人工智能第二次实验

郑子浩 2017202117

目录

一、实验一回顾：

Gym 重要的四个函数回顾：

reset(self)：重置环境的状态，返回观察。

step(self, action)：推进一个时间步长，返回 observation, reward, done, info。

render(self, mode=‘human’, close=False)：重绘环境的一帧。默认模式一般比较友好，如弹出一个窗口。

close(self)：关闭环境，并清除内存。

在 Gym 中，env.step()会返回 4 个参数：

◉观测 Observation (Object)：当前 step 执行后，环境的观测(类型为对象)。例如，从相机获取的像素点，机器人各个关节的角度或棋盘游戏当前的状态等；

◉奖励 Reward (Float): 执行上一步动作(action)后，智能体(agent)获得的奖励(浮点类型)，不同的环境中奖励值变化范围也不相同，但是强化学习的目标就是使得总奖励值最大；

◉完成 Done (Boolen): 表示是否需要将环境重置 env.reset。大多数情况下，当 Done 为 True 时，就表明当前回合(episode)或者试验(tial)结束。例如当机器人摔倒或者掉出台面，就应当终止当前回合进行重置(reset);

◉信息 Info (Dict): 针对调试过程的诊断信息。在标准的智体仿真评估当中不会使用到这个 info，具体用到的时候再说。

总结来说，它形成了一个"agent-environment loop"，在每个时间点上，智能体（可以认为是你写的算法）选择一个动作（action），环境返回上一次 action 的观测（Observation）和奖励（Reward），用图表示为

动作（action）：左移（0）；右移（1）

状态变量（state variables）：

x：小车在轨道上的位置（position of the cart on the track）

θ：杆子与竖直方向的夹角（angle of the pole with the vertical）

$\dot{x}$：小车速度（cart velocity）

$\dot{\theta}$：角度变化率（rate of change of the angle）

每次调用 env.reset( )将重新产生一个初始状态。返回值 observation 的四个元素分别表示了小车位置、小车速度、杆子夹角及角变化率。

二、数据集和概念介绍：

机器学习可以分为：

监督学习：给出定义好的标签，程序「学习」标签和数据之间的映射关系
非监督学习：没有标签的数据集
强化学习：达到目标会有正向反馈

本次实验目的是通过机器学习探索优化 open AI gym 的倒立摆问题。对于本次实验的监督学习方法，我使用的是自己准备的数据集 500.csv。该数据集来自于实验一，实验一中实现得分 200 的每一个 episode 的每一步构成了数据集的每一行，总共 1024 行。

三、回归分析：

3.1 标准线性回归：

通过线性回归构造出来的函数一般称之为了线性回归模型。线性回归模型的函数一般写作为：

$$h_\theta(x) = \theta_0 + \theta_1 x$$

通过线性回归算法，我们可能会得到很多的线性回归模型，但是不同的模型对于数据的拟合或者是描述能力是不一样的。我们的目的最终是需要找到一个能够最精确地描述数据之间关系的线性回归模型。这是就需要用到代价函数。代价函数就是用来描述线性回归模型与正式数据之前的差异。如果完全没有差异，则说明此线性回归模型完全描述数据之前的关系。如果需要找到最佳拟合的线性回归模型，就需要使得对应的代价函数最小，相关的公式描述如下：

Hypothesis:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

核心代码如下：

线性回归代码：

```
from sklearn.linear_model import LinearRegression
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
lin_reg.intercept_, lin_reg.coef_
predict = lin_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

Gym 中调用线性回归：

