集成学习

刘德华

2023 年 6 月 25 日

例子 1

使用 bagging 集成方法构造决策树回归模型进行回归[a]。

[a]实现的程序见例1

例子 2

使用随机森林和 adaboost 集成方法在 iris 数据集上绘制决策面[a]。

[a]实现的程序见例1

例子 3

使用梯度提升回归集成方法进行回归[a]。

[a]实现的程序见例1

例子 4

使用梯度提升 out-of-bag 分类集成方法进行分类[a]。

[a]实现的程序见例1

例子 5

为 Iris 数据集的两个特征绘制 VotingClassifier 的决策边界[a]。

[a]实现的程序见例1

例子 6

在糖尿病数据集上使用 VotingRegression 作回归[a]。

[a]实现的程序见例1

1 代码

Python 代码 1

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 bagging 回归器
from sklearn.ensemble import BaggingRegressor
# 导入决策树回归
from sklearn.tree import DecisionTreeRegressor
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 计算期望的迭代次数
n_repeat = 50
# 训练集样本量
n_train = 50
# 测试集样本量
n_test = 1000
# 标准差
noise = 0.1
np.random.seed(0)
# 构造模型
estimators = [
    ("Tree", DecisionTreeRegressor()),
    ("Bagging(Tree)", BaggingRegressor(DecisionTreeRegressor())),
]
# 回归器的长度
n_estimators = len(estimators)
# 生成 x 数据
def f(x):
    x = x.ravel()
    return np.exp(-(x**2)) + 1.5 * np.exp(-((x - 2) ** 2))
# 生成 XY 数据
def generate(n_samples, noise, n_repeat=1):
```

```
    X = np.random.rand(n_samples) * 10 - 5
    X = np.sort(X)
    if n_repeat == 1:
        y = f(X) + np.random.normal(0.0, noise, n_samples)
    else:
        y = np.zeros((n_samples, n_repeat))

        for i in range(n_repeat):
            y[:, i] = f(X) + np.random.normal(0.0, noise, n_samples)
    X = X.reshape((n_samples, 1))
    return X, y
# 训练集列表
X_train = []
y_train = []
# 多个训练集
for i in range(n_repeat):
    X, y = generate(n_samples=n_train, noise=noise)
    X_train.append(X)
    y_train.append(y)
# 生成一个测试集
X_test, y_test = generate(n_samples=n_test, noise=noise, n_repeat=n_repeat)
# 开始绘图
fig, axs = plt.subplots(2, 2, figsize=(10, 8), tight_layout=True)
for n, (name, estimator) in enumerate(estimators):
    # 初始化预测值
    y_predict = np.zeros((n_test, n_repeat))
    for i in range(n_repeat):
        # 模型拟合
        estimator.fit(X_train[i], y_train[i])
        # 模型预测
        y_predict[:, i] = estimator.predict(X_test)
    # 误差分解，初始化误差值
    y_error = np.zeros(n_test)
    for i in range(n_repeat):
        for j in range(n_repeat):
            # 计算预测值和真实值之间的误差
            y_error += (y_test[:, j] - y_predict[:, i]) ** 2
    # rescale 一下
    y_error /= n_repeat * n_repeat
    # test 数据集的噪声，方差
    y_noise = np.var(y_test, axis=1)
    # 偏差平方和
    y_bias = (f(X_test) - np.mean(y_predict, axis=1)) ** 2
```

```
    # 预测值的方差
    y_var = np.var(y_predict, axis=1)
    print(
        "{0}: {1:.4f} (error) = {2:.4f} (bias^2) "
        " + {3:.4f} (var) + {4:.4f} (noise)".format(
            name, np.mean(y_error), np.mean(y_bias),
            np.mean(y_var), np.mean(y_noise)
        )
    )
    # Plot figures
    ax1=axs.flatten()[n]
    # 绘制测试集的真实值，无噪声
    ax1.plot(X_test, f(X_test), "b", label="$f(x)$")
    # 绘制训练集散点图，有噪声
    ax1.plot(X_train[0], y_train[0], ".b", label="LS ~ $y = f(x)+noise$")

    for i in range(n_repeat):
        if i == 0:
            # 测试集上的预测值
            ax1.plot(X_test, y_predict[:, i], "r", label=r"$\^y(x)$")
        else:
            # 测试集上的预测值
            ax1.plot(X_test, y_predict[:, i], "r", alpha=0.05)
    # 测试集上的预测值的均值
    ax1.plot(X_test, np.mean(y_predict, axis=1), "c", label=r"$\mathbb{E}_{LS}␣
↪\^y(x)$")
    ax1.set_xlim([-5, 5])
    ax1.set_title(name)
    if n == n_estimators - 1:
        ax1.legend(loc=(1.1, 0.5))

    ax2 = axs.flatten()[n_estimators + n]
    # 测试集上的误差，偏差，方差，噪声
    ax2.plot(X_test, y_error, "r", label="$error(x)$")
    ax2.plot(X_test, y_bias, "b", label="$bias^2(x)$"),
    ax2.plot(X_test, y_var, "g", label="$variance(x)$"),
    ax2.plot(X_test, y_noise, "c", label="$noise(x)$")
    ax2.set_xlim([-5, 5])
    ax2.set_ylim([0, 0.1])
    if n == n_estimators - 1:
        ax2.legend(loc=(1.1, 0.5))

plt.show()
```

```
fig.savefig("../codeimage/code1.pdf")

