**云南大学数学与统计学院**

**《数据挖掘与决策支持实验》上机实践报告**

|  |  |  |
| --- | --- | --- |
| 课程名称：数据挖掘与决策支持实验 | 年级：2015级 | 上机实践成绩： |
| 指导教师：彭程 | 姓名：刘鹏 | 专业：信息与计算科学 |
| 上机实践名称：对心脏病数据进行决策树分析 | 学号：20151910042 | 上机实践日期：2018-07-04 |
| 上机实践编号：06 | 组号： |  |

一、实验目的

学习使用R语言进行变量选择。

二、实验内容

对心脏病数据进行决策树分析

三、实验平台

Windows 10 Pro 1803；

Microsoft© Visual Studio 2017 Enterprise。

Version 1.1.442 – © 2009-2018 RStudio, Inc.

# 四、算法设计

摘 要：鸢尾花数据集是常用的分类实验数据集，包含150个数据集，分为3类，每类50个数据，每个数据包含4个属性。我们通过pca和relief算法选取其中两个属性，训练svm分类模型，对鸢尾花是否属于山鸢尾进行识别。

关键词：维归约，PCA，relief特征排列，SVM分类

# 文献综述

文献[1]说明了数值标准化对挖掘结果的影响，以及维归约的意义；文献[2]作为一本中译本的机器学习专著，详细得介绍了几种维归约算法与分类算法的数学原理及实现方法。

# 数据预处理

数据预处理（data preprocessing）是指在主要的处理以前对数据进行的一些处理。在真实数据中，我们拿到的数据可能包含了大量的缺失值，可能包含大量的噪音，也可能因为人工录入错误导致有异常点存在，非常不利于算法模型的训练。数据预处理的结果是对各种脏数据进行对应方式的处理，得到标准的、干净的、连续的数据，提供给数据统计、数据挖掘等使用。

数据预处理方法包括去除唯一属性，处理缺失值，特征编码，数据标准化，特征选择等等。因为我们选择的数据集已经进行了先期处理，故只进行标准化和特征选择的过程。

## 数据标准化

数据标准化是将样本的属性缩放到某个指定的范围。在多指标评价体系中，由于各评价指标的性质不同，通常具有不同的量纲和数量级。当各指标间的水平相差很大时，如果直接用原始指标值进行分析，就会突出数值较高的指标在综合分析中的作用，相对削弱数值水平较低指标的作用。因此，为了保证结果的可靠性，需要对原始指标数据进行标准化处理。

我们使用z-scores标准化方法，基于原始数据的均值（mean）和标准差（standarddeviation）进行数据的标准化。将的原始值使用z-score标准化到。

均值和标准差都是在样本集上定义的，而不是在单个样本上定义的。标准化是针对某个属性的，需要用到所有样本在该属性上的值。

实现如下：

def featureNormalize(X):

'''归一化数据标准差'''

n = X.shape[1]

mu = zeros((1,n));

sigma = zeros((1,n))

mu = mean(X,axis=0)

sigma = std(X,axis=0)

for i in range(n):

X[:,i] = (X[:,i]-mu[i])/sigma[i]

return (X)

## 特征选择

特征选择，即是指从全部特征中选取一个特征子集，剔除不相关、冗余、没有差异刻画能力的特征，从而达到减少特征个数、减少训练或者运行时间、提高模型精确度的作用。

如何做特征选择呢，如果要从全部特征中选择一个最优的子集，使得其在一定的评价标准下，在当前训练和测试数据上表现最好。

从这个层面上理解，特征选择可以看作三个问题：

1.从原始特征集中选出固定数目的特征，使得分类器的错误率最小这是一个无约束的组合优化问题；

2.对于给定的允许错误率，求维数最小的特征子集，这是一种有约束的最优化问题；

3.在错误率和特征子集的维数之间进行折中。

上述3个问题都是一个难问题，当特征维度较小时，实现起来可行，但是当维度较大时，实现起来的复杂度很大，所以实际应用中很难实用。上述三种特征选择都属于难的问题。由于求最优解的计算量太大，需要在一定的时间限制下寻找能得到较好次优解的算法。

