目录

一、数据集

1.1 数据集简介

安德森鸢尾花卉数据集（英文：Anderson's Iris data set），也称鸢尾花卉数据集（英文：Iris flower data set）或费雪鸢尾花卉数据集（英文：Fisher's Iris data set），是一类多重变量分析的数据集。它最初是埃德加·安德森从加拿大加斯帕半岛上的鸢尾属花朵中提取的形态学变异数据，后由罗纳德·费雪作为判别分析的一个例子，运用到统计学中。

其数据集包含了 150 个样本,通过花萼长度，花萼宽度，花瓣长度，花瓣宽度 4 个属性预测鸢尾花卉属于（Setosa，Versicolour，Virginica）三个种类中的哪一类。

数据集来源：http://archive.ics.uci.edu/ml/datasets/Iris

1.2 数据集分析及预处理

导入数据后，分析数据内容。首先查看前五行数据，共有 6 个属性（Unnamed:0, Speal.Length, Speal.Width, Petal.Length, Petal.Width, Species）。根据数据集已知的介绍，第一个属性 Unnamed:0 仅是数据的行号，可以删除。

```
In [4]:  1  df.head()
executed in 31ms, finished 19:57:54 2019-04-03
```

Out[4]:

	Unnamed: 0	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	1	5.1	3.5	1.4	0.2	setosa
1	2	4.9	3.0	1.4	0.2	setosa
2	3	4.7	3.2	1.3	0.2	setosa
3	4	4.6	3.1	1.5	0.2	setosa
4	5	5.0	3.6	1.4	0.2	setosa

图 1-1 原始数据前五行

删除属性 Unnamed:0 后再次查看数据前五行。

```
In [5]:  1  iris = df.drop(labels='Unnamed: 0', axis=1)
executed in 9ms, finished 20:15:39 2019-04-03

In [6]:  1  iris.head()
executed in 12ms, finished 20:15:40 2019-04-03
```

Out[6]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

图 1-2 删除属性 Unnamed:0 后数据前五行

通过 describe()方法显示数值属性摘要。

```
In [8]:  1  iris.describe()
executed in 37ms, finished 20:20:47 2019-04-03
```

Out[8]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

图 1-3 数值属性摘要

查看花卉各种类数据量，得知每类花卉 50 组数据，分布合理。

```
In [9]:  1  iris['Species'].value_counts()
executed in 9ms, finished 20:39:57 2019-04-03
```

Out[9]:

```
setosa        50
versicolor    50
virginica     50
Name: Species, dtype: int64
```

图 1-4 花卉各种类数据量统计

通过 Violinplot，绘制小提琴图，查看数据分布和概率密度，见图 1-5。通过 Pairplot，绘制散点图矩阵，查看数据两两特征之间的关系，见图 1-6。根据图 1-5、1-6 可知，Setosa 与其他两类区别较明显，而 Versicolour 与 Virginica 部分数据存在重合，后续处理需要注意。

