華中科技大學

项目开发报告

题目：疫情期间网民情绪识别

课程名称：机器学习

专业班级：**CS1703**

学　　号：**U201714670**

姓　　名：范唯

指导教师：李玉华

报告日期：**2020 年 7 月 25 日**

计算机科学与技术学院

项目开发报告

1.1 项目目的

(1) 掌握SVM、Logistics、Neural Network等底层算法

(2) 加强使用python、shell等语言的熟练程度

(3) 加强自身的工程开发与调试能力

(4) 掌握通过Python分析数据、绘制图表的能力

(5) 分析预测疫情期间网民们的情绪走势

1.2 问题分析

1.2.1 题目背景

2019 新型冠状病毒(COVID-19)感染的肺炎疫情发生对人们生活生产的方方面面产生了重要影响，并引发国内舆论的广泛关注，众多网民参与疫情相关话 题的讨论。为了帮助政府掌握真实社会舆论情况，科学高效地做好防控宣传和舆 情引导工作，本题目针对疫情相关话题开展网民情绪识别的任务。

1.2.2 数据集

据集依据与“新冠肺炎”相关的 230 个主题关键词进行数据采集，抓取了 2020 年 1 月 1 日—2020 年 2 月 20 日期间微博数据，并对其进行人工标注，标注 分为三类，分别为:1(积极)，0(中性)和-1(消极)。

训练数据以 csv 格式存储在 train.csv 文件中，其中包含 45000 条微博数 据，具体格式如下:

微博中文内容	情感倾向
1.微博中文内容，格式为字符串	2.情感倾向，取值为{1,0,-1}

1.2.3 任务描述

根据 train.csv 文件中的微博数据，设计算法对 test.csv 文件中的 4500 条

微博内容进行情绪识别，判断微博内容是积极的(1)、消极的(-1)还是中性的 (0)。

将结果存储在 csv 文件中，编码采用 UTF-8 编码，格式如下:

微博中文内容	情感倾向
新冠肺炎.....	1

1.2.4 思路分析

通过NLP来进行情感分析在市面上已经有了非常成熟的体系。SVM、Logistics、Neural Network等算法都是非常好的解决方案，而scikit-learn和SnowNLP都是比较成熟的情感分析框架了。以下是在sklearn和SnowNLP两者进行选择的心路历程:

- 一开始是打算使用BosonNLP的库来做的，但是后来发现效果奇差无比。所以后来便改用了sklearn了。
- 其实直接用百度的自然语言分析的黑盒API也可以，但是那样的话就完全越过了训练步骤，纯粹的是测试了，没有意义。
- 7.11 sklearn的训练速度很慢，所以再次选择SnowNLP来进行训练
- 7.12 SVM & 词向量方法加入
- 7.15 SnowNLP数据清洗，微博的很多奇怪的title会非常影响判断结果

所以最后选择了SnowNLP来作为主要的设计方法，其他算法仅做基础实现而不用过度优化。

而本次项目设计中比较新颖的一点是采用了 -1 / 0 / 1 三分类来进行情感划分，市面上许多算法也都是直接根据Pos/Neg来进行划分的。而SnowNLP可以不仅可以直接对一句话的极性进行打分（0 ~ 1），还可以通过内置函数来训练自己的训练集。

```
def train(neg_file, pos_file):
    neg_docs = codecs.open(neg_file, 'r', 'utf-8').readlines()
    pos_docs = codecs.open(pos_file, 'r', 'utf-8').readlines()
    classifier.train(neg_docs, pos_docs)


def save(fname, iszip=True):
    classifier.save(fname, iszip)


def load(fname, iszip=True):
    classifier.load(fname, iszip)


def classify(sent):
    return classifier.classify(sent)
```

因为给出的是一个浮点数分数而不是具体分类，所以我们需要自己额外训练一个映射模型来进行归一化处理。将浮点数均匀的映射在-1 / 0 / 1 三分类上。

```
def normalize(normalizeResult, left, right):
    if normalizeResult > left:
        normalizeResult = 1
    elif normalizeResult <= left and normalizeResult >= right:
        normalizeResult = 0
    else:
        normalizeResult = -1
    return normalizeResult
```

