第4章 机器学习算法

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 人工智能基础与应用 | | 章名 | | | 机器学习算法 | |
| 教学内容 | 机器学习算法1 | | | | 课时 | 2 | |
| 项目性质 | √演示性 □验证性 □设计性 □综合性 | | | | | | |
| 授课班级 |  | 授课日期 | |  | | 授课地点 |  |
| 教学目标 | 掌握线性回归  掌握逻辑回归 | | | | | | |
| 教学内容 | 1. 线性回归 2. 逻辑回归 | | | | | | |
| 教学重点 | 线性回归、逻辑回归 | | | | | | |
| 教学难点 | 线性回归、逻辑回归 | | | | | | |
| 教学准备 | 装有Python的计算机  教学课件PPT  教材：《人工智能基础与应用（微课版）》 | | | | | | |
| 作业设计 |  | | | | | | |

教学过程

|  |  |
| --- | --- |
| **教学环节** | **教学内容与过程**  **（教学内容、教学方法、组织形式、教学手段）** |
| **课前组织** | 做好上课前的各项准备工作（打开计算机、打开课件、打开软件、打开授课计划、教案等），吸引学生注意力。 |
| **课程说明** | 【课前说明】  从机器学习的基本概念引入本节课学习内容。  【目的】  使学生从了解本节课的学习目标、学习重点、考评方式等方面明确课程学习的要求和目标。 |
| **课程内容描述** | 4.1 线性回归  机器学习分为监督学习（或称为有监督学习）、无监督学习、半监督学习，线性回归属于监督学习。  监督学习在训练时，不仅需要将训练的数据输入计算机，还需要将与该训练数据对应的结果（标签）输入计算机。训练完成后，将测试的数据输入计算机，计算机会对测试数据的标签进行预测，例如，要完成一个能够认识小狗的算法，在使用监督学习算法时需要将小狗的照片作为训练数据输入计算机，同时还要告诉计算机这张照片结果（标签）是小狗，算法才可以进行训练。常见的监督学习算法有线性回归、逻辑回归、KNN、SVM等。  无监督学习是给计算机提供需要训练的数据，不会将数据对应的结果（标签）输入，所以计算机无法直接确定哪个数据对应着什么标签，只能通过分析数据的特征进行特征描述。常见的无监督学习算法有聚类算法、K-均值聚类算法等。  半监督学习中提供的数据，一些有标签，一些没有标签，但没有标签的数据要多于有标签的数据，通过局部特征分析，从而获取整体特征分布。  4.1.1 什么是线性回归  线性回归是利用数理统计中的回归分析来确定两种或两种以上变量间相互依赖定量关系的一种统计分析方法，其运用领域十分广泛。  简单来说，线性回归就是要完成一个函数，该函数有输入和输出，当输入一些参数时，该函数就能得到一个结果作为输出，这个输出就是对输入的预测，且这个预测的结果是连续的。  在平面直角坐标系中确定一条直线至少需要两个点，如果在平面内绘制一条直线，那么这条直线的表达式可以是*y*=*kx*+*b*（斜截式），在斜率*k*和截距*b*已知的情况下，只要输入一个*x*，就有与之对应的*y*。线性回归是在*N*维的空间找到一条直线、一个平面或者一个超平面，使其能够拟合提供的数据，从而可以预测新的数据。在线性回归中，公式为*y*=*Wx*+*b*，*W*是权重，*b*是偏置。  4.1.2 线性回归例子引入  在复习过程中，投入不同的复习时间，最后考试的成绩也不尽相同，而且由于复习的内容与快慢不同，所以不会有时间和成绩完全成正比的情况。随机生成一些数据*x*作为时间投入，*y*作为考试成绩，找到一条直线，让这条直线尽可能地拟合图中的数据点，复习过程中时间投入与考试成绩的图像如图所示。  4-1  复习过程中时间投入与考试成绩图  在这个回归任务中，需要找到“最佳拟合线”使得误差最小化。假设找到了这条最佳的拟合线，方程为*y*=*kx*+*b*，则对于每一个样本点*x*(*p*)，根据拟合的线可知对应的预测值为，其真值为*y*，预测值为。拟合后的线如图所示。  4-2  拟合后的线  希望拟合后的线上每一个*y*和的差距都尽可能小，即这条线上所有样本点*x*(*p*)对应的真值和预测值之差的和是最小的，真值与预测值的差距反映在二维空间上，表现为两者距离的长短，从而将求最优化的拟合线问题转换为求最短距离问题。计算距离常用的方法有L1距离和L2距离等。  L1距离是一个拐角距离，对它而言只有上、下、左、右4个方向，所以求取的并不是两点之间的最短距离。