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：**

回归任务——地震等级预测

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：**

回归分析：指一种预测性的建模技术，主要是研究自变量和因变量的关系。通常使用线/曲线来拟合数据点，然后研究如何使曲线到数据点的距离差异最小。线性回归是回归分析的一种。假设目标值（因变量）与特征值（自变量）之间线性相关（即满足一个多元一次方程，如：f(x)=w1x1+…+wnxn+b）；然后构建损失函数；最后通过令损失函数最小来确定参数。

SVR是SVM的一种运用，基本的思路是一致，除了一些细微的区别。使用SVR作回归分析，与SVM一样，我们需要找到一个超平面，不同的是：在SVM中要找出一个间隔（gap）最大的超平面，而在SVR，我们定义一个ε，定义虚线内区域的数据点的残差为0，而虚线区域外的数据点（支持向量）到虚线的边界的距离为残差（ζ）。与[线性模型](https://www.zhihu.com/search?q=%E7%BA%BF%E6%80%A7%E6%A8%A1%E5%9E%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra={"sourceType":"article","sourceId":33692660}" \t "/Users/jackieshi/Documents\\x/_blank)类似，我们希望这些残差（ζ）最小。所以大致上来说，SVR就是要找出一个最佳的条状区域（2ε宽度），再对区域外的点进行回归。

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

1. **实验内容：**
2. 数据集：quake.dat
   1. 三个特征值：Focal\_depth（震源深度），Latitude（纬度），Longitude（经度）；
   2. 真实值：Richter（里氏震级）。
3. 数据集划分（共2178项）：前2000项为训练集，后178项为测试集。
4. 衡量指标：均方误差（MSE）和均方根误差（RMSE）
5. **实验器材（设备、元器件）：**

1. 硬件：MacBook Pro笔记本电脑

2. 软件：MAC OSX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Focal\_depth,、Latitude、Longitude和Richter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，前三个数据放入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target放入Y列表中；
3. 分割数据：根据任务要求，将数据分隔为2000:178，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：(2000, 3)和(2000, 1)；
4. 定义模型：
   1. 方法一——线性回归：使用sklearn包下的linear\_mode的类LinearRegression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mean\_squared\_error()获得均方误差，由于metrics中没有方法直接计算RMSE，这里我是用numpy方法sqrt()求MSE的平方根，得到均方根误差。
   2. 方法二——基于线性核的SVR：使用sklearn包中的svm模块搭建模型。其中包含SVR类，我定义了一个svr对象，同时设置参数kernel=‘linear’，也就是定义核函数为linear。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，同样使用mean\_squared\_error()获得均方误差，用numpy方法sqrt()求MSE的平方根，得到均方根误差。
   3. 方法三——基于径向基核的SVR：使用sklearn包中的svm模块搭建模型。其中包含SVR类，我定义了一个svr对象，同时设置参数kernel=‘rbf，也就是定义核函数为rbf。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，同样使用mean\_squared\_error()获得均方误差，用numpy方法sqrt()求MSE的平方根，得到均方根误差。
   4. 方法四——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MeanSquaredError()；最后，matrics设置为mse，rmse；最后，利用fit方法训练模型，设置epoch=20，batch\_size=16；
5. 模型训练：对这四个模型进行训练，并且分别得到四个模型对应的MSE和RMSE。
6. **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）**
7. 重要数据结果分析
8. 数据可视化结果

Chart

Description automatically generated

结果分析：

代码成功画出了测试数据集中预测值和实际值的拟合图像。可以看到所画出的图像拟合程度并不是很高，经过分析应该是因为数据量过少，导致模型还不是特别完善，所以会出现偏离。

（2）代码运行结果

对模型进行训练后，得到了不同模型的MSE和RMSE的值，下图所示：

|  |
| --- |
| Text  Description automatically generated  Text  Description automatically generated  图 各模型MSE和RMSE |

经过分析，我们可以看到，在这个四个模型中进行MSE以及RMSE两个参数的比较，我们可以看出，基于rbf核函数的SVR模型表现最好，MSE和RMSE的值都是最低的，表明你和程度最高，浮动最小，而最差的是基于线性核函数的SVR。

1. 核心代码流程分析
2. 线性回归代码

class LR(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.linreg = linear\_model.LinearRegression()  
  
 def train(self):  
 self.linreg.fit(self.train\_X, self.train\_y)  
 print('截距: ', self.linreg.intercept\_)  
 print('系数: ', self.linreg.coef\_)  
 def evaluate(self):  
 y\_pred = self.linreg.predict(self.test\_X)  
 MSE\_score = metrics.mean\_squared\_error(self.test\_y, y\_pred)  
 RMSE\_score = np.sqrt(metrics.mean\_squared\_error(self.test\_y, y\_pred))  
 print('MSE of LR: ', MSE\_score)  
 print('RMSE of LR: ', RMSE\_score)

