EDU

CSDN学院　IT实战派

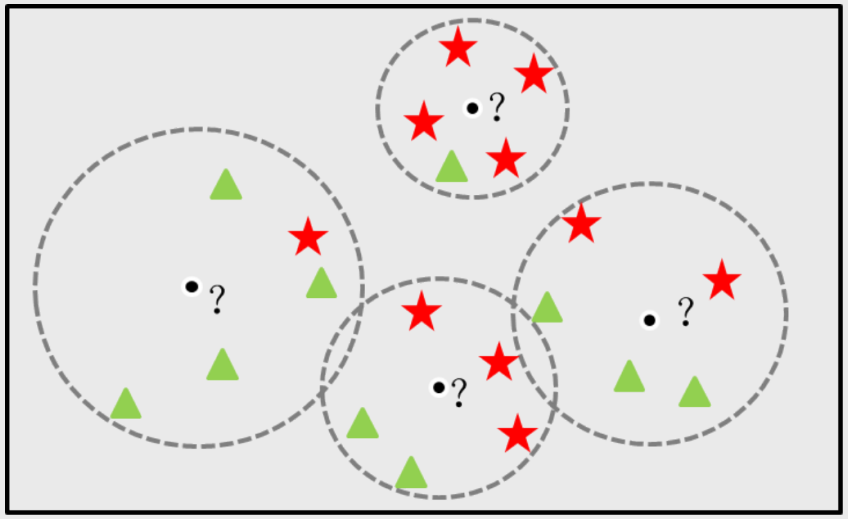
K近邻模型

讲师：刘顺祥

课程目标

1. 理解K近邻模型的思想
2. 掌握K近邻模型中最佳K值的选择
3. 了解几种常用的距离度量方法
4. K近邻模型的应用实战

模型思想



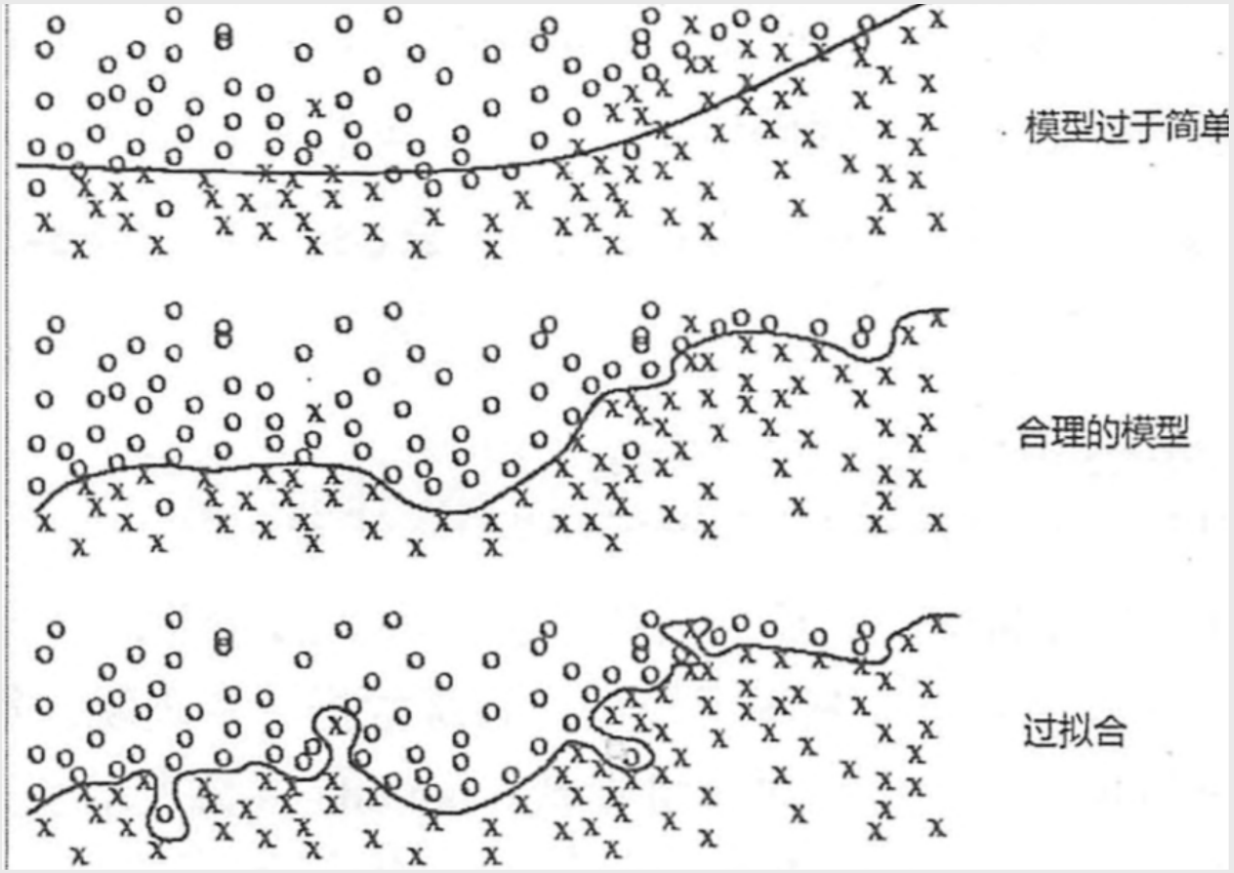
如图所示，模型的本质就是寻找k个最近样本，然后基于最近样本做“预测”。对于离散型的因变量来说，从k个最近的已知类别样本中挑选出频率最高的类别用于未知样本的判断；对于连续型的因变量来说，则是将k个最近的已知样本均值用作未知样本的预测。

