# 随机森林算法

## 1. 实验目的

了解随机森林算法的原理，并且可以简单应用

## 2. 算法原理

首先，随机森林(Random Forest,以下简称 RF)使用了 CART 决策树作为弱学习器，这让我们想到了梯度提升树 GBDT。第二，在使用决策树的基础上，RF 对决策树的建立做了改进，对于普通的决策树，我们会在节点上所有的 n 个样本特征中选择一个最优的特征来做决策树的左右子树划分，但是 RF 通过随机选择节点上的一部分样本特征，这个数字小于 n，假设为 nsub，然后在这些随机选择的 nsub 个样本特征中，选择一个最优的特征来做决策树的左右子树划分。这样进一步增强了模型的泛化能力。

如果 nsub=n，则此时 RF 的 CART 决策树和普通的 CART 决策树没有区别。nsub 越小，则模型越健壮，当然此时对于训练集的拟合程度会变差。也就是说 nsub 越小，模型的方差会减小，但是偏倚会增大。在实际案例中，一般会通过交叉验证调参获取一个合适的 nsub 的值。

除了上面两点，RF 和普通的 bagging 算法没有什么不同，下面简单总结下 RF 的算法。

输入为样本集 D=(x,y1),(x2,y2),...(xm,ym)，弱分类器迭代次数 T。输出为最终的强分类器 f(x)

1）对于 t=1,2...,T:

a)对训练集进行第 t 次随机采样，共采集 m 次，得到包含 m 个样本的采样集 Dm

b)用采样集 Dm 训练第 m 个决策树模型 Gm(x)，在训练决策树模型的节点的时候， 在节点上所有的样本特征中选择一部分样本特征， 在这些随机选择的部分样本特征中选择一个最优的特征来做决策树的左右子树划分。

2)如果是分类算法预测，则 T 个弱学习器投出最多票数的类别或者类别之一为最终类别。如果是回归算法，T 个弱学习器得到的回归结果进行算术平均得到的值为最终的模型输出。

## 3. 实验环境

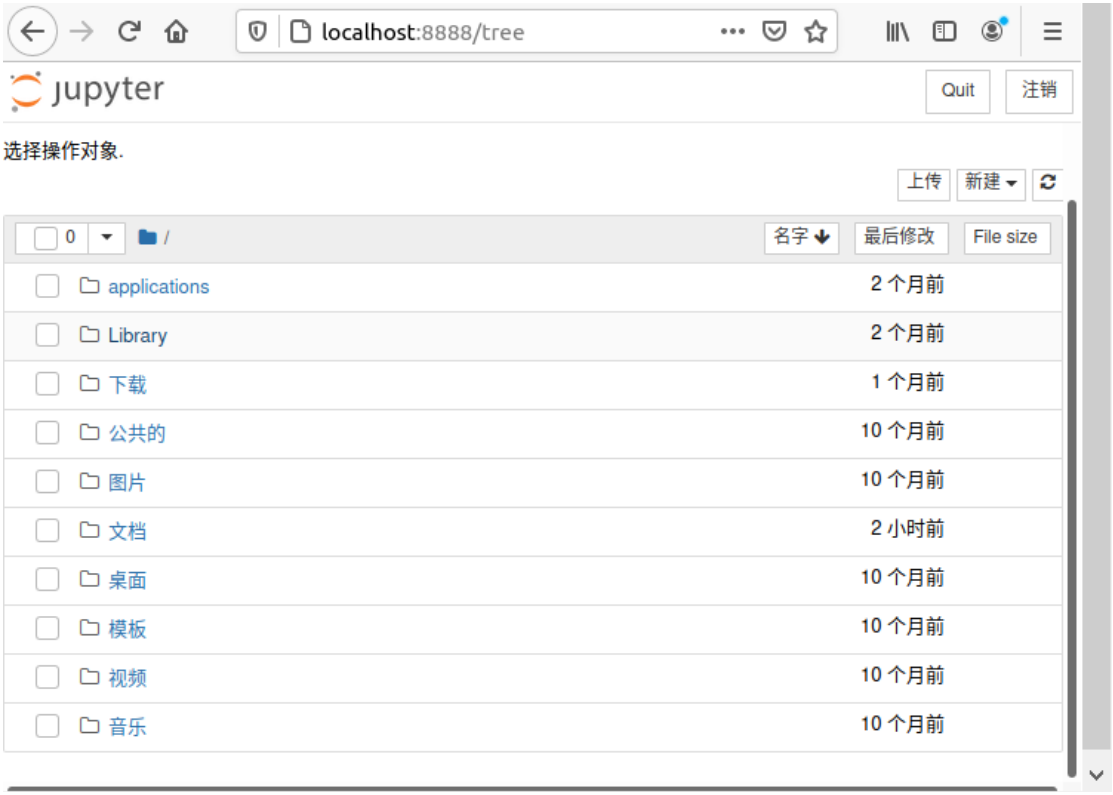
Ubuntu 20.04

Python 3.6