```
import gym
env = gym.make('CartPole-v0') # 环境导入
env.reset()
act = 0
total = 0
for i in range(1000):
    env.render()
    observation, reward, done, info = env.step(act) # 随便动一动~~~
    #print(i,":","action",act,"reward",reward)
    act_pred = []
    act_pred.append(observation)
    #print(regressor.predict([[0.02,-0.03,0.03,0.01]]))
    act = lin_reg.predict(act_pred)[0]
    if act>=0.5:
        act = 1
    else:
        act = 0
    total = total + reward
    if done == 1:
        print("stop after",i+1,"steps\n","the total reward is",total)
        break

env.close()
```

结果显示：

```
测试集准确率：95.967742%
stop after 200 steps
 the total reward is 200.0
```

3.2 多项式回归：

线性回归的局限性是只能应用于存在线性关系的数据中，但是有可能数据之间是非线性关系，虽然我再次尝试多项式回归。

核心代码：

第 1 步：导入重要的库和我们用于执行多项式回归的数据集。

```
from sklearn.preprocessing import PolynomialFeatures
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
```

第 2 步：将数据集划分为两个组件，即 X 和 y。X 将包含 1 到倒数第二列之间的列，y 将包含最后列。

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

第 3 步：将多项式回归模型拟合到两个分量 X 和 y 上。

```
poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X_train)
lin_reg = LinearRegression()
```

第 4 步：使用线性和多项式回归预测新结果。

```
X_test = poly_features.fit_transform(X_test)
predict = lin_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：94.354839%
stop after 200 steps
 the total reward is 200.0
```

3.3 正则线性模型

3.3.1 岭回归：

岭回归的代价函数如下：

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - (wx^{(i)} + b))^2 + \lambda||w||_2^2 = MSE(\theta) + \lambda\sum_{i=1}^{n}\theta_i^2$$

为了方便计算导数，通常也写成下面的形式：

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - (wx^{(i)} + b))^2 + \frac{\lambda}{2}||w||_2^2 = \frac{1}{2}MSE(\theta) + \frac{\lambda}{2}\sum_{i=1}^{n}\theta_i^2$$

上式中的 w 是长度为 n 的向量，不包括截距项的系数 θ 0； θ 是长度为 n+1 的向量，包括截距项的系数 θ 0；m 为样本数；n 为特征数。

岭回归的代价函数仍然是一个凸函数，因此可以利用梯度等于 0 的方式求得全局最优解（正规方程）：

$$\theta = (X^TX + \lambda I)^{-1}(X^Ty)$$

上述正规方程与一般线性回归的正规方程相比，多了一项 λ I，其中 I 表示单位矩阵。假如 XTX 是一个奇异矩阵（不满秩），添加这一项后可以保证该项可逆。

核心代码如下：

```
ridge_reg = Ridge(alpha=1, solver="cholesky", random_state=42)
ridge_reg.fit(X_train, y_train)
predict = ridge_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果如下：

```
测试集准确率：93.951613%
stop after 200 steps
 the total reward is 200.0
```

3.3.2 套索回归：

套索回归与岭回归有一点不同，它在惩罚部分使用的是绝对值，而不是平方值。这导致惩罚（即用以约束估计的绝对值之和）值使一些参数估计结果等于零。使用的惩罚值越大，估计值会越趋近于零。这将导致我们要从给定的 n 个变量之外选择变量。

其惩罚函数为：

$$J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

核心代码：

```
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X_train, y_train)
predict = lasso_reg.predict(X_test)
```

结果显示：

```
测试集准确率：49.596774%
stop after 9 steps
 the total reward is 9.0
```

3.3.3 弹性网络：

弹性网络是岭回归和套索回归的中间地带，其正则项是岭回归和套索回归的正则项混合，混合比例通过 r 来控制。当 r 为 0 时等同于岭回归，r 为 1 时等同于套索回归。

其惩罚函数为：

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

核心代码如下：

```
from sklearn.linear_model import ElasticNet
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5, random_state=42)
elastic_net.fit(X_train, y_train)
predict = elastic_net.predict(X_test)
```

