# 14.scikit-learn 库

scikit-learn 库是当今最流行的机器学习算法库之一

可用来解决分类与回归问题

本章以鸢尾花数据集为例，简单了解八大传统机器学习分类算法的sk-learn实现

八大传统分类算法：

K近邻：最近的k个邻居

朴素贝叶斯：后验概率最大化

决策树：向着纯净的类别，不断分裂

逻辑回归：特征映射成概率，全体概率之积最大化

支持向量机：最小间隔的最大化

集成方法-随机森林：多次有放回取样，弱分类器组合强分类器

集成方法-Adaboost：根据上轮弱分类器效果，更新数据权重，弱分类器加加权求和

集成方法-GBDT：不断地拟合残差

欲深入了解传统机器算法的原理和公式推导，请继续学习《统计学习方法》或《西瓜书》

## 14.1.鸢尾花数据集

【1】下载数据集

**import** **seaborn** **as** **sns**

*#iris = sns.load\_dataset("iris")*

iris = pd.read\_csv("data/iris.csv")

【2】数据集的查看

type(iris)

pandas.core.frame.DataFrame

iris.shape

(150, 5)

iris.head()

|  | **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 |

iris.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 5 columns):

sepal\_length 150 non-null float64

sepal\_width 150 non-null float64

petal\_length 150 non-null float64

petal\_width 150 non-null float64

species 150 non-null object

dtypes: float64(4), object(1)

memory usage: 5.9+ KB

iris.describe()

|  | **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 |

iris.species.value\_counts()

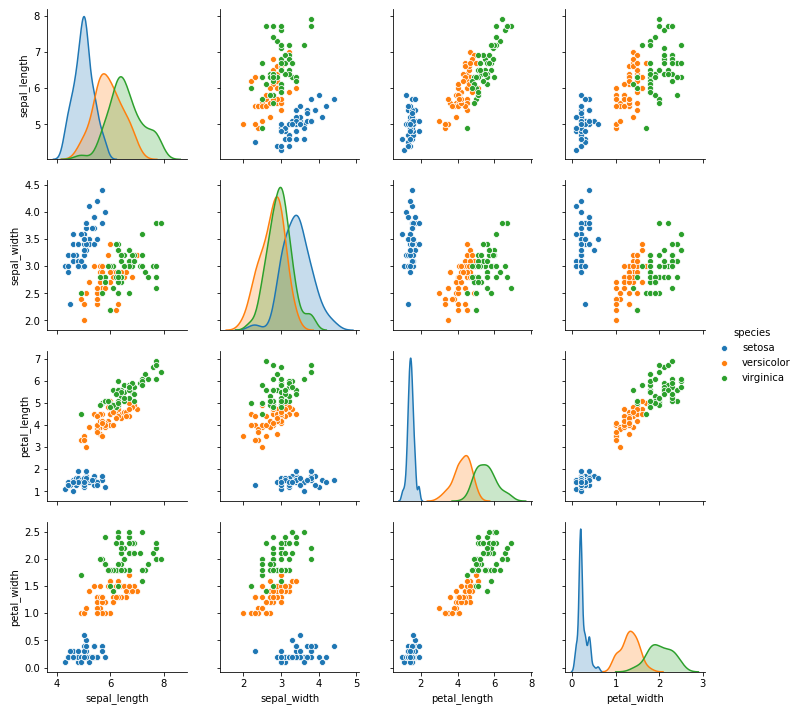
virginica 50

versicolor 50

setosa 50

Name: species, dtype: int64

sns.pairplot(data=iris, hue="species")



**【3】数据清洗**

iris\_simple = iris.drop(["sepal\_length", "sepal\_width"], axis=1)

iris\_simple.head()

|  | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- |
| **0** | 1.4 | 0.2 | setosa |
| **1** | 1.4 | 0.2 | setosa |
| **2** | 1.3 | 0.2 | setosa |
| **3** | 1.5 | 0.2 | setosa |
| **4** | 1.4 | 0.2 | setosa |