所以对于SnowNLP方法来说，主要的工作量集中于以下几点：

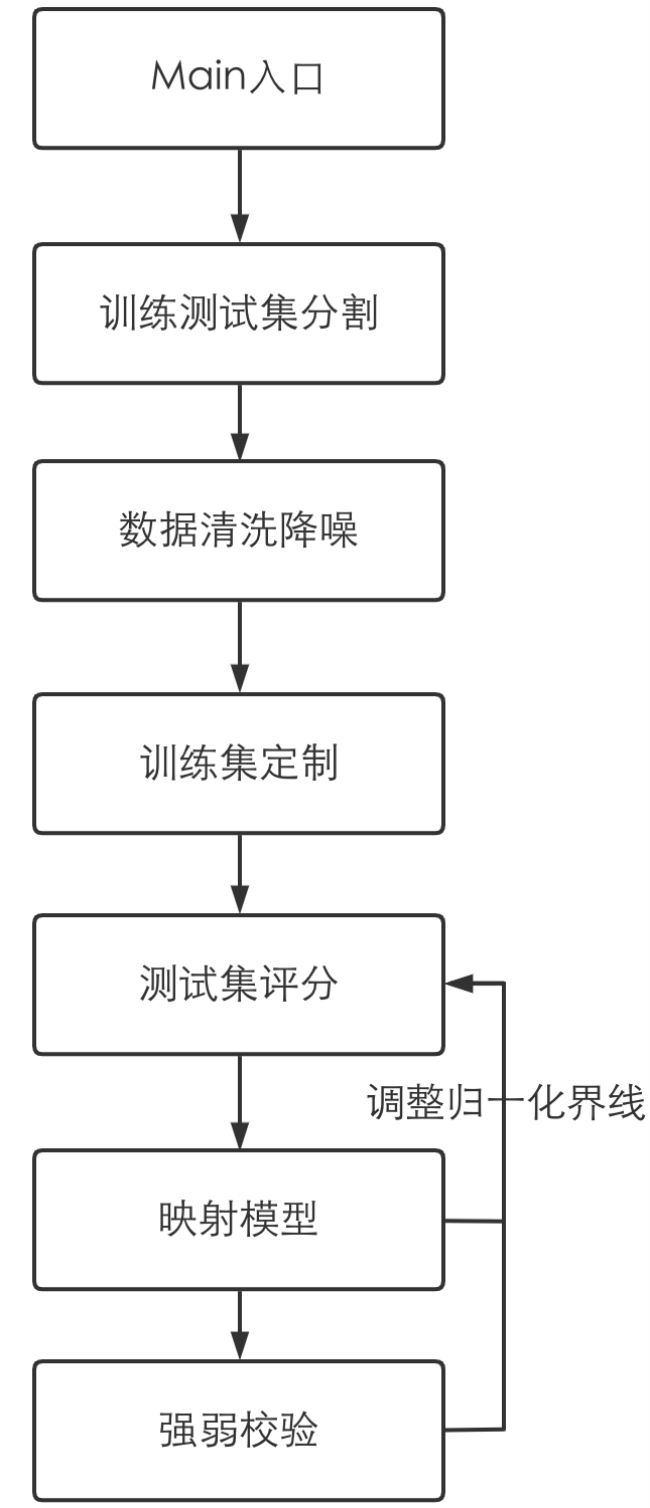
1. 分割训练集中的-1 / 0 / 1三类文本并保存至Pos.txt / Neu.txt / Neg.txt三个文件中
2. 使用停用词以及正则匹配去除噪声（微博上的用户评论前的各种title会非常影响结果）
3. 训练自己的训练集并保存至文件「sentiment.marshal.3」中
4. 对自己的测试集进行测试，得出归一化之前的句子分数
5. 训练映射模型，将分数均匀映射至-1 / 0 / 1三个分类上
6. 通过强弱校验来分别对结果正确率进行判断。

> 这里没采用混淆矩阵分析的原因是因为这里采用的是三分类而非二分类。

此外还选择了SVM、Logistics、Word2Vec+LSTM三分类来分别训练模型来观察是否能达到比较好的效果。

1.3 基于SnowNLP

根据1.2.4中的思路分析，现在我们对此进行扩展详细说明。



> SnowNLP是一个python写的类库，可以方便的处理中文文本内容，是受到了TextBlob的启发而写的，由于现在大部分的自然语言处理库基本都是针对英文的，于是写了一个方便处理中文的类库，并且和TextBlob不同的是，这里没有用NLTK，所有的算法都是自己实现的，并且自带了一些训练好的字典。注意本程序都是处理的unicode编码，所以使用时请自行decode成unicode。

以上是官方对snownlp的描述，简单地说，snownlp是一个中文的自然语言处理的Python库，支持的中文自然语言操作包括：

- 中文分词
- 词性标注
- 情感分析
- 文本分类
- 转换成拼音
- 繁体转简体
- 提取文本关键词
- 提取文本摘要
- tf，idf
- Tokenization
- 文本相似

1.3.1 训练测试集分割

训练集的分割方法保存在工程中的covid19/splitNegPos.py中，如需查看源码可自行取用。

```
def startSplit(self, posnum, negnum, midnum, testnum):
        file = open("train&test/pos.txt", 'w')
        for comment in self.pos[0:posnum]:
            comment = self.cleanComment(comment)
            if comment != '':
                file.write(comment)
                file.write('\n')
        file.close()

        file = open("train&test/neg.txt", 'w')
        ............

        file = open("train&test/mid.txt", 'w')
        ............

        data = pd.read_csv('train&test/test.csv', encoding='utf-8')
        with open('train&test/test.txt', 'w', encoding='utf-8') as f:
            for line in data.values[0:testnum]:
                testline = self.cleanComment(str(line[0]))
                f.write((testline + '\n'))
```

该部分的实现主要依赖于pandas下的read_csv()函数来进行拆分合并文件。我们要做的工作主要是将文件-1 / 0 / 1三个分类分别划分到pos.txt / neu.txt / neg.txt三个文件中。并且将test.csv中的正确结果导入进txt中。

1.3.2 数据清洗降噪

在微博中，许多文本都包含许多奇怪的title和转发链接，所以我们需要通过正则表达式以及停用词消除一些对情感没有太大影响的词汇。


```
def cleanComment(self, comment):
    comment = re.sub('#.*#', '', comment)
    comment = re.sub('//@.*:', '', comment)
    comment = re.sub('//@.*： ', '', comment)
    comment = re.sub('//.*:', '', comment)
    comment = re.sub('//.*： ', '', comment)
    comment = re.sub('【.*】', '', comment)
    comment = re.sub('《.*》', '', comment)
    comment = re.sub('//.*//', '', comment)
    comment = re.sub('@.*： ', '', comment)
    comment = re.sub('@.*:', '', comment)
    comment = re.sub('『.*』', '', comment)
    comment = re.sub(r'\d', '', comment)
    return comment
```