平面内从（1，1）点到（4，4）点的L1距离如图所示。  4-3  从（1，1）点到（4，4）点的L1距离  L1距离公式为，（*x*1，*y*1）是起点的坐标，（*x*2，*y*2）是终点的坐标。  L2距离是两点之间或多点之间距离的计算方法，从几何学的角度，可以理解为它在计算两个向量间的欧氏距离。平面内从（1，1）点到（4，4）点的L2距离如图所示。  4-4  从（1，1）点到（4，4）点的L2距离  L2距离公式为，（*x*1，*y*1）是起点的坐标，（*x*2，*y*2）是终点的坐标。  综上，在寻找复习时间和考试成绩的最佳拟合曲线时，是不是也可以用L1距离或者L2距离来做呢？答案是肯定的，本书中使用L2距离完成拟合。  此时目标就确立了，即所有的样本点*x*(*p*)对应的拟合线上预测值与真实值*y*的距离之和要尽可能小，也就是说要尽可能小，结合，使得尽可能小，由于*x*和*y*是已知的，所以需要找到*k*和*b*。在机器学习中，斜率*k*一般称为权重*W*，截距*b*称为偏置*b*，*y*(*p*)是真值，是预测值。这个过程就是使用最小二乘法解决问题的思路。  在机器学习中，需要通过分析问题确定问题的损失函数，通过最优化损失函数获得机器学习的模型。在本问题中，选择为损失函数，由于*x*和*y*都是已知的，所以需要找到参数*k*和*b*，使得损失函数的值最小。  4.1.3 数学方法解决线性回归问题  利用数学的方法寻找损失函数的最小值，即求函数的极值。求函数的极值问题可以转换为对损失函数的各个未知分量进行求导，得到*k*和*b*的值，这里先给出最后结果，即，*。*  推导过程需要使用微积分，读者也可以跳过推导过程直接学习方法实现。  1．导入必要的库  在PyCharm中新建项目machine\_learning，在machine\_learning项目下新建Linear\_regression.py，在PyCharm中编写以下代码。  import matplotlib.pyplot as plt  import numpy as np  需要NumPy库转换数据类型，需要Matplotlib的pyplot实现画图可视化。  2．设置一个初始的*W*和*b*  # y = W \* x + b  W = 0.9  b = 15  写代码的时候使用*W*代表权重，不使用*k*。为了保证名词统一以及专业性，之后的斜率*k*统一改为权重*W*。  3．在二维平面上创造一些数据  train\_data = []  for i in range(150):  # 从[0.0,80.0)中随机采样，注意定义域是左闭右开，即包含0.0，不包含80.0  tr\_x = np.random.uniform(0.0, 80.0)  # 高斯分布的概率密度函数  # tr\_x：float，此概率分布的均值（对应着整个分布的中心center）  # 3：float，此概率分布的标准差（对应于分布的宽度）  tr\_y = tr\_x \* W + b  train\_data.append([np.random.normal(tr\_x, 3), np.random.normal(tr\_y, 3)])  随机生成一些横纵坐标数据，横坐标为学习的小时数，纵坐标为最后的成绩，生成150个点进行拟合。由于直接使用随机数生成的点还是分布在一条直线上，所以采用高斯分布的密度函数对生成的*x*和*y*值进行范围性的随机化，使这些点不完全在一条直线上，而是散落在该直线的两侧。  4．将数据在水平方向上平铺，形成一个新的数组  data = np.hstack(train\_data).reshape(-1, 2)  5．绘制散点图  plt.xlim((0, 85))  plt.ylim((10, 95))  plt.xlabel('time/h')  plt.ylabel('fraction')  # 冒号左边是行范围，冒号右边是列范围  # 第一维全取，第二维取第0个和第1个方向  plt.scatter(data[:, 0], data[:, 1], color="cyan", edgecolor="white")  plt.show()  plt.xlim()函数规定了*x*轴范围，plt.xlabel()函数绘制*x*轴的标签，plt.scatter()函数绘制散点图，颜色为cyan（青色），边缘为white（白色）。由于点比较多，所以连在一起难以区分，用边缘可便于区分，绘制后的散点图如图所示。  4-5  绘制后的散点图  6．