**【4】标签编码**

**from** **sklearn.preprocessing** **import** LabelEncoder

encoder = LabelEncoder()

iris\_simple["species"] = encoder.fit\_transform(iris\_simple["species"])

iris\_simple

|  | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- |
| **0** | 1.4 | 0.2 | 0 |
| **1** | 1.4 | 0.2 | 0 |
| **...** | ... | ... | ... |
| **149** | 5.1 | 1.8 | 2 |

【5】数据集的标准化（本数据集特征比较接近，实际处理过程中未标准化）

**from** **sklearn.preprocessing** **import** StandardScaler

**import** **pandas** **as** **pd**

trans = StandardScaler()

\_iris\_simple = trans.fit\_transform(iris\_simple[["petal\_length", "petal\_width"]])

\_iris\_simple = pd.DataFrame(\_iris\_simple, columns = ["petal\_length", "petal\_width"])

\_iris\_simple.describe()

|  | **petal\_length** | **petal\_width** |
| --- | --- | --- |
| **count** | 1.500000e+02 | 1.500000e+02 |
| **mean** | -8.652338e-16 | -4.662937e-16 |
| **std** | 1.003350e+00 | 1.003350e+00 |
| **min** | -1.567576e+00 | -1.447076e+00 |
| **25%** | -1.226552e+00 | -1.183812e+00 |
| **50%** | 3.364776e-01 | 1.325097e-01 |
| **75%** | 7.627583e-01 | 7.906707e-01 |
| **max** | 1.785832e+00 | 1.712096e+00 |

【6】构建训练集和测试集（本课暂不考虑验证集）

**from** **sklearn.model\_selection** **import** train\_test\_split

train\_set, test\_set = train\_test\_split(iris\_simple, test\_size=0.2)

test\_set.head()

|  | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- |
| **3** | 1.5 | 0.2 | 0 |
| **111** | 5.3 | 1.9 | 2 |
| **24** | 1.9 | 0.2 | 0 |
| **5** | 1.7 | 0.4 | 0 |
| **92** | 4.0 | 1.2 | 1 |

iris\_x\_train = train\_set[["petal\_length", "petal\_width"]]

iris\_x\_train.head()

|  | **petal\_length** | **petal\_width** |
| --- | --- | --- |
| **63** | 4.7 | 1.4 |
| **93** | 3.3 | 1.0 |
| **34** | 1.5 | 0.2 |
| **35** | 1.2 | 0.2 |
| **126** | 4.8 | 1.8 |

iris\_y\_train = train\_set["species"].copy()

iris\_y\_train.head()

63 1

93 1

34 0

35 0

126 2

Name: species, dtype: int32

iris\_x\_test = test\_set[["petal\_length", "petal\_width"]]

iris\_x\_test.head()

|  | **petal\_length** | **petal\_width** |
| --- | --- | --- |
| **3** | 1.5 | 0.2 |
| **111** | 5.3 | 1.9 |
| **24** | 1.9 | 0.2 |
| **5** | 1.7 | 0.4 |
| **92** | 4.0 | 1.2 |

iris\_y\_test = test\_set["species"].copy()

iris\_y\_test.head()

3 0

111 2

24 0

5 0

92 1

Name: species, dtype: int32

## 14.2.k近邻算法

**【1】基本思想**

与待预测点最近的训练数据集中的k个邻居

把k个近邻中最常见的类别预测为带预测点的类别

**【2】sklearn实现**

**from** **sklearn.neighbors** **import** KNeighborsClassifier

构建分类器对象

clf = KNeighborsClassifier()

clf

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,

weights='uniform')

训练

clf.fit(iris\_x\_train, iris\_y\_train)

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,

weights='uniform')

