

神经网络编程作业

神经网络编程作业

TASK1:使用Logistic回归估计马疝病的死亡率

一、软件设计

1. 设计目标

- ① 数据集讲解
- ② 目标要求

2. 算法设计分析

- ① Logistic回归
- ② 代码实现

```
=====>> 数据预处理 <<=====
=====>> 逻辑回归算法 <<=====
=====>> main函数 <<=====
```

二、运行结果

截图一_运行结果，单次运行
截图二_运行结果，循环求均值

三、总结体会

程序设计心得:

TASK2: 使用神经网络完成新闻分类

一、软件设计

1. 设计目标

- ① 数据集讲解:
- ② 设计提示思路
- ③ **目标要求**

2. 算法设计分析

- ① 神经网络基本概念
- ② TD-IDF
- ③ 代码实现

```
=====>> 数据处理 <<=====
=====>> 神经网络 <<=====
=====>> main函数 <<=====
```

二、运行结果

三、总结体会

程序设计心得:

TASK1:使用Logistic回归估计马疝病的死亡率

一、软件设计

1. 设计目标

① 数据集讲解

- **训练集**: 包含于文件horseColicTraining.txt中，用于训练得到模型的最佳系数。训练集包含299个样本（299行），每个样本含有21个特征（前21列），这些特征包含医院检测马疝病的指标；最后1列为类别标签，表示病马的死亡情况；部分样本含有缺失值；
- **测试集**: 包含于文件horseColicTest.txt中，通过预测测试样本中病马的死亡情况，来评估训练模型的优劣。测试集包含69个样本，部分样本部分特征缺失；最后1列为类别标签，用于计算错误率。

- **特征值缺失**：在该示例中，可以选择实数0来替换所有缺失值，理由是：
 - 1, 在系数更新时不会影响到系数的值（后面会介绍）；
 - 2, 由于 $\text{sigmoid}(0)=0.5$ ，对结果的预测不具有任何倾向性；
 - 3, 该数据集中的特征值一般不为0，因此某种意义上也满足‘特殊值’这个要求。
- **训练集中类别标签缺失**：在Logistic回归中，可将该样本直接丢弃。

② 目标要求

所需提交材料：任务一需要编程实现Logistic回归完成对测试集的预测，并且计算得到精度，编写实验报告简述算法原理，实验结果等。

2. 算法设计分析

所用项目环境 IDE: VS code Python版本: python3.7.5

① Logistic回归

• Logistic回归概述

Logistic回归的主要思想是，根据现有的数据对分类边界建立回归公式，从而实现分类（一般两类）。“回归”的意思就是要找到最佳拟合参数，其中涉及的数学原理和步骤如下：

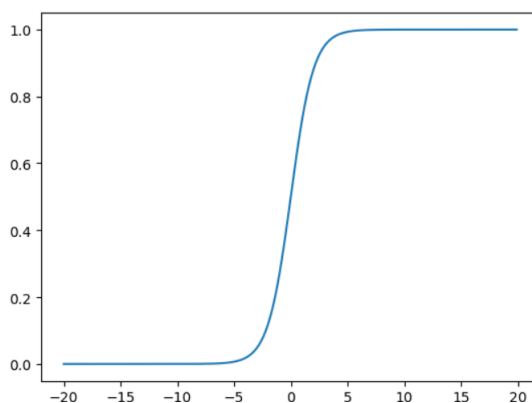
- (1) 需要一个合适的分类函数来实现分类【单位阶跃函数、Sigmoid函数】
- (2) 损失函数（Cost函数）来表示预测值($h(x)$)与实际值(y)的偏差($h - y$)，要使得回归最佳拟合，那么偏差要尽可能小（偏差求和或取均值）。
- (3) 记 $J(w)$ 表示回归系数为 w 时的偏差，那么求最佳回归参数 w 就转换成了求 $J(w)$ 的最小值。
【梯度下降、上升法】

