情感分析的新方法

情感分析

f(x) 2016-09-14 23 阅读

情感分析是自然语言处理（NLP）中的常见的方法应用，是文本分析的一种。所谓文本分析指的是从文字来源中推导出知识和洞见的过程。而情感分析则进一步从文字中自动识别出人们对特定主题的主观看法、情绪以及态度等等资讯。通过这种方式，情感分析可以被视为利用一些情感得分指标来量化定性数据的方法。尽管情绪在很大程度上是主观的，但是情感量化分析已经有很多有用的实践，比如企业分析消费者对产品的反馈信息，或者检测在线评论中的差评信息。

0 闲话

最简单的情感分析方法是利用词语的正负属性来判定。句子中的每个单词都有一个得分，乐观的单词得分为 `+1` ，悲观的单词则为 `-1` 。然后我们对句子中所有单词得分进行加总求和得到一个最终的情感总分。很明显，这种方法有许多局限之处，最重要的一点在于它忽略了上下文的信息。例如，在这个简易模型中，因为“not”的得分为 `-1` ，而“good”的得分为 `+1` ，所以词组“not good”将被归类到中性词组中。尽管词组“not good”中包含单词“good”，但是人们仍倾向于将其归类到悲观词组中。

另外一个常见的方法是将文本视为一个“词袋”。我们将每个文本看出一个 `1xN` 的向量，其中N表示文本词汇的数量。该向量中每一列都是一个单词，其对应的值为该单词出现的频数。例如，词组“bag of bag of words”可以被编码为 `[2, 2, 1]` 。这些数据可以被应用到机器学习分类算法中（比如罗吉斯回归或者支持向量机），从而预测未知数据的情感状况。需要注意的是，这种有监督学习的方法要求利用已知情感状况的数据作为训练集。虽然这个方法改进了之前的模型，但是它仍然忽略了上下文的信息和数据集的规模情况。

还有一种是基于word2vec的方法。它可以将一段句子表征为实数值向量。在获得词向量后，对词向量进行平均处理，最终获取到句子向量。然后，利用机器学习的分类算法预测句子的情感倾向。这种方法在微博等短文上的应用效果十分不错，这是因为微博通常只有十几个单词，所以即使经过平均化处理仍能保持相关的特性。一旦我们开始分析段落数据时，如果忽略上下文和单词顺序的信息，那么我们将会丢掉许多重要的信息。对于分析长篇评论，更好的方法是采用doc2vec来创建输入信息。

1 Word2vec

word2vec可以在捕捉语境信息的同时压缩数据规模。word2vec实际上是两种不同的方法：Continuous Bag of Words(CBOW)和Skip-gram。CBOW的目标是根据上下文来预测当前词语的概率。Skip-gram刚好相反：根据当前词语来预测上下文的概率。

1.1 word2vec载入

谷歌预训练好的词向量数据 GoogleNews-vectors-negative300.bin.gz 。该词向量是基于谷歌新闻数据训练所得：

```
from gensim.models.word2vec import Word2Vec
model = Word2Vec.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
```

1.2 word2vec测试

在运用到情感分析前，我们先来测试下word2vec对于单词分类的能力。我们将利用 Word Banks 三个分类样本集：食物，运动和天气的单词集合。首先获得这些词向量：

```
import numpy as np
with open('test-word2vec/food.txt', 'r') as infile:
    food_words = infile.readlines()

with open('test-word2vec/sports.txt', 'r') as infile:
    sports_words = infile.readlines()

with open('test-word2vec/weather.txt', 'r') as infile:
    weather_words = infile.readlines()

def get_word_vecs(words):
    vecs = []
    for word in words:
        word = word.strip()
        try:
            vecs.append(model[word].reshape((1, 300)))
        except KeyError:
            continue
    vecs = np.concatenate(vecs) #数组
    return np.array(vecs, dtype='float') #TSNE expects float type values
```

```
food_vecs = get_word_vecs(food_words)
sports_vecs = get_word_vecs(sports_words)
weather_vecs = get_word_vecs(weather_words)
```