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

翻转

encoder.inverse\_transform(res)

array(['setosa', 'virginica', 'setosa', 'setosa', 'versicolor',

'versicolor', 'setosa', 'virginica', 'versicolor', 'virginica',

'versicolor', 'virginica', 'virginica', 'virginica', 'versicolor',

'setosa', 'setosa', 'setosa', 'versicolor', 'setosa', 'virginica',

'setosa', 'virginica', 'versicolor', 'setosa', 'versicolor',

'setosa', 'setosa', 'versicolor', 'versicolor'], dtype=object)

评估

accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

存储数据

out = iris\_x\_test.copy()

out["y"] = iris\_y\_test

out["pre"] = res

out

|  | **petal\_length** | **petal\_width** | **y** | **pre** |
| --- | --- | --- | --- | --- |
| **3** | 1.5 | 0.2 | 0 | 0 |
| **111** | 5.3 | 1.9 | 2 | 2 |
| **...** | ... | ... | ... | ... |
| **90** | 4.4 | 1.2 | 1 | 1 |

out.to\_csv("iris\_predict.csv")

**【3】可视化**

**import** **numpy** **as** **np**

**import** **matplotlib** **as** **mpl**

**import** **matplotlib.pyplot** **as** **plt**

**def** draw(clf):

*# 网格化*

M, N = 500, 500

x1\_min, x2\_min = iris\_simple[["petal\_length", "petal\_width"]].min(axis=0)

x1\_max, x2\_max = iris\_simple[["petal\_length", "petal\_width"]].max(axis=0)

t1 = np.linspace(x1\_min, x1\_max, M)

t2 = np.linspace(x2\_min, x2\_max, N)

x1, x2 = np.meshgrid(t1, t2)

*# 预测*

x\_show = np.stack((x1.flat, x2.flat), axis=1)

y\_predict = clf.predict(x\_show)

*# 配色*

cm\_light = mpl.colors.ListedColormap(["#A0FFA0", "#FFA0A0", "#A0A0FF"])

cm\_dark = mpl.colors.ListedColormap(["g", "r", "b"])

*# 绘制预测区域图*

plt.figure(figsize=(10, 6))

plt.pcolormesh(t1, t2, y\_predict.reshape(x1.shape), cmap=cm\_light)

*# 绘制原始数据点*

plt.scatter(iris\_simple["petal\_length"], iris\_simple["petal\_width"], label=**None**,

c=iris\_simple["species"], cmap=cm\_dark, marker='o', edgecolors='k')

plt.xlabel("petal\_length")

plt.ylabel("petal\_width")

*# 绘制图例*

color = ["g", "r", "b"]

species = ["setosa", "virginica", "versicolor"]

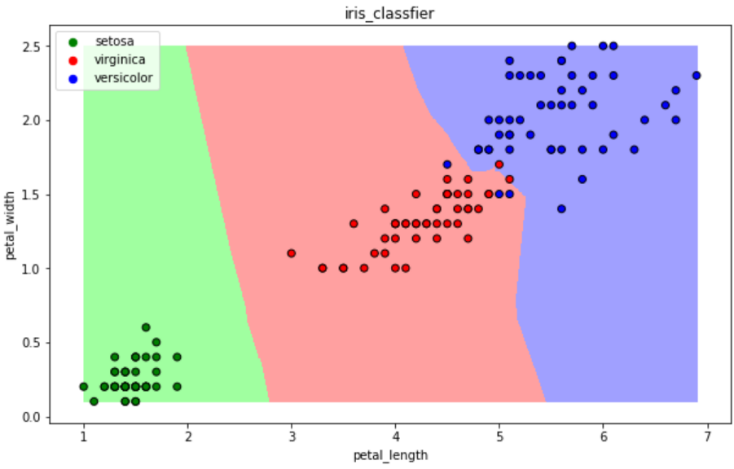
**for** i **in** range(3):

plt.scatter([], [], c=color[i], s=40, label=species[i]) *# 利用空点绘制图例*

plt.legend(loc="best")

plt.title('iris\_classfier')

draw(clf)



