分类。

---

4 关于分类效果

还有个问题，就是为何fastText的分类效果常常不输于传统的非线性分类器?

假设我们有两段文本：

我 来到 达观数据

俺 去了 达而观信息科技

这两段文本意思几乎一模一样，如果要分类，肯定要分到同一个类中去。但在传统的分类器中，用来表征这两段文本的向量可能差距非常大。传统的文本分类中，你需要计算出每个词的权重，比如tfidf值， “我”和“俺” 算出的tfidf值相差可能会比较大，其它词类似，于是，VSM（向量空间模型）中用来表征这两段文本的文本向量差别可能比较大。

但是fastText就不一样了，它是用单词的embedding叠加获得的文档向量，词向量的重要特点就是向量的距离可以用来衡量单词间的语义相似程度，于是，在fastText模型中，这两段文本的向量应该是非常相似的，于是，它们很大概率会被分到同一个类中。

使用词embedding而非词本身作为特征，这是fastText效果好的一个原因；另一个原因就是字符级n-gram特征的引入对分类效果会有一些提升 。

NO.4

手写一个fastText

keras是一个抽象层次很高的神经网络API，由python编写，底层可以基于Tensorflow、Theano或者CNTK。它的优点在于：用户友好、模块性好、易扩展等。所以下面我会用keras简单搭一个fastText的demo版，生产可用的fastText请移步https://github.com/facebookresearch/fastText。

如果你弄懂了上面所讲的它的原理，下面的demo对你来讲应该是非常明了的。

为了简化我们的任务：

1. 训练词向量时，我们使用正常的word2vec方法，而真实的fastText还附加了字符级别的n-gram作为特征输入；

2. 我们的输出层使用简单的softmax分类，而真实的fastText使用的是Hierarchical Softmax。

首先定义几个常量：

VOCAB_SIZE = 2000

EMBEDDING_DIM =100

MAX_WORDS = 500

CLASS_NUM = 5

VOCAB_SIZE表示词汇表大小，这里简单设置为2000；

EMBEDDING_DIM表示经过embedding层输出，每个词被分布式表示的向量的维度，这里设置为100。比如对于“达观”这个词，会被一个长度为100的类似于[0.97860014, 5.93589592, 0.22342691, -3.83102846, -0.23053935, ...]的实值向量来表示；

MAX_WORDS表示一篇文档最多使用的词个数，因为文档可能长短不一（即词数不同），为了能feed到一个固定维度的神经网络，我们需要设置一个最大词数，对于词数少于这个阈值的文档，我们需要用“未知词”去填充。比如可以设置词汇表中索引为0的词为“未知词”，用0去填充少于阈值的部分；

CLASS_NUM表示类别数，多分类问题，这里简单设置为5。

模型搭建遵循以下步骤：

1. 添加输入层（embedding层）。Embedding层的输入是一批文档，每个文档由一个词汇索引序列构成。例如：[10, 30, 80, 1000] 可能表示“我 昨天 来到 达观数据”这个短文本，其中“我”、“昨天”、“来到”、“达观数据”在词汇表中的索引分别是10、30、80、1000；Embedding层将每个单词映射成

EMBEDDING_DIM维的向量。于是：input_shape=(BATCH_SIZE, MAX_WORDS), output_shape=(BATCH_SIZE,MAX_WORDS, EMBEDDING_DIM)；

2. 添加隐含层（投影层）。投影层对一个文档中所有单词的向量进行叠加平均。keras提供的GlobalAveragePooling1D类可以帮我们实现这个功能。这层的input_shape是Embedding层的output_shape，这层的output_shape=(BATCH_SIZE, EMBEDDING_DIM)；

3. 添加输出层（softmax层）。真实的fastText这层是Hierarchical Softmax，因为keras原生并没有支持Hierarchical Softmax，所以这里用Softmax代替。这层指定了CLASS_NUM，对于一篇文档，输出层会产生CLASS_NUM个概率值，分别表示此文档属于当前类的可能性。这层的output_shape=(BATCH_SIZE, CLASS_NUM)