创建训练函数  一般来说，机器学习的程序算法中需要两个自定义函数：一个是训练函数train(x,y)，该函数的作用是将需要训练的数据和标签输入后进行训练；另一个函数是predict(x)，它的作用是预测输入的新数据的分类标签。  def train\_math(train\_data):  # 获取数据  trainx = [train\_d[0] for train\_d in train\_data] # list  trainy = [train\_d[1] for train\_d in train\_data]  xtr = np.hstack(trainx)  ytr = np.hstack(trainy)  x\_mean = np.mean(xtr)  y\_mean = np.mean(ytr)  numerator = 0.0 # 分子  denominator = 0.0 # 分母  for x\_i, y\_i in zip(xtr, ytr):  numerator += (x\_i - x\_mean) \* (y\_i - y\_mean)  denominator += (x\_i - x\_mean) \*\* 2  W = numerator / denominator  b = y\_mean - W \* x\_mean  print("W = ", W)  print("b = ", b)  return W, b  7．创建预测函数  def predict(x, W, b, data):  y = W \* x + b  plt.xlim((0, 85))  plt.ylim((10, 95))  plt.xlabel('时间/h')  plt.ylabel('分数（分）')  plt.scatter(data[:, 0], data[:, 1], color="c", edgecolor="white")  plt.plot([0, 85], [b, W \* 85 + b], color='y', linewidth=2)  plt.scatter(x, y, color="r", edgecolor="white")  plt.show()  传入需要预测的*x*，同时为了更加直观地了解拟合后的直线，将data传入函数中，*W*和*b*作为模型也传入函数中，绘制生成的点、拟合的直线以及预测学习50h的成绩点。  8．调用训练函数  W, b = train\_math(data)  predict(50, W, b, data)  关掉Matplotlib打印的图像，得到*W*和*b*的值，结果如图所示。    结果  设置的*W*约为0.9，*b*约为15，由于在创建随机点时加入了高斯分布的密度函数，所以所有的点不会直接分布在直线上，结果比较接近实际的权重和偏置。需要注意的是，由于每次的初始化数据都是随机生成的，所以得到的*W*和*b*都是不一致的。拟合的直线以及预测的点如图所示。  4-7  拟合的直线以及预测的点  4.1.4 利用TensorFlow解决线性回归问题  利用TensorFlow解决线性回归问题之前需要介绍一些概念。  4.1.3小节中用数学方法解决线性回归问题，其本质是最小化一个损失函数，通过最小二乘法直接计算损失函数对应参数的数学解。但是很多机器学习的模型是求不到这样的数学解的，那么基于这样的模型，就需要使用一种基于搜索的策略来找到这个最优解。梯度下降法就是机器学习领域最小化损失函数的一个最为常用的方法。  线性函数的形式为，根据这个函数形式推导出它的损失函数为。但是这个损失函数是有问题的，如果有150个点，则可得到一个损失值*L*，而如果有15000个点，那么损失*L*会很大。损失函数的值与样本数据的数量是不应该有关系的，所以，对损失函数的值求平均值得到新的损失函数值，那么损失函数值的大小和样本数据的数量就没有关系了，得到新的损失函数为。寻找使损失值*L*最小的*W*和*b*的过程称为最优化（Optimization）。最优化的过程可以比作：一个蒙着眼睛的人在山地上行走，目标是要走到山底，他只能通过感受地形的变化一点一点地完成目标。在数学上，地形变化是不需要猜测的，可以计算出来，最陡峭的方向称为损失函数的梯度（Gradient）。在一维函数中，斜率是函数在某一点的瞬时变化率，梯度是函数斜率的一般化表达，它是一个向量，表示某一函数在该点处的方向，导数沿着该方向取得最大值，即函数在该点处沿着该方向（该梯度的方向）变化最快，变化率最大（为该梯度的模）。由于损失函数值需要减小，所以需要向着梯度的负方向更新，这个方法称为梯度下降法。  