## 14.3.朴素贝叶斯算法

**【1】基本思想**

当发生的时候，哪一个发生的概率最大

**【2】sklearn实现**

**from** **sklearn.naive\_bayes** **import** GaussianNB

构建分类器对象

clf = GaussianNB()

clf

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

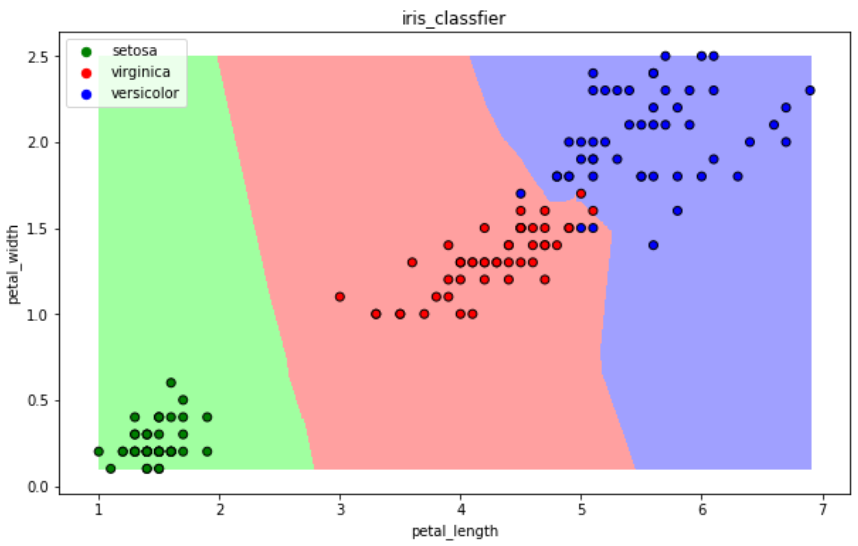
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.4.决策树算法

【1】基本思想

CART算法：每次通过一个特征，将数据尽可能的分为纯净的两类，递归的分下去

【2】sklearn实现

**from** **sklearn.tree** **import** DecisionTreeClassifier

构建分类器对象

clf = DecisionTreeClassifier()

clf

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False,

random\_state=None, splitter='best')

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

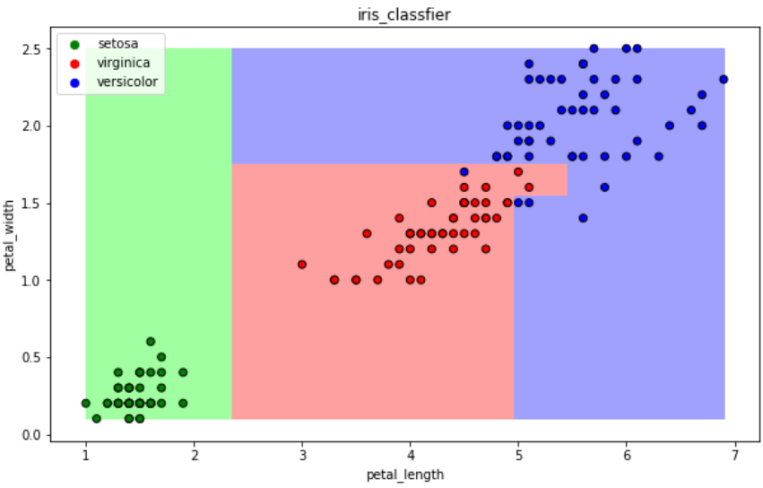
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.5.逻辑回归算法

**【1】基本思想**

一种解释：

训练：通过一个映射方式，将特征 映射成 ，求使得所有概率之积最大化的映射方式里的参数

预测：计算取概率最大的那个类别作为预测对象的分类

**【2】sklearn实现**

**from** **sklearn.linear\_model** **import** LogisticRegression

构建分类器对象

clf = LogisticRegression(solver='saga', max\_iter=1000)

clf

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, l1\_ratio=None, max\_iter=1000,

multi\_class='warn', n\_jobs=None, penalty='l2',

random\_state=None, solver='saga', tol=0.0001, verbose=0,

warm\_start=False)