然后我们利用TSNE和matplotlib对分类结果进行可视化处理：

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

ts = TSNE(2)
reduced_vecs = ts.fit_transform(np.concatenate((food_vecs, sports_vecs, weather_vecs)))
for i in range(len(reduced_vecs)):
    if i < len(food_vecs):
        #food words colored blue
        color = 'b'
    elif i >= len(food_vecs) and i < (len(food_vecs) + len(sports_vecs)):
        #sports words colored red
        color = 'r'
    else:
        #weather words colored green
        color = 'g'
    plt.plot(reduced_vecs[i, 0], reduced_vecs[i, 1], marker='o', color=color, markersize=8)
```

1.3 Emoji推文的情感分析

我们将分析带有Emoji表情推文的情感状况。我们利用emoji表情对我们的数据添加模糊的标签。笑脸表情表示乐观情绪，皱眉标签表示悲观情绪。总的400000条推文被分为乐观和悲观两组数据。我们随机从这两组数据中抽取样本，构建比例为 `8:2` 的训练集和测试集。随后，我们对训练集数据构建Word2Vec模型，其中分类器的输入值为推文中所有词向量的加权平均值。我们可以利用 **`Scikit-Learn`** 构建许多机器学习模型。

首先，我们导入数据并构建Word2Vec模型：

```
from sklearn.cross_validation import train_test_split
from gensim.models.word2vec import Word2Vec

with open('twitter_data/pos_tweets.txt', 'r') as infile:
    pos_tweets = infile.readlines()
```

```
with open('twitter_data/neg_tweets.txt', 'r') as infile:
    neg_tweets = infile.readlines()

#use 1 for positive sentiment, 0 for negative
x = np.concatenate((pos_tweets, neg_tweets)) #数组
y = np.concatenate((np.ones(len(pos_tweets)), np.zeros(len(neg_tweets))))
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

#Do some very minor text preprocessing
def clean_text(corpus):
    corpus = [z.lower().replace('\n', '').split() for z in corpus]
    return corpus

x_train = clean_text(x_train)
x_test = clean_text(x_test)

#Initialize model and build vocab
n_dim = 300
tweet_w2v = Word2Vec(size=n_dim, min_count=10)
tweet_w2v.build_vocab(x_train)
#Train the model over train_reviews (this may take several minutes)
tweet_w2v.train(x_train)
```

接下来，为了利用下面的函数获得推文中所有词向量的平均值，我们必须构建作为输入文本的词向量：

```
def build_word_vector(text, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in text:
        try:
            vec += tweet_w2v[word].reshape((1, size))
            count += 1.
        except KeyError:
            continue
    if count > 1.:
        vec /= count
    return vec
```

为了使模型更有效，许多机器学习模型需要预先处理数据集的量纲，特别是文本分类器这类具有许多变量的模型。建立训练集和测试集向量并对其标准化处理：

```
from sklearn.preprocessing import scale
train_vecs = np.concatenate([build_word_vector(z, n_dim) for z in x_train])
train_vecs = scale(train_vecs)
test_vecs = np.concatenate([build_word_vector(z, n_dim) for z in x_test])
test_vecs = scale(test_vecs)
```

在这个案例中我们使用罗吉斯回归的随机梯度下降法作为分类器算法：

```
from sklearn.linear_model import SGDClassifier

lr = SGDClassifier(loss='log', penalty='l1')
lr.fit(train_vecs, y_train)
print 'Test Accuracy: %.2f' % lr.score(test_vecs, y_test) #77%

#Create ROC curve
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

pred_probas = lr.predict_proba(test_vecs)[:, 1]
fpr, tpr, _ = roc_curve(y_test, pred_probas)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, label='area = %.2f' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.legend(loc='lower right')
plt.show()
```

2 Doc2vec

Doc2vec又名paragraph2vec或sentence2vec，除了增加一个段落向量以外，这个方法几乎等同于Word2Vec。和Word2Vec一样，该模型也存在两种方法：Distributed Memory(DM)和Distributed Bag of Words(DBOW)。DM试图在给定上下文和段落向量的情况下预测单词的概率。在一个句子或者文档的训练过程中，段落ID保持不变，共享着同一个段落向量。DBOW则在仅给定段落向量的情况下预测段落中一组随机单词的概率。

2.1 Doc2vec工具

输入到doc2vec是一个迭代LabeledSentence对象。虽然这种结构允许每个句子有多个标签，但建议每个句子一个标签（句子唯一标识）。每个这样的对象表示一个句子，如下：

```
from gensim.models import doc2vec
sentence = doc2vec.LabeledSentence(words=[u'some', u'words', u'here'], tags=[u'SENT_1'])
```

该算法遍历sentence迭代器两次：一次构建词汇；一次训练模型，学习向量表示对每个词和标签。可以通过下面类的用例作为训练数据实现，每行一个句子中的文件：

```
class LabeledLineSentence(object):
    def __init__(self, filename):
        self.filename = filename
    def __iter__(self):
        for uid, line in enumerate(open(self.filename)):
            yield doc2vec.LabeledSentence(words=line.split(), tags=['SENT_%s' % uid])

sentences = LabeledLineSentence('lee_background.cor') #https://github.com/RaRe-Technologies/gensim/blob/develop/gensim/test/test_data/lee_ba
for sentence in sentences:
    print sentence
```

doc2vec同时学习词和标签的向量表示，如果你希望只学习词向量，则设置 `train_lbls=False` ，如果你希望只学习标签向量，则设置 `train_words=False` ：