结果显示：

```
测试集准确率：68.548387%
stop after 176 steps
 the total reward is 176.0
```

3.4 逻辑回归：

逻辑回归的假设函数为：

$$h_\theta(X) = g(X\theta) = \frac{1}{1+e^{-X\theta}}$$

其中 X 为样本输入，$h_\theta(X)$为模型输出 θ为要求解的模型参数。设 0.5 为临界值，当 $h_\theta(X) > 0.5$时，即 $X\theta > 0$时，y 为 1；当 $h_\theta(X) < 0.5$时，即$X\theta < 0$时，y 为 0。

模型输出值$h_\theta(X)$在[0,1]区间内取值，因此可从概率角度进行解释：$h_\theta(X)$越接近于 0，则分类为 0 的概率越高；$h_\theta(X)$越接近于 1，则分类为 1 的概率越高；$h_\theta(X)$越接近于临界值 0.5，则无法判断，分类准确率会下降。

逻辑回归采用对数似然损失函数：

$$J(Y, P(Y|X)) = -log(P(Y|X))$$

假设样本输出 y 是 0,1 两类，则

$$P(y=1|x;\theta) = h_\theta(x)$$

$$P(y=0|x;\theta) = 1 - h_\theta(x)$$

即：

$$P(y|x;\theta) = [h_\theta(x)]^y[1-h_\theta(x)]^{1-y}$$

则损失函数为：

$$J(\theta) = -\sum_{i=1}^{n} log([h_\theta(x^{(i)})]^{y^{(i)}}[1-h_\theta(x^{(i)})]^{1-y^{(i)}})$$

$$= -\sum_{i=1}^{n}[y^{(i)}log([h_\theta(x^{(i)})]) + (1-y^{(i)})log([1-h_\theta(x^{(i)})])]$$

写成向量形式即：

$$J(\theta) = -Y^T log(h_\theta(X)) - (E-Y)^T log(E - h_\theta(X))$$

其中 E 为全 1 向量。

核心代码：

```
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

log_reg = LogisticRegression(solver="liblinear", random_state=42)
log_reg.fit(X_train, y_train)
predict = log_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

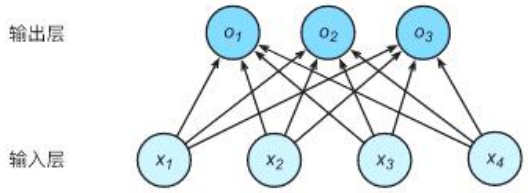
结果显示：

```
测试集准确率：87.500000%
stop after 200 steps
 the total reward is 200.0
```

3.5Softmax 回归：

与线性回归的不同在于 Softmax 回归的输出单元从一个变成了多个，同时引入 Softmax 运算使得输出更加适合离散值的预测和训练；

Softmax 回归模型：



与线性回归相同，都是将输入特征与权重做线性叠加，，其输出层也是一个全连接层。与线性回归的最大不同在于：Softmax 回归的输出值个数等于标签中的类别数。

核心代码：

```
from sklearn.linear_model import LogisticRegression
import pandas as pd
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

softmax_reg = LogisticRegression(multi_class="multinomial",solver="lbfgs", C=10, random_state=42)
softmax_reg.fit(X_train, y_train)
predict = softmax_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

显示结果：

```
测试集准确率：93.951613%
stop after 200 steps
 the total reward is 200.0
```

四、分类与聚类：

4.1 线性 SVM 分类：

在感知器模型中,SVM 算法是在数据中找出一个划分超平面,让尽可能多的数据分布在这个平面的两侧，从而达到分类的效果,但是在实际数据中这个符合我们要求的超平面是可能存在多个的。

SVM 的解决问题的思路是找到离超平面的最近点，通过其约束条件求出最优解。

支持向量满足函数：

$$w^T x+b=\pm 1$$

支持向量点到超平面的距离：

$$\frac{\left|y(w^T x+b)\right|}{\|w\|_2}=\frac{1}{\|w\|_2}$$

我们解题的思路是：让所有分类的点各自在支持向量的两边，同时要求尽量使得支持向量远离超平面，优化问题可以用数学公式可以表示如下：

$$\max_{w,b}\frac{1}{\|w\|_2}$$

$$s.t: y^{(i)}(w^T * x^{(i)}+b)\geq 1, i=1,2,...,m$$

以上优化问题可以转化为：

$$\min_{\omega,b} \frac{1}{2}\|\omega\|_2^2$$
$$s.t: y^{(i)}(\omega^T * x^{(i)}+b) \geq 1, i=1,2,...,m$$