4. 指定损失函数、优化器类型、评价指标，编译模型。损失函数我们设置为categorical_crossentropy，它就是我们上面所说的softmax回归的损失函数；优化器我们设置为SGD，表示随机梯度下降优化器；评价指标选择accuracy，表示精度。

用训练数据feed模型时，你需要：

1. 将文档分好词，构建词汇表。词汇表中每个词用一个整数（索引）来代替，并预留“未知词”索引，假设为0；

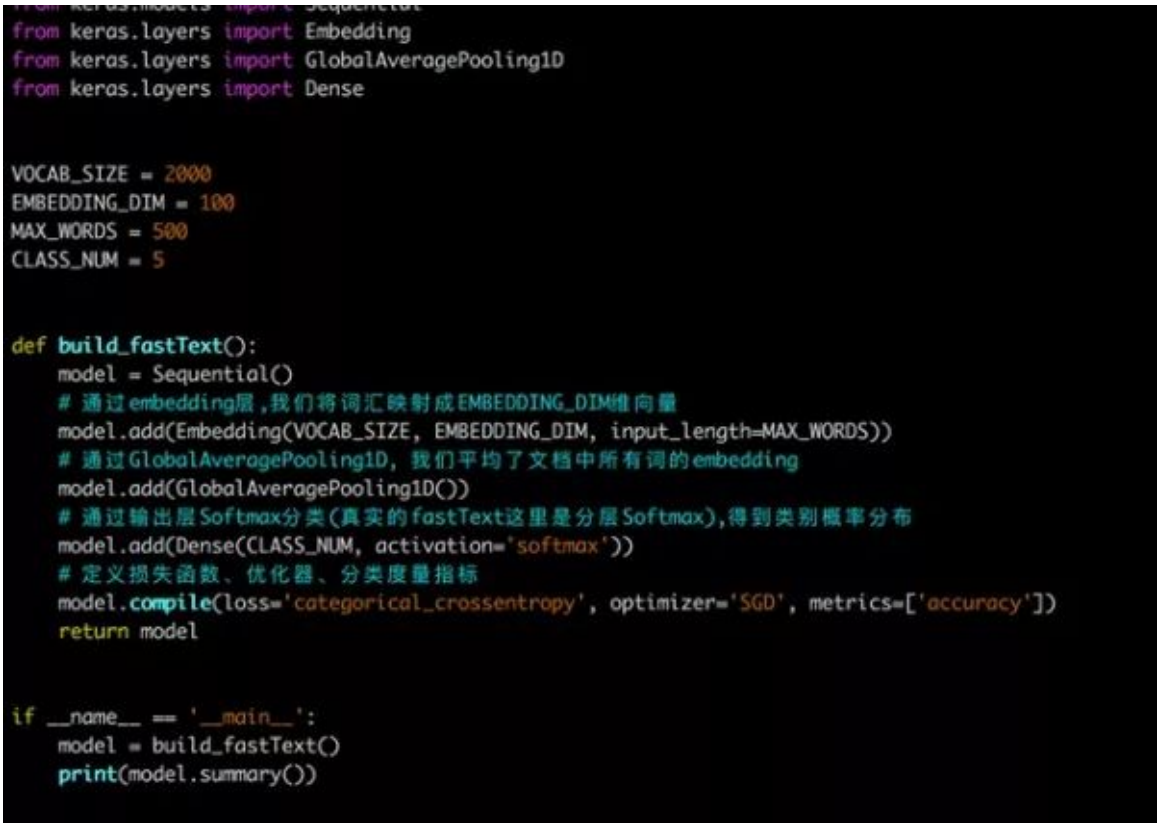
2. 对类标进行onehot化。假设我们文本数据总共有3个类别，对应的类标分别是1、2、3，那么这三个类标对应的onehot向量分别是[1, 0, 0]、[0, 1, 0]、[0, 0, 1]；

3. 对一批文本，将每个文本转化为词索引序列，每个类标转化为onehot向量。就像之前的例子，“我 昨天 来到 达观数据”可能被转化为[10, 30, 80, 1000]；它属于类别1，它的类标就是[1, 0, 0]。由于我们设置了MAX_WORDS=500，这个短文本向量后面就需要补496个0，即[10, 30, 80, 1000, 0, 0, 0, ..., 0]。因此，batch_xs的 维度为(BATCH_SIZE,MAX_WORDS)，batch_ys的维度为（BATCH_SIZE, CLASS_NUM）。