• 分类函数

分类函数选择sigmoid函数,公式如下：

$$h(x) = \frac{1}{1 + e^{-x}}$$

这个函数的特点是，当 $x=0$ 时， $h(x)=0.5$ ，而 x 越大， $h(x)$ 越接近1， x 越小， $h(x)$ 越接近0。因此，**当 $x>0$ ，就可以将数据分入1类；当 $x<0$ ，就可以将数据分入0类。**函数图如下：



• Cost函数

具体数学推导过程省略，只列出结果

输入的特征向量： $z = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = w^T x$

分类函数(sigmoid函数)： $h(z) = \frac{1}{1 + e^{-z}}$

因此预测函数： $h(z) = h_w(x) = \frac{1}{1 + e^{-w^T x}}$

对于任意确定的 x 和 w , 有: $P(y = 1|x, w) = h_w(x)$ $P(y = 0|x, w) = 1 - h_w(x)$

可以等价于 $P(y|x, w) = (h_w(x))^y(1 - h_w(x))^{1-y}$

对数似然函数:

$$J(w) = \log L(w) = \sum_{i=1}^m \left(y^{(i)} \log h_w(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_w(x^{(i)})) \right)$$

推导出代价函数Cost函数:

$$Cost(h_w^x, y) = \begin{cases} \log h_w(x) \\ \log(1 - h_w(x)) \end{cases}$$

逻辑回归的实现最重要的就是是偏差最小, 我们通常用 $J(w)$ 表示预测值与实际值的偏差, 也就是Cost函数, 则需要 $J(w)$ 最大

- 梯度上升法

求解 $J(w)$ 最大值的方法, 沿着函数梯度方向寻找, 梯度 $\nabla f(x, y)$

梯度上升法中, 梯度算子沿着函数增长最快的方向移动 (**移动方向**), 如果移动大小为 α (**步长**), 那么梯度上升法的迭代公式是:

$$w := w + \alpha \nabla_w f(w)$$

转化问题为:

$$w_j := w_j + \alpha \frac{\partial J(w)}{\partial w_j} (j = 1, 2 \dots n)$$

求得结果为:

$$w_j := w_j + \alpha \sum_{i=1}^m \left(y^{(i)} - h_w(x^{(i)}) \right) x_j^{(i)}$$

②代码实现

===== >> 数据预处理 << =====

算法介绍: 对提供的训练文件进行读取为矩阵, 并将标签提取出来

数据整理: 对于缺失值的情况, 作业中的数据已经将缺失值全部补0了, 根据设计目标中的数据集讲解可得, 设为0即可不影响后续。

- 数据读取函数:

```
def init_data(filepath):
    data = np.loadtxt(filepath, delimiter='\t')
    dataMatIn = data[:, 0:-1]
    classLabels = data[:, -1]
    dataMatIn = np.insert(dataMatIn, 0, 1, axis=1)
    # 特征数据集, 添加1是构造常数项x0
    return dataMatIn, classLabels
```

- 代码细节:

- (1) 采用`np.loadtxt()`导入数据, 分隔符为`\t`, 同时使用列表的`[m:n]`操作读取标签
- (2) 采用`np.insert`在数据最前的第一列插入一列全1数据, 目的是为了构建回归方程中的常数项 w_0

=====>> 逻辑回归算法 <<=====

算法介绍：实现逻辑回归，对处理好的数据，计算出回归方程，采用梯度上升的方法

参考书本内容已经网上博客，编写了普通梯度上升，随机梯度上升，随机改进梯度上升三种方法，参考原理公式如下：

$$\text{sigmoid函数 } h(z) = \frac{1}{1 + e^{-z}}$$
$$w_j := w_j + \alpha \sum_{i=1}^m \left(y^{(i)} - h_w(x^{(i)}) \right) x_j^{(i)}$$