模型执行步骤

- 确定未知样本近邻的个数k值。
- 根据某种度量样本间相似度的指标（如欧氏距离）将每一个未知类别样本的最近k个已知样本搜寻出来，形成一个个簇。
- 对搜寻出来的已知样本进行投票，将各簇下类别最多的分类用作未知样本点的预测。

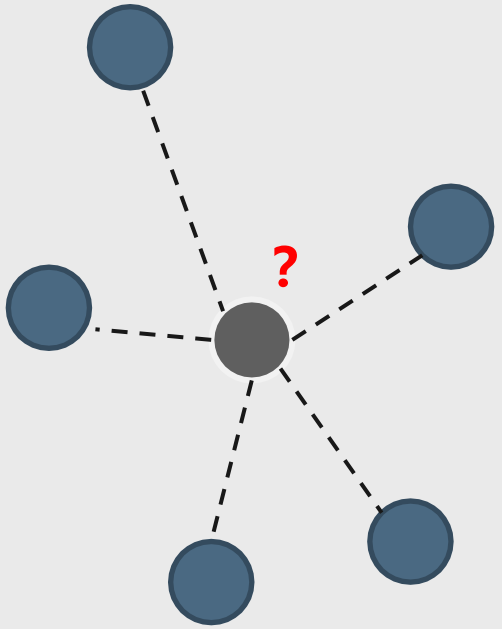
K值的选择

根据经验发现，不同的k值对模型的预测准确性会有比较大的影响，如果k值过于偏小，可能会导致模型的过拟合；反之，又可能会使模型进入欠拟合状态。



K值的选择

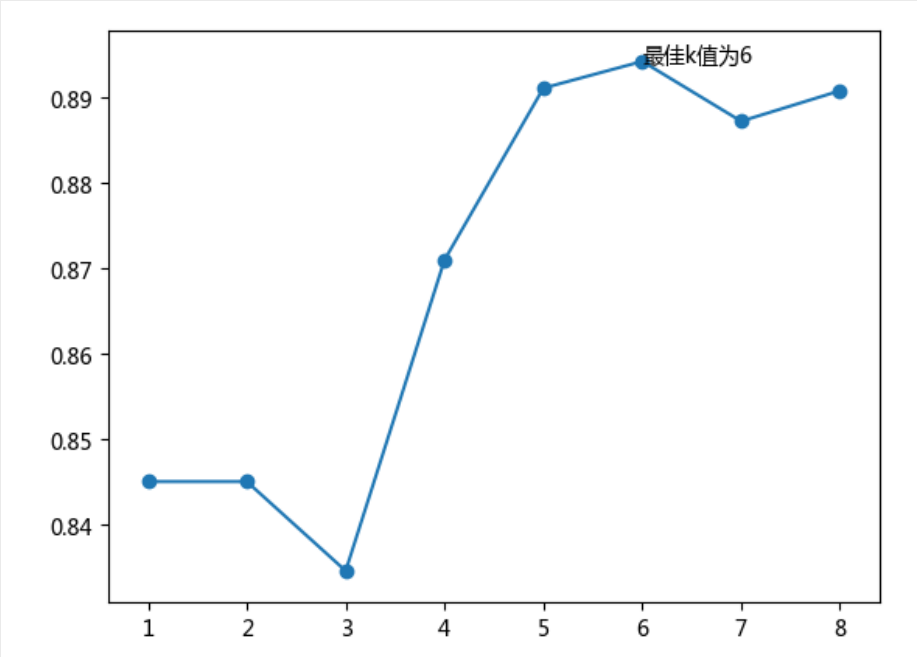
一种是设置k近邻样本的投票权重，使用KNN算法进行分类或预测时设置的k值比较大，担心模型发生欠拟合的现象，一个简单有效的处理办法就是设置近邻样本的投票权重，如果已知样本距离未知样本比较远，则对应的权重就设置得低一些，否则权重就高一些，通常可以将权重设置为距离的倒数。