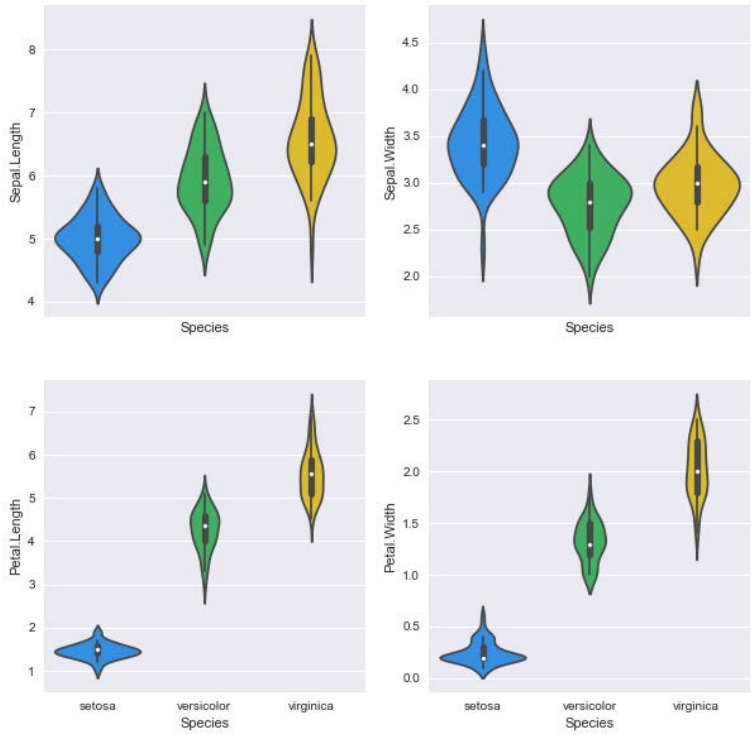

图 1-5 数据分布及概率密度图

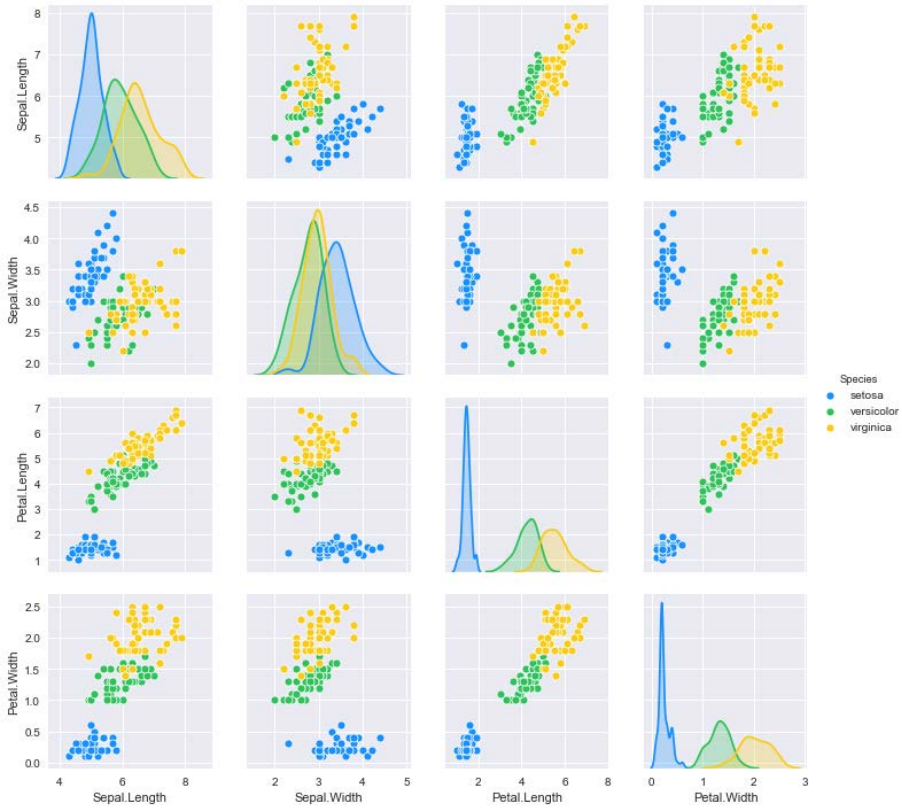

图 1-6 数据散点图矩阵

因为数据的分类属性 Species 为文本属性，不方便处理，需要先将这些文本标签转化为数字。Setosa、versicolor、virginica 分别对应 0、1、2。

```
In [53]:
# 载入特征和标签集
X = iris[['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width']]
y = iris['Species']
# 对标签集进行编码
encoder = LabelEncoder()
y = encoder.fit_transform(y)
print(y)
```

executed in 8ms, finished 21:25:05 2019-04-03

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

图 1-7 数据散点图矩阵

1.3 训练集和测试集划分

为了尽可能保持数据分布的一致性，避免因数据划分过程中引入额外的偏差而对最终结果产生影响，故在此次分类任务中采用分层采样，保持类别比例。即训练集和测试集中 Setosa：Versicolour：Virginica=1:1:1，其中训练集：验证集=7:3。

In [12]:
```
split = StratifiedShuffleSplit(test_size=0.3, random_state=42)
for train_index, test_index in split.split(iris, iris['Species']):
    strat_train_set = iris.loc[train_index]
    strat_test_set = iris.loc[test_index]
train_data = np.array(strat_train_set)
test_data = np.array(strat_test_set)
train_X = train_data[:,0:-1]
train_y = train_data[:,-1]
test_X = test_data[:,0:-1]
test_y = test_data[:,-1]
```

executed in 26ms, finished 22:28:12 2019-04-03

In [13]:
```
strat_test_set['Species'].value_counts()
```

executed in 10ms, finished 22:28:14 2019-04-03

Out[13]:
```
setosa        15
virginica     15
versicolor    15
Name: Species, dtype: int64
```

In [14]:
```
strat_train_set['Species'].value_counts()
```

executed in 8ms, finished 22:28:16 2019-04-03

Out[14]:
```
setosa        35
virginica     35
versicolor    35
Name: Species, dtype: int64
```

图 1-8 数据集划分

二、k 近邻法

2.1 算法简介

k 近邻法（k-nearest neighbor, k-NN）是一种基本分类与回归方法，k 近邻法的输入为实例的特征向量，对应于特征空间的点；输出为实例的类别，可以取多类。k 近邻法假设给定一个训练数据集，其中的实例类别已定。分类时，对新的实例，根据其 k 个最近邻的训练实例的类别，通过多数表决等方式进行预测。因此，k 近邻法不具有显式的学习过程。k 近邻法实际上利用训练数据集对特征向量空间进行划分，并作为其分类的“模型”。

k 值的选择、距离度量及分类决策规则是 k 近邻法的三个基本要素。

2.2 实验

2.2.1 特征归一化

为了消除数据特征之间的量纲影响，我们需要对特征进行归一化处理，使得不同指标之间具有可比性。对数值类型的特征做归一化可以将所有的特征都统一到一个大致相同的数值区间内。

本文采用线性函数归一化（Min-Max Scaling），对原始数据进行线性变换，使结果映射到[0, 1]的范围，实现对原始数据的等比缩放。

归一化公式如下：

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

其中 X 为原始数据，X_{max}、X_{min}分别为数据最大值和最小值。

图 2-1 是原始训练数据前五行，图 2-2 是特征归一化后的训练数据前五行，可以看出数据已经映射到[0, 1]的范围。

```
In [28]:  1  train_X[:5,:]
          executed in 8ms, finished 22:31:28 2019-04-04
Out[28]: array([[6.9, 3.2, 5.7, 2.3],
                [6.3, 2.8, 5.1, 1.5],
                [6.1, 2.8, 4.0, 1.3],
                [5.5, 2.4, 3.7, 1.0],
                [7.1, 3.0, 5.9, 2.1]], dtype=object)
```