可以转化为求损失函数 J(w）的最小值，如下表示：

$$J(\omega) = \frac{1}{2}\|\omega\|_2^2$$
$$s.t: y^{(i)}(\omega^T * x^{(i)}+b) \geq 1, i=1,2,...,m$$

以上问题可以用 KKT 条件求解：

$$L(\omega, b, \beta) = \frac{1}{2}\|\omega\|_2^2 + \sum_{i=1}^{m} \beta_i[1 - y^{(i)}(\omega^T * x^{(i)}+b)], \beta \geq 0$$

核心代码：

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge", random_state=42)),
    ])

svm_clf.fit(X_train, y_train)
svm_clf.predict(X_test)
predict = svm_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0])        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：95.161290%
stop after 200 steps
 the total reward is 200.0
```

4.2 非线性 SVM 分类：

线性 svm 分类器是有效的，但是现实中不是所有数据集都可以线性分离的。处理非线性数据集的方法之一是添加更多特征。初步尝试搭建一条流水线：一个 PolynomicalFeatures 转换器，接着一个 StandardScaler，然后是 linearSVC。

核心代码：

```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split

dataset = pd.read_csv('C:/Users/ASUS/Desktop/500.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])

polynomial_svm_clf.fit(X_train, y_train)
predict = svm_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：95.161290%
stop after 200 steps
 the total reward is 200.0
```

4.2.1 多项式核：

本次尝试使用一个 3 阶多项式内核训练 SVM 分类器。这个方法在没有实际添加任何特征的前提下，排除了数量爆炸的影响。如果最后模型拟合过度应降低阶数，反过来，拟合不足时则可以尝试提升。

核心代码：

```
from sklearn.svm import SVC

poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
predict = poly_kernel_svm_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：97.177419%
stop after 200 steps
 the total reward is 200.0
```

4.2.2 高斯 RBF 核函数：

尝试使用高斯 RBF 核函数。核函数意义：按一定规律统一改变样本的特征数据得到新的样本，新的样本按新的特征数据能更好的分类，由于新的样本的特征数据与原始样本的特征数据呈一定规律的对应关系，因此根据新的样本的分布及分类情况，得出原始样本的分类情

况。

高斯核本质是在衡量样本和样本之间的“相似度”，在一个刻画“相似度”的空间中，让同类样本更好的聚在一起，进而线性可分。

核心代码：

```
from sklearn.svm import SVC
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
predict = rbf_kernel_svm_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：74.193548%
stop after 87 steps
 the total reward is 87.0
```

4.3SVM 回归：

对于回归模型，优化目标函数和分类模型保持一致，依然是$\min_{w,b} \frac{\|w\|^2}{2}$，但是约束条件不同。我们知道回归模型的目标是让训练集中的每个样本点(x_i, y_i)，尽量拟合到一个线性模型$y_i = w^T x_i + b$上。对于一般的回归模型，我们是用均方误差作为损失函数的，但 SVM 不是这样定义损失函数的。

SVM 回归算法采用$\varepsilon - insensitive$误差函数，$\varepsilon - insensitive$误差函数定义为，如果预测值$\hat{y}_i$与真实值$y_i$之间的差值$|\hat{y}_i - y_i| \leq \epsilon$，则不产生损失，否则，损失代价为$|\hat{y}_i - y_i| - \epsilon$。

总结一下，SVM 回归模型的损失函数度量为：

$$\hat{y}_i = w^T x_i + b$$

$$E_\epsilon(\hat{y}_i - y_i) = \begin{cases} 0, & if\ |\hat{y}_i - y_i| < \epsilon; \\ |\hat{y}_i - y_i| - \epsilon, & otherwise \end{cases}$$

核心代码：

