机器学习导论

# 编程大作业

姓名：陶威

班级：电信1604班

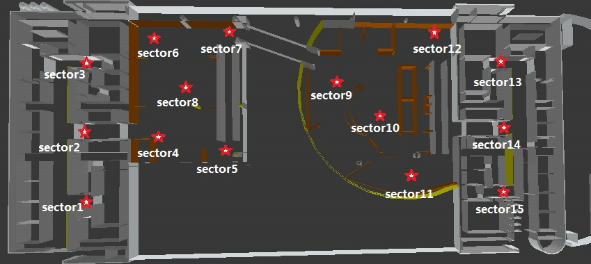
学号：U201613331

指导老师：王邦

## 实验题目：

1. **布局图**

下图室内部署了15个sector，每个sector的具体位置如图中五角星所示。每个sector能够发射2.1G频段和3.5G频段信号。



1. **数据集**

给出的数据包括训练集和测试集，这些数据均是该区域中的仿真数据集。

**训练集：**存于dataAll.csv,共包括3863个样本，每个样本所在的点间间隔为0.6m。每个样本中，前30列表示15个sector在两个频段的信号强度，如21005表示第6个sector中发射的2.1G频段信号，350011表示第12个sector中发射的3.5G频段信号。列‘x’和‘y’表示该样本所在点的坐标位置。

**测试集：**存于testAll.csv,共包括497个样本。所有列表示15个sector在两个频段的信号强度，其标签含义与dataAll中相同。

在给定的数据集中，-126.23dBm代表该sector的该频段的无线信号无法传播到该点。

1. **评估标准**

采用尽量少的sector，在满足覆盖率>95%的要求下，实现高准确度定位。

**覆盖率：**在一个点的样本中，如果存在至少一个sector的任意频段的信号强度大于-105dBm，则表示该点被覆盖。覆盖率则定义为，被覆盖的点占所有点的比例。

**定位准确度：**设测试样本实际坐标为(x\_n,y\_n),评估坐标为(x\_ne,y\_ne)，则平均定位误差=，平均定位误差越小，定位准确度越高。

## 准备事项——工具函数

1. 计算平均定位误差，根据题目要求中的误差计算公式来实现的

*def* calc\_aver\_error(list1, list2) -> float: *assert* len(list1) == len(list2)  
 num = len(list1)  
 error\_sum = 0  
 *for* i *in* range(num):  
 error\_sum += np.linalg.norm(np.array(list1[i]) - np.array(list2[i]))  
 *return* error\_sum / num

1. 计算覆盖率

1） 计算每个坐标点被多少个sector覆盖

*def* calc\_coverage\_num(matrix):nums = []  
 *for* line *in* matrix:  
 num = 0  
 *for* i *in* range(len(line) // 2):  
 *if* is\_covered(line[i]) *or* is\_covered(line[i + len(line) // 2]):  
 num += 1  
 nums.append(num)  
 *return* nums

2） 计算样本覆盖率，使用python内置reduce函数迭代计算

*def* calc\_coverage\_rate(arr): *return* reduce(*lambda* acc, num: acc + 1 *if* num > 0 *else* acc, calc\_coverage\_num(arr), 0) / len(arr)

1. 针对某种算法采用留出法计算得分，用于后续评估不同算法的准确度

*def* calc\_score(X, y, estimator):  
 *"""  
 采用train\_test\_split计算得分，越小越好  
 """  
 from* sklearn.model\_selection *import* train\_test\_split  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)  
 estimator.fit(X\_train, y\_train)  
 y\_predict = estimator.predict(X\_test)  
 *return* calc\_aver\_error(y\_test, y\_predict)

## 数据处理——预处理

**思路一：**

原始数据本身的维度较高，同时又含有两种频段的数据（2.1G和3.5G），所以想办法去掉一种频段的数据。可以采取合并和分割的办法。

**合并：**计算了样本数据中每个sector在两种频段的信号强度的平均值后，发现其差别都不大，所以考虑对两种频段的信号强度取平均值来达到降维的目的，合并代码如下：

*def* merge(X):  
 *"""  
 合并两种频率的信号强度，采用求平均值的方法  
 """* length = len(X)  
 matrix = [*None*] \* length  
 *for* index, line *in* enumerate(X):  
 new\_len = len(line) // 2  
 new\_line = [*None*] \* new\_len  
 *for* i *in* range(new\_len):  
 new\_line[i] = (line[i] + line[i + new\_len]) / 2  
 matrix[index] = new\_line  
 *return* matrix