图 2-1 原始训练数据前 5 行

```
In [41]:
1 scaler = MinMaxScaler().fit(train_X)
2 train_X_scaled = scaler.fit_transform(train_X)
3 test_X_scaled = scaler.fit_transform(test_X)
executed in 7ms, finished 20:54:59 2019-04-05

C:\Software\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595:
DataConversionWarning: Data with input dtype object was converted to flo
at64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Software\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595:
DataConversionWarning: Data with input dtype object was converted to flo
at64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Software\Anaconda3\lib\site-packages\sklearn\utils\validation.py:595:
DataConversionWarning: Data with input dtype object was converted to flo
at64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)

In [42]:
1 train_X_scaled[:5,:]
executed in 8ms, finished 20:55:02 2019-04-05

Out[42]: array([[0.72222222, 0.54545455, 0.79661017, 0.91666667],
       [0.55555556, 0.36363636, 0.69491525, 0.58333333],
       [0.5       , 0.36363636, 0.50847458, 0.5       ],
       [0.33333333, 0.18181818, 0.45762712, 0.375     ],
       [0.77777778, 0.45454545, 0.83050847, 0.83333333]])
```

图 2-2 特征归一化后训练数据前 5 行

2.2.2 交叉验证

鸢尾花数据集只有 150 条数据，数据较少，故采用交叉验证法。

"交叉验证法"(cross validation)先将数据集 D 划分为 k 个大小相似的互斥子集。每个子集 D_i 都尽可能保持数据分布的一致性，即从 D 中通过分层采样得到。然后，每次用 k-1 个子集的并集作为训练集，余下的那个子集作为测试集；这样就可获得 k 组训练/测试集，从而可进行 k 次训练和测试，最终返回的是这 k 个测试结果的均值。显然，交叉验证法评估结果的稳定性和保真性在很大程度上取决于 k 的取值，为强调这一点，通常把交叉验证法称为"k 折交叉验证" (k-fold crossvalidat ion)。k 最常用的取值是 10，此时称为 10 折交叉验证。