以下我们使用基于维归约的主成分分析和基于加权特征选择的Relief算法求取次优解。

### PCA算法

主成分分析（Principal Component Analysis），是一种用于探索高维数据的技术。PCA通常用于高维数据集的探索与可视化。还可以用于数据压缩，数据预处理等。PCA可以把可能具有线性相关性的高维变量合成为线性无关的低维变量，称为主成分（principal components），新的低维数据集会尽可能的保留原始数据的变量，可以将高维数据集映射到低维空间的同时，尽可能的保留更多变量。

数学原理：

基

二维空间默认和为一组基。 其实任何两个线性无关的二维向量都可以成为一组基。因为正交基有较好的性质，所以一般使用的基都是正交的。

内积

向量和的内积公式为：

我们希望基的模是1，因为从内积的意义可以看到，如果基的模是，那么就可以方便的用向量点乘基而直接获得其在新基上的坐标了，即 向量在基上的投影=向量与基的内积=坐标。

基变换的矩阵表示

将一组向量的基变换表示为矩阵的相乘。一般的，如果我们有个维向量，想将其变换为由个维向量（个基）表示的新空间中，那么首先将个基按行组成矩阵，然后将待变换向量按列组成矩阵，那么两矩阵的乘积就是变换结果。可以小于，而决定了变换后数据的维数。也就是说，我们可以将一维数据变换到更低维度的空间中去，变换后的维度取决于基的数量。

因此这种矩阵相乘可以表示降维变换：

两个矩阵相乘的意义：将右边矩阵中的每一列列向量变换到左边矩阵中每一行行向量为基所表示的空间中。

优化目标

数学上用协方差表示两个维度的相关性，当协方差为0时，表示两个维度完全独立。为了让协方差为0，我们选择第二个基时只能在与第一个基正交的方向上选择，因此最终选择的两个方向一定是正交的。

降维问题的优化目标：将一组N维向量降为R维，其目标是选择R个单位正交基，使得原始数据变换到这组基上后，各维度两两间的协方差为0，而每个维度的方差则尽可能大。

设原始矩阵为，表示个维向量，其协方差矩阵为；为变换矩阵；为目标矩阵, 其协方差矩阵为。我们要求降维后的矩阵的每一维包含的数据足够分散，也就是每一行（维）方差足够大，而且要求行之间的元素线性无关，也就是要求行之间的协方差全部为0，这就要求协方差矩阵的对角线元素足够大，除对角线外元素都为0。相当于对进行协方差矩阵对角化。

算法实现

设有个维数据:

1.将原始数据按列组成行列矩阵X将X的每一行进行零均值化，即减去每一行的均值

2.求出的协方差矩阵

3.求出协方差矩阵的特征值及对应的特征向量，的特征值就是的每维元素的方差，也是的对角线元素，从大到小沿对角线排列构成。

4.将特征向量按对应特征值大小从上到下按行排列成矩阵，根据实际业务场景，取前行组成矩阵

5.即为降到维后的目标矩阵

我们使用Python下著名的机器学习库Scikit-learn中的PCA函数实现。

代码与运行结果如下：

#PCA

pca = PCA(n\_components='mle')

pca.fit(x\_vals)

print(pca.explained\_variance\_ratio\_)

plt.figure()

plt.plot(pca.explained\_variance\_ratio\_, 'k', linewidth=2)

plt.xlabel('n\_components', fontsize=16)

plt.ylabel('explained\_variance\_ratio\_', fontsize=16)

plt.show()