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

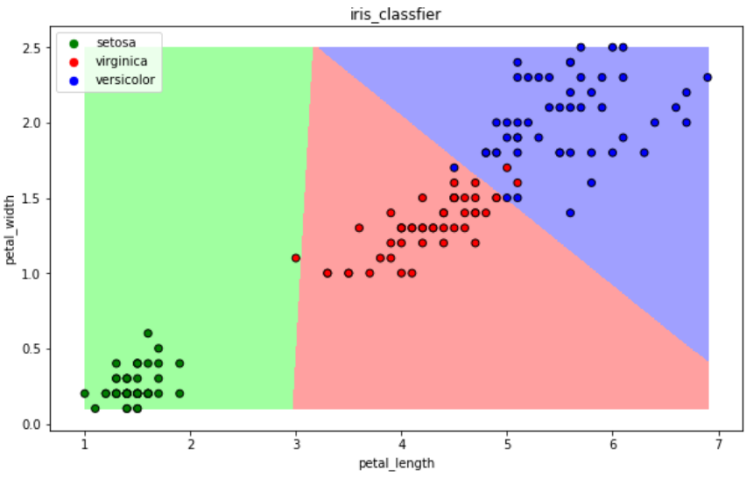
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.6.支持向量机算法

**【1】基本思想**

以二分类为例，假设数据可用完全分开：

用一个超平面将两类数据完全分开，且最近点到平面的距离最大

**【2】sklearn实现**

**from** **sklearn.svm** **import** SVC

构建分类器对象

clf = SVC()

clf

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma='auto\_deprecated',

kernel='rbf', max\_iter=-1, probability=False, random\_state=None,

shrinking=True, tol=0.001, verbose=False)

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

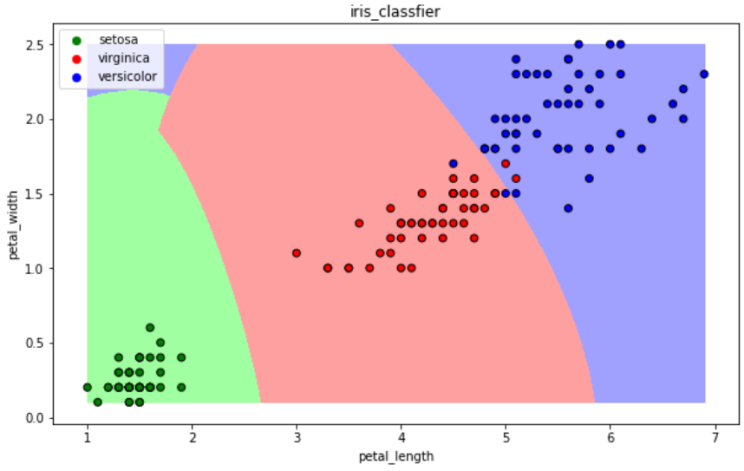
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.7.集成方法——随机森林

【1】基本思想

训练集m，有放回的随机抽取m个数据，构成一组，共抽取n组采样集

n组采样集训练得到n个弱分类器 弱分类器一般用决策树或神经网络

将n个弱分类器进行组合得到强分类器

【2】sklearn实现

**from** **sklearn.ensemble** **import** RandomForestClassifier

构建分类器对象

clf = RandomForestClassifier()

clf

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators='warn',

n\_jobs=None, oob\_score=False, random\_state=None,

verbose=0, warm\_start=False)

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

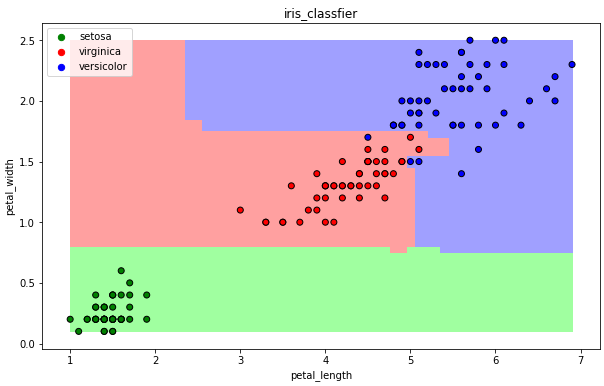
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.8.集成方法——Adaboost

