华中科技大学

机器学习导论作业报告

专业院系 ：电子信息与通信学院
专业班级 ：电信1805班
学生学号 ：U201813372
学生姓名 ：朱浩然

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

- **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系数的值（后面会介绍）；

  2，由于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**

①Logistic回归

- **Logistic回归概述**

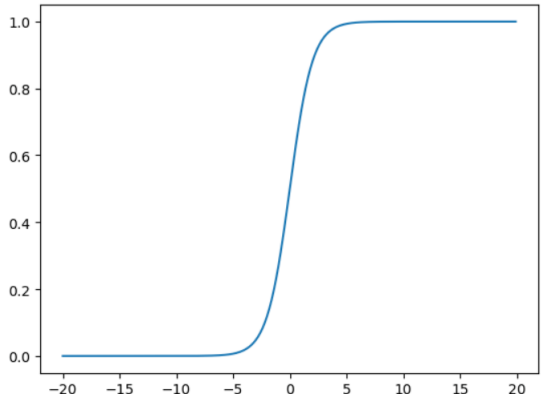
  Logistic回归的主要思想是，根据现有的数据对分类边界建立回归公式，从而实现分类（一般两类）。"回归"的意思就是要找到最佳拟合参数，其中涉及的数学原理和步骤如下：
  （1）需要一个合适的分类函数来实现分类【单位阶跃函数、Sigmoid函数】
  （2）损失函数（Cost函数）来表示预测值$(h(x))$与实际值(y)的偏差$(h-y)$,要使得回归最佳拟合，那么偏差要尽可能小（偏差求和或取均值）。
  （3）记$J(w)$表示回归系数为ω时的偏差，那么求最佳回归参数ω就转换成了求$J(w)$的最小值。【梯度下降、上升法】

- **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：$z=w_0x_0+w_1x_1+w_2x_2+\ldots+w_nx_n=w^Tx$

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

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

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

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：

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

推导出代价函数Cost函数：

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

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Cos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αα（**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:

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

求得结果为：

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

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W0

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

- **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 = sigmoid(sum(dataMatIn[i] * 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 ,alpha动态设置，根据每次循环的序号改变

- **分类器估计函数:**

> y = w0 + w1x1 + w2x2 + ...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.