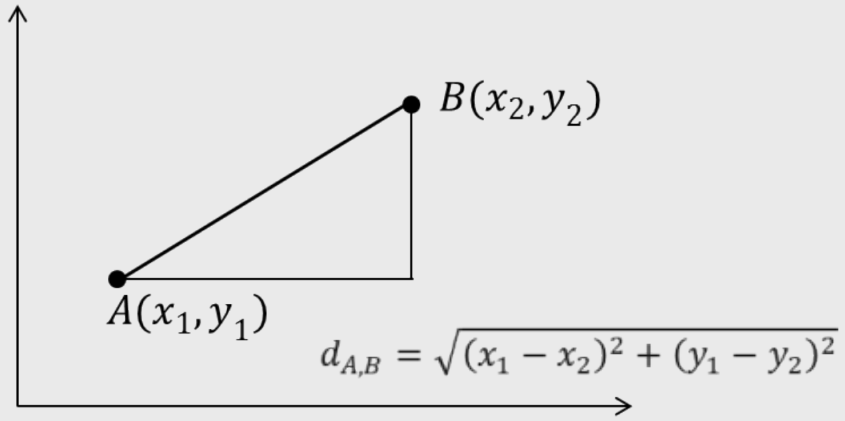
K值的选择

采用多重交叉验证法，该方法是目前比较流行的方案，其核心就是将k取不同的值，然后在每种值下执行m重的交叉验证，最后选出平均误差最小的k值。

当然，还可以将两种方法的优点相结合，选出理想的k值。

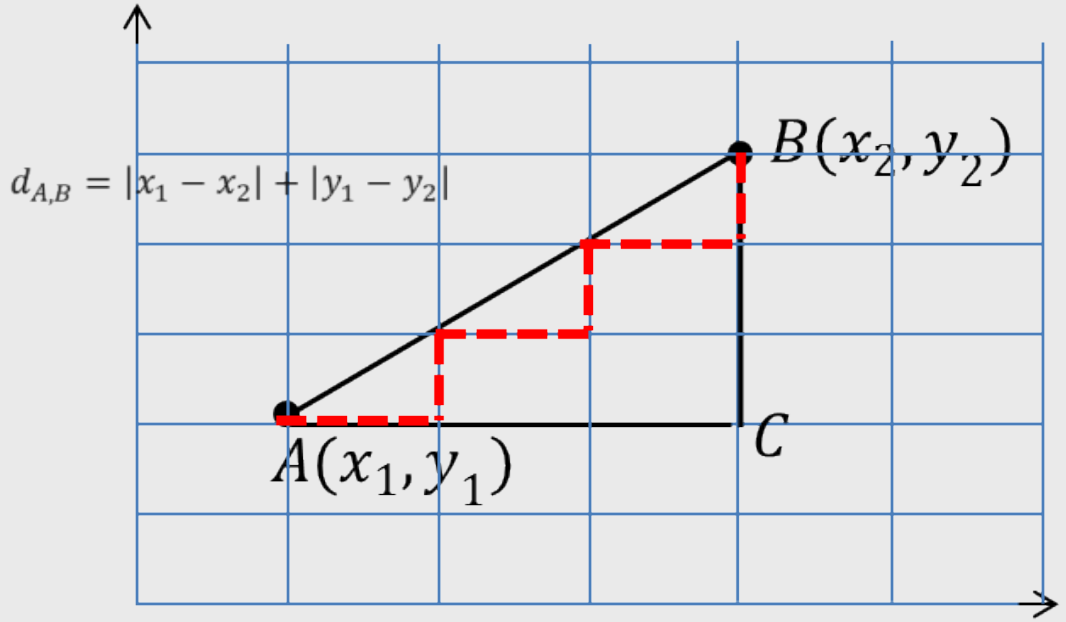


欧式距离



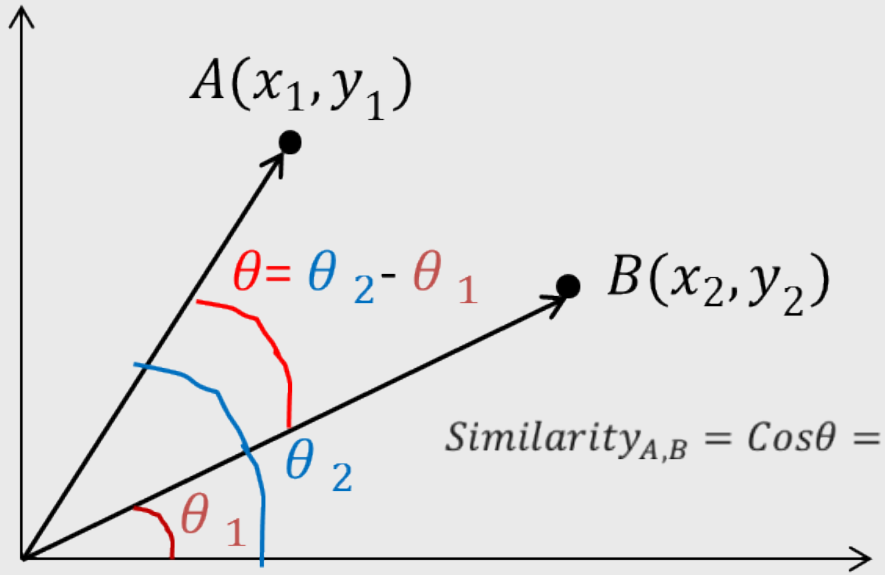
$$d_{A,B} = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_n - x_n)^2}$$

曼哈顿距离



$$d_{A,B} = |y_1 - x_1| + |y_2 - x_2| + \cdots + |y_n - x_n|$$

余弦相似度



$$Similarity_{A,B} = Cos\theta = \frac{x_1x_2 + y_1y_2}{\sqrt{x_1^2 + y_1^2}\sqrt{x_2^2 + y_2^2}}$$

$$Similarity_{A,B} = Cos\theta = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

K近邻模型

函数说明

```
neighbors.KNeighborsClassifier(n_neighbors=5, weights='uniform',
                               p=2, metric='minkowski',)
```

n_neighbors：用于指定近邻样本个数K，默认为5

weights：用于指定近邻样本的投票权重，默认为'uniform'，表示所有近邻样本的投票权重一样；如果为'distance'，则表示投票权重与距离成反比，即近邻样本与未知类别的样本点距离越远，权重越小，反之，权重越大