Tree: 0.0255 (error) = 0.0003 (bias^2)  + 0.0152 (var) + 0.0098 (noise)
Bagging(Tree): 0.0196 (error) = 0.0004 (bias^2)  + 0.0092 (var) + 0.0098 (noise)
```

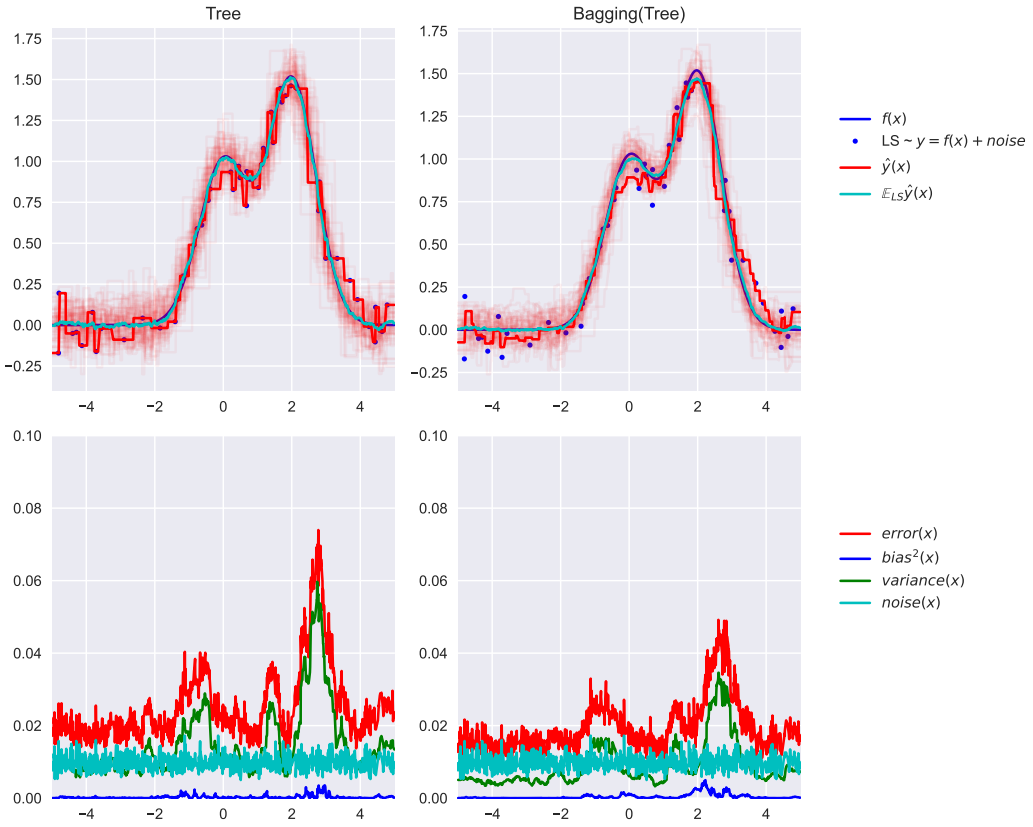

图 1: code1

Python 代码 2

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入颜色库
from matplotlib.colors import ListedColormap
# 导入数据集工具
from sklearn.datasets import load_iris
# 导入集成模型
from sklearn.ensemble import (
```

```
    RandomForestClassifier, # 随机森林
    ExtraTreesClassifier, # 额外树
    AdaBoostClassifier, # adaboost 模型
)
# 导入决策树分类器
from sklearn.tree import DecisionTreeClassifier
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 类别
n_classes = 3
# 估计器数量
n_estimators = 30
# 选择颜色
cmap = plt.cm.RdYlBu
# step width for decision surface contours
plot_step = 0.02
# step widths for coarse classifier guesses
plot_step_coarser = 0.5
# fix the seed on each iteration
RANDOM_SEED = 13
# 加载数据
iris = load_iris()
# 图形编号
plot_idx = 0
# 构建模型
models = [
    DecisionTreeClassifier(max_depth=None),
    RandomForestClassifier(n_estimators=n_estimators),
    ExtraTreesClassifier(n_estimators=n_estimators),
    AdaBoostClassifier(DecisionTreeClassifier(max_depth=3),␣
 ↪n_estimators=n_estimators),
]
# 开始绘图
fig, axs = plt.subplots(3, 4, figsize=(12, 14), tight_layout=True)
for pair in ([0, 1], [0, 2], [2, 3]):
    for model in models:
        # 只提取两个维度的变量
        X = iris.data[:, pair]
        y = iris.target
```

```
        # 构造训练集
        idx = np.arange(X.shape[0])
        np.random.seed(RANDOM_SEED)
        np.random.shuffle(idx)
        X = X[idx]
        y = y[idx]
        # 标准化
        mean = X.mean(axis=0)
        std = X.std(axis=0)
        X = (X - mean) / std
        # 模型拟合
        model.fit(X, y)
        # 模型在训练集上的预测准确率
        scores = model.score(X, y)
        # Create a title for each column and the console by using str() and
        # slicing away useless parts of the string
        model_title = str(type(model)).split(".")[-1][:-2][: -len("Classifier")]
        model_details = model_title
        if hasattr(model, "estimators_"):
            model_details += " with {} estimators".format(len(model.estimators_))
        print(model_details + " with features", pair, "has a score of", scores)

        ax = axs.flatten()[plot_idx]
        if plot_idx <= len(models):
            # Add a title at the top of each column
            ax.set_title(model_title, fontsize=9)
        # 绘制等高线，决策边界
        # filled contour plot
        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(
            np.arange(x_min, x_max, plot_step), np.arange(y_min, y_max, plot_step)
        )
        # Plot either a single DecisionTreeClassifier or alpha blend the
        # decision surfaces of the ensemble of classifiers
        if isinstance(model, DecisionTreeClassifier):
            # 模型预测
            Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