```
from sklearn.svm import SVR

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1, gamma="auto")
svm_poly_reg.fit(X, y)
predict = svm_poly_reg.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：52.419355%
stop after 32 steps
 the total reward is 32.0
```

五、集成学习和随机森林：

5.1bagging 和 pasting：

获得多样分类器的一种方法是使用不同的算法，另一种方法是使用相同的算法，但是在训练数据的不同子集上进行训练。如果针对训练数据的抽样是有放回的抽样，那么这样的方法是 bagging（bootstrap aggregation 的缩写），如果针对训练数据的抽样是没有放回的抽样，那么这样的方法是 pasting。换句话说，bagging 和 pasting 都允许特定的训练数据多次出现在不同的训练过程，但是只有 bagging 允许特定的训练数据多次出现在相同的训练过程。

当所有的模型训练完成后，最后的结果通过最多模型输出的结果（分类问题）或模型输出结果的平均值（回归问题）得到。相比于使用完整的训练数据进行训练，使用部分训练数据训练得到的模型具有较高的偏差，但是，集成学习可以同时降低偏差和方差。一般来说，相比单独的模型在完整的训练数据上得到的结果，集成学习得到的结果具有类似的偏差和较低的方差。

scikit-learn 通过 BaggingClassifier（分类问题）/BaggingRegressor（回归问题）实现 bagging 和 pasting（默认实现 bagging，实现 pasting 需要设置 bootstrap=False）。

核心代码：

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
predict = bag_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：81.854839%
stop after 200 steps
 the total reward is 200.0
```

5.2 随机森林：

随机森林是在 bagging 算法的基础之上加了一点小小的改动演化过来的。我们知道 bagging 算法是在原始的数据集上采用有放回的随机取样的方式来抽取 m 个子样本，从而利用这 m 个子样本训练 m 个基学习器，从而降低了模型的方差。而我们的随机森林的改动有两处，第一：不仅随机的从原始数据集中随机的抽取 m 个子样本，而且在训练每个基学习器的时候，不是从所有特征中选择最优特征来进行节点的切分，而是随机的选取 k 个特征，从这 k 个特征中选择最优特征来切分节点，从而更进一步的降低了模型的方差；第二：随机

森林使用的基学习器是 CART 决策树。

随机森林随机选择的样本子集大小 m 越小模型的方差就会越小，但是偏差会越大，所以在实际应用中，我们一般会通过交叉验证的方式来调参，从而获取一个合适的样本子集的大小。所以随机森林除了基学习器使用 CART 决策树和特征的随机选择以外，其他方面与 bagging 方法没有什么不同。

核心代码：

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

predict = rnd_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：79.838710%
stop after 200 steps
 the total reward is 200.0
```

5.3 提升法：

提升方法基于这样一种思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断好。

对于分类问题而言，给定一个训练样本集，求比较粗糙的分类规则（弱分类器）要比求精确的分类规则（强分类器）容易得多。提升方法就是从弱学习算法出发，反复学习，得到一系列弱分类器（又称为基本分类器），然后组合这些弱分类器，构成一个强分类器。大多数的提升方法都是改变训练数据的概率分布（训练数据的权值分布），针对不同的训练数据分布调用弱学习算法学习一系列弱分类器。

核心代码：

```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=200,
    algorithm="SAMME.R", learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
predict = ada_clf.predict(X_test)
right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))        #计算在测试集上的准确度
```

结果显示：

```
测试集准确率：88.306452%
stop after 200 steps
 the total reward is 200.0
```

5.4 梯度提升：

GBDT 属于 boosting 算法，也是迭代，使用了前向分布算法，但是弱学习器限定了只能使用 CART 回归树模型，同时迭代思路和 Adaboost 也有所不同。一是效果确实挺不错。二是

即可以用于分类也可以用于回归。三是可以筛选特征。

核心代码：

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
predict = gbrt.predict(X_test)
right=0
for i in range(len(predict)):
    if predict[i] - y_test[i] < 0.5 and predict[i] - y_test[i] > -0.5:
        right=right+1
#right = sum(predict == y_test)
print ('测试集准确率：%f%%'%(right*100.0/predict.shape[0]))       #计算在测试集上的准确度
```