下面是构建模型的代码，数据处理、feed数据到模型的代码比较繁琐，这里不展示。

```
# coding: utf-8
from __future__ import unicode_literals

from keras.models import Sequential
```



```
from keras.layers import Embedding
from keras.layers import GlobalAveragePooling1D
from keras.layers import Dense


VOCAB_SIZE = 2000
EMBEDDING_DIM = 100
MAX_WORDS = 500
CLASS_NUM = 5


def build_fastText():
    model = Sequential()
    # 通过embedding层,我们将词汇映射成EMBEDDING_DIM维向量
    model.add(Embedding(VOCAB_SIZE, EMBEDDING_DIM, input_length=MAX_WORDS))
    # 通过GlobalAveragePooling1D, 我们平均了文档中所有词的embedding
    model.add(GlobalAveragePooling1D())
    # 通过输出层Softmax分类(真实的fastText这里是分层Softmax),得到类别概率分布
    model.add(Dense(CLASS_NUM, activation='softmax'))
    # 定义损失函数、优化器、分类度量指标
    model.compile(loss='categorical_crossentropy', optimizer='SGD', metrics=['accuracy'])
    return model


if __name__ == '__main__':
    model = build_fastText()
    print(model.summary())
```

NO.5

fastText原理及实践

fastText在达观数据的应用
fastText作为诞生不久的词向量训练、文本分类工具，在达观得到了比较深入的应用。主要被用在以下两个系统：

1. 同近义词挖掘。Facebook开源的fastText工具也实现了词向量的训练，达观基于各种垂直领域的语料，使用其挖掘出一批同近义词；

2. 文本分类系统。在类标数、数据量都比较大时，达观会选择fastText 来做文本分类，以实现快速训练预测、节省内存的目的。

本条目发布于2018年01月25号。属于数据挖掘、文本分类、自然语言处理分类，被贴了 fastText、文本分类 标签。作者是recommender。

背景

CIKM Cup(或者称为CIKM Competition)是ACM CIKM举办的国际数据挖掘竞赛的名称。CIKM全称是International Conference on Information and Knowledge Management，属于信息检索和数据挖掘领域的国际著名学术会议，由ACM SIGIR分会（ACM Special Interest Group on Information Retrieval）主办。

随着数据挖掘技术越来越重要，CIKM会议的影响力也水涨船高，逐渐逼近KDD、WWW、ICDE。2014年是CIKM第一次在中国大陆举办，邀请了Google大神Jeff Dean，微软EVP陆奇博士和德国Max Planck Institute的Gerhard Weikum教授担任Keynote Speaker，盛况空前。CIKM很重视工业界的运用，既有面向工业届的Tutorial/Workshop，也有CIKM Cup这样面向实战的国际数据挖掘竞赛（类似另一个著名的数据挖掘竞赛KDD Cup），比赛使用真实的工业界数据和应用课题，让全世界的数据挖掘选手们一较高下。

今年的CIKM Cup竞赛的题目是自动识别用户的查询意图（Query Intent Detection，QID），主办方提供了来自百度线上的真实的用户查询和点击的数据（总行数为6141万行），竞赛目标是根据已标注的用户行为数据，来判断其中用户查询时的真实意图，要求识别的准确率和召回率越高越好。比赛历时2个半月，共吸引了520支队伍参赛，最终我们的队伍Topdata脱颖而出，所提出的算法以F1值0.9296排名Final Leaderboard第一获得冠军！

Final LeaderBoard

Rank	Name	Best Quiz Score	Best Submit Time
1	topdata	0.9296	Sep 30 2014 23:59:15 (PDT)
2	FAndy	0.9245	Sep 30 2014 23:15:04 (PDT)
3	adfr	0.9222	Sep 30 2014 03:44:32 (PDT)
4	yingwei_xin	0.9220	Sep 30 2014 23:57:42 (PDT)
5	fancyspeed	0.9203	Sep 30 2014 12:49:38 (PDT)
6	senochow	0.9181	Sep 30 2014 08:47:20 (PDT)
7	lhd911107	0.9165	Sep 29 2014 06:13:01 (PDT)