            # 将 Z 重塑
            Z = Z.reshape(xx.shape)
            # 等高线
            cs = ax.contourf(xx, yy, Z, cmap=cmap)
        else:
```

```
            # Choose alpha blend level with respect to the number
            # of estimators
            # that are in use (noting that AdaBoost can use fewer estimators
            # than its maximum if it achieves a good enough fit early on)
            estimator_alpha = 1.0 / len(model.estimators_)
            for tree in model.estimators_:
                Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
                Z = Z.reshape(xx.shape)
                cs = ax.contourf(xx, yy, Z, alpha=estimator_alpha, cmap=cmap)

        # Build a coarser grid to plot a set of ensemble classifications
        # to show how these are different to what we see in the decision
        # surfaces. These points are regularly space and do not have a
        # black outline
        xx_coarser, yy_coarser = np.meshgrid(
            np.arange(x_min, x_max, plot_step_coarser),
            np.arange(y_min, y_max, plot_step_coarser),
        )
        Z_points_coarser = model.predict(
            np.c_[xx_coarser.ravel(), yy_coarser.ravel()]
        ).reshape(xx_coarser.shape)
        cs_points = ax.scatter(
            xx_coarser,
            yy_coarser,
            s=15,
            c=Z_points_coarser,
            cmap=cmap,
            edgecolors="none",
        )
        # Plot the training points, these are clustered together and have a
        # black outline
        ax.scatter(
            X[:, 0],
            X[:, 1],
            c=y,
            cmap=ListedColormap(["r", "y", "b"]),
            edgecolor="k",
            s=20,
        )
        plot_idx += 1  # move on to the next plot in sequence

plt.suptitle("Classifiers on feature subsets of the Iris dataset", fontsize=12)
plt.show()
```

```
fig.savefig("../codeimage/code2.pdf")
```

```
DecisionTree with features [0, 1] has a score of 0.9266666666666666
RandomForest with 30 estimators with features [0, 1] has a score of
0.9266666666666666
ExtraTrees with 30 estimators with features [0, 1] has a score of
0.9266666666666666
AdaBoost with 30 estimators with features [0, 1] has a score of
0.8666666666666667
DecisionTree with features [0, 2] has a score of 0.9933333333333333
RandomForest with 30 estimators with features [0, 2] has a score of
0.9933333333333333
ExtraTrees with 30 estimators with features [0, 2] has a score of
0.9933333333333333
AdaBoost with 30 estimators with features [0, 2] has a score of
0.9933333333333333
DecisionTree with features [2, 3] has a score of 0.9933333333333333
RandomForest with 30 estimators with features [2, 3] has a score of
0.9933333333333333
ExtraTrees with 30 estimators with features [2, 3] has a score of
0.9933333333333333
AdaBoost with 30 estimators with features [2, 3] has a score of
0.9933333333333333
```