- **sigmoid函数**

```
def sigmoid(z):  
    return 1 / (1 + np.exp(-z))
```

- - **代码细节：**代码比较简单，直接使用公式代入即可

- **普通梯度上升函数：**

```
def grad_ascent(dataMatIn, classLabels):  
    dataMatrix = np.mat(dataMatIn) # (m,n)  
    labelMat = np.mat(classLabels).transpose()  
    m, n = np.shape(dataMatrix)  
    weights = np.ones((n, 1)) # 初始化回归系数 (n, 1)  
    alpha = 0.001 # 步长  
    maxCycle = 500 # 最大循环次数  
  
    for i in range(maxCycle):  
        h = sigmoid(dataMatrix * weights) # sigmoid 函数  
        error = labelMat - h # y-h, (m - 1)  
        weights = weights + alpha * dataMatrix.transpose() * error  
    return weights
```

- - **代码细节：**

(1) 首先读取训练数据的尺寸，设计循环，每次循环按照原理中的数学公式计算即可

- **随机梯度上升函数：**

普通算法中，每次循环矩阵都会进行 $m * n$ 次乘法计算，时间复杂度是 $\text{maxCycles} * m * n$ 。当数据量很大时，时间复杂度是很大。这里尝试使用随机梯度上升法来进行改进。随机梯度上升法的思想是，每次只使用一个数据样本点来更新回归系数。这样就大大减小计算开销。

```
def stoc_grad_ascent(dataMatIn, classLabels):
    m, n = np.shape(dataMatIn)
    alpha = 0.01
    weights = np.ones(n)
    for i in range(m):
        h = sigmoid(sum(dataMatIn[i] * weights)) # 数值计算
        error = classLabels[i] - h
        weights = weights + alpha * error * dataMatIn[i]
    return weights
```

- 代码细节:

(1) 循环部分, 不在每次计算整个训练数据与回归系数相乘, 而是每次训练数据中的一个数据进行计算 $h = \text{sigmoid}(\text{sum}(\text{dataMatIn}[i] * \text{weights}))$

- 随机梯度上升改进函数:

进行随机的样本选取, 解决参数周期性波动问题。

α 的取值问题, 将其设置为一个递减的动态参数, 对于如此调整的原因, 这是由于, 在学习的过程中, 一开始可以设置较大的步长对参数进行调整, 但是随着学习过程的加深, 应当减小参数的变化程度使得参数达到稳定。同时为了使后面循环的影响不为0, 设置了一个最小值 0.0001, 即使再加大循环次数, 也会持续的对参数进行修正

```
def stoc_grad_ascent_one(dataMatIn, classLabels, numIter=150):
    m, n = np.shape(dataMatIn)
    weights = np.ones(n)
    for j in range(numIter):
        dataIndex = list(range(m))
        for i in range(m):
            alpha = 4 / (1 + i + j) + 0.0001 # alpha动态设置
            randIndex = int(np.random.uniform(0, len(dataIndex)))
            h = sigmoid(sum(dataMatIn[i] * weights)) # 数值计算
            error = classLabels[randIndex] - h
            weights = weights + alpha * dataMatIn[randIndex] * error
            del (dataIndex[randIndex])
    return weights
```

- 代码细节:

(1) 采用两层循环, 每次对于数据随机取值, 采用 `np.random.uniform()`
 (2) $\alpha = 4 / (1 + i + j) + 0.0001$, α 动态设置, 根据每次循环的序号改变

- 分类器估计函数:

$y = w_0 + w_1x_1 + w_2x_2 + \dots$ 求和回归系数和计算数据的矩阵乘积

```
def classifyVector(inX, trainWeights):
    prob = sigmoid(sum(inX * trainWeights))
    if prob > 0.5:
        return 1
    else:
        return 0
```

- 代码细节:

(1) sigmoid函数在 $x=0$ 的时候为0.5, 所以当结果 >0.5 时, 判定为1, 反之为0

- 精度估计函数

```
def Test_predict(weights, test_dataMatIn, test_classLabels):
    error = 0
    for test_data, test_label in zip(test_dataMatIn, test_classLabels):
        label = classifyVector(test_data, weights)
        if label != test_label:
            error = error + 1
    error_rate = float(error) / len(test_classLabels)
    print('测试集预测结果错误率为: ', error_rate)
    return error_rate
```

- 代码细节:

(1) 将分类器预测的结果与测试集的真实值循环依次比较是否相等, 计算错误个数, 与总数相除即可得出错误率

===== >> main函数 <<=====

算法介绍: 调用上述编写的算法函数, 实现邮件分类

```
if __name__ == '__main__':
    train_dataMatIn, train_classLabels =
    init_data('Task1/horseColicTraining.txt')
    print("训练集数据导入完成")
    weights = stoc_grad_ascent_one(train_dataMatIn, train_classLabels) # 计算权值
    print("回归系数w计算完成")
    test_dataMatIn, test_classLabels = init_data('Task1/horseColicTest.txt')
    print("训练集数据导入完成")
    rate_list = []
    for i in range(10):
        rate_list.append(Test_predict(weights, test_dataMatIn,
test_classLabels)) # 进行预测
    rate_ave = np.mean(rate_list)
    print('平均错误率为: ', rate_ave)
```

- 代码细节:

(1) 循环调用10次, 对于每一次预测错误概率保存成列表

(2) 使用np.mean()函数求列表平均值, 计算平均精度.

二、运行结果

截图一_运行结果, 单次运行

```
f:\own_file\课程文件\机器学习导论\神经网络\神经网络\Task1\task1.py:1
    return 1 / (1 + np.exp(-z))
回归系数w计算完成
训练集数据导入完成
测试集预测结果错误率为:  0.3283582089552239
PS F:\own_file\课程文件\机器学习导论\神经网络\神经网络>
```

- 图片分析：将主函数循环执行一次，打印出结果为**错误率为0.328**，由于特征的缺失值很多，可以得出手写算法的精度还是比较高的

截图二_运行结果，循环求均值

```
124     rate_list = []
125     for i in range(10):
终端    问题 2    输出    调试控制台    2: Code
PS F:\own_file\课程文件\机器学习导论\神经网络\神经网络> python -u "f:\own_file\课程文件\机器学习导论\神经网络\神经网络\Task1\task1.py"
训练集数据导入完成
f:\own_file\课程文件\机器学习导论\神经网络\神经网络\Task1\task1.py:17: SyntaxWarning: 'exp' is no longer a keyword
entered in exp
    return 1 / (1 + np.exp(-z))
回归系数w计算完成
训练集数据导入完成
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
测试集预测结果错误率为: 0.3283582089552239
平均错误率为: 0.3283582089552239
PS F:\own_file\课程文件\机器学习导论\神经网络\神经网络>
```

- 图片分析：将主函数循环10次取循环，保留每次测试错误率，共同结果如上图所示。**错误率比较稳定，错误率平均值仍然为0.328**

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
3. 逻辑回归算法的构建，重点在于对于数学公式的理解，遇到难以解决的问题可以先单步调试，找到问题一步步修改

TASK2: 使用神经网络完成新闻分类

一、软件设计

1. 设计目标

① 数据集讲解：

该数据集用于文本分类，包括大约20000个左右的新闻文档，均匀分为20个不同主题的新闻组集合，其中：

训练集：包括11314个新闻文档及其主题分类标签。训练数据在文件train目录下，训练新闻文档在train_texts.dat文件中，训练新闻文档标签在train_labels.txt文档中，编号为0~19，表示该文档分属的主题标号。

