电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2019091604016

姓 名 夏野

（实验） 课程名称 人工智能

实验时间 12月1日

实验教师 周帆

**电 子 科 技 大 学**

**实 验 报 告**

1. **实验名称：**

分类任务——真假钞票鉴别（二分类）

1. **实验学时：**

4学时

1. **实验目的：**

结合前面基础实验所学习到的各种模型，对实际问题进行应用，同时比较这些机器学习模型在实际问题中的表现情况。

1. **实验原理：**

逻辑回归：属于机器学习里面的监督学习，它是以回归的思想来解决分类问题的一种非常经典的二分类分类器。由于其训练后的参数有较强的可解释性，在诸多领域中，逻辑回归通常用作 baseline 模型，以方便后期更好的挖掘业务相关信息或提升模型性能。

SVM：支持向量机（support vector machines，SVM）是一种二分类模型，它将实例的特征向量映射为空间中的一些点，SVM 的目的就是想要画出一条线，以 “最好地” 区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。SVM 适合中小型数据样本、非线性、高维的分类问题。

梯度下降：用来最小化我们的损失函数的。在直线方程中，导数就代表斜率；而在曲线方程中，导数代表的是该点处切线的斜率。因此，为了不断更新我们的参数，也就是w、b，我们要求损失函数的梯度，并沿着梯度的反方向就行更新，这样，就可以到达一个局部最低点。

Sklearn ：(全称 Scikit-Learn) 是基于 [Python](https://so.csdn.net/so/search?from=pc_blog_highlight&q=Python) 语言的机器学习工具。它建立在 NumPy, SciPy, Pandas 和 Matplotlib 之上，里面的 API 的设计非常好，所有对象的接口简单，很适合新手上路。在 Sklearn 里面有六大任务模块：分别是分类、回归、聚类、降维、模型选择和预处理

TensorFlow：是一个使用数据流图进行数值计算的开放源代码软件库。图中的节点代表数学运算，而图中的边则代表在这些节点之间传递的多维数组（张量）。借助这种灵活的架构，您可以通过一个 API 将计算工作部署到桌面设备、服务器或移动设备中的一个或多个 CPU 或 GPU。TensorFlow 最初是由 Google Brain 团队（隶属于 Google 机器智能研究部门）中的研究人员和工程师开发的，旨在用于进行机器学习和深度神经网络研究。但该系统具有很好的通用性，还可以应用于众多其他领域。

1. **实验内容：**
2. 数据集：data\_banknote\_authentication.txt
   1. 四个特征值（连续型数值）：Variance of Wavelet Transformed image；Skewness of Wavelet Transformed image；Kurtosis of Wavelet Transformed image；Entropy of image；
   2. 标签：0为真钞，1为假钞
3. 数据集划分（共1372项）：前1200项为训练集，后172项为测试集。
4. 衡量指标：查准率（Precision）和查全率（Recall）。
5. **实验器材（设备、元器件）：**