Python 代码 3

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集工具
from sklearn.datasets import load_diabetes
# 导入集成模型
from sklearn.ensemble import GradientBoostingRegressor
# 导入均方误差工具
from sklearn.metrics import mean_squared_error
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入绘图库中的字体管理包
from matplotlib import font_manager
```

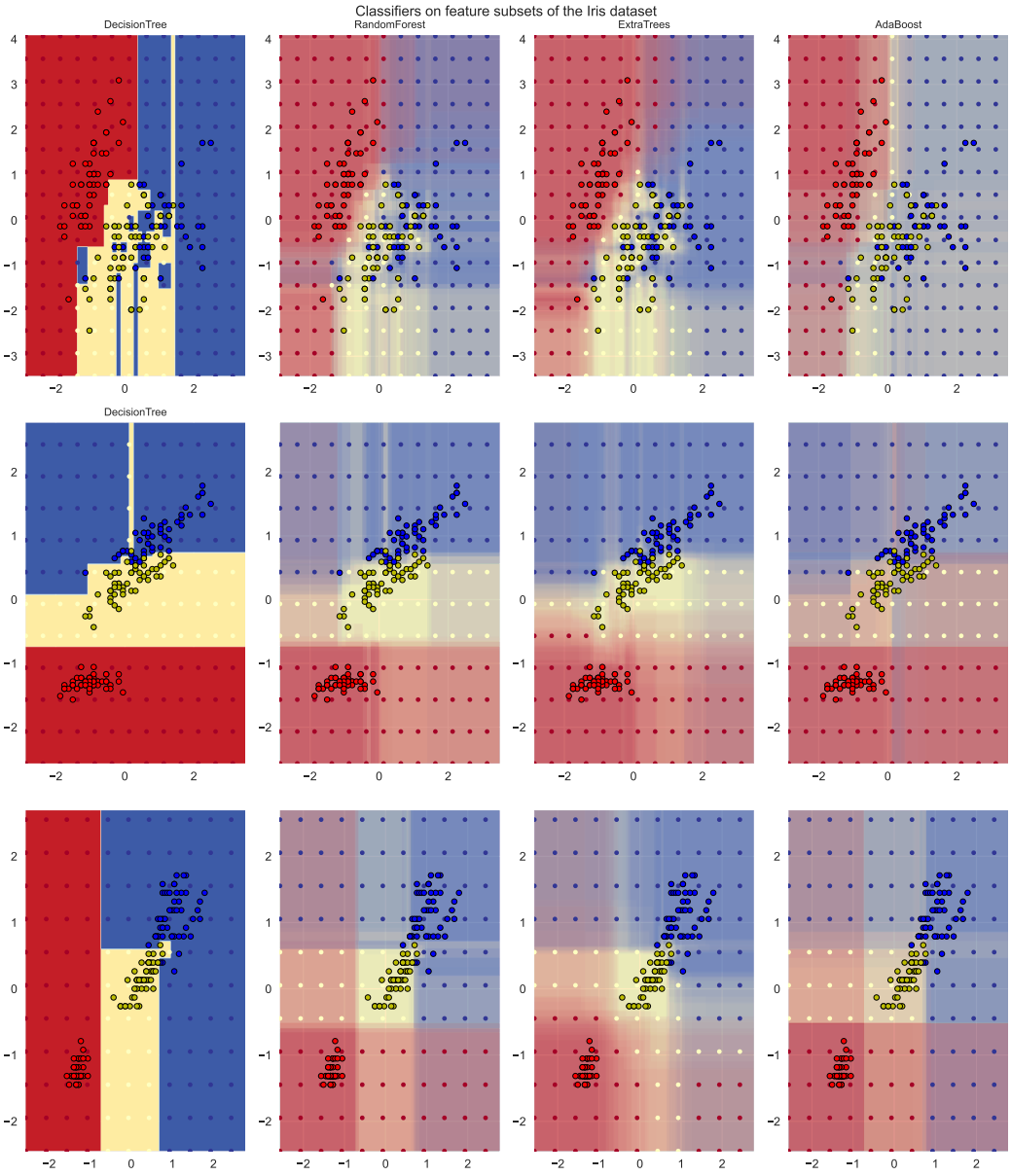
Classifiers on feature subsets of the Iris dataset
DecisionTree
RandomForest
ExtraTrees
AdaBoost
DecisionTree

图 2: code2

```
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 加载数据集
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target
# 划分数据集
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1, random_state=13
)
# 设置一些参数
params = {
    "n_estimators": 500,
    "max_depth": 4,
    "min_samples_split": 5,
    "learning_rate": 0.01,
    "loss": "squared_error",
}
# 构建模型
reg = GradientBoostingRegressor(**params)
# 模型拟合
reg.fit(X_train, y_train)
# 预测
y_pred = reg.predict(X_test)
# 均方误差
mse = mean_squared_error(y_test, y_pred)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
# 初始化测试集上的 mse
test_score = np.zeros((params["n_estimators"],), dtype=np.float64)
# 阶段性预测 X_test
for i, y_pred in enumerate(reg.staged_predict(X_test)):
    # 得到回归的 mse
    test_score[i] = mean_squared_error(y_test, y_pred)

fig, ax = plt.subplots(figsize=(6, 6), tight_layout=True)
ax.set_title("Deviance")
ax.plot(
    np.arange(params["n_estimators"]) + 1,
    reg.train_score_,
    "b-",
    label="Training Set Deviance",
)
```

```
ax.plot(
    np.arange(params["n_estimators"]) + 1,
    test_score, "r-", label="Test Set Deviance"
)
ax.legend(loc="upper right")
ax.set_xlabel("Boosting Iterations")
ax.set_ylabel("Deviance")
plt.show()
fig.savefig("../codeimage/code3.pdf")