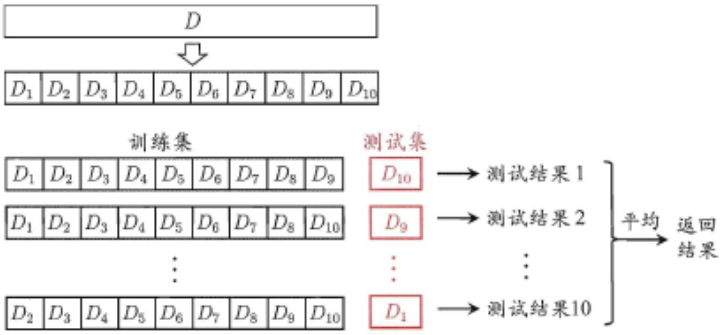

图 2-3 10 折交叉验证示意图

2.2.3 KNN 实验

先讨论 k 取值不同对分类准确率的影响。

取 k 值范围为[1, 30]，画出不同 k 值时的准确率。

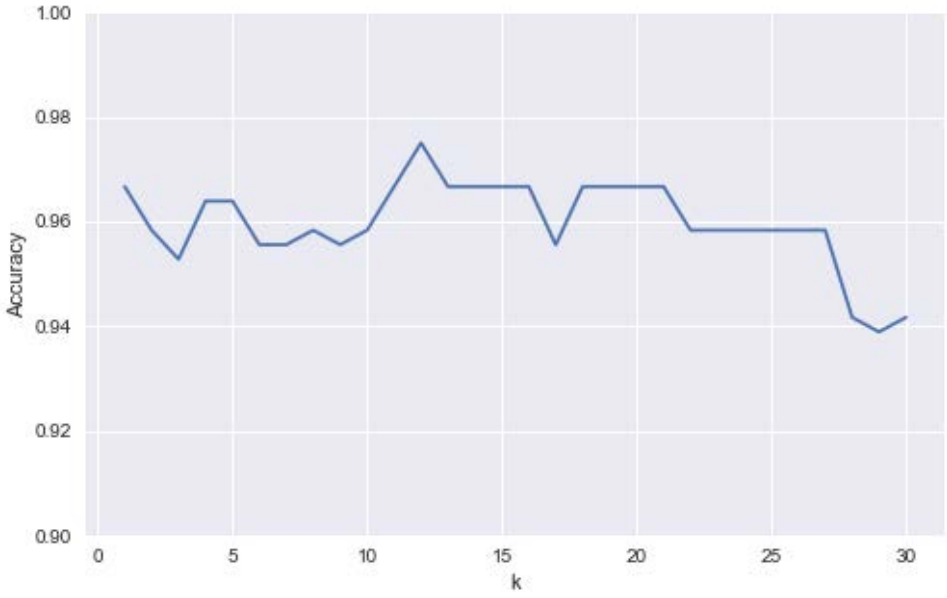

图 2-4 p=2 时不同 k 取值时准确率

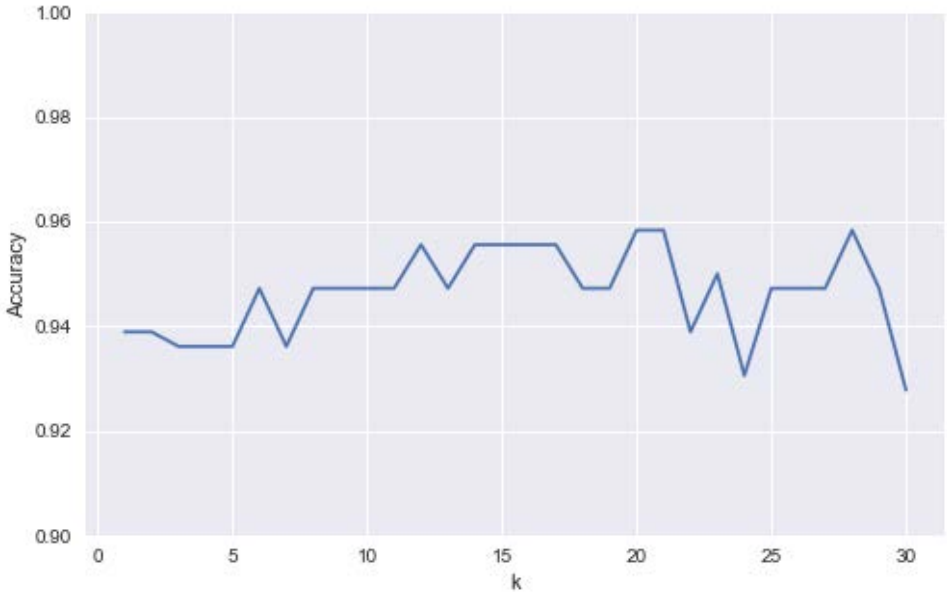

图 2-5 p=1 时不同 k 取值时准确率

图 2-4 和图 2-5 分别是不同距离度量生成的图。p=2 是 L2 距离，p=1 是 L1 距离。根据这两幅图可以看出，在本数据集中 L2 距离普遍有更高的准确率。故使用 L2 距离。

图 2-4 中 k=12 时，交叉验证的准确率最高，故取 k=12。

下面查看 k=12，p=2 时 KNN 的相关性能度量。

The accuracy of K-Nearest Neighbor Classifier is 0.9777777777777777

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	0.94	1.00	0.97	15
virginica	1.00	0.93	0.97	15
micro avg	0.98	0.98	0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

图 2-6 p=2，k=12 的性能度量

由图 2-6 可知，我们的 KNN 分类器精确率 98%，召回率 98%，F1-Score98%，准确率 97.8%。

接下来，我们可视化决策边界，查看 k 取值不同时，决策边界的变化。

我们为了方便可视化，只取两维数据（Sepal.Length、Sepal.Width）。由图 2-7 和图 2-8 可以看出，k 取值越大时，决策边界越光滑。

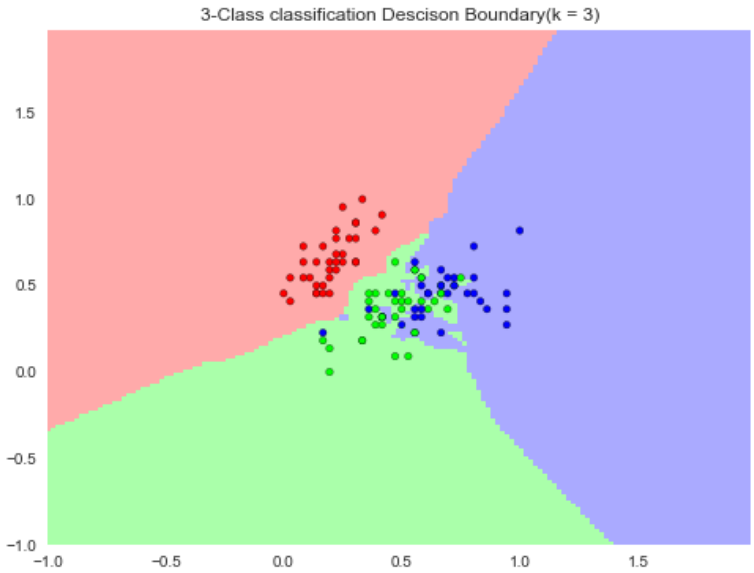

图 2-7 k=3 时决策边界

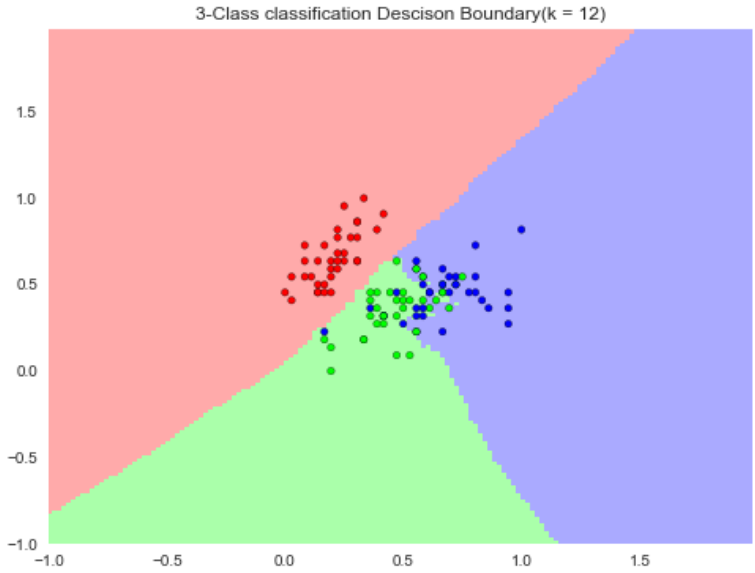

图 2-8 *k*=12 时决策边界

三、决策树

3.1 算法简介

一般的，一棵决策树包含一个根结点、若干个内部结点和若干个叶结点。叶结点对应于决策结果，其他每个结点则对应于一个属性测试；每个结点包含的样本集合根据属性测试的结果被划分到子结点中；根结点包含样本全集。从根结点到每个叶结点的路径对应了一个判定测试序列。决策树学习的目的是为了产生一棵泛化能力强，即处理未见示例能力强的决策树，如图 3-1 所示。