**【1】基本思想**

训练集m，用初始数据权重训练得到第一个弱分类器，根据误差率计算弱分类器系数，更新数据的权重

使用新的权重训练得到第二个弱分类器，以此类推

根据各自系数，将所有弱分类器加权求和获得强分类器

**【2】sklearn实现**

**from** **sklearn.ensemble** **import** AdaBoostClassifier

构建分类器对象

clf = AdaBoostClassifier()

clf

AdaBoostClassifier(algorithm='SAMME.R', base\_estimator=None, learning\_rate=1.0,

n\_estimators=50, random\_state=None)

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

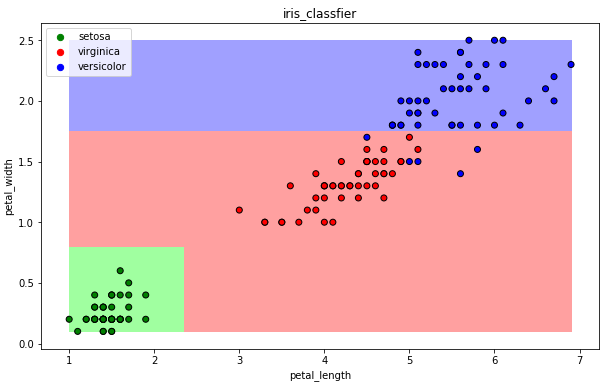
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.9.集成方法——梯度提升树GBDT

【1】基本思想

训练集m，获得第一个弱分类器，获得残差，然后不断地拟合残差

所有弱分类器相加得到强分类器

【2】sklearn实现

**from** **sklearn.ensemble** **import** GradientBoostingClassifier

构建分类器对象

clf = GradientBoostingClassifier()

clf

GradientBoostingClassifier(criterion='friedman\_mse', init=None,

learning\_rate=0.1, loss='deviance', max\_depth=3,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_iter\_no\_change=None, presort='auto',

random\_state=None, subsample=1.0, tol=0.0001,

validation\_fraction=0.1, verbose=0,

warm\_start=False)

训练

clf.fit(iris\_x\_train, iris\_y\_train)

预测

res = clf.predict(iris\_x\_test)

print(res)

print(iris\_y\_test.values)

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

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

评估

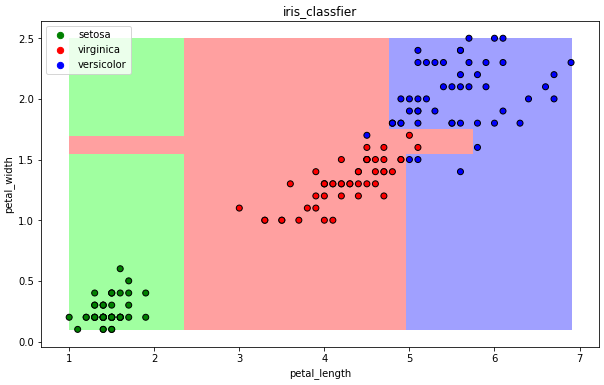
accuracy = clf.score(iris\_x\_test, iris\_y\_test)

print("预测正确率:**{:.0%}**".format(accuracy))

预测正确率:97%

可视化

draw(clf)



## 14.10.扩展

**【1】xgboost**

GBDT的损失函数只对误差部分做负梯度（一阶泰勒）展开

XGBoost损失函数对误差部分做二阶泰勒展开，更加准确，更快收敛

**【2】lightgbm**

微软：快速的，分布式的，高性能的基于决策树算法的梯度提升框架

速度更快

**【3】stacking**

堆叠或者叫模型融合

先建立几个简单的模型进行训练，第二级学习器会基于前级模型的预测结果进行再训练

**【4】神经网络**