The mean squared error (MSE) on test set: 3041.0505
```

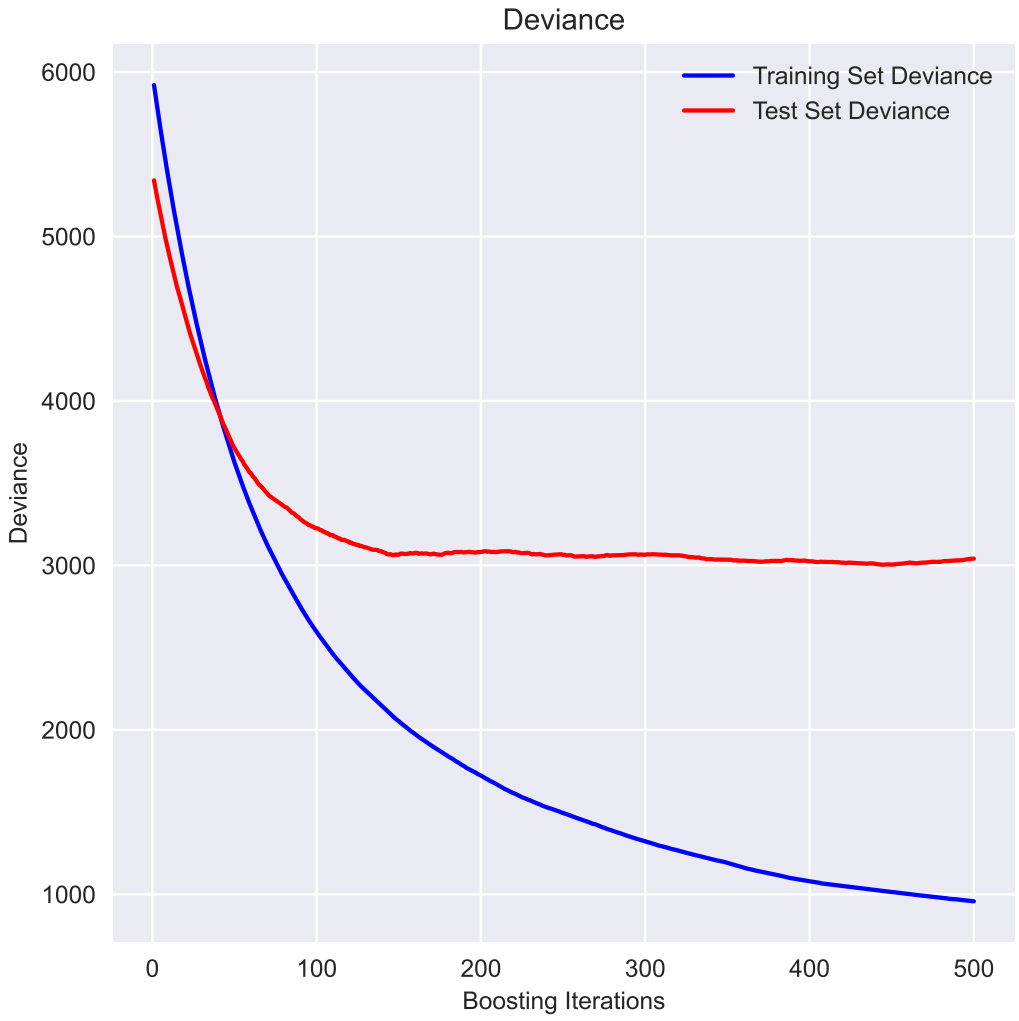

图 3: code3

Python 代码 4

```
# 导入操作系统库
```

```
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
# 导入基础计算库
import numpy as np
# 导入绘图库
import matplotlib.pyplot as plt
# 导入 K 折交叉验证
from sklearn.model_selection import KFold
# 导入梯度提升分类器
from sklearn.ensemble import GradientBoostingClassifier
# 导入数据集划分工具
from sklearn.model_selection import train_test_split
# 导入 log_loss 函数
from sklearn.metrics import log_loss
# 导入 expit 函数
from scipy.special import expit
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 样本量
n_samples = 1000
np.random.seed(13)
# 生成 x
x1 = np.random.uniform(size=n_samples)
x2 = np.random.uniform(size=n_samples)
x3 = np.random.randint(0, 4, size=n_samples)
# 计算概率
p = expit(np.sin(3 * x1) - 4 * x2 + x3)
# 生成 y
y = np.random.binomial(1, p, size=n_samples)
# 合并
X = np.c_[x1, x2, x3]
X = X.astype(np.float32)
# 数据集划分
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=9
)
# Fit classifier with out-of-bag estimates
params = {
```

```
    "n_estimators": 1200,
    "max_depth": 3,
    "subsample": 0.5,
    "learning_rate": 0.01,
    "min_samples_leaf": 1,
    "random_state": 3,
}
# 构造模型
clf = GradientBoostingClassifier(**params)
# 模型拟合
clf.fit(X_train, y_train)
# 分类准确率
acc = clf.score(X_test, y_test)
print("Accuracy: {:.4f}".format(acc))
# 估计器的数量
n_estimators = params["n_estimators"]
x = np.arange(n_estimators) + 1
def heldout_score(clf, X_test, y_test):
    """compute deviance scores on ``X_test`` and ``y_test``."""
    score = np.zeros((n_estimators,), dtype=np.float64)
    # 阶段性预测概率
    for i, y_proba in enumerate(clf.staged_predict_proba(X_test)):
        score[i] = 2 * log_loss(y_test, y_proba[:, 1])
    return score
# CV 估计
def cv_estimate(n_splits=None):
    # k 折
    cv = KFold(n_splits=n_splits)
    # 构建模型
    cv_clf = GradientBoostingClassifier(**params)
    val_scores = np.zeros((n_estimators,), dtype=np.float64)
    for train, test in cv.split(X_train, y_train):
        # 每一折下进行模型拟合
        cv_clf.fit(X_train[train], y_train[train])
        # 得分
        val_scores += heldout_score(cv_clf, X_train[test], y_train[test])
    val_scores /= n_splits
    return val_scores