以上是调用re中正则表达式匹配库中的sub()方法，来消除一些奇怪的title。比如：`//@整点电影：//@【整点电影】：`

然后在清洗完了之后，再进行去除停用词。GitHub上有许多停用词库，本项目选用的是哈工大的停用词库和百度的停用词库。以下是去停用词的cut2wd()方法。

```
def cut2wd(sentence):
    wordcut = jieba.cut(sentence)
    wordcheck = []
    for w in wordcut:
        wordcheck.append(w)
    stopwords = readfile('train&test/stopwords.txt')
    newword = []
    for w in wordcheck:
        if w in stopwords:
            continue
        else:
            newword.append(w)
    return newword
```

1.3.3 训练集定制

经过去停之后，我们可以去定制自己专属的训练集。其核心的训练方法还是基于贝叶斯模型的。

```
def train(self, data):
    # data 中既包含正样本，也包含负样本
    for d in data: # data中是list
        # d[0]:分词的结果，list
        # d[1]:正/负样本的标记
        c = d[1]
        if c not in self.d:
            self.d[c] = AddOneProb() # 类的初始化
        for word in d[0]: # 分词结果中的每一个词
            self.d[c].add(word, 1)
    # 返回的是正类和负类之和
    self.total = sum(map(lambda x: self.d[x].getsum(), self.d.keys())) # 取得所有的d中的sum之和
```

在实际的项目中，需要根据实际的数据重新训练情感分析的模型，大致分为如下的几个步骤：

- 准备正负样本，并分别保存，如正样本保存到pos.txt，负样本保存到neg.txt；
- 利用snownlp训练新的模型
- 保存好新的模型

但是我们需要在/Users/***/anaconda3/lib/python3.7/site-packages/snownlp/sentiment/__init__.py下对于SnowNLP的sentiment.marshal路径进行修改。不再使用模型自带的sentiment.marshal。

```
_init_.py
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

import os
import codecs

from .. import normal
from .. import seg
from ..classification.bayes import Bayes

#data_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),'sentiment.marshal')
data_path = '/Users/alexfan/HUSTER-CS/机器学习/结课项目/疫情期间网民情绪识别/covid19/sentiment.marshal'
```

情感分类的基本模型是贝叶斯模型Bayes，对于贝叶斯模型，可以参见文章简单易学的机器学习算法——朴素贝叶斯。对于有两个类别c1和c2的分类问题来说，其特征为w1,…,wn，特征之间是相互独立的，属于类别c1的贝叶斯模型的基本过程为：

$$P(c_1 \mid w_1, \cdots, w_n) = \frac{P(w_1, \cdots, w_n \mid c_1) \cdot P(c_1)}{P(w_1, \cdots, w_n)}$$

其中：

$$P(w_1, \cdots, w_n) = P(w_1, \cdots, w_n \mid c_1) \cdot P(c_1) + P(w_1, \cdots, w_n \mid c_2) \cdot P(c_2)$$

训练之后，保存sentiment.marshal.3文件在我们自定义的data_path中。

1.3.4 测试集评分

在训练完成之后，可以通过内置的snownlp()函数来进行评分。但是得到的都是0～1的浮点数的结果

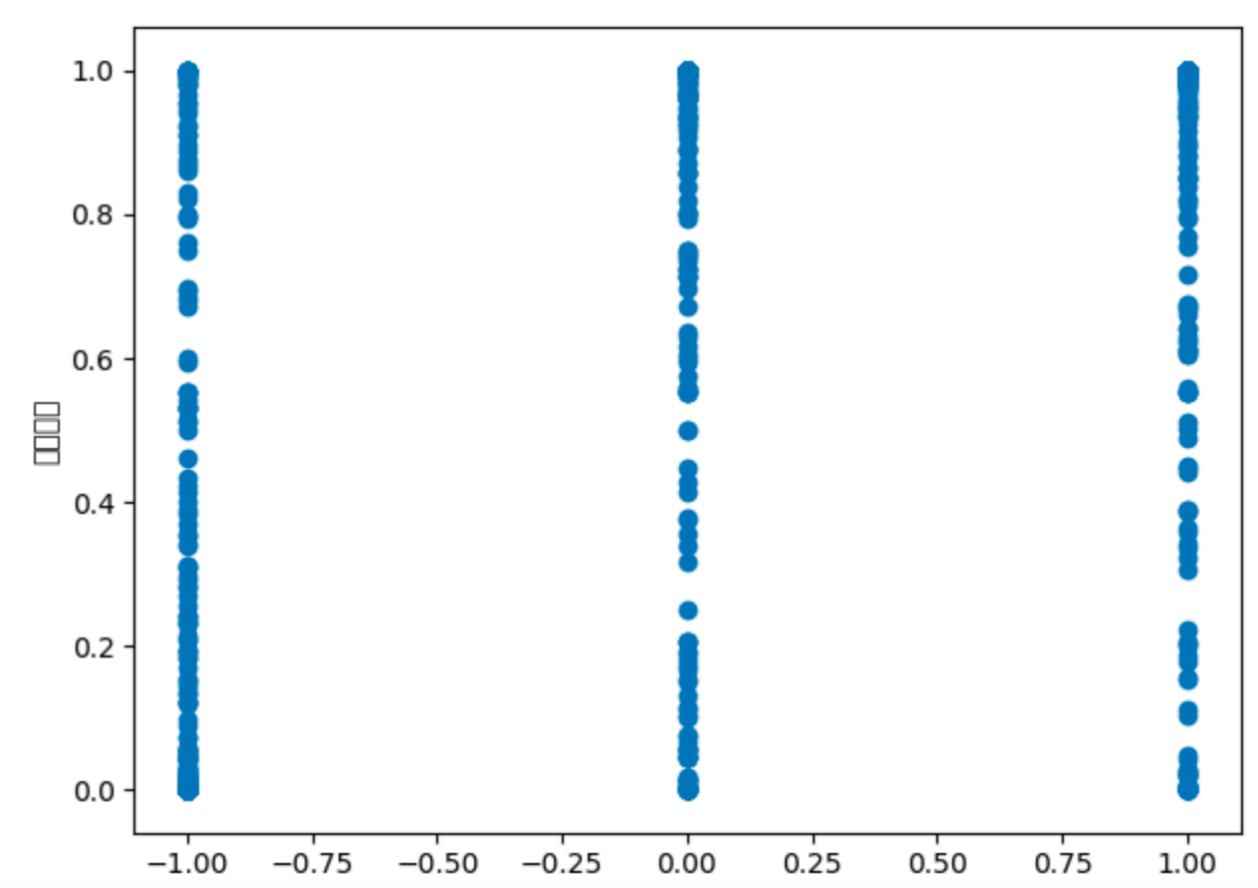
```
result = [0.44289849564187256, 0.9994543029440242, 0.5, 0.003862980768790547, 0.0018132847554932496,
```