```
d2v_model = doc2vec.Doc2Vec(alpha=0.025, min_alpha=0.025)  # use fixed learning rate
d2v_model.build_vocab(sentences)
for epoch in range(10):
    d2v_model.train(sentences)
    d2v_model.alpha -= 0.002  # decrease the learning rate
    d2v_model.min_alpha = model.alpha  # fix the learning rate, no decay

docvec = d2v_model.docvecs[99]
docvec = d2v_model.docvecs['SENT_99']  # if string tag used in training
sims = d2v_model.docvecs.most_similar([99])
sims = d2v_model.docvecs.most_similar(['SENT_99'])
```

```
sims = d2v_model.docvecs.most_similar([docvec])
# 推断新句子/文档的向量表示
d2v_model.infer_vector(['Infer', 'a', 'vector', 'for', 'given', 'post-bulk', 'training', 'document'])
# 计算两个句子/文档的相似度
from gensim import matutils
d1 = d2v_model.infer_vector(['rome', 'italy'])
d2 = d2v_model.infer_vector(['car'])
dot(matutils.unitvec(d1), matutils.unitvec(d2))
```

doc2vec实例的保存和载入：

```
model.save('/tmp/my_model.doc2vec') # store the model to mmap-able files
model_loaded = Doc2Vec.load('/tmp/my_model.doc2vec') # load the model back
```

类似word2vec，如 `model.most_similar()/model.doesnt_match()/model.similarity()` 也存在。原始词和标签向量可以单独通过 `model['word']` 访问，或通过 `model.syn0` 一次访问全部。

2.2 Doc2vec情感分析

Doc2vec进行情感分类任务，我们使用IMDB电影评论数据集 aclImdb_v1.tar.gz ，数据集中包含25000条正向评价，25000条负面评价以及50000条未标注评价，正面评价和负面评价在train和test目录各12500条。解压文件：

```
$ tar -zxf aclImdb_v1.tar.gz
$ cd aclImdb
$ ls
imdbEr.txt  imdb.vocab  README  test  train
$ head train/neg/99_1.txt
Some films that you pick up for a pound turn out to be rather good - 23rd Century films released dozens of obscure Italian and American movi
```

我们这里只加载了train目录数据，12500条正向评价，12500条负面评价，50000条未标注的评价：

```
import os
from gensim.models.doc2vec import LabeledSentence
from sklearn.cross_validation import train_test_split
import numpy as np
```

```
def get_reviews(dirname):
    docs = []
    for fname in os.listdir(dirname):
        infile = open(os.path.join(dirname, fname), 'r')
        docs.append(' '.join(infile.readlines()))
        infile.close()
    return docs

pos_reviews = get_reviews('aclImdb/train/pos')
neg_reviews = get_reviews('aclImdb/train/neg')
unsup_reviews = get_reviews('aclImdb/train/unsup')
#use 1 for positive sentiment, 0 for negative
x = np.concatenate((pos_reviews, neg_reviews)) #数组
y = np.concatenate((np.ones(len(pos_reviews)), np.zeros(len(neg_reviews))))
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

def clean_text(corpus):
    punctuation = """.,?!:;(){}[]"""
    corpus = [z.lower().replace('\n', '') for z in corpus]
    corpus = [z.replace('<br />', ' ') for z in corpus]
    #treat punctuation as individual words
    for c in punctuation:
        corpus = [z.replace(c, ' %s ' % c) for z in corpus]
    corpus = [z.split() for z in corpus]
    return corpus
```

接下来，我们实例化doc2vec的两个模型，DM和DBOW。gensim的说明文档建议多次训练数据集并调整学习速率或在每次训练中打乱输入信息的顺序。我们从Doc2Vec模型中获得电影评论向量：

```
from gensim.models.doc2vec import Doc2Vec
import random

size = 400
#instantiate our DM and DBOW models
model_dm = Doc2Vec(min_count=1, window=10, size=size, sample=1e-3, negative=5, workers=6)
model_dbow = Doc2Vec(min_count=1, window=10, size=size, sample=1e-3, negative=5, dm=0, workers=6)

#build vocab over all reviews
model_dm.build_vocab(x_train + x_test + unsup_reviews)
```

```
model_dbow.build_vocab(x_train + x_test + unsup_reviews)