# Estimate best n_estimator using cross-validation
cv_score = cv_estimate(3)
# Compute best n_estimator for test data
test_score = heldout_score(clf, X_test, y_test)
```

```
# negative cumulative sum of oob improvements
cumsum = -np.cumsum(clf.oob_improvement_)
# min loss according to OOB
oob_best_iter = x[np.argmin(cumsum)]
# min loss according to test (normalize such that first loss is 0)
test_score -= test_score[0]
test_best_iter = x[np.argmin(test_score)]
# min loss according to cv (normalize such that first loss is 0)
cv_score -= cv_score[0]
cv_best_iter = x[np.argmin(cv_score)]
# 设置曲线的颜色
oob_color = list(map(lambda x: x / 256.0, (190, 174, 212)))
test_color = list(map(lambda x: x / 256.0, (127, 201, 127)))
cv_color = list(map(lambda x: x / 256.0, (253, 192, 134)))
# 设置线型
oob_line = "dashed"
test_line = "solid"
cv_line = "dashdot"
# plot curves and vertical lines for best iterations
fig, ax = plt.subplots(figsize=(8, 6), tight_layout=True)
ax.plot(
    x, cumsum, label="OOB loss",
    color=oob_color, linestyle=oob_line
)
ax.plot(
    x, test_score, label="Test loss",
    color=test_color, linestyle=test_line
)
ax.plot(
    x, cv_score, label="CV loss",
    color=cv_color, linestyle=cv_line
)
ax.axvline(x=oob_best_iter, color=oob_color, linestyle=oob_line)
ax.axvline(x=test_best_iter, color=test_color, linestyle=test_line)
ax.axvline(x=cv_best_iter, color=cv_color, linestyle=cv_line)
# add three vertical lines to xticks
xticks = plt.xticks()
xticks_pos = np.array(
    xticks[0].tolist() + [oob_best_iter, cv_best_iter, test_best_iter]
)
xticks_label = np.array(list(
    map(lambda t: int(t), xticks[0])
    ) + ["OOB", "CV", "Test"]
```

```
)
ind = np.argsort(xticks_pos)
xticks_pos = xticks_pos[ind]
xticks_label = xticks_label[ind]
plt.xticks(xticks_pos, xticks_label, rotation=90)
ax.legend(loc="upper center")
ax.set_ylabel("normalized loss")
ax.set_xlabel("number of iterations")
plt.show()
fig.savefig("../codeimage/code4.pdf")