- 返回值为正面情绪的概率
- 越接近1表示正面情绪
- 越接近0表示负面情绪

但是我们需要的结果是三分类的-1 / 0 / 1 的结果，所以在测试结束后，把所有的浮点数结果存放在result数组里，等待下面的映射模型来进行转换。

1.3.5 映射模型

为了解决三分类的映射关系问题，我们需要将浮点结果映射到-1 / 0 / 1上，最好的办法就是先查看每种结果在三条线上的分布图，再按照0.02的步长来选择left和right的界线（通过chooseLeftAndRight()）。对每种情况划分后来统计结果。



以上chooseLeftAndRight方法则是按照指定的num步长来划分归一化处理。

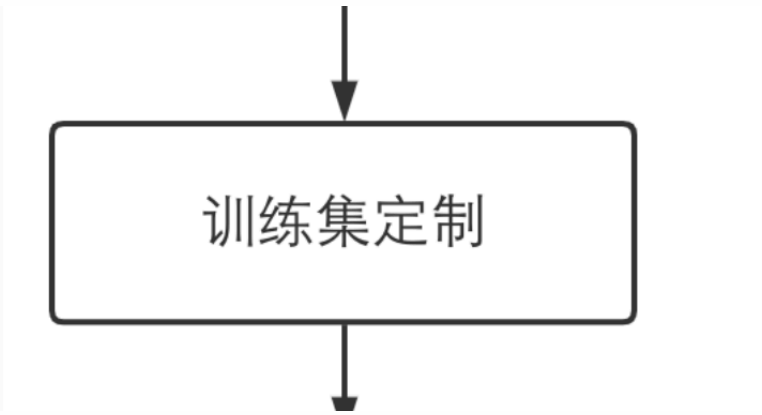
```
def chooseLeftAndRight(pos, neu, neg, num):
    left = []
    right = []
    sum = []
    for l in range(num):
        for r in range(l, num):
            correct = 0
            for n in range(0, l):
                correct += neg[n]
            for n in range(l, r):
                correct += neu[n]
            for p in range(r, num+1):
                correct += pos[p]
            print(correct)
            sum.append(correct)
            left.append(l*(1/num))
            right.append(r*(1/num))
    return left, right, sum
```

部分结果如下图：

```
左界: 0.9 右界: 0.98 正确结果数: 2388
左界: 0.92 右界: 0.92 正确结果数: 2366
左界: 0.92 右界: 0.94 正确结果数: 2369
左界: 0.92 右界: 0.96 正确结果数: 2373
左界: 0.92 右界: 0.98 正确结果数: 2380
左界: 0.94 右界: 0.94 正确结果数: 2352
左界: 0.94 右界: 0.96 正确结果数: 2356
左界: 0.94 右界: 0.98 正确结果数: 2363
左界: 0.96 右界: 0.96 正确结果数: 2344
左界: 0.96 右界: 0.98 正确结果数: 2351
左界: 0.98 右界: 0.98 正确结果数: 2308
最大正确率:  2456
```

1.4 (支持向量机) SVM 方法

因为1.3中已经讲过了关于数据预处理，并且校验方法可以复用，所以只需要改动训练集中训练部分的**API**。将其中的算法替换成SVM即可。



> 支持向量机（support vector machines, SVM）是一种二分类模型，它的基本模型是定义在特征空间上的**间隔最大的线性分类器**，间隔最大使它有别于感知机；SVM还包括**核技巧**，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。SVM的的学习算法就是求解凸二次规划的最优化算法。

具体实现步骤如下：

- 对数据进行降维:PCA，文本分类:SVM，导入numpy以及pandas

```
import sys
import numpy as np
import pandas as pd
from sklearn.decomposition import pca
from sklearn import svm
from sklearn import metrics
```

- 前期数据准备，按照datapath读取训练数据。

```
# 数据准备
datapath = ''
df = pd.read_csv(datapath)
x = df.iloc[:, 2:]
y = df.iloc[:, 1]
```