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
　　属性集 $A = \{a_1, a_2, \ldots, a_d\}$.
过程： 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3: 　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A = \varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6: 　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10: 　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11: 　**if** D_v 为空 **then**
12: 　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13: 　**else**
14: 　　以 TreeGenerate(D_v, $A \setminus \{a_*\}$)为分支结点
15: 　**end if**
16: **end for**
输出： 以 node 为根结点的一棵决策树

图 3-1 决策树学习基本算法

显然，决策树的生成是一个递归过程。在决策树基本算法中，有三种情形会导致递归返回：(1)当前结点包含的样本全属于同一类别，无需划分；(2)当前属性集为空，或是所有样本在所有属性上取值相同，无法划分；(3)当前结点包含的样本集合为空，不能划分。

3.2 划分选择

由算法 3-1 可看出，决策树学习的关键是第 8 行，即如何选择最优划分属性。一般而言，随着划分过程不断进行，我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的"纯度"(purity)越来越高。

“信息熵”(information entropy)是度量样本集合纯度最常用的一种指标。$Ent(D)$的值越小，则 D 的纯度越高。

$$Ent(D) = -\sum_{k=1}^{|\gamma|} p_k log_2 p_k$$

属性 a 对样本集 D 进行划分所获得的"信息增益"(information gain)：

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

一般而言，信息增益越大，则意味着使周属性 a 来进行划分所获得的“纯度提升”越大。因此，我们可用信息增益来进行决策树的划分属性选择，即在图 3-1 算法第 8 行选择属性。著名的 ID3 决策树学习算法就是以信息增益为准则来选择划分属性。

信息增益准则对可取值数目较多的属性有所偏好，为减少这种偏好可能带来的不利影响，著名的 C4.5 决策树算法不直接使用信息增益，而是使用“增益率”(gain ratio)来选择最优划分属性。采用与式相同的符号表示，增益率定义为：

$$Gain\_ratio(D,a) = \frac{Gain(D,a)}{IV(a)}$$

其中

$$IV(a) = -\sum_{v=1}^{V} \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$$

CART 决策树使用"基尼指数" (Gini index)来选择划分属性。采数据集 D 的纯度可用基尼值来度量：

$$Gini(D) = \sum_{k=1}^{|y|} \sum_{k' \neq k} p_k p_{k'} = 1 - \sum_{k=1}^{|y|} p_k^2$$

属性 a 的基尼指数定义为：

$$Gini_{index(D,a)} = \sum_{v=1}^{V} \frac{|D^v|}{|D|} Gini(D^v)$$

于是，我们在候选属性集合 A 中，选择那个使得划分后基尼指数最小的属性作为最优划分属性。

3.3 剪枝处理

剪枝(pruning)是决策树学习算法对付“过拟合”的主要手段。在决策树学习，为了尽可能正确分类训练样本，结点划分过程将不断重复，有时会造成决策树分支过多，这时就可能因训练样本学得“太好”了，以致于把训练集自身的一些特点当作所有数据都具有的一般性质而导致过拟合。因此，可通过主动去掉一些分支来降低过拟合的风险。

决策树剪枝的基本策略有“预剪枝”(prepruning)和“后剪枝”(post-pruning)。预剪枝是指在决策树生成过程中，对每个结点在划分前先进行估计，若当前结点的划分不能带来决策树泛化性能提升，则停止划分并将当前结点标记为叶结点；后剪枝则是先从训练集生成一棵完整的决策树，然后自底向上地对非叶结点进行考察，若将该结点对应的子树替换为叶结点能带来决策树泛化性能提升，则将该子树替换为叶结点。

3.4 实验

决策树算法不需要将数据归一化，因为决策树的划分依据是信息增益或者基尼指数。我们根据不同标准画出其准确率曲线。其中 gini、entropy 分别对应基尼指数和信息增益。splitter 可以使用"best"或者"random"。前者在特征的所有划分点中找出最优的划分点。后者是随机的在部分划分点中找局部最优的划分点。默认的"best"适合样本量不大的时候，而如果样本数据量非常大，此时决策树构建推荐"random"。

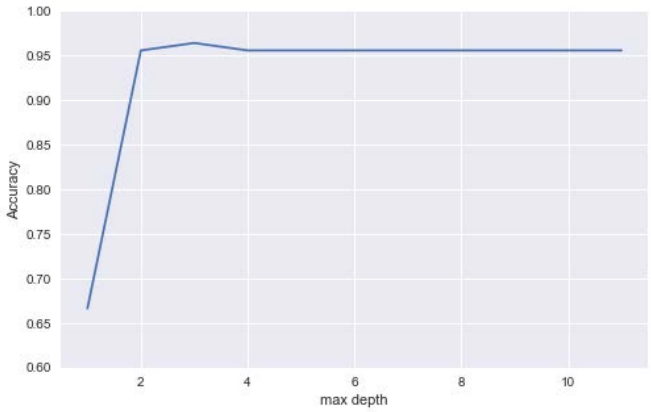

图 3-2 criterion='gini',splitter='best'