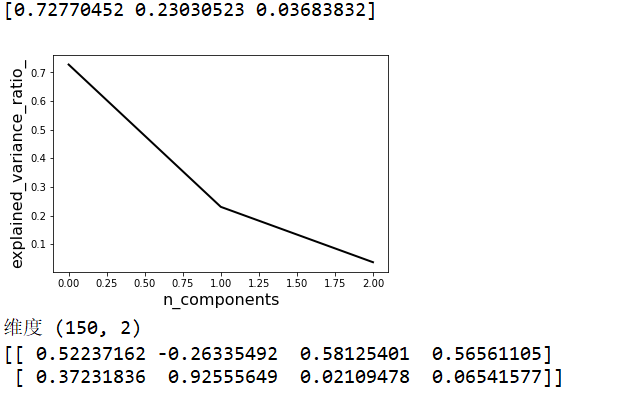
pca=PCA(n\_components=2)

pca.fit(x\_vals)

newdata=pca.transform(x\_vals)

print('维度',newdata.shape)

print(pca.components\_) # 投影方向向量



可见将四维压缩成二维时，误差率小于5%，选择将原数据降为二维，下面用relief算法进行验证。

### Relief算法交叉验证

Relief算法最早由Kira提出，最初局限于两类数据的分类问题。Relief算法是一种特征权重算法(Feature weighting algorithms)，根据各个特征和类别的相关性赋予特征不同的权重，权重小于某个阈值的特征将被移除。Relief算法中特征和类别的相关性是基于特征对近距离样本的区分能力。算法从训练集中随机选择一个样本，然后从和同类的样本中寻找最近邻样本，称为，从和不同类的样本中寻找最近邻样本，称为，然后根据以下规则更新每个特征的权重：如果和在某个特征上的距离小于和上的距离，则说明该特征对区分同类和不同类的最近邻是有益的，则增加该特征的权重；反之，如果和在某个特征的距离大于和上的距离，说明该特征对区分同类和不同类的最近邻起负面作用，则降低该特征的权重。以上过程重复次，最后得到各特征的平均权重。特征的权重越大，表示该特征的分类能力越强，反之，表示该特征分类能力越弱。

假设我们有一个已知标签（两类问题）的数据集，个sample，每个sample有维的特征。首先我们需要将特征进行标准化（归一化）处理，保证特征的取值区间为（二进制的特征则保证不是0则是1）。算法会迭代次。这里我们初始化一个维度为的向量来存放每个特征的权重值。

在每一次迭代都会随机选择一个sample的特征向量，并在各类中寻找与向量最近邻的样本（这里采用的是欧氏距离）。在相同类中最近邻的sample称为，在不同类中最近邻的sample称为。按照下面的公式对权重向量W进行更新：

当随机选择sample的某个特征与的相应特征的距离比与的相应特征的距离要小的时候，这个特征的权重就会被降低。换句话说就是，随机选择sample在某个特征上与不同类更近，与同类反而远的时候，这个特征的权重会被降低。经过次迭代之后，将权重向量的每一个权重分别除以就是最终的相关性向量了。

代码及运行结果如下：

#relief 交叉验证

def distanceNorm(Norm,D\_value):

if Norm == '1':

counter = absolute(D\_value);

counter = sum(counter);

elif Norm == '2':

counter = power(D\_value,2);

counter = sum(counter);

counter = sqrt(counter);

elif Norm == 'Infinity':

counter = absolute(D\_value);

counter = max(counter);

else:

raise Exception('We will program this later......');

return counter;

def fit(features,labels,iter\_ratio):

(n\_samples,n\_features) = shape(features);

distance = zeros((n\_samples,n\_samples));

weight = zeros(n\_features);

if iter\_ratio >= 0.5:

# compute distance

for index\_i in range(n\_samples):

for index\_j in range(index\_i+1,n\_samples):

D\_value = features[index\_i] - features[index\_j];

distance[index\_i,index\_j] = distanceNorm('2',D\_value);

distance += distance.T;

else:

pass;

# start iteration

for iter\_num in range(int(iter\_ratio\*n\_samples)):

nearHit = list();

nearMiss = list();

distance\_sort = list();

# random extract a sample

index\_i = randrange(0,n\_samples,1);

self\_features = features[index\_i];