Accuracy: 0.6820
```

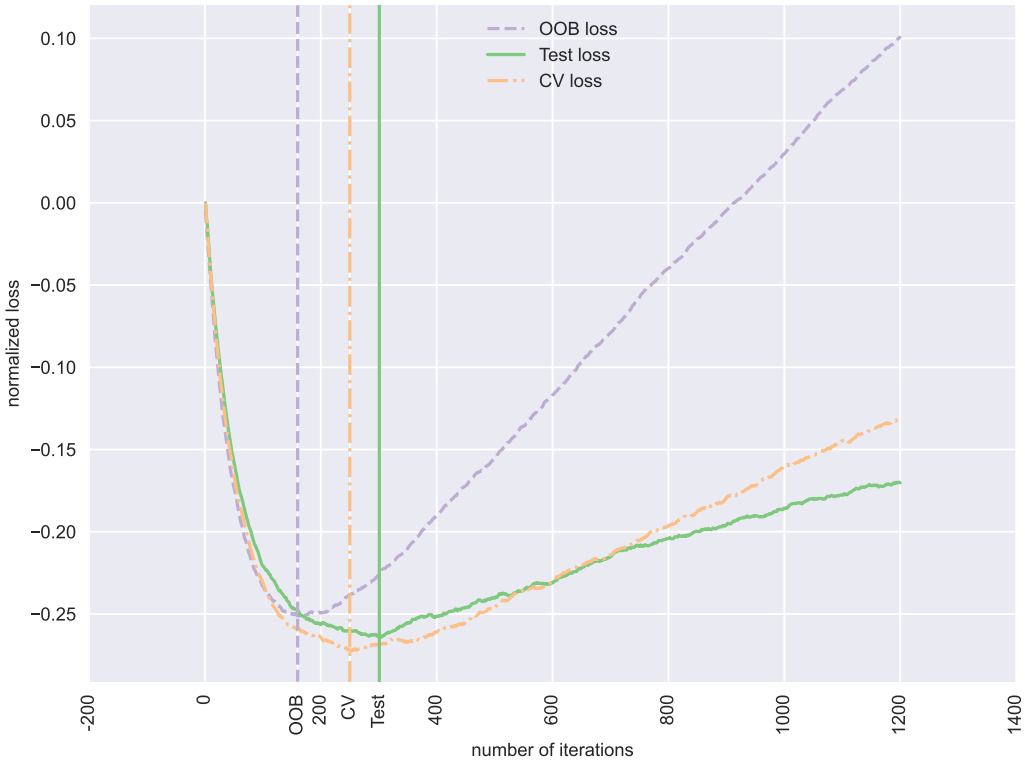

图 4: code4

Python 代码 5

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集
```

```
from sklearn.datasets import load_iris
# 导入决策树分类
from sklearn.tree import DecisionTreeClassifier
# 导入 KNN 分类
from sklearn.neighbors import KNeighborsClassifier
# 导入 SVC
from sklearn.svm import SVC
# 导入投票投票分类器
from sklearn.ensemble import VotingClassifier
# 导入决策边界显示
from sklearn.inspection import DecisionBoundaryDisplay
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 导入 product 工具
from itertools import product
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
# 加载数据集
iris = load_iris()
# X,  Y
X = iris.data[:, [0, 2]]
y = iris.target
# 决策树
clf1 = DecisionTreeClassifier(max_depth=4)
# KNN
clf2 = KNeighborsClassifier(n_neighbors=7)
# SVC
clf3 = SVC(gamma=0.1, kernel="rbf", probability=True)
# 投票选择分类器
eclf = VotingClassifier(
    estimators=[("dt", clf1), ("knn", clf2), ("svc", clf3)],
    voting="soft", # 软投票
    weights=[2, 1, 2], # 权重
)
# 模型拟合
clf1.fit(X, y)
clf2.fit(X, y)
clf3.fit(X, y)
eclf.fit(X, y)
# 绘制投票边界
fig, axs = plt.subplots(2, 2, sharex="col", sharey="row", figsize=(10, 8))
```

```
for idx, clf, tt in zip(
    product([0, 1], [0, 1]),
    [clf1, clf2, clf3, eclf],
    ["Decision Tree (depth=4)", "KNN (k=7)", "Kernel SVM", "Soft Voting"],
):
    DecisionBoundaryDisplay.from_estimator(
        clf, X, alpha=0.4, ax=axs[idx[0], idx[1]], response_method="predict"
    )
    axs[idx[0], idx[1]].scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor="k")
    axs[idx[0], idx[1]].set_title(tt)

plt.show()
fig.savefig("../codeimage/code5.pdf")
```