下面依然来看蒙着眼睛下山的例子，下山时迈不同大小的步子会导致下到山底所花费的时间不一样。如果迈的步子太大，会导致找不到最低点，一直在最低点左右徘徊；步子太小，虽然能更好地找到最低点，但是过程会比较漫长。这个步长就是学习率。  接下来使用TensorFlow解决引入的线性回归问题。  1．导入必要的库  import tensorflow as tf  在之前的基础上，还需要导入TensorFlow的库。  2．创建一个训练函数  def train\_tf(train\_data):  # 1.获取数据  trainx = [train\_d[0] for train\_d in train\_data] # list  trainy = [train\_d[1] for train\_d in train\_data]  # 2.构造预测的线性回归函数：y= W \* x + b  W = tf.Variable(tf.random\_uniform([1])) # 从均匀分布中返回随机值，即[0，1)  b = tf.Variable(tf.zeros([1])) # 在一维数组里放一个值  y = W \* trainx + b  # 3.判断假设的函数的好坏  cost = tf.reduce\_mean(tf.square(y - trainy))  # 4.优化函数  optimizer = tf.train.AdamOptimizer(0.05)  train = optimizer.minimize(cost)  # 5.开始训练  with tf.Session() as sess:  # 初始化所有变量值  sess.run(tf.global\_variables\_initializer())  # 将画图模式改为交互模式  plt.ion()  for k in range(1000):  sess.run(train)  # 构造图形结构  # 实时地输出训练好的W和b  if k % 50 == 0:  print("第", k ,"步：","cost=", sess.run(cost), "W=",  sess.run(W), "b=", sess.run(b))  plt.cla() # 清除原有图像  plt.plot(trainx, trainy, 'co', label='train data') # 显示数据  plt.plot(trainx, sess.run(y), 'y', label='train result') # 显示拟合  plt.pause(0.01)  plt.ioff() # 关闭交互模式  plt.close() # 关闭当前窗口  print("训练完成！")  # 输出训练好的W和b  print("finally\_cost=", sess.run(cost), "finally\_W=", sess.run(W),  "finally\_b=", sess.run(b))  return sess.run(W)[0], sess.run(b)[0]  tf.reduce\_mean()函数用于计算张量Tensor沿着指定数轴（Tensor的某一维度）的平均值，主要用于降维或计算结果的平均值。第4步中，用梯度下降算法找最优解，通过梯度下降法为最小化损失函数增加了相关的优化操作。在训练过程中，先实例化一个优化函数，并基于一定的学习率进行梯度优化训练，如tf.train.AdamOptimizer()，该优化函数是一个寻找全局最优点的优化算法，引入了二次方梯度校正；使用minimize()操作，不仅可以优化及更新训练的模型参数，也可以为全局步骤（Global Step）计数，函数的参数传入损失值节点cost，再启动一个外层的循环，优化器就会按照循环的次数沿着cost最小值的方向优化参数。第5步开始训练，先初始化所有变量值和操作，打开plt的交互模式，开始训练并实时显示拟合的效果。  3．调用训练函数  W, b = train\_tf(data)  predict(50, W, b, data)  完成调用后执行该程序，经过1000步的训练之后，得到*W*和*b*，预测结果如图所示。  4-8  预测结果  *W*和*b*的值如图所示。由于代码中每50步打印1次，1000步为0～999，所以图4-9中只打印到第950步。    *W*和*b*的值  基于TensorFlow实现简单线性回归，样本只有一个特征值。如果样本有多个特征值，就需要进行多元线性回归。  在简单线性回归中，横坐标*x*(*p*)表示的是一个值;如果横坐标*x*(*p*)对应的是一组向量，则公式的形式就变为，这个公式就是多元线性回归的公式。