结果显示：

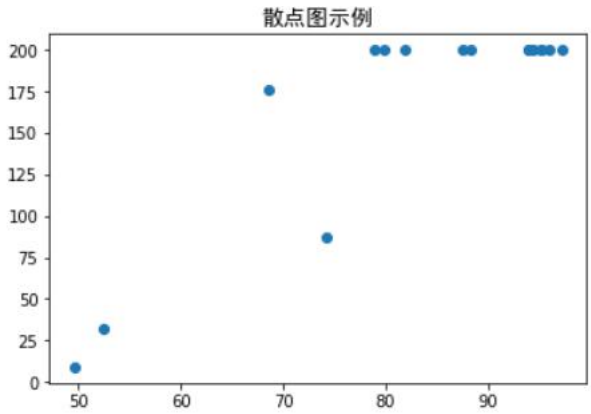
```
测试集准确率：78.629032%
stop after 200 steps
 the total reward is 200.0
```

六、总结：

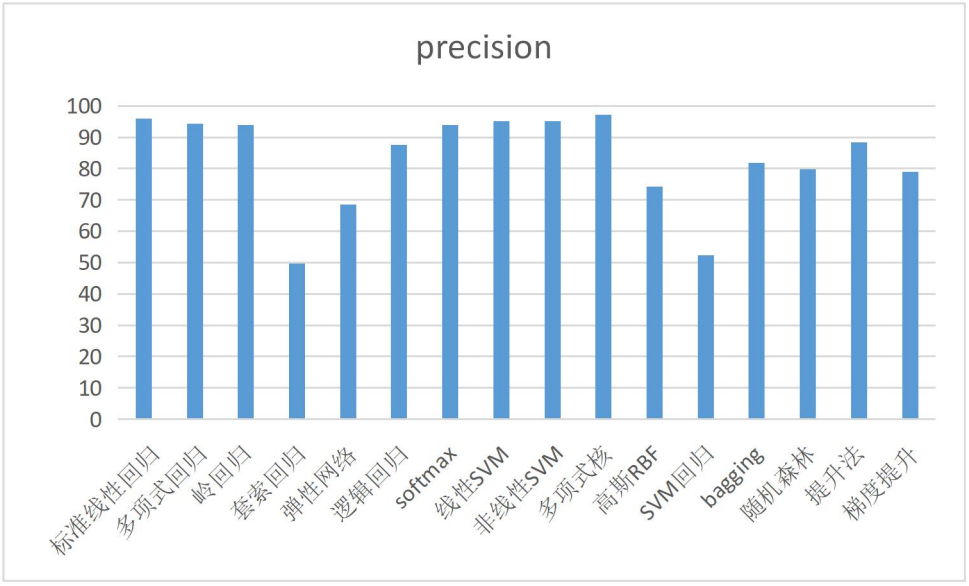
目前本实验所有尝试结果总结如下：

方法	Precision（%）	reward
标准线性回归	95.97	200
多项式回归	94.35	200
岭回归	93.95	200
套索回归	49.6	9
弹性网络	68.55	176
逻辑回归	87.5	200
softmax	93.95	200
线性 SVM	95.16	200
非线性 SVM	95.16	200
多项式核	97.18	200
高斯 RBF	74.19	87
SVM 回归	52.42	32
bagging	81.85	200
随机森林	79.84	200
提升法	88.31	200
梯度提升	78.93	200

散点图如下：



精确度的柱状图如下：



得分的柱状图如下：

可以看出，非线性 SVM+多项式核是目前最好的方法。准确度可达 97%，得分 200。