#We pass through the data set multiple times, shuffling the training reviews each time to improve accuracy.
all_train_reviews = x_train + unsup_reviews
for epoch in range(10):
    random.shuffle(all_train_reviews)
    model_dm.train(all_train_reviews)
    model_dbow.train(all_train_reviews)

#Get training set vectors from our models
def get_vecs(model, corpus, size):
    vecs = [np.array(model.docvecs[z.tags[0]]).reshape((1, size)) for z in corpus]
    return np.concatenate(vecs) #数组

train_vecs_dm = get_vecs(model_dm, x_train, size)
train_vecs_dbow = get_vecs(model_dbow, x_train, size)
train_vecs = np.hstack((train_vecs_dm, train_vecs_dbow))

#train over test set
all_test = x_test + []
```

复制代码
保存代码

现在我们用评论向量构建分类器模型，我们将再次使用sklearn中的SGDClassifier：

```
from sklearn.linear_model import SGDClassifier

lr = SGDClassifier(loss='log', penalty='l1')
lr.fit(train_vecs, y_train)

print 'Test Accuracy: %.2f' % lr.score(test_vecs, y_test) #82%
```

参考资料：

情感分析的新方法

Doc2vec tutorial

Deep learning with paragraph2vec

Distributed Representations of Sentences and Documents

情感分析是自然语言处理（NLP）中的常见的方法应用，是文本分析的一种。所谓文本分析指的是从文字来源中推导出知识和洞见的过程。而情感分析则进一步从文字中自动识别出人们对特定主题的主观看法、情绪以及态度等等资讯。通过这种方式，情感分析可以被视为利用一些情感得分指标来量化定性数据的方法。尽管情绪在很大程度上是主观的，但是情感量化分析已经有很多有用的实践，比如企业分析消费者对产品的反馈信息，或者检测在线评论中的差评信息。

情感分析

点赞

作者：f(x)

精于心，简于形

原文地址：情感分析的新方法, 感谢原作者分享。

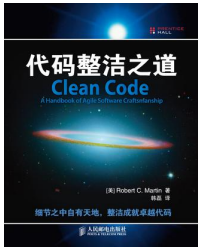
←特征工程七种常用方法

→混合型数据聚类

发表评论

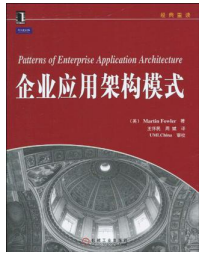
发表评论

好书推荐



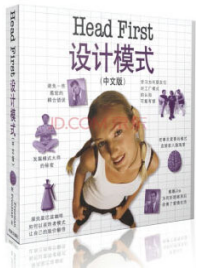
代码整洁之道

[亚马逊]



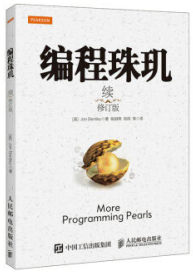
企业应用架构模式

[京东 亚马逊]



Head First设计模式

[京东 亚马逊]



编程珠玑（续 修订版）

[京东 亚马逊]

您可能感兴趣的博客

【案例】情感分析在用研中的初步探索	baozhu 发表2月前
情感分析的新方法	博主 发表1年前
读懂你心的情感型AI	数据有意思 发表6月前
大数据舆情情感分析，如何提取情感并使用什么样的工具？（贴情感标签）	小数点 发表9月前
如何使用KNIME进行情感分析	中	数控小V 发表1年前
如何用KNIME进行情感分析	下	莫扎特 发表1年前
电商评论情感分析(上）	雪姬 发表1年前
为什么美国非结构化数据分析比国内领先三年?	雪姬 发表1年前
英国脱欧，民众是悲是喜?机器学习告诉你答案	数控小V 发表1年前
高速发展的机器学习，会给企业运营带来怎样的改变?	莫扎特 发表1年前

为了挖掘大数据中的商业价值，“玻森数据” 开放了一个中文语义分析 数控小V 发表2年前

关于情感分析，你不得不知道的11件事 LinkinPark 发表2年前

您可能感兴趣的代码

表变量，代替临时表 by xuleaper 4天前

触发器 by 吴红军 5天前

Python的socket编程 by 阮小七 5天前

sql中order by（true or false）类似 group by 的方式 by YuChao 5天前

Android 调节屏幕亮度代码 by 朱凯迪 5天前

java 实现的Boyer-Moore(BM)算法 by 落叶随风 5天前

Android监听Gps设置变化方法二 by 法名空虚 5天前

python实现插入排序 by 山药 6天前

判断是否汉字的Java代码 by Koon.LY 6天前

python计算windows的cpu使用率 by 蟋蟀MM 6天前

一条比较SQL by xuleaper 6天前

从百度地图API接口批量获取地点的经纬度 by jack.chen 6天前