如果将该公式降维为简单线性回归，就是偏置*b*，就是权重*W*。  对于多元线性回归，将权重*W*作为一个矩阵处理，输入的*x*也必须是一个矩阵，如输入的*x*矩阵为，权重*W*为，那么的结果就是一个的矩阵，加上偏置*b*，就可以得到*y*的值。  4.2 逻辑回归  逻辑回归虽然被称为回归，但一般被用来处理分类问题，它可将样本的特征和样本发生的概率联系起来，是一种监督学习的方法。  4.2.1 什么是逻辑回归  在线性回归中，输入和输出呈一定的关系且输出为连续的，即。如果输出的*y*不表示为一个值，而是一个概率*p*，则公式更改为，从线性回归输出一个具体的值到输出概率的过程需要一个中间公式，这个中间公式为。当没有中间公式的转换时，就是一个回归问题；加了中间公式之后，就是一个分类问题。加入这个公式后，对于连续的输入，输出是不连续的，即每一个输入都会被分到某一类输出中，这个中间公式可以解决二分类问题，如果想解决多分类问题，可以使用其他中间公式。  线性回归中有公式，在逻辑回归中，*x*可以为任意数（或多维数组），但是输出的概率值域为[0,1]。将线性回归的公式更改为，将作为新的特征可得到概率，往往使用sigmoid()函数，称为激活函数，sigmoid()函数的形式为，sigmoid()函数图像如图所示。  4-10  sigmoid()函数图像  函数的值域为(0，1)，左右都是开区间。它的特点是：当*h*无限大时，分母无限趋近于1，整体也是无限趋近于1的；当*h*无限小时，分母无限趋近于无穷大，整体是无限趋近于0的。将sigmoid()函数在坐标轴上绘制出来后，拥有坐标轴的sigmoid()函数图像如图所示。  4-11  拥有坐标轴的sigmoid()函数图像  根据图4-11可知，当横坐标*h*大于0时，纵坐标大于0.5，当横坐标*h*小于0时，纵坐标小于0.5，逻辑回归公式为，所以当大于0.5时，分类为1；小于0.5时，分类为0。  4.2.2 逻辑回归例子引入  银行基于一个人的信用度决定是否为其发放信用卡，这里假设横坐标*x*是信用度，纵坐标*y*是工作稳定性。设置一些点，三角形代表银行不发给他信用卡，五角星代表银行发给他信用卡，如图所示。  4-12  是否发放信用卡  4.2.3 数学方法解决逻辑回归问题  在线性回归中，结果是线性的，期望*y*和的差距尽可能小，可以使用距离求解。但是在逻辑回归中，得到的结果是概率，没办法直接求距离，这时候需要做一些处理，使用损失函数，其中*x*是逻辑回归的结果。对于该公式，损失函数的图像如图所示。  4-13  损失函数的图像  上图中，的*x*代表逻辑回归分类后的结果，即，值域为(0, 1)，在*y*=0曲线上，当估计出来的趋近于1时，整个函数趋近于正无穷，算法会将分类结果分到。但是此时是在*y*=0曲线上，即*y*的分类应该是0，所以分类分错了，损失函数值会非常大，随着减小，损失函数值越来越小，直到趋近于0甚至等于0，分类正确。这样就将损失函数值的大小与分类的正确与否关联起来了。对于另外一条曲线，思路与之类似。  损失函数是两个函数，在编程的时候会造成困难，所以将其合并为一个函数： 。当*y*=1时，；当*y*=0时，。这里有一个样本，当有*n*个样本时，损失函数为，其中，，按照解决线性回归问题的思路，应该算出它的数学公式解，但是这个式子是没有数学公式解的，所以这里使用梯度下降的方法解决该逻辑回归问题。  4.2.4 利用TensorFlow解决逻辑回归问题  使用梯度下降法实现逻辑回归，绘制一条线将是否发给用户信用卡两类数据分开，在machine\_learning项目下新建Logistic\_regression.py。  1．导入必要的库  import matplotlib.pyplot as plt  import tensorflow as tf  import numpy as np  2．