metric：用于指定距离的度量指标，默认为闵可夫斯基距离

p：当参数metric为闵可夫斯基距离时，p=1，表示计算点之间的曼哈顿距离；p=2，表示计算点之间的欧氏距离；该参数的默认值为2

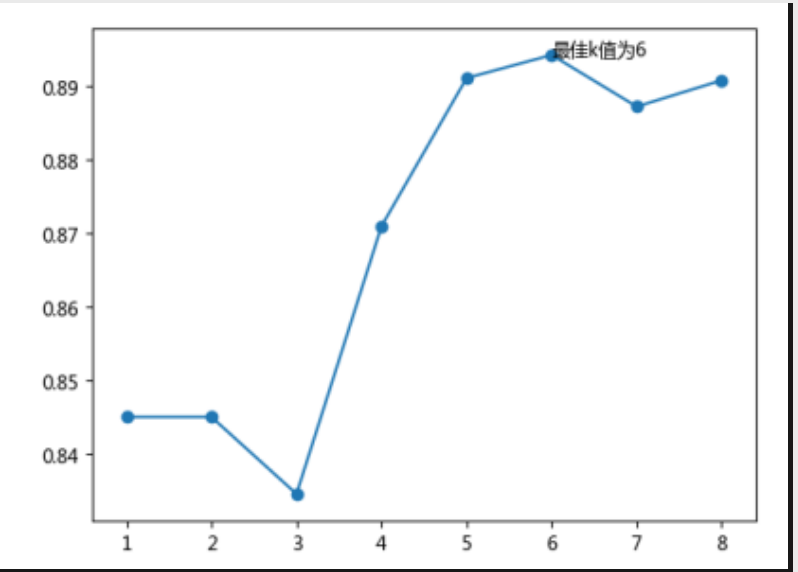
代码演示

```
# 导入数据
Knowledge = pd.read_excel(r'C:\Users\Administrator\Desktop\Knowledge.xlsx’ )
# 拆分数据
X_train, X_test, y_train, y_test = model_selection.train_test_split(Knowledge[predictors],Knowledge.UNS,
                                                                    test_size = 0.25, random_state =
1234)
# 设置待测试的不同k值
K = np.arange(1,np.ceil(np.log2(Knowledge.shape[0])))

# 构建空的列表，用于存储平均准确率
accuracy = []
for k in K:
    # 使用10重交叉验证的方法，比对每一个k值下KNN模型的预测准确率
    cv_result = model_selection.cross_val_score(neighbors.KNeighborsClassifier(n_neighbors = k,
                                                                                weights = 'distance’ ),
                                                X_train, y_train, cv = 10, scoring='accuracy')
    accuracy.append(cv_result.mean())
```

代码演示

```
# 从k个平均准确率中挑选出最大值所对应的下标
arg_max = np.array(accuracy).argmax()
# 中文和负号的正常显示
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
# 绘制不同K值与平均预测准确率之间的折线图
plt.plot(K, accuracy)
# 添加点图
plt.scatter(K, accuracy)
# 添加文字说明
plt.text(K[arg_max], accuracy[arg_max], '最佳k值为%s' %int(K[arg_max]))
# 显示图形
plt.show()
```



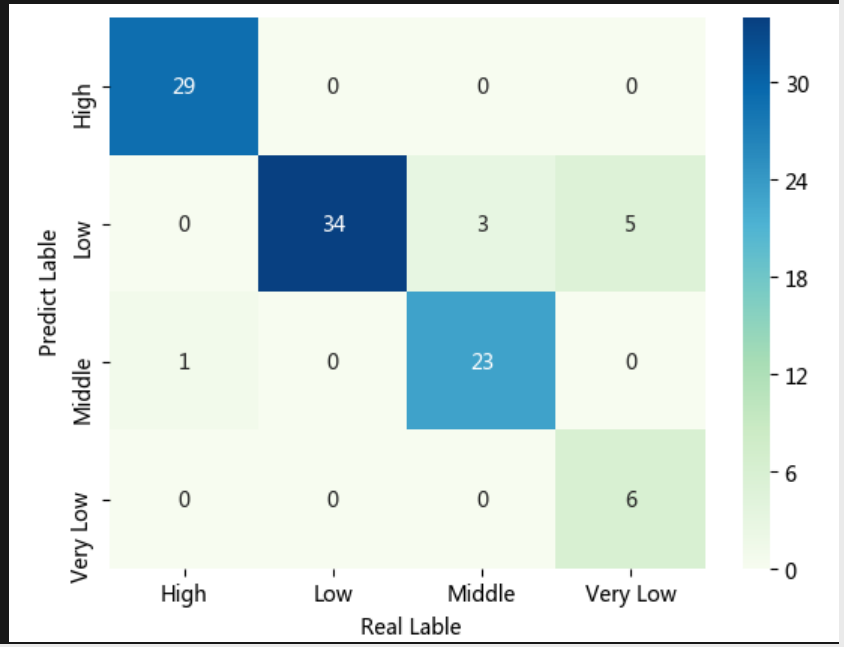
代码演示

```
# 重新构建模型，并将最佳的近邻个数设置为6
knn_class = neighbors.KNeighborsClassifier(n_neighbors = 6, weights = 'distance')
# 模型拟合
knn_class.fit(X_train, y_train)
# 模型在测试数据集上的预测
predict = knn_class.predict(X_test)
# 构建混淆矩阵
cm = pd.crosstab(predict,y_test)
```