测试集：包括7532个新闻文档，标签并未给出。测试集文件在test目录下，测试集新闻文档在test_texts.dat文件中。

② 设计提示思路

1. 数据集读取：读取文件train_texts.dat和test_texts.dat方式如下，以train_texts.dat为例，test_texts.dat读取方式相同：(标签文件为正常txt文件，读取方式按照读取txt文件即可)

```
import pickle
file_name = 'train_texts.dat'
with open(file_name, 'rb') as f:
    train_texts = pickle.load(f)
```

2. 文档特征提取：因为每篇新闻都是由英文字符表示而成，因此需要首先提取每篇文档的特征，把每篇文档抽取为特征向量，可以采用多种特征提取方式，这里给出建议为提取文档的TF-IDF特征，即词频 (TF) -逆文本频率 (IDF)

提取文档的TF-IDF特征可以通过sklearn.feature_extraction.text中的TfidfVectorizer来完成，具体实现代码如下：

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=10000)
vectors_train = vectorizer.fit_transform(train_texts)
```

3. 后续算法：在完成每篇新闻文档的特征提取后，就可以构建神经网络模型进行训练，具体的网络模型结构，大家可以自行尝试

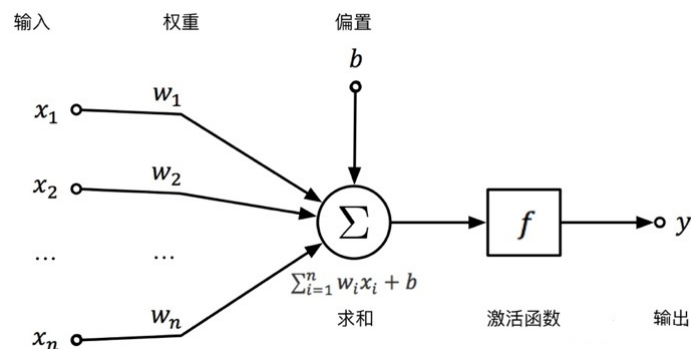
③ 目标要求

- 需编写实验报告说明具体实验流程，模型结构等
- 预测测试集新闻文档分类结果，将预测结果与训练集标签相同的存储方式进行存储，存储为txt文件
- 每一行表示一个新闻的分类标号，将预测结果txt文件上传到系统作业指定栏中，如下所示。

2. 算法设计分析

所用项目环境 IDE：VS code Python版本：python3.7.5

①神经网络基本概念



- 连接 (Connection)：神经元中数据流动的表达方式。
- 求和结点 (Summation Node)：对输入信号和权值的乘积进行求和。
- 激活函数 (Activate Function)：一个非线性函数，对输出信号进行控制（后文将会详细介绍激活函数）。
- x_1, x_2, \dots, x_n 为输入信号的各个分量；
- w_1, w_2, \dots, w_n 为神经元各个突触的权值；
- b 为神经元的偏置参数；

- Σ 为求和节点， $z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b = \sum_{i=1}^n w_i x_i + b$ ；
- f 为激活函数，一般为非线性函数，如 Sigmoid、Tanh 函数等；
- y 为该神经元的输出。

进行抽象，得到神经元的数学基本表达式为：

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

假设 $W = [w_1, w_2, \dots, w_n]$ 为权值向量， $X = [x_1, x_2, \dots, x_n]$ 为输入向量。一个神经元的基本功能是对输入向量 X 与权值向量 W 内积求和后并加上偏置参数 b ，经过非线性的激活函数 f ，得到 y 作为输出结果。因此神经元又可以使用矩阵形式表达为：

$$y = f(W^T X + b)$$

② TD-IDF

1. 概念

- (1)TF-IDF (term frequency-inverse document frequency) 是一种用于资讯检索与资讯探勘的常用加权技术。
- (2)TF-IDF是一种统计方法，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。
- (3)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中出现的频率成反比下降。
- (4)TF-IDF加权的各种形式常被搜寻引擎应用，作为文件与用户查询之间相关程度的度量或评级。

2. 主要思想

- (1)如果某个词或短语在一篇文章中出现的频率TF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。

(2)TFIDF实际上是： $TF * IDF$ ，TF词频(Term Frequency)，IDF反文档频率(Inverse Document Frequency)。TF表示词条在文档d中出现的频率（另一说：TF词频(Term Frequency)指的是某一个给定的词语在该文件中出现的次数）。

(3)IDF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（见后续公式），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。（另一说：IDF反文档频率(Inverse Document Frequency)是指果包含词条的文档越少，IDF越大，则说明词条具有很好的类别区分能力。）

(4)但是实际上，有时候，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。这就是IDF的不足之处。

③代码实现

较长的函数具体代码没有给出，给出函数定义及返回值

=====>> 数据处理 <<=====

- **txt文件读取算法：**