- 使用sklearn下的pca方法进行数据降维。不然会严重的拖慢速度，加长训练的时间。

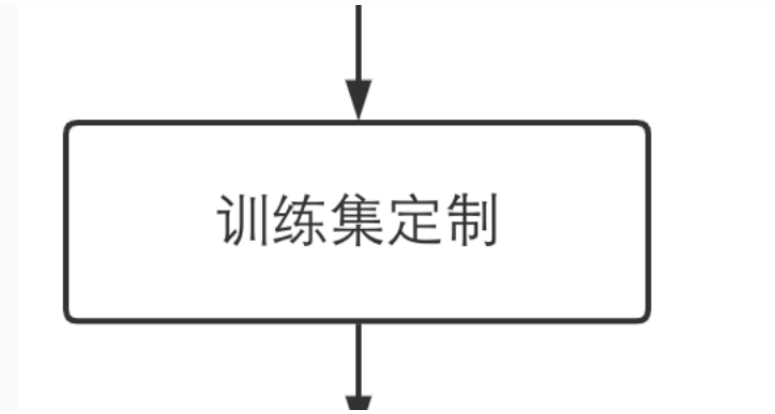
```
# PCA降维。计算全部贡献率
n_components = 400
pca_model = pca.PCA(n_components)
pca_model.fit(x)
```

- 使用sklearn下的pca方法取数据的100维作为分类器的输入，拟合曲线后再进行数据的标准化。最后使用svm库中的SVC方法来训练。

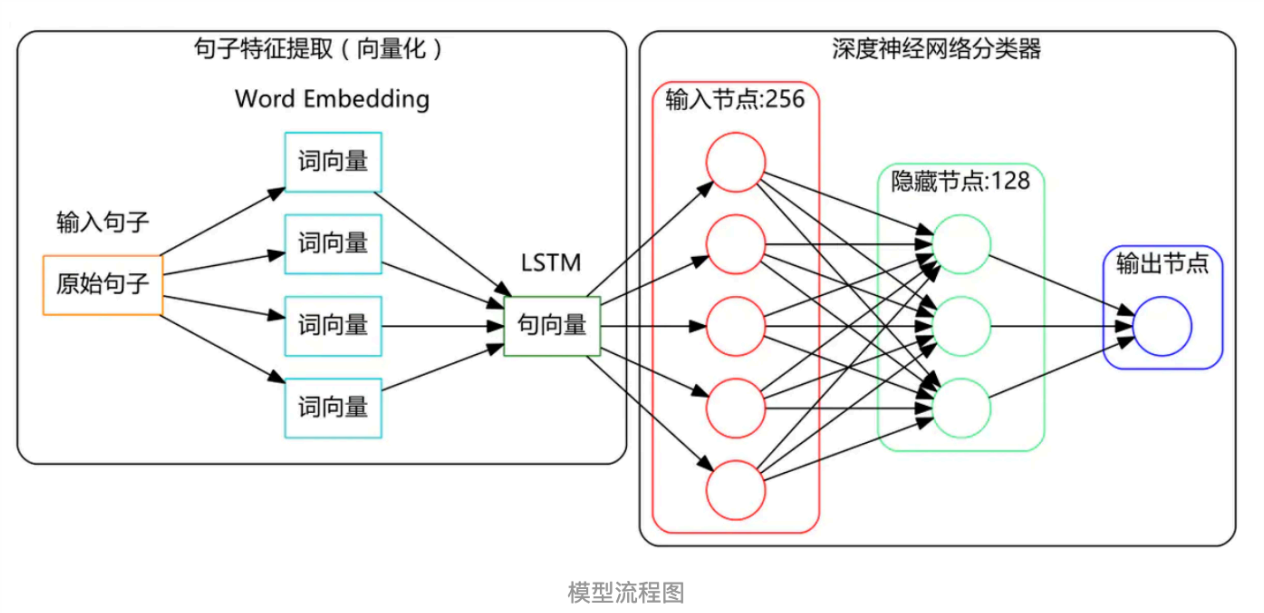
```
# pca作图
# 取100维作为分类器输入
x_pca = pca.PCA(n_components=100).fit_transform(x)  # 先拟合数据再标准化
# SVM分类
clf = svm.SVC(C=2.0, probability=True)
clf.fit(x_pca, y)
print('Test Accuracy is: %.2f' % clf.score(x_pca, y))
```

1.5 基于LSTM三分类方法

同上述1.4一样因为1.3中已经讲过了关于数据预处理，并且校验方法可以复用，所以只需要改动训练集中训练部分的**API**。将其中的算法替换成LSTM三分类即可。



模型的大致流程图如下图所示:



模型流程图

Word2Vec完成了用高维向量（词向量，Word Embedding）表示词语，并把相近意思的词语放在相近的位置，而且用的是实数向量（不局限于整数）。我们只需要有大量的某语言的语料，就可以用它来训练模型，获得词向量。

接下来要解决的问题是：我们已经分好词，并且已经将词语转换为高维向量，那么句子就对应着词向量的集合，也就是矩阵，类似于图像处理，图像数字化后也对应一个像素矩阵；可是模型的输入一般只接受一维的特征，那怎么办呢？一个比较简单的想法是将矩阵展平，也就是将词向量一个接一个，组成一个更长的向量。这个思路是可以，但是这样就会使得我们的输入维度高达几千维甚至几万维，事实上是难以实现的。在自然语言处理中，通常用到的方法是递归神经网络或循环神经网络（都叫RNNs）。它们的作用跟卷积神经网络是一样的，将矩阵形式的输入编码为较低维度的一维向量，而保留大多数有用信息。