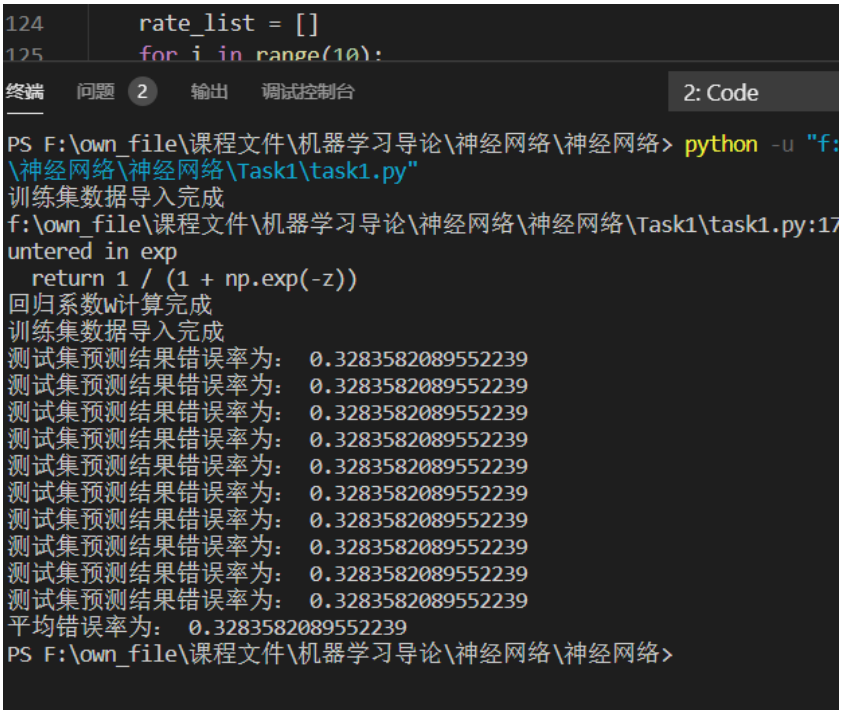
二、运行结果

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



- 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

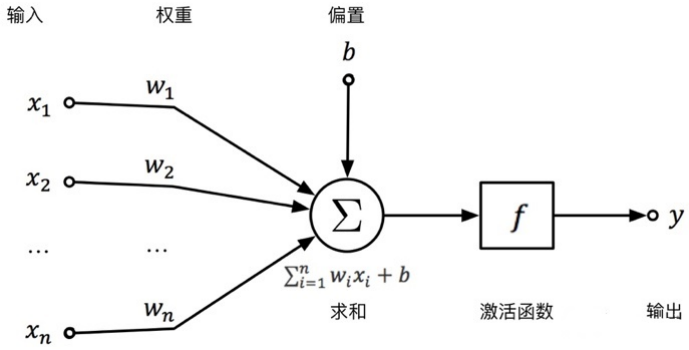
③ 目标要求

- 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）。
- $x_1, x_2, \ldots, x_n$ 为输入信号的各个分量；
- $w_1, w_2, \ldots, w_n$ 为神经元各个突触的权值；
- b 为神经元的偏置参数；
- $\sum$ 为求和节点，$z = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b = \sum_{i=1}^{n} w_i x_i + b$；
- 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：

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

假设 $W = [w_1, w_2, \cdots, w_n]$ 为权值向量，$X = [x_1, x_2, \cdots, x_n]$ 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'  # 去除单引号，逗号，每行末尾追加换行符
        file.write(s)
    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TD_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)
```

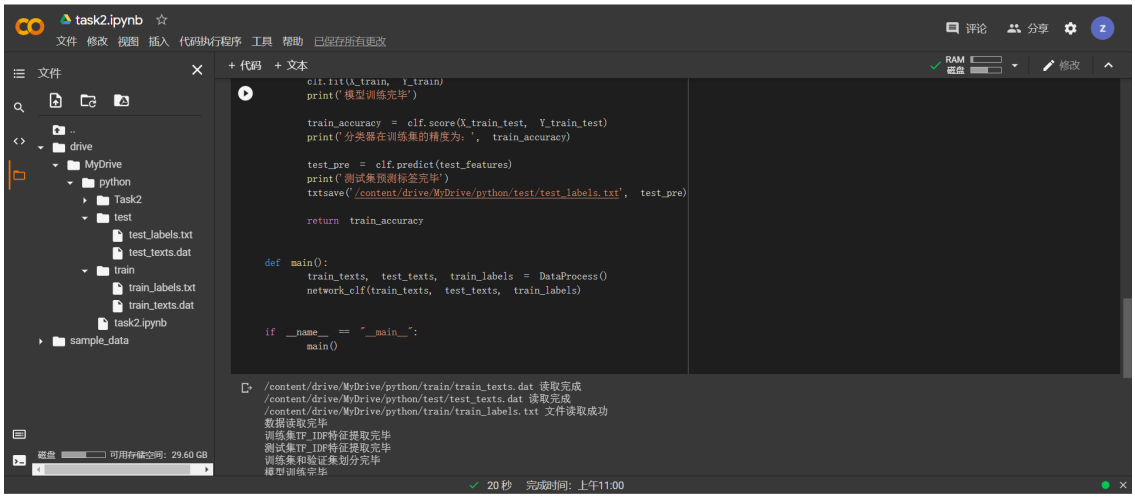
- ○ **代码细节**：

  (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：



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

```
/content/drive/MyDrive/python/train/train_texts.dat 读取完成
/content/drive/MyDrive/python/test/test_texts.dat 读取完成
/content/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
/content/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