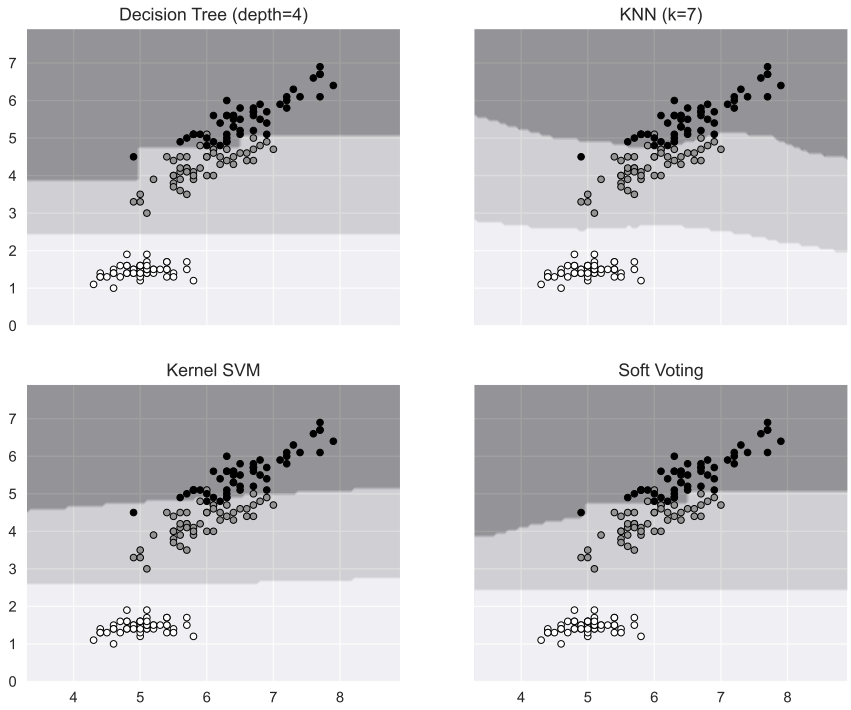

图 5: code5

Python 代码 6

```
# 导入操作系统库
import os
# 更改工作目录
os.chdir(r"D:\softwares\applied statistics\pythoncodelearning\chap6\sourcecode")
```

```
# 导入绘图库
import matplotlib.pyplot as plt
# 导入数据集
from sklearn.datasets import load_diabetes
# 导入梯度提升回归
from sklearn.ensemble import GradientBoostingRegressor
# 导入随机森林
from sklearn.ensemble import RandomForestRegressor
# 导入线性模型
from sklearn.linear_model import LinearRegression
# 导入投票回归
from sklearn.ensemble import VotingRegressor
# 导入绘图库中的字体管理包
from matplotlib import font_manager
# 实现中文字符正常显示
font = font_manager.FontProperties(fname=r"C:\Windows\Fonts\SimKai.ttf")
# 使用 seaborn 风格绘图
plt.style.use("seaborn-v0_8")
X, y = load_diabetes(return_X_y=True)
# 构建回归模型
reg1 = GradientBoostingRegressor(random_state=1)
reg2 = RandomForestRegressor(random_state=1)
reg3 = LinearRegression()
# 模型拟合
reg1.fit(X, y)
reg2.fit(X, y)
reg3.fit(X, y)
# 投票回归模型
ereg = VotingRegressor([("gb", reg1), ("rf", reg2), ("lr", reg3)])
# 模型拟合
ereg.fit(X, y)
# 用于预测的 X
xt = X[:20]
# 预测值
pred1 = reg1.predict(xt)
pred2 = reg2.predict(xt)
pred3 = reg3.predict(xt)
pred4 = ereg.predict(xt)
fig, ax = plt.subplots(figsize=(6,6), tight_layout=True)
# 绘制预测的散点图
ax.plot(pred1, "gd", label="GradientBoostingRegressor")
ax.plot(pred2, "b^", label="RandomForestRegressor")
ax.plot(pred3, "ys", label="LinearRegression")
```

```
ax.plot(pred4, "r*", ms=10, label="VotingRegressor")
plt.tick_params(
    axis="x", which="both", bottom=False,
    top=False, labelbottom=False
)
ax.set_ylabel("predicted")
ax.set_xlabel("training samples")
ax.legend(loc="best")
ax.set_title("Regressor predictions and their average")
plt.show()
fig.savefig("../codeimage/code6.pdf")
```

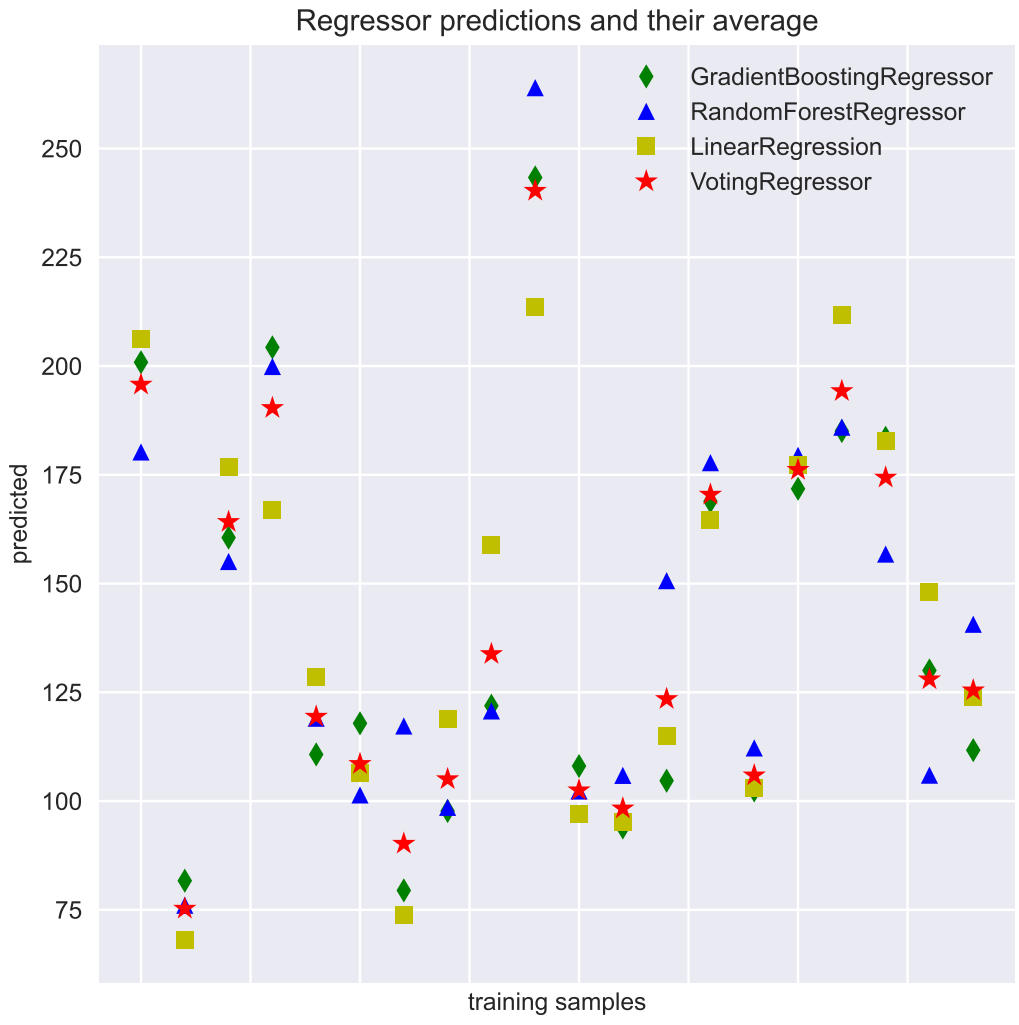

图 6: code6