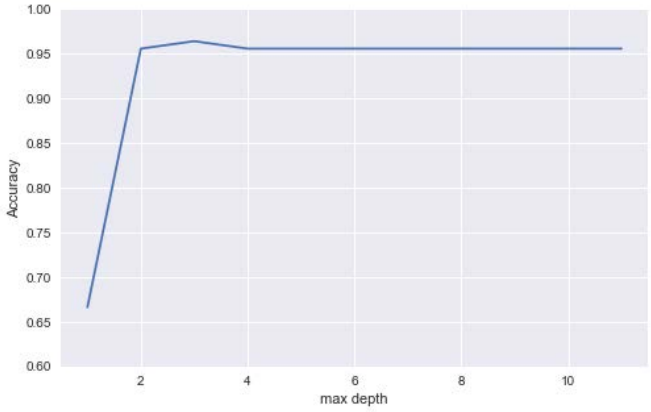

图 3-3 criterion='entropy',splitter='best'

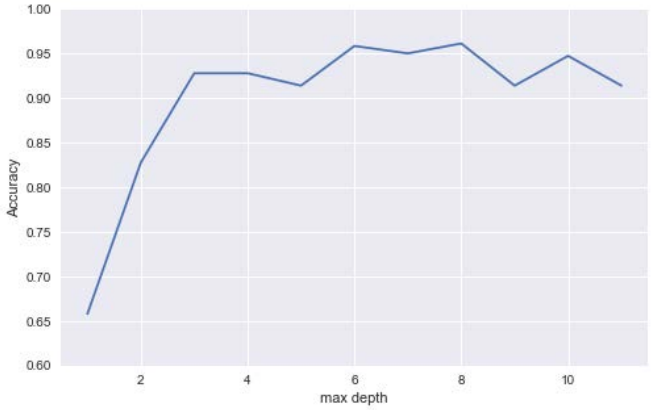

图 3-4 criterion='gini',splitter='random'

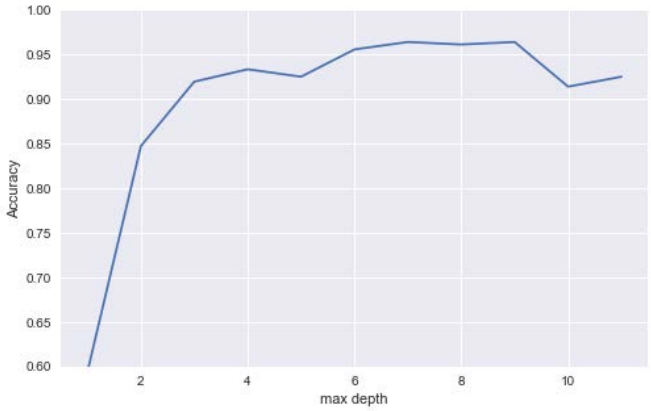

图 3-5 criterion='entropy',splitter='random'

由图 3-2、3-3 可知，在这个数据集下，划分点选择最优的划分点时，基尼指数和信息增益表现一样好。再根据图 3-4、3-5 可知，划分点选择随机局部最优点时表现不如图 3-2 和 3-3。

故决策树分类器参数选择为：criterion='gini'，splitter='best'，max_depth=3。这个分类器的表现如下图。精确率 96%、召回率 96%、F1-Score96%、准确率为 95.6%。决策树结构如图 3-7 所示。

The accuracy of Decision Tree Classifier is 0.9555555555555556

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	0.93	0.93	0.93	15
virginica	0.93	0.93	0.93	15
micro avg	0.96	0.96	0.96	45
macro avg	0.96	0.96	0.96	45
weighted avg	0.96	0.96	0.96	45