代码需要注意的几点是，第一是，标签需要使用keras.utils.to_categorical来yummy，第二是LSTM二分类的参数设置跟二分有区别，选用softmax，并且loss函数也要改成categorical_crossentropy，代码如下：

```
def get_data(index_dict,word_vectors,combined,y):

    n_symbols = len(index_dict) + 1  # 所有单词的索引数，频数小于10的词语索引为0，所以加1
    embedding_weights = np.zeros((n_symbols, vocab_dim)) # 初始化 索引为0的词语，词向量全为0
    for word, index in index_dict.items(): # 从索引为1的词语开始，对每个词语对应其词向量
        embedding_weights[index, :] = word_vectors[word]
    x_train, x_test, y_train, y_test = train_test_split(combined, y, test_size=0.2)
    y_train = keras.utils.to_categorical(y_train,num_classes=3)
    y_test = keras.utils.to_categorical(y_test,num_classes=3)
    # print x_train.shape,y_train.shape
    return n_symbols,embedding_weights,x_train,y_train,x_test,y_test
```

定义网络结构

```
##定义网络结构
def train_lstm(n_symbols,embedding_weights,x_train,y_train,x_test,y_test):
    print 'Defining a Simple Keras Model...'
    model = Sequential()  # or Graph or whatever
    model.add(Embedding(output_dim=vocab_dim,
                        input_dim=n_symbols,
                        mask_zero=True,
                        weights=[embedding_weights],
                        input_length=input_length))  # Adding Input Length
```

Dense=>全连接层,输出维度=3

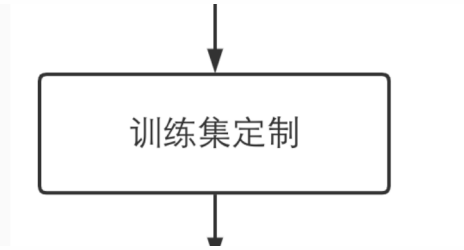
```
    model.add(LSTM(output_dim=50, activation='tanh'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))
    model.add(Activation('softmax'))

    print 'Compiling the Model...'
    model.compile(loss='categorical_crossentropy',
                  optimizer='adam',metrics=['accuracy'])

    print "Train..." # batch_size=32
    model.fit(x_train, y_train, batch_size=batch_size, epochs=n_epoch,verbose=1)

    print "Evaluate..."
    score = model.evaluate(x_test, y_test,batch_size=batch_size)

    yaml_string = model.to_yaml()
    with open('../model/lstm.yml', 'w') as outfile:
        outfile.write( yaml.dump(yaml_string, default_flow_style=True) )
```

```
    model.save_weights('../model/lstm.h5')
    print 'Test score:', score
```

后期分析可以得到LSTM应该是解决这个问题的最好的办法，毕竟没有怎么调参就已经达到了0.51的正确率。但是时间不够了，所以暂时还是把SnowNLP放在第一优先级。在课程结束后还得好好来调优一下神经网络。

1.6 基于Logistic回归模型

同上述1.5和1.4一样因为1.3中已经讲过了关于数据预处理，并且校验方法可以复用，所以**只需要改动训练集中训练部分的API**。将其中的算法替换成LSTM三分类即可。

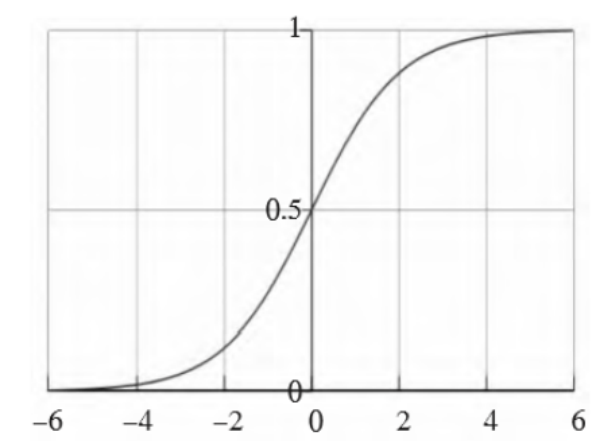


首先对其进行文本预处理，将经过处理后的微博文本进行人工情感类别标记，然后基于多远情感词典抽取微博的情感特征词和使用布尔权值方法计算情感特征权重值，导入到Logistic回归模型，经过训练后得到LR（Logistics Regression）分类器，用它来预测未知类别的微博文本的情感极性。

为了实现Logistic回归，需要根据现有的数据，构建分类边界回归公式，不断进行参数训练。找到最佳的拟合参数，得到LR分类器，最后用它来实现未知类别微博文本的情感分类。首先需要使用Sigmoid函数来进行分类是主管还是客观的微博。公式如下

$$\sigma(z)=\frac{1}{1+e^{-z}}$$

Sigmoid函数图如下：



之后再基于改进的上升梯度算法确定最佳回归系数：

对分类边界建立回归公式，具体如式（2）所 示。其中 X 表示特征，W 表示回归系数，对每一个 特征乘以回归系数，然后所有值相加得到 Z 值，然 后将其带入到Sigmoid函数，此时σ（z）的取值范围介于 0 到 1之间，对σ（z）大于 0.5，划归为“1”类，σ（z）小于 0.5，则划归到“0”类。

$$Z = W_0X_0 + W_1X_1 + \ldots\ldots + W_nX_n (2)$$

选择随机梯度算法作为最优化算法，其伪代码如下：

- 随机回归系数初始化为l
- 对数据集中每个样本
- 计算该样本的梯度
- 使用alpha*gradient更新回归系数值
- 返回回归系数值

最终得到的也是二分类的结果，但是依然可以使用1.3中的映射模型来将二分类映射到三分类上。不过结果很差，只有不到0.4的正确率。

1.7 结果分析

1.7.1 SnowNLP结果分析

在1.3中我们得到的最优左界: 0.54 右界: 0.98，测试结果如下

校验函数如下：

- 强校验

```
def checkResult(result, realResult):
    correct = 0
    for i in range(len(result)):
        if result[i] == realResult[i]:
            correct += 1
    print("\n强校验正确率=====================================>")
    print(correct/len(result))
    print("正确结果数量：{}".format(correct))
    print("全部结果数量：{}".format(len(realResult)))
    return correct, len(result)
```