1. 硬件：mac笔记本电脑

2. 软件：ios系统，pycharm

1. **实验步骤：**
2. 数据处理：原始数据是文本数据，以逗号为分隔符，每一行包含Variance of Wavelet Transformed image、Skewness of Wavelet Transformed image、Kurtosis of Wavelet Transformed image、Entropy of image和label这四个数据，因此使用open()函数打开文件，并使用with语句方便最后读取文件结束之后不用显式的关闭文件。将数据分隔开后，前四个数据放入X列表中，同时进行数据类型转换，将string类型转换成float数据，将最后一个数据作为label放入Y列表中。
3. 数据归一化：由于特征的数值存在较大的波动，而数据的过大浮动会让模型训练的效果很差，主要原因在于不同量纲的数据不满足正态分布，因此无法很好的拟合模型，因此这里我才用StandardScaler对数据进行归一化；
4. 分割数据：根据任务要求，将数据分隔为1200:172，分别作为训练数据和测试数据，同时，使用numpy方法array()，将数据转换为ndarray，方便后续模型的训练。此时，打印形状为：(1200, 4)和(1200, 1)；
5. 定义模型：
   1. 方法一——逻辑回归：使用sklearn包下的linear\_mode的类LogisticRegression进行模型加载，将训练数据和target丢进fit()方法中进行训练；模型训练好之后，使用predict()方法将测试数据进行预测，得到y\_pred，然后利用metrics中的precision\_score()和recall\_score()得到预测结果的Precision和Recall。
   2. 方法二——基于径向基核的SVC：使用sklearn包中的svm模块搭建模型。其中包含SVC类，我定义了一个svc对象，同时设置参数kernel=‘rbf’，也就是定义核函数为rbf，之后使用fit()方法，将训练数据和对应target作为参数，开始训练模型。之后，进行模型评估，利用metrics中的precision\_score()和recall\_score()得到预测结果的Precision和Recall。
   3. 方法三——基于深度学习前馈神经网络：tensorflow.keras模块中提供了非常容易的模型搭建方法——直接使用Sequential、functional API，我采用简单方法Sequential进行搭建。定义三层Dense，最后一层Dense单元数为1，作为预测输出。使用compile函数进行模型的编译，其中需要设置优化器，选择adam；损失函数设置为BinaryCrossentropy()。最后，matrics设置为Precision和Recall；最后，利用fit方法训练模型，设置epoch=20，batch\_size=16。为了减少过拟合的影响，我又采用了K折交叉验证，K具体值为5，每一次都对模型进行训练和预测，将预测结果保存起来最后采用投票得到最终预测结果。
6. 模型训练：对这三个模型进行训练，并且分别得到三个模型对应的查准率和查全率。
7. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**
8. 重要数据结果分析

（1）代码运行结果

对模型进行训练后，得到了不同模型的Precision和Recall，如图1所示：

|  |
| --- |
| Text  Description automatically generated  Text  Description automatically generated with low confidence  图 1 三个模型Precision和Recall的值 |

通过运行结果，我们可以对比三个不同模型之间的Precision以及Recall的值。可以看到在三个模型中基于径向基核的SVC模型的表现是最好的。

1. 核心代码流程分析：

（1）逻辑回归代码：

class LR(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.train\_X, self.test\_X = dataProcess.process(self.train\_X, self.test\_X)  
 self.train\_y = np.squeeze(self.train\_y)  
 self.lr = LogisticRegression(C=10.0, random\_state=0)  
 def train(self):  
 self.lr.fit(self.train\_X, self.train\_y)  
 def evaluate(self):  
 y\_pred = self.lr.predict(self.test\_X)  
 Precision = metrics.precision\_score(self.test\_y, y\_pred)  
 Recall = metrics.recall\_score(self.test\_y, y\_pred)  
 print('Precision of LR:', Precision)  
 print('Recall of LR:', Recall)

代码分析：

对于LR类代码来说，分为了三个部分，分别是模型初始化函数\_\_init\_\_(),模型个训练函数train()以及模型评估函数evaluate()。整个类的搭建方法和前面几个实验类似，这个不再赘述。

对与逻辑回归而言，逻辑回归是假设数据服从伯努利分布的,所以当我们通过极大化似然函数的方法、运用梯度下降来求解参数从而来达到将数据二分类的目的。逻辑回归的基本工作原理就是，在逻辑回归中y值确实是一个连续的变量。逻辑回归的做法是划定一个阈值，y值大于这个阈值的是一类，而y值小于这个阈值的是另外一类。阈值具体如何调整根据实际情况选择。通常来说会选择0.5作为我们的阈值来划分。因此其分类形式非常简单，类似于线性模型，很难去拟合数据的真实分布。但是，对于非线性分类任务来说，效果就不是很好了。

（2）rbf核函数代码：

class SVM\_model(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.train\_X, self.test\_X = dataProcess.process(self.train\_X, self.test\_X)  
 self.train\_y = np.squeeze(self.train\_y)  
 self.clf = SVC(gamma=0.01, kernel='rbf')  
 def train(self):  
 self.clf.fit(self.train\_X, self.train\_y)  
 def evaluate(self):  
 y\_pred = self.clf.predict(self.test\_X)  
 Precision = metrics.precision\_score(self.test\_y, y\_pred)  
 Recall = metrics.recall\_score(self.test\_y, y\_pred)  
 print('Precision of SVC:', Precision)  
 print('Recall of SVC:', Recall)