图 3-6 决策树性能度量

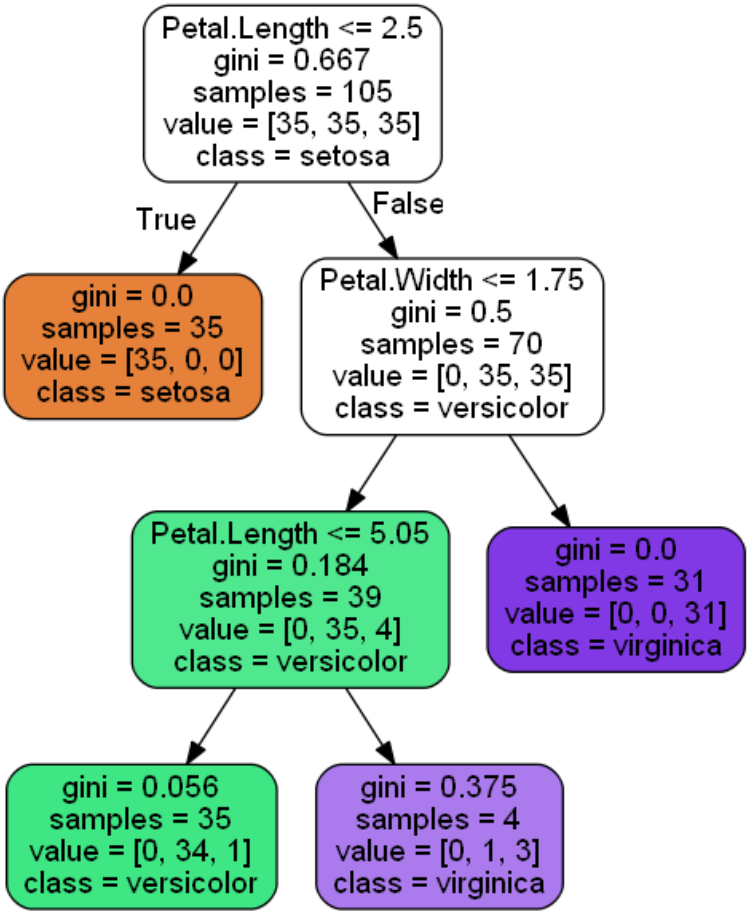

图 3-7 决策树可视化

接下来，我们可视化决策边界，查看 max_depth 取值不同时，决策边界的变化。为了方便可视化，只取两维数据（Sepal.Length、Petal.Length）。由图 3-8 和图 3-9 可以看出，max_depth 取值越小时，决策边界越光滑。max_depth 越大时，模型越复杂，更容易过拟合。

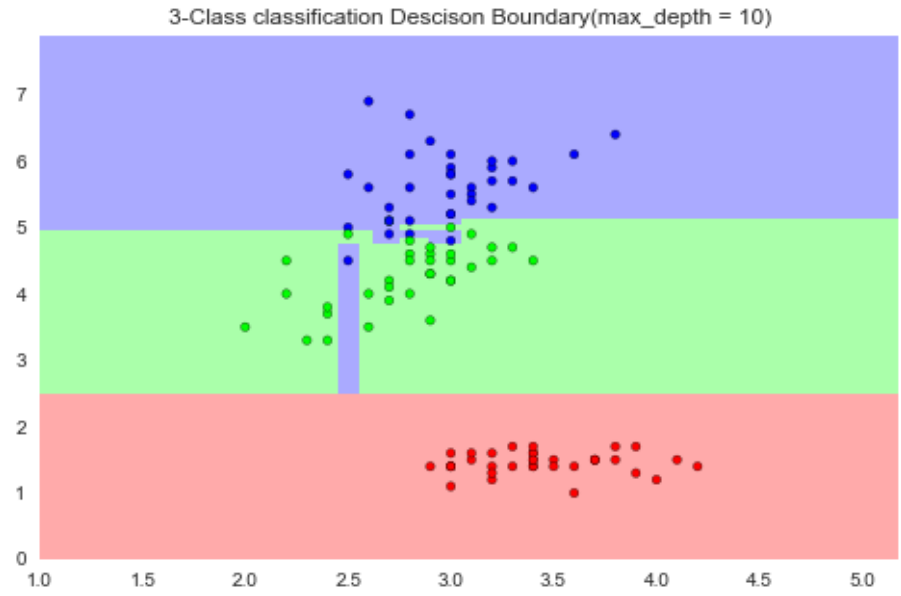

图 3-8 max_depth=10

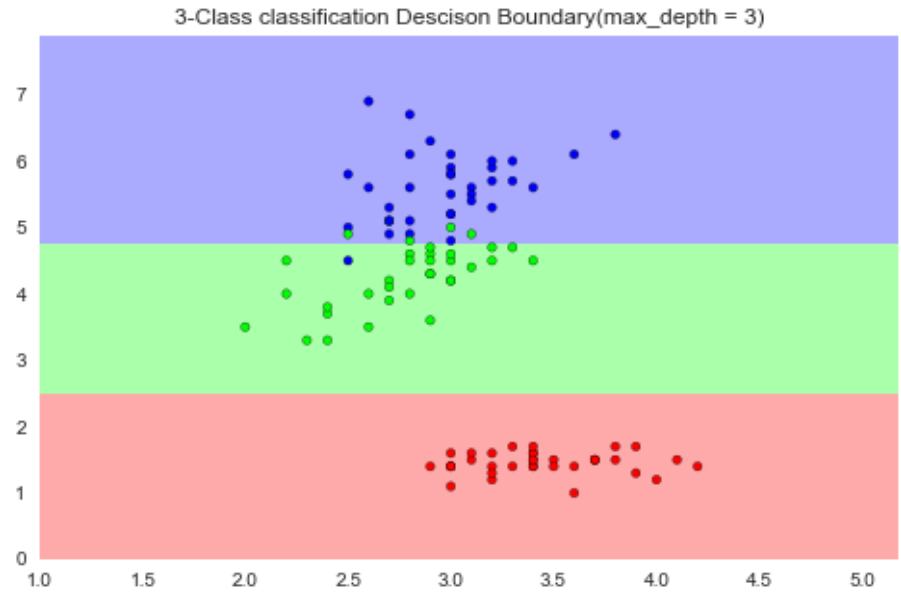

图 3-9 max_depth=3

四、SVM

4.1 算法简介

支持向量机(supportvector machines, SVM)是一种二类分类模型。它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；支持向量机还包括核技巧，这使它成为实质上的非线性分类器。支持向量机的学习策略就是间隔最大化，可形式化为一个求解凸二次规(convexquadratic programming)的问题，也等价于正则化的合页损失函数的最小化问题。支持向鱼机的学习算法是求解凸二次规划的最优化算法。

支持向量机学习方法包含构建由简至繁的模型：线性可分支持向量机（linear support vector machine in linearly separable case）、线性支持向量机（linear support vector machine）及非线性支持向量机(non-linear support vector machine)。简单模型是复杂模型的基础，也是复杂模型的特殊情况。当训练数据线性可分时，通过硬间隔最大化(hard margin maximization)，学习一个线性的分类器，即线性可分支持向量机，又称为硬间隔支持向量机；当训练数据近似线性可分时，通过软间隔最大化(soft margin maximization)，也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向景机；当训练数据线性不可分时，通过使用核技巧(kernel trick)及软间隔最大化，学习非线性支持向量机。