- 无中性校验

```
def checkResultNoNeu(result, realResult):
    correct = 0
    sum = 0
    for i in range(len(result)):
        if realResult[i] != 0:
            sum += 1
        if result[i] == realResult[i]:
            correct += 1
    print("\n无中性校验正确率=====================================>")
    print(correct/sum)
    print("正确结果数量：{}".format(correct))
    print("全部结果数量：{}".format(sum))
    return correct, sum
```

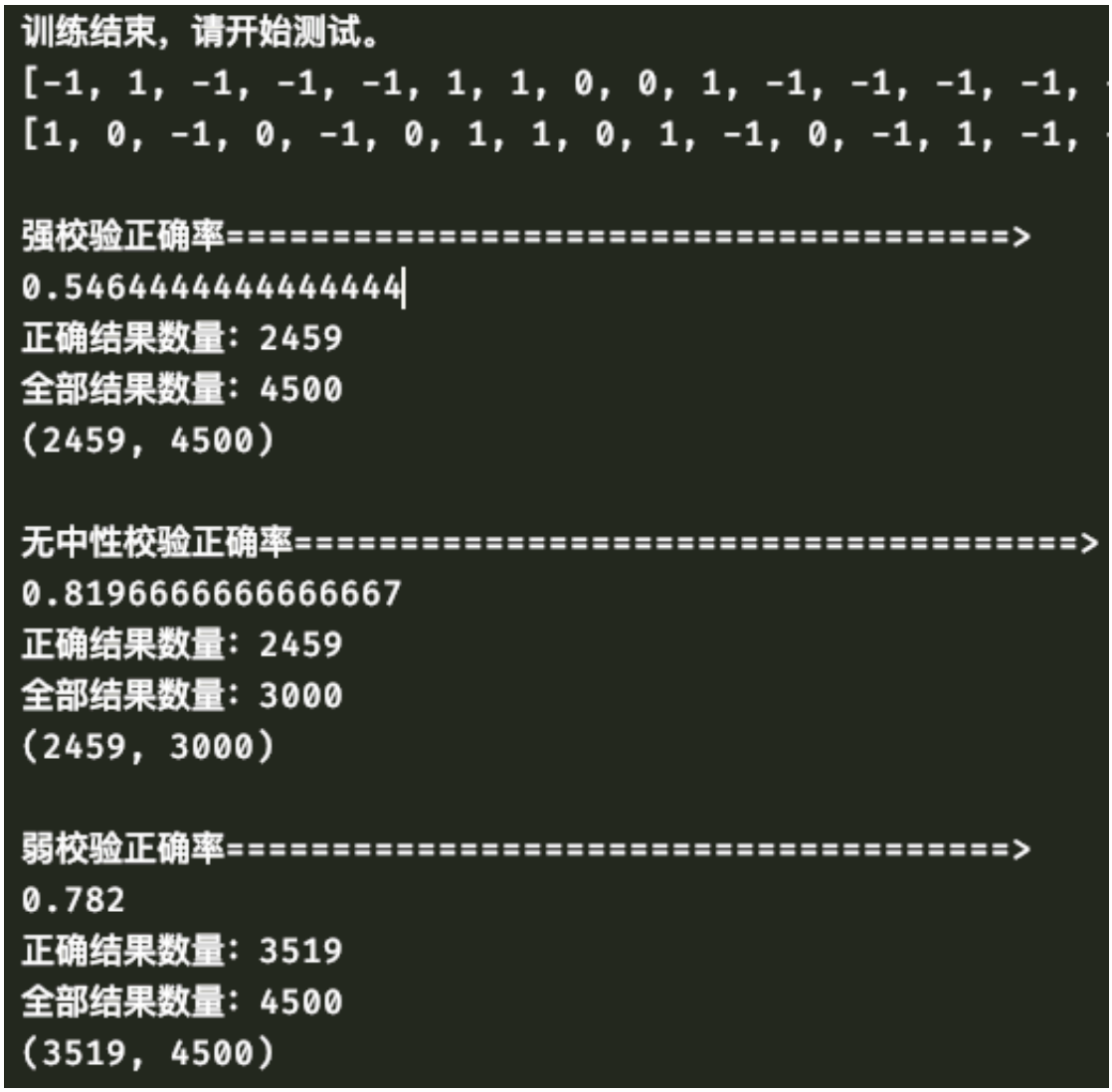
- 弱校验

```
def checkResultCombNeuAndPos(result, realResult):
    correct = 0
    for i in range(len(result)):
        if realResult[i] == -1:
            if result[i] == -1:
                correct += 1
        else:
            if result[i] == 1 or result[i] == 0:
                correct += 1
    print("\n弱校验正确率=====================================>")
    print(correct/len(result))
    print("正确结果数量：{}".format(correct))
    print("全部结果数量：{}".format(len(realResult)))
    return correct, len(realResult)
```

强校验正确率:0.546

无中性校验正确率:0.81967

弱校验正确率:0.782



分析可以得出其预测结果正确率大致为0.55，如果只校验pos和neg的结果的话，正确率却会大幅度上升，分析可以发现是因为在SnowNLP模型下对于pos / neu语句的评分相差无几，所以才会导致强校验结果下正确率趋近0.55而弱校验的正确率很高。