**分割：**另一种思路则是将每一条数据一分为二，只保留一个频段的值，达到增加样本数量而减少数据维度的目的，分割代码如下：

*def* split(X):  
 *"""  
 分开两种频率的信号强度，分开为两个样本  
 """* length = len(X)  
 matrix = [*None*] \* (length \* 2)  
 *for* index, line *in* enumerate(X):  
 half\_len = len(line) // 2  
 matrix[index] = line[0:half\_len]  
 matrix[index + length] = line[half\_len:]  
 *return* matrix

**结论：**对思路一的想法采用控制变量法测试，测试代码如下，测试结果如图，发现这两种算法的效果都比原来有所下降，故舍弃。

测试代码：

*def* data\_processing(X, y, estimator):  
 *"""  
 对合并和分割进行测试  
 """* scores = [  
 calc\_score(X, y, estimator),  
 calc\_score(merge(X), y, estimator),  
 calc\_score(split(X), y \* 2, estimator)  
 ]  
 methods = ['raw', 'merge', 'split']  
 plt.scatter(methods, scores)  
 plt.xlabel('methods')  
 plt.ylabel('scores')  
 plt.show()

**测试结果：**



**思路二：**

类比决策树一章编程作业的题目思路，将RSS信号量二值化，以达到将回归问题转化为分类问题的目的，同时设置了不同的阈值（来源于题目中的某些特殊值）来二值化RSS，对比其误差。

**测试代码：**

*def* binarize(X, y, estimator):  
 *from* sklearn *import* preprocessing  
 scores = []  
 thresholds = []  
 scores.append(calc\_score(X, y, estimator))  
 thresholds.append('None')  
 *for* threshold *in* [-105, -126.23]:  
 binarizer = preprocessing.Binarizer(threshold=threshold)  
 scores.append(calc\_score(binarizer.transform(X), y, estimator))  
 thresholds.append(str(threshold))  
 plt.scatter(thresholds, scores)  
 plt.xlabel('thresholds')  
 plt.ylabel('scores')  
 plt.show()

**测试结果：**



**结论：**因二值化过程中相当于丢失了部分特征，故最后误差有所增大，所以仍不可行。

## 数据处理——sector选取

根据题目要求，需要在满足覆盖率>95%的要求下，采用尽量少的sector，实现高准确度定位。

**思路一：**

采用最直观的思路，即先计算出每个sector的覆盖率，再从覆盖率最小的sector中逐个删除，直到不满足整体覆盖率>95%的情况发生。

**测试代码：**

*def* get\_min\_sector(matrix):  
 *"""  
 计算能保持覆盖率 > 95%的最少sector个数  
 """* sector\_num = len(matrix[0]) // 2  
 *while* sector\_num > 0:  
 sector\_rate = calc\_sector\_coverage\_rate(matrix)  
 sorted\_sector\_rate = np.argsort(sector\_rate)  
 index = sorted\_sector\_rate[0]  
 print(index)  
 matrix = np.delete(matrix, [index, index + sector\_num], axis=1)  
 print(matrix[0])  
 sector\_num -= 1  
 print(calc\_coverage\_rate(matrix))  
 *if* calc\_coverage\_rate(matrix) <= 0.95:  
 *return* sector\_num + 1

**结论：**这种思路看似合理，实测效果并不好，因为覆盖率小的sector可能刚好覆盖了其它sector没有覆盖到的地方，故此思路被舍弃。

**思路二：**

采用递归回溯+深度优先遍历，遍历所有可能的情况，以选取符合题意的结果。

**测试代码：**

*def* dfs(raw\_matrix, matrix, last, indexes, res):  
 *if* calc\_coverage\_rate(matrix) <= 0.95:  
 indexes[last] = 1  
 *return  
 if* sum(res) > sum(indexes):  
 res = indexes[:]  
 print(sum(res))  
 length = len(indexes)  
 *for* i *in* range(length):  
 *if* indexes[i] == 1:  
 indexes[i] = 0  
 delete\_index = np.array(  
 reduce(*lambda* acc, item: acc + [item[0]] *if* item[1] == 0 *else* acc, enumerate(indexes), [])  
 )  
 new\_matrix = np.delete(raw\_matrix, np.append(delete\_index, delete\_index + length), axis=1)  
 dfs(raw\_matrix, new\_matrix, i, indexes, res)  
 indexes[i] = 1