对于SVM\_model类代码来说，分为了三个部分，分别是模型初始化函数\_\_init\_\_(),模型个训练函数train()以及模型评估函数evaluate()。整个类的搭建方法和前面几个实验类似，这个不再赘述。

对于rbf核函数而言，我们需要先将原始的数据点（x， y）映射为新的样本（x'，y'）；然后将新的特征向量点乘（x',y'），返回其点乘结果；只与为什么我们要使用计算点积的原因，它将每一个样本点映射到一个无穷维的特征空间，因此rbf核函数的svr最大特点就是支撑集（支集）较小，影响范围较小，对于给定样本，只对样本的附近样本有影响，存在“局部响应特性”。

（3）神经网络代码：

class DeepLearning(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.train\_X, self.test\_X = dataProcess.process(self.train\_X, self.test\_X)  
  
 def build\_model(self, lr=0.001):  
 model = Sequential(  
 [  
 layers.InputLayer(input\_shape=(4,)),  
 layers.Dense(64, activation='relu'),  
 layers.Dense(32, activation='relu'),  
 layers.Dense(1, activation='sigmoid')  
 ]  
 )  
 model.compile(  
 optimizer=optimizers.Adam(lr),  
 loss=losses.BinaryCrossentropy(),  
 metrics=['Precision', 'Recall']  
 )  
 # model.summary()  
 return model  
 def train(self):  
 kf = StratifiedKFold(n\_splits=5, shuffle=True)  
 y\_pred\_all = []  
 for train\_idx, test\_idx in kf.split(self.train\_X, self.train\_y):  
 x\_train, x\_dev = self.train\_X[train\_idx], self.train\_X[test\_idx]  
 y\_train, y\_dev = self.train\_y[train\_idx], self.train\_y[test\_idx]  
 model = self.build\_model()  
 model.fit(x\_train, y\_train, epochs=5, batch\_size=16, validation\_data=(x\_dev, y\_dev))  
 y\_pred = model.predict(self.test\_X)  
 # print(y\_pred.shape)  
 y\_pred\_all.append(list(np.squeeze(y\_pred)))  
 print(y\_pred\_all)  
 for preds in y\_pred\_all:  
 for index, num in enumerate(preds):  
 preds[index] = 0 if num <= 0.5 else 1  
 y\_pred\_all = np.array(y\_pred\_all)  
 y\_pred\_vote = []  
 for i in range(len(y\_pred\_all[0])):  
 one = np.count\_nonzero(y\_pred\_all[:, i])  
 zero = 5 - one  
 y\_pred\_vote.append(1 if one > zero else 0)  
 Precision = metrics.precision\_score(self.test\_y, y\_pred\_vote)  
 Recall = metrics.recall\_score(self.test\_y, y\_pred\_vote)  
 print('Precision of MLP:', Precision)  
 print('Recall of MLP:', Recall)

代码分析

对于神经网络的代码，tensorflow.keras的模块中提供了非常容易的模型搭建方法——直接使用Sequential进行搭建。定义三层Dense，最后一层Dense单元数为1，作为预测输出。使用compile函数进行模型的编译，其中需要设置优化器，选择adam；损失函数设置为BinaryCrossentropy()。最后，matrics设置为Precision和Recall；最后，利用fit方法训练模型，设置epoch=20，batch\_size=16。为了减少过拟合的影响，我又采用了K折交叉验证，K具体值为5，每一次都对模型进行训练和预测，将预测结果保存起来最后采用投票得到最终预测结果。

最后，针对深度神经网络，看我们的数据集数量，我们可以推断MLP的效果不会很理想，尤其是迭代了50次后，训练的结果的Precision和recall都非常的高，以至于到后面Recall已经是1了，但是当对测试数据进行evaluate时，我们的结果并不是很满意，出现了过拟合的状态，因此可以看出，对于这种小样本数据，如果不做数据增强，那么机器学习的算法会是更好选择。

1. **总结及心得体会：**

通过本次任务二，我自己对于机器学习的算法有了更深的理解，并且能够自己对一些情况做出自己的分析和判断。