```
def txtread(filepath):  
    with open(filepath, 'r') as fr:  
        file_data = [inst.strip() for inst in fr.readlines()]  
    print(filepath, '文件读取成功')  
    return file_data
```

- - **代码细节：** 代码简单，使用文件操作，循环写入即可

- **txt文件保存算法：**

```
def txtsave(filename, data):  
    file = open(filename, 'w')  
    for i in range(len(data)):  
        s = str(data[i]).replace('[', '').replace(']', '').replace(',', '\t') # 去除  
        s = s.replace('"', '').replace("'", '').replace('.', '\t') + '\n' # 去除  
        file.write(s) # 去除  
    file.close()  
    print(filename, "保存文件成功")
```

- - **代码细节：** 代码简单，使用文件操作，使用.replace() 去除不需要的换行符等

- **dat文件读取算法：**

```
def DataRead(filename):  
    with open(filename, 'rb') as f:  
        data_texts = pickle.load(f)  
    print(filename, '读取完成')  
    return data_texts
```

- - **代码细节：** 代码简单，使用pickle.load()读取数据并返回

- 总数据处理函数:

```
def DataProcess():
    train_texts = DataRead('Task2/train/train_texts.dat')
    test_texts = DataRead('Task2/test/test_texts.dat')
    train_labels = txtread('Task2/train/train_labels.txt')
    print('数据读取完毕')
    return train_texts, test_texts, train_labels
```

- ○ 代码细节: 调用设计好的DataRead函数, txtread函数读取数据即可

Returns:
train_texts, 训练集特征数据
test_texts, 测试集特征数据
train_labels, 训练集标签列表

===== >> 神经网络 << =====

算法介绍: 调用sklearn库使用神经网络, 对训练集生成分类器

- TF_IDF特征提取

```
Vectorizer = TfidfVectorizer(max_features=10000)
Vectors_train = Vectorizer.fit_transform(train_texts)
train_features = csr_matrix(Vectors_train).toarray()
print("训练集TF_IDF特征提取完毕")

Vectors_test = Vectorizer.transform(test_texts)
test_features = csr_matrix(Vectors_test).toarray()
print("测试集TF_IDF特征提取完毕")
```

- ○ 代码细节:

- (1) 使用TfidfVectorizer.fit_transform对训练集数据TF_IDF提取
- (2) 使用csr_matrix().toarray()对提取的数据转化为二维数据array

- 分类器生成

```
X_train, X_train_test, Y_train, Y_train_test =
train_test_split(train_features, train_labels, test_size=0.2,)
print('训练集和验证集划分完毕')
# clf = MLPClassifier()
clf = MLPClassifier(hidden_layer_sizes=(100,),
                    activation="relu",
                    solver='adam',
                    alpha=0.0001,
                    batch_size='auto',
                    learning_rate="constant",
                    learning_rate_init=0.001, )
clf.fit(X_train, Y_train)
print('模型训练完毕')

train_accuracy = clf.score(X_train_test, Y_train_test)
```

```
print('分类器在训练集的精度为: ', train_accuracy)