**结论：**此方法效率降低，但准确度最高，最终计算出最多可删除1, 3, 6, 8, 10, 11, 13, 14这8个sector仍可保持95%以上的覆盖率。

## 模型选择与评估

为了测试出最适合的模型，直接尝试这门课上所学到的所有机器学习算法来调参、训练并预测测试集，并且逐一采用留出法验证其准确度，最后再综合所有结果并画图比较，测试代码如下。

**随机森林：**

*from* sklearn.ensemble *import* RandomForestRegressor  
*from* sklearn.model\_selection *import* GridSearchCV  
param\_grid = [  
 {  
 'bootstrap': [*True*, *False*],  
 'n\_estimators': [i *for* i *in* range(10, 110, 10)]  
 }  
]  
*# 网格搜索，后续因为篇幅省略*grid\_search = GridSearchCV(RandomForestRegressor(), param\_grid, cv=5, n\_jobs=-1)  
grid\_search.fit(X\_train, y\_train)  
print(grid\_search.best\_params\_)  
models.append('RandomForest')  
scores.append(calc\_score(X\_train, y\_train, RandomForestRegressor(n\_estimators=100)))

**kNN：**

*from* sklearn.neighbors *import* KNeighborsRegressormodels.append('kNN')  
scores.append(calc\_score(X\_train, y\_train, KNeighborsRegressor(n\_jobs=-1, n\_neighbors=2)))

**梯度提升：**

*from* sklearn.multioutput *import* MultiOutputRegressor  
models.append('GradientBoosting')  
scores.append(  
 calc\_score(X\_train, y\_train, MultiOutputRegressor(GradientBoostingRegressor(n\_estimators=100, max\_depth=10)))  
)

**神经网络多层感知机：**

*from* sklearn.neural\_network *import* MLPRegressor  
models.append('NeuralNetwork')  
scores.append(calc\_score(X\_train, y\_train, MLPRegressor(hidden\_layer\_sizes=(100, 100))))

**SVM**（因其不能预测多维数据，故分别基于x和y坐标进行预测）**：**

*from* sklearn *import* svm  
clf\_x = svm.SVR(C=1000, gamma=0.01)  
clf\_y = svm.SVR(C=1000, gamma=0.01)  
*from* sklearn.model\_selection *import* train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_train, y\_train)  
y\_train = np.array(y\_train)  
clf\_x.fit(X\_train, y\_train[:, 0])  
clf\_y.fit(X\_train, y\_train[:, 1])  
x = clf\_x.predict(X\_test)  
y = clf\_y.predict(X\_test)  
models.append('SVM')  
scores.append(calc\_aver\_error(y\_test, np.column\_stack((x, y))))

**线性回归：**

*from* sklearn.linear\_model *import* LinearRegression  
models.append('LinearRegression')  
scores.append(calc\_score(X\_train, y\_train, LinearRegression()))

**贝叶斯岭回归：**

*from* sklearn.linear\_model *import* BayesianRidge  
*from* sklearn.multioutput *import* MultiOutputRegressor  
models.append('BayesianRidge')  
scores.append(calc\_score(X\_train, y\_train, MultiOutputRegressor(BayesianRidge())))

**根据结果画图展示：**

plt.scatter(models, scores)  
plt.xlabel('models')  
plt.ylabel('scores')  
plt.show()

**图例：**



**结论：**从图例上可以看出，在本题中随机森林，kNN和梯度提升法效果较好，分析其原因可能是：kNN算法与定位这个应用场景比较吻合，从位置上选择最近的点进行定位参考，符合生活常识。微软早在2000年就开始进行WiFi位置指纹法定位的研究工作（Bahl and Padmanabhan 2000），他们采用的就是最近邻算法，可见kNN与生俱来的优越性。不过kNN的缺点是预测时间复杂度和训练集大小成正比。而随机森林则是因为其“随机”的特点，即随机选择特征，可以避免过拟合，因而比较适合处理这种高维的数据集。