1.7.2 SVM结果分析

将训练接口转到SVM之后，正确率有所下降，一部分原因是因为没有优化，一部分原因来自于我们的训练集数据有50%是中性句子，这使得向量机中的聚类变的非常冗余。而且对于三分类的句子效果也不太好。

强校验正确率:0.481

无中性校验正确率:0.8497

弱校验正确率:0.722

```
<************************强校验正确率************************>
0.48123111
正确结果数量: 2109
全部结果数量: 4500
(2109, 4500)

<************************无中性校验正确率************************>
0.8496666666666667
正确结果数量: 2659
全部结果数量: 3000
(2659, 3000)

<************************弱校验正确率************************>
0.722
正确结果数量: 3249
全部结果数量: 4500
(3249, 4500)
```

1.7.3 基于LSTM三分类结果分析

强校验正确率:0.531

无中性校验正确率:0.921

弱校验正确率:0.745

```
<************************强校验正确率************************>
0.5311435
正确结果数量: 2390
全部结果数量: 4500
(2300, 4500)

<************************无中性校验正确率************************>
0.921666667
正确结果数量: 2765
全部结果数量: 3000
(2765, 3000)

<************************弱校验正确率************************>
0.745333333
正确结果数量: 3354
全部结果数量: 4500
(3354, 4500)
```

LSTM通过词向量->句向量->神经网络三部曲，在不调优的前提下已经能达到和SnowNLP框架近乎相同的结果。

1.7.4 基于Logistic回归模型结果分析

强校验正确率:0.362

无中性校验正确率:0.594

弱校验正确率:0.4321

```
<************************强校验正确率************************>
0.362666667
正确结果数量: 1632
全部结果数量: 4500
(1632, 4500)

<************************无中性校验正确率************************>
0.594
正确结果数量: 1782
全部结果数量: 3000
(1782, 3000)

<************************弱校验正确率************************>
0.431231
正确结果数量: 1940
全部结果数量: 4500
(1940, 4500)
```

纯粹的逻辑回归的效果似乎看起来很差，因为三分类的预测结果一旦接近0.33就近乎等于无效果，所以这种方法在后期也不再进行维护优化了。其他几种方法中其实或多或少都融入了逻辑回归的步骤。

1.8 思考和总结

整个项目做完之后的感觉思路还是十分顺畅的，除了少许地方用到的一些数学知识需要补一下，其他地方的难度不是特别高。对于情感分析，市面上已经用了许多非常成熟的开源框架，所以我们也选用了SnowNLP这种半定制化的框架来作为主要的调优对象，而SVM、Logistic回归模型、LSTM三分类法都是用来辅助参考的。（事后发现LSTM三分类法应该才是最优解），整个项目的遗憾可能也就是没有继续调优LSTM模型，感觉它可以把正确率上到0.6+。不过理解了其中的逻辑之后，已经达到了这门课的学习目的了。

SnowNLP是一个python写的类库，可以方便的处理中文文本内容，是受到了TextBlob的启发而写的，由于现在大部分的自然语言处理库基本都是针对英文的，于是写了一个方便处理中文的类库，并且和TextBlob不同的是，这里没有用NLTK，所有的算法都是自己实现的，并且自带了一些训练好的字典。

SVM（支持向量机）它的基本模型是定义在特征空间上的**间隔最大的线性分类器**，间隔最大使它有别于感知机；SVM还包括**核技巧**，这使它成为实质上的非线性分类器。而和贝叶斯略微区别开来。

Logistic则在本次实验中表现的很糟糕，最终分析还是吃了二分类的亏，导致其实力不能完全发挥。其实在SnowNLP中也有逻辑回归的步骤，所以单纯的使用Logistics是不足以和其他方法匹敌的。

LSTM三分类则是一个非常好的解决办法，从词向量到句向量，以及后续的神经网络。都是目前业界最通用的方法，只可惜没有更多的时间去调优了，不调参的情况下它的正确率已经能达到SnowNLP的水准了。

而机器学习来分析文本的情感其实很多时候是个伪命题，在业界的应用价值其实没有想象中那么高。而且一句话的情感程度在不同人眼里是完全不一样的，并且微博上许多人喜欢阴阳怪气的去发表一些观点。而且打标签的时候，会受到一句话的各种title的影响。而预测的结果也会出现千奇百怪的偏差。

不过这门课，这个选题的核心是为了帮助我们去理解分类器是如何工作的。通过贝叶斯等分类方法来进行特征的分类与聚合，将视线从纯粹的代码逻辑拉回到数学逻辑上，这是一件非常精妙的事情。这也让机器学习这门课变的十分有意义而且趣味性更高了。

最后也十分感谢老师和同学的帮助，让我在机器学习的领域更进了一步。

1.9 参考文献

- 《基于SVM的文本词句情感分析》
  https://www.ixueshu.com/document/cd7838079876fcad2f5b8f1a50494211318947a18e7f9386.html
- 《基于Logistic回归模型的微博情感分析研究》
  https://www.ixueshu.com/document/68abee163511291ecb8dc96b1e58de1a318947a18e7f9386.html
- 《基于LSTM三分类的文本情感分析》
  https://www.jianshu.com/p/158c3f02a15b
- 《情感分析方法之nltk情感分析器和SVM分类器》
  https://blog.csdn.net/sinat_36972314/article/details/79621591
- 《情感分析——深入snownlp原理和实践》
- https://blog.csdn.net/google19890102/article/details/80091502