test_pre = clf.predict(test_features)
print('测试集预测标签完毕')
txtsave('Task2/test/test_labels.txt', test_pre)

return train_accuracy
```

- 代码细节:

- (1) 调用sklearn库函数train_test_split将提取特征完毕的训练集数据划分为训练集和验证集, 验证集用于评级分类器的精度, 以及调参
- (2) 调用sklearn库函数MLPClassifier生成神经网络分类器, 使用.fit拟合训练数据, 使用.score()函数计算分类器在验证集上的精度, 使用.predict()预测测试集标签, 并保存为txt文件

=====>> main函数 <<=====

算法介绍: 调用上述编写的算法函数, 实现新闻文本分类

```
def main():
    train_texts, test_texts, train_labels = DataProcess()
    network_clf(train_texts, test_texts, train_labels)
```

- 代码细节:

- (1) 调用之前设计的数个函数即可

二、运行结果

采用Google colab运行代码(自己的笔记本性能较差)

- 多次调整参数, 取得最好结果的时候如下:

```
task2.ipynb
文件 修改 视图 插入 代码块执行程序 工具 帮助 已保存所有更改
RAM 磁盘
+ 代码 + 文本
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2
```

```
if __name__ == "__main__":
    main()
```

```

/contnet/drive/MyDrive/python/train/train_texts.dat 读取完成
/contnet/drive/MyDrive/python/test/test_texts.dat 读取完成
/contnet/drive/MyDrive/python/train/train_labels.txt 文件读取成功
数据读取完毕
训练集TF_IDF特征提取完毕
测试集TF_IDF特征提取完毕
训练集和验证集划分完毕
模型训练完毕
分类器在训练集的精度为: 0.9142730888201502
测试集预测标签完毕
/contnet/drive/MyDrive/python/test/test_labels.txt 保存文件成功

```

• 参数分析

```

clf = MLPClassifier(hidden_layer_sizes=(100,),
                    activation="relu",
                    solver='adam',
                    alpha=0.0001,
                    batch_size='auto',
                    learning_rate="constant",
                    learning_rate_init=0.001, )

```

参数	备注
hidden_layer_sizes	第i个元素表示第i个隐藏层中的神经元数量。
activation	{'identity', 'logistic', 'tanh', 'relu'}, 'relu' 隐藏层的激活函数: 'identity', 无操作激活, 对实现线性瓶颈很有用, 返回 $f(x) = x$; 'logistic', logistic sigmoid函数, 返回 $f(x) = 1 / (1 + \exp(-x))$; 'tanh', 双曲tan函数, 返回 $f(x) = \tanh(x)$; 'relu', 整流后的线性单位函数, 返回 $f(x) = \max(0, x)$
slover	{'lbfgs', 'sgd', 'adam'}, 'adam'。权重优化的求解器: 'lbfgs'是准牛顿方法族的优化器; 'sgd'指的是随机梯度下降。'adam'是指由Kingma, Diederik和Jimmy Ba提出的基于随机梯度的优化器。注意: 默认解算器“adam”在相对较大的数据集（包含数千个训练样本或更多）方面在训练时间和验证分数方面都能很好地工作。但是, 对于小型数据集, “lbfgs”可以更快地收敛并且表现更好。
alpha	float, 可选。L2惩罚 (正则化项) 参数。
batch_size	用于随机优化器的minibatch的大小。如果slover是'lbfgs', 则分类器将不使用minibatch。设置为“auto”时, $batch_size = \min(200, n_samples)$
learning_rate	用于权重更新。仅在solver = 'sgd'时使用。'constant'是'learning_rate_init'给出的恒定学习率;
learning_rate_init	使用初始学习率。它控制更新权重的步长。仅在solver = 'sgd'或'adam'时使用。

- **图片分析:** 上图为使用TF_IDF、MLPClassifier生成的分类器精度, 可以看出**精度在0.914左右**, 实验代码的精度还是比较高的

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力