训练逻辑回归  # 创造数据  raw\_data\_X = [[1.85, 1.05],  [1.57, 2.63],  [2.28, 1.42],  [2.28, 3.64],  [1.94, 3.68],  [2.09, 2.66],  [1.49, 3.66],  [0.12, 1.12],  [0.25, 1.04],  [0.23, 0.54],  [0.83, 1.49],  [0.95, 0.09],  [0.46, 1.63],  [0.26, 1.03],  ]  raw\_data\_Y = [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1];  data = np.array(raw\_data\_X)  label = np.array(raw\_data\_Y)  data = np.hstack(data).reshape(-1,2)  label = np.hstack(label).reshape(-1,1)  label1 = label.reshape(1,-1)[0]  plt.scatter(data[label1 == 0, 0], data[label1 == 0, 1], marker="\*")  plt.scatter(data[label1 == 1, 0], data[label1 == 1, 1], marker="^")  plt.show()  x = tf.placeholder(tf.float32,shape=(None,2))  y\_ = tf.placeholder(tf.float32,shape=(None,1))  # tf.random\_normal()函数用于从服从指定正态分布的数值中取出指定个数的值  # tf.random\_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None,  name=None)  weight = tf.Variable(tf.random\_normal([2,1]), dtype=tf.float32)  bias = tf.Variable(tf.constant(0.1, shape=[1]))  # tf.nn.sigmoid()是激活函数  y\_hat = tf.nn.sigmoid(tf.matmul(x, weight) + bias)  # 不适用该损失函数  # cost = tf.reduce\_sum(tf.square(y\_ - y\_hat))  # 损失函数  cost = - tf.reduce\_mean(y\_ \* tf.log(tf.clip\_by\_value(y\_hat, 1e-10, 1.0)) + \  (1 - y\_) \* tf.log(tf.clip\_by\_value((1 - y\_hat), 1e-10, 1.0)))  # 梯度下降  optimizer = tf.train.AdamOptimizer(0.001)  train = optimizer.minimize(cost)  # 开始训练  with tf.Session() as sess:  sess.run(tf.global\_variables\_initializer())  plt.ion()  for i in range(8000):  sess.run(train,feed\_dict={x:data,y\_:label})  #画出训练后的分割函数  #mgrid()函数产生两个300×400的数组：0～3每隔0.1取一个数，共300×400个  xx, yy = np.mgrid[0:3:.1,0:4:.1]  if (i % 20) == 0:  # np.c\_用于合并两个数组  # ravel()函数将多维数组降为一维，仍返回array数组，元素以列排列  grid = np.c\_[xx.ravel(), yy.ravel()]  probs = sess.run(y\_hat, feed\_dict={x:grid})  # print(probs)  probs = probs.reshape(xx.shape)  plt.cla() # 清除原有图像  plt.scatter(data[label1 == 0, 0], data[label1 == 0, 1], marker="\*")  plt.scatter(data[label1 == 1, 0], data[label1 == 1, 1], marker="^")  plt.contour(xx, yy, probs, levels=[.5])  plt.pause(0.00001)  print("After %d steps, cost:%f" % (i, sess.run(cost, feed\_dict= {x:data,y\_:label})))  plt.close()  经过8000步的训练后，逻辑回归结果如图所示。  4-14  逻辑回归结果 |
| **总结评价** | 本节课主要讲解了机器学习的2个算法，即线性回归、逻辑回归，分析了监督学习和非监督学习的特点以及区别。 |