8	tbinjiayou	0.9155	Sep 23 2014 19:19:01 (PDT)
9	klb3713	0.9155	Sep 23 2014 20:31:01 (PDT)
10	db2george	0.9145	Sep 29 2014 00:01:19 (PDT)

应很多朋友的邀请，发表这篇文章详细介绍我们使用的方法，给对大数据挖掘算法感兴趣的朋友们作个参考。另外在领奖现场我们和其他参赛队伍作了愉快的交流，因此本文也吸收了其他队伍的一些优秀思路，可以看作是这次竞赛整体方法和对策的总结。文章最后还附上了一些我个人的参赛感言（陈运文）。
继续阅读 →

本条目发布于2014年11月20号。属于中文信息处理、数据挖掘、文本分类、机器学习、自然语言处理、计算语言学、随笔分类，被贴了 数据挖掘、文本分析 标签。作者是recommender。

注：博文转载、语料库使用，请注明提供者、来源以及空间提供方。

免责声明：此语料库仅供自然语言处理的业余爱好者研究和交流，禁止用于任何商业用途（包括在资源内部链接广告等行为）。

感谢网易新闻中心、腾讯新闻中心、凤凰新闻中心以及新浪新闻中心提供新闻素材。新闻著作权归以上网站所有，任何人未经上述公司允许不得抄袭。

语料库下载地址：http://download.cnblogs.com/finallyliuyu/corpus.rar

语料素材来源：　　凤凰新闻中心、网易新闻中心、腾讯新闻中心、新浪新闻中心。

语料库整理提供者：　finallyliuyu

语料库空间提供方：　博客园（无偿提供）

说明：

1、此语料库非职务作品，由本人在业余时间搜集整理，免费提供给对NLP狂热的业余爱好者学习研究使用；本人是自然语言处理的业余爱好者，在类别定义等方面都可能存在一些欠缺，欢迎大家提出宝贵意见和建议；

2、下载地址提供的是MS SQL2000数据库的备份文件。使用此数据库，您需要安

装 MS SQL2000 server，然后将corpus.rar解压并还原。压缩包大小为54.8M，共包含39247篇新闻，分为历史、军事、文化、读书、教育、IT、娱乐、社会与法制等八个类别。历史类、文化类、读书类新闻来自于凤凰网，IT类的新闻全部来自tech.qq，教育类的新闻来自edu.qq，娱乐类的新闻来自网易。社会与法制类的新闻来自于新浪和腾讯的几个版面；

3、需要特别注意的是，有的新闻在开头处有大量空白，因此在查询数据库ArticleText字段中有大片空白的，不是空新闻，是整个新闻体截断显示的缘故。

4、有关语料库的其他情况，请参考《献给热衷于自然语言处理的业余爱好者的中文新闻分类语料库之一》。

我本人在此语料库做过的验证性实验有：《KL语义距离计算系列》，《Kmeans聚类系列》以及《文本分类和特征词选择系列》。

感谢DUDU在博客园无偿帮忙提供空间；也感谢博客园团队。衷心祝愿你们越办越好！

本条目发布于2010年12月11号。属于文本分类、语料库分类，被贴了 文本分类、新闻语料 标签。作者是finallyliuyu。

前些天一个学弟发邮件咨询有关自动作文评分的问题，在了解了这是他们导师布置的一个任务后，出于做统计机器翻译的惯性思维，我马上想到的是利用语言模型对作文进行流利度方面的打分，但也意识到这是一个粗糙的甚至是错误的评分系统，因为它连最基本的作文长度都没有考虑。 继续阅读 →

本条目发布于2009年06月28号。属于文本分类、自然语言处理分类，被贴了 BETSY、Electronic Essay Rater、Intelligent Essay Assessor、IntelliMetric、Larkey、Project Essay Grade、文本分类、自动作为评分、自然语言处理 标签。作者是52nlp。