代码流程分析：

首先针对线性回归和基于线性核函数的SVR：线性回归就是要找一条直线，并且让这条直线尽可能地拟合图中的数据点。我们调用dataProcess.read\_file()函数读取文件数据，并调用linear\_model.LinearRegression()函数建立模型调用线性回归模型进行计算后，就可以通过fit()函数训练得到该直线的截距和系数。调用mean\_squared\_error()函数以及sqrt()函数可以计算到我们得出结果的MSE值以及RMSE值。

1. 基于线性核函数的SVR代码

lass SVM\_model\_linear(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.train\_y = np.squeeze(self.train\_y)  
 self.linear\_svr = SVR(kernel='linear')  
 def train(self):  
 self.linear\_svr.fit(self.train\_X, self.train\_y)  
 def evaluate(self):  
 y\_pred = self.linear\_svr.predict(self.test\_X)  
 MSE\_score = metrics.mean\_squared\_error(self.test\_y, y\_pred)  
 RMSE\_score = np.sqrt(metrics.mean\_squared\_error(self.test\_y, y\_pred))  
 print('MSE of linear:', MSE\_score)  
 print('RMSE of linear:', RMSE\_score)

代码分析：

同样对于基于线性核函数的SVR而言，也是希望找到一个分割方法让其你和数据点。他数据初始化的方法和上一个模型的方法是差不多的。同样也是调用fit()方法进行训练，然后调用同样的方法来得到MSE以及RMSE。

1. rbf核函数代码

class SVM\_model\_rbf(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 self.train\_y = np.squeeze(self.train\_y)  
 self.rbf\_svr = SVR(kernel='rbf')  
 def train(self):  
 self.rbf\_svr.fit(self.train\_X, self.train\_y)  
 def evaluate(self):  
 y\_pred = self.rbf\_svr.predict(self.test\_X)  
 MSE\_score = metrics.mean\_squared\_error(self.test\_y, y\_pred)  
 RMSE\_score = np.sqrt(metrics.mean\_squared\_error(self.test\_y, y\_pred))  
 print('MSE of rbf:', MSE\_score)  
 print('RMSE of rbf:', RMSE\_score)

代码分析：

装载数据、搭建模型、训练、得出结果的代码形式和前面两个类似，这里不再赘述。主要介绍影响。

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'）；再将新的特征向量点乘（x',y'），返回其点乘结果。

至于我们为什么要计算点积，是因为此处只针对 SVM 中的应用；它会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”，同时也就实现非线性映射，所以效果会比非线性回归好一点。

1. 深度神经网络代码

class DeepLearning(object):  
 def \_\_init\_\_(self, FILE):  
 self.train\_X, self.train\_y, self.test\_X, self.test\_y = dataProcess.read\_file(FILE)  
 def build\_model(self, lr=0.1):  
 model = Sequential(  
 [  
 layers.InputLayer(input\_shape=(3,)),  
 # layers.Dense(64, activation='relu'),  
 layers.Dense(32, activation='relu'),  
 layers.Dense(1)  
 ]  
 )  
 model.compile(  
 optimizer=optimizers.Adam(lr=lr),  
 loss=losses.MeanSquaredError(),  
 metrics=['mse', 'acc']  
 )  
 model.summary()  
 return model  
 def train(self):  
 model = self.build\_model()  
 model.fit(self.train\_X, self.train\_y, epochs=40, batch\_size=16)  
 model.evaluate(self.test\_X, self.test\_y)  
 y\_pred = model.predict(self.test\_X)  
 y\_pred = np.squeeze(y\_pred)  
 self.test\_y = np.squeeze(self.test\_y)  
 x = [i for i in range(len(y\_pred))]  
 plt.figure()  
 plt.plot(x, y\_pred, label='predict', color='red')  
 plt.plot(x, self.test\_y, label='true', color='blue')  
 plt.legend(loc=2)  
 plt.show()

代码分析：

对于深度神经网络的代码来说，在搭建模型是与前面几个代码有所不同。我们这里选择使用的是基于深度学习前馈神经网络：tensorflow.keras，这里我采用简单方法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MeanSquaredError()；最后，matrics设置为mse，rmse；最后，利用fit方法训练模型，设置epoch=20，batch\_size=16；

理论上MLP网络的结果应该是最好的，既包含了函数的非线性性，又能够通过多层网络获得特征与target之间的联系，获得其规律性，但是结果竟然和线性回归差不多。但是由于数据量太少，不能发挥深度神经网络最好的作用，所以效果看起来并没有rbf效果好。

1. **总结及心得体会：**

通过完成任务一，我更加深刻的理解了不同模型之间的区别以及利弊。对于不同模型的搭建、训练方法更加书序。代码能力得到了很大的提高。

并且在进行模型评估时，对于MSE和RMSE两个指标有了更深的理解。前者计算可能造成量纲的不同，因此引入RMSE。