当输入空间为欧氏空间或离散集合、特征空间为希尔伯特空间时，核函数(kernel function)表示将输入从输入空间映射到特征空间得到的特征向量之间的内积。通过使用核函数可以学习非线性支持向量机，等价于隐式地在高维的特征空间中学习线性支持向量机．这样的方法称为核技巧。核方法(kernel method)是比支持向量机更为一般的机器学习方法。

4.2 实验

首先使用 LinearSVC。

为了选择合适的损失函数，我们使用交叉验证的方法，验证不同损失函数作时的准确率。由下图可知，损失函数为 hinge 时，交叉验证准确率为 76.94%，而选择 squared_hinge 时，准确率为 91.67%，故损失函数选择 squared_hinge。

```
In [172]:
1  cv_scores = []
2  for loss in ['hinge', 'squared_hinge']:
3      model = svm.LinearSVC(loss=loss)
4      scores = cross_val_score(model, train_X_scaled, train_y, cv=10, scorir
5      cv_scores.append(scores.mean())
6  print(cv_scores)

executed in 56ms, finished 20:46:51 2019-04-06

[0.7694444444444446, 0.9166666666666667]
```

图 4-1 不同损失函数时的准确率

接下来适应交叉验证的方法来选择合适的惩罚参数 C。由下图可知，选择 C=3。

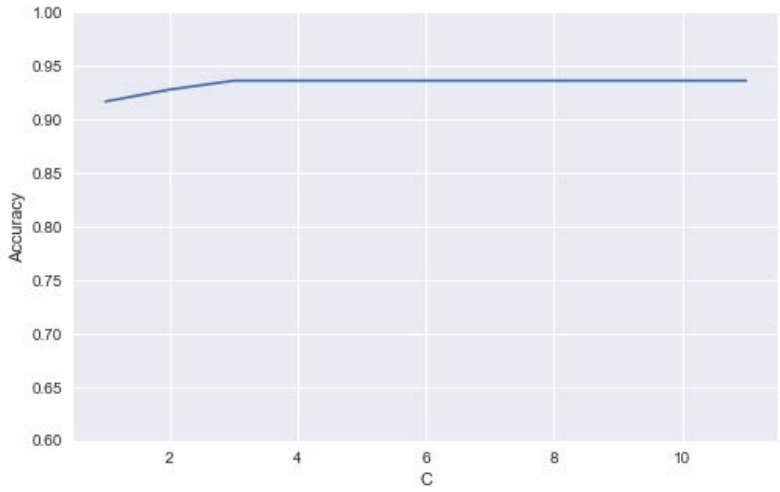

图 4-2 LinearSVC 不同惩罚参数 C 时的准确率

此分类器准确率为 93.3%。

The accuracy of linearSVC Classifier is 0.9333333333333333

	precision	recall	f1-score	support
setosa	1.00	0.93	0.97	15
versicolor	0.88	0.93	0.90	15
virginica	0.93	0.93	0.93	15
micro avg	0.93	0.93	0.93	45
macro avg	0.94	0.93	0.93	45
weighted avg	0.94	0.93	0.93	45

图 4-3 LinearSVC 准确率

使用 SVC 分类器。

使用交叉验证法，确定 C=3。

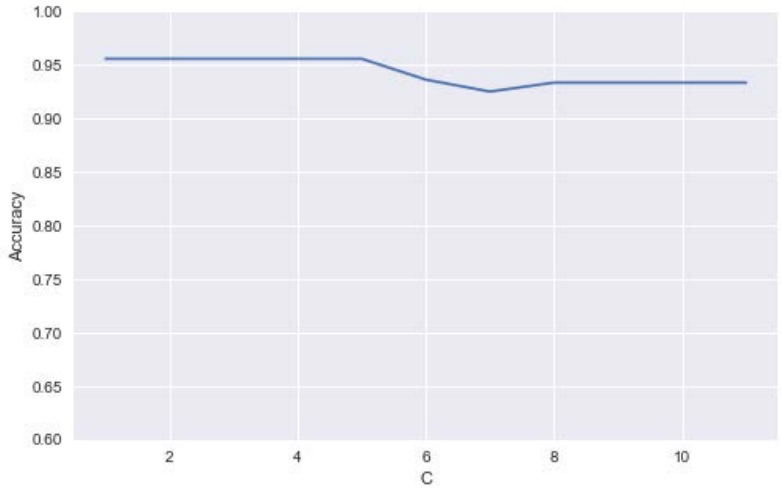

图 4-4 SVC 不同惩罚参数 C 时的准确率

此分类器准确率、精确率、召回率、F1-Score 均为 100%。

The accuracy of SVC Classifier is 1.0

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	1.00	1.00	1.00	15
virginica	1.00	1.00	1.00	15
micro avg	1.00	1.00	1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

图 4-5 SVC 准确率

使用非线性 SVC。

此分类器准确率、精确率、召回率、F1-Score 均为 100%。

The accuracy of NuSVC Classifier is 1.0

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	1.00	1.00	1.00	15
virginica	1.00	1.00	1.00	15
micro avg	1.00	1.00	1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

图 4-6 NuSVC 准确率