	High	Low	Middle	Very Low
High	29	0	0	0
Low	0	34	3	5
Middle	1	0	23	0
Very Low	0	0	0	6

代码演示

```
# 将混淆矩阵构造成数据框，并加上字段名和行名称，用于行或列的含义说明
cm = pd.DataFrame(cm, columns = ['High','Low','Middle','Very Low'],
                      index = ['High','Low','Middle','Very Low'])
# 绘制热力图
sns.heatmap(cm, annot = True,cmap = 'GnBu')
# 添加x轴和y轴的标签
plt.xlabel(' Real Lable')
plt.ylabel(' Predict Lable')
# 图形显示
plt.show()
```



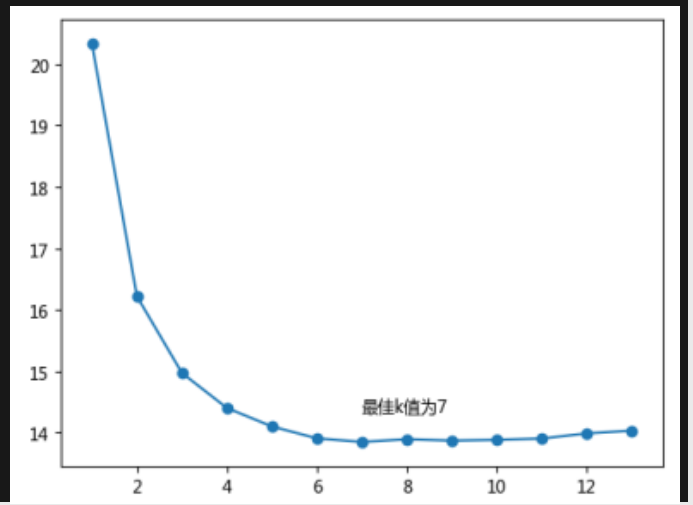
代码演示

```
# 读入数据
ccpp = pd.read_excel(r'C:\Users\Administrator\Desktop\CCPP.xlsx')
# 对所有自变量数据做标准化处理
predictors = ccpp.columns[:-1]
X = minmax_scale(ccpp[predictors])

# 设置待测试的不同k值
K = np.arange(1,np.ceil(np.log2(ccpp.shape[0])))
# 构建空的列表，用于存储平均MSE
mse = []
for k in K:
    # 使用10重交叉验证的方法，比对每一个k值下KNN模型的计算MSE
    cv_result = model_selection.cross_val_score(neighbors.KNeighborsRegressor(n_neighbors = k,
                                                                               weights = 'distance’ ),
                                                X_train, y_train, cv = 10, scoring='neg_mean_squared_error')
    mse.append((-1*cv_result).mean())
```

代码演示

```
# 从k个平均MSE中挑选出最小值所对应的下标
arg_min = np.array(mse).argmin()
# 绘制不同K值与平均MSE之间的折线图
plt.plot(K, mse)
# 添加点图
plt.scatter(K, mse)
# 添加文字说明
plt.text(K[arg_min], mse[arg_min] + 0.5, '最佳k值为%s' %int(K[arg_min]))
# 显示图形
plt.show()
```



代码演示

```
# 重新构建模型，并将最佳的近邻个数设置为7
knn_reg = neighbors.KNeighborsRegressor(n_neighbors = 7, weights = 'distance')
# 模型拟合
knn_reg.fit(X_train, y_train)
# 模型在测试集上的预测
predict = knn_reg.predict(X_test)
# 计算MSE值
metrics.mean_squared_error(y_test, predict)

out:
12.814094947334913
```

代码演示

```
# 对比真实值和实际值（前10组数据）
pd.DataFrame({'Real':y_test,'Predict':predict}, columns=['Real','Predict']).head(10)
```

实际值	435.68	442.90	449.01	449.75	455.20	453.49	479.14	446.71	429.80	474.40
预测值	437.68	443.10	448.76	445.56	453.01	455.46	476.54	445.57	430.82	474.40

EDU

CSDN学院　IT实战派