# search for nearHit and nearMiss

if iter\_ratio >= 0.5:

distance[index\_i,index\_i] = max(distance[index\_i]); # filter self-distance

for index in range(n\_samples):

distance\_sort.append([distance[index\_i,index],index,labels[index]]);

else:

# compute distance respectively

distance = zeros(n\_samples);

for index\_j in range(n\_samples):

D\_value = features[index\_i] - features[index\_j];

distance[index\_j] = distanceNorm('2',D\_value);

distance[index\_i] = max(distance); # filter self-distance

for index in range(n\_samples):

distance\_sort.append([distance[index],index,labels[index]]);

distance\_sort.sort(key = lambda x:x[0]);

for index in range(n\_samples):

if nearHit == [] and distance\_sort[index][2] == labels[index\_i]:

# nearHit = distance\_sort[index][1];

nearHit = features[distance\_sort[index][1]];

elif nearMiss == [] and distance\_sort[index][2] != labels[index\_i]:

# nearMiss = distance\_sort[index][1]

nearMiss = features[distance\_sort[index][1]];

elif nearHit != [] and nearMiss != []:

break;

else:

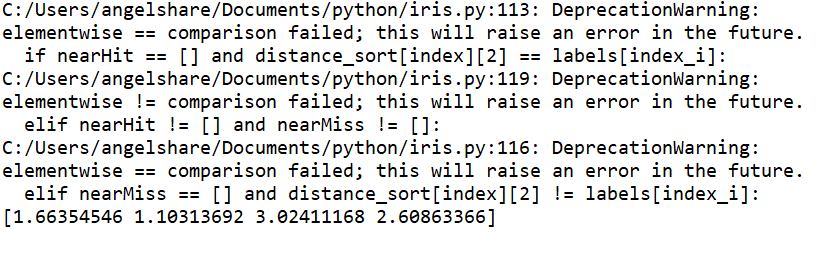
continue;

# update weight

weight = weight - power(self\_features - nearHit,2) + power(self\_features - nearMiss,2);

print (weight/(iter\_ratio\*n\_samples));

return (weight/(iter\_ratio\*n\_samples));



可见第三、四两个属性相关度最高，不妨选择这两个属性进行分类器训练。

# SVM分类

支持向量机是在所有知名的数据挖掘算法中最健壮，最准确的方法之一，它属于二分类算法，可以支持线性和非线性的分类。我们只使用线性分类器。

## 工作原理

它分类的思想是，给定给一个包含正例和反例的样本集合，svm的目的是寻找一个超平面来对样本根据正例和反例进行分割。

在样本空间中，划分超平面可通过如下线性方程来描述：

假设它已经完成了对样本的分隔，且两种样本的标签分别是，那么对于一个分类器来说，和就可以分别代表两个不同的类别。

但光是分开是不够的，SVM的核心思想是尽最大努力使分开的两个类别有最大间隔，这样才使得分隔具有更高的可信度。而且对于未知的新样本才有很好的分类预测能力。

那么怎么描述这个间隔，并且让它最大呢？SVM的办法是：让离分隔面最近的数据点具有最大的距离。

为了描述离分隔超平面最近的数据点，需要找到两个和这个超平面平行和距离相等的超平面：

在这两个超平面上的样本点也就是理论上离分隔超平面最近的点，是它们的存在决定了H1和H2的位置，支撑起了分界线，它们就是所谓的支持向量，这就是支持向量机的由来

有了这两个超平面就可以顺理成章的定义上面提到的间隔（margin）了

二维情况下 和两条平行线的距离公式为：

可以推出H1和H2两个超平面的间隔为，即现在的目的是要最大化这个间隔。

等价于最小化, 为了之后的求导和计算方便，进一步等价于最小化:

假设超平面能将样本正确分类，则可令：

综合两个式子得：

这就是目标函数的约束条件。现在这个问题就变成了一个最优化问题，可以用现成的优化工具包求解。

## 代码和结果

我们使用tensorflow来建立模型

sess = tf.Session()

# 分离训练和测试集

train\_indices = random.choice(len(x\_vals),

round(len(x\_vals)\*0.8),

replace=False)

test\_indices = array(list(set(range(len(x\_vals))) - set(train\_indices)))

x\_vals\_train = x\_vals[train\_indices]

x\_vals\_test = x\_vals[test\_indices]

y\_vals\_train = y\_vals[train\_indices]

y\_vals\_test = y\_vals[test\_indices]

batch\_size = 100

# 初始化feedin

x\_data = tf.placeholder(shape=[None, 2], dtype=tf.float32)

y\_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)

# 创建变量

A = tf.Variable(tf.random\_normal(shape=[2, 1]))

b = tf.Variable(tf.random\_normal(shape=[1, 1]))

# 定义线性模型

model\_output = tf.subtract(tf.matmul(x\_data, A), b)

# Declare vector L2 'norm' function squared

l2\_norm = tf.reduce\_sum(tf.square(A))

# Loss = max(0, 1-pred\*actual) + alpha \* L2\_norm(A)^2

alpha = tf.constant([0.01])

classification\_term = tf.reduce\_mean(tf.maximum(0., tf.subtract(1., tf.multiply(model\_output, y\_target))))

loss = tf.add(classification\_term, tf.multiply(alpha, l2\_norm))

my\_opt = tf.train.GradientDescentOptimizer(0.01)

train\_step = my\_opt.minimize(loss)

init = tf.global\_variables\_initializer()

sess.run(init)

# Training loop

loss\_vec = []

train\_accuracy = []

test\_accuracy = []

for i in range(20000):

rand\_index = np.random.choice(len(x\_vals\_train), size=batch\_size)

rand\_x = x\_vals\_train[rand\_index]

rand\_y = np.transpose([y\_vals\_train[rand\_index]])

sess.run(train\_step, feed\_dict={x\_data: rand\_x, y\_target: rand\_y})

[[a1], [a2]] = sess.run(A)

[[b]] = sess.run(b)

slope = -a2/a1

y\_intercept = b/a1

best\_fit = []

x1\_vals = [d[1] for d in x\_vals]

for i in x1\_vals:

best\_fit.append(slope\*i+y\_intercept)

# Separate I. setosa

setosa\_x = [d[1] for i, d in enumerate(x\_vals) if y\_vals[i] == 1]

setosa\_y = [d[0] for i, d in enumerate(x\_vals) if y\_vals[i] == 1]

not\_setosa\_x = [d[1] for i, d in enumerate(x\_vals) if y\_vals[i] == -1]

not\_setosa\_y = [d[0] for i, d in enumerate(x\_vals) if y\_vals[i] == -1]

plt.plot(setosa\_x, setosa\_y, 'o', label='I. setosa')

plt.plot(not\_setosa\_x, not\_setosa\_y, 'x', label='Non-setosa')

plt.plot(x1\_vals, best\_fit, 'r-', label='Linear Separator', linewidth=3)

plt.ylim([0, 10])

plt.legend(loc='lower right')

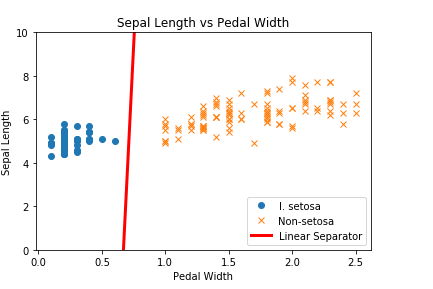
plt.title('Sepal Length vs Pedal Width')

plt.xlabel('Pedal Width')

plt.ylabel('Sepal Length')

plt.show()

得到划分为



# 参考文献

1. Mehmed Kantardzic, *数据挖掘：概念、模型、方法和算法*, 2013, 北京：清华大学出版社.

2. Ian Goodfellow, Yoshua Bengio,Aaron COurville,  *DEEP LEARNING*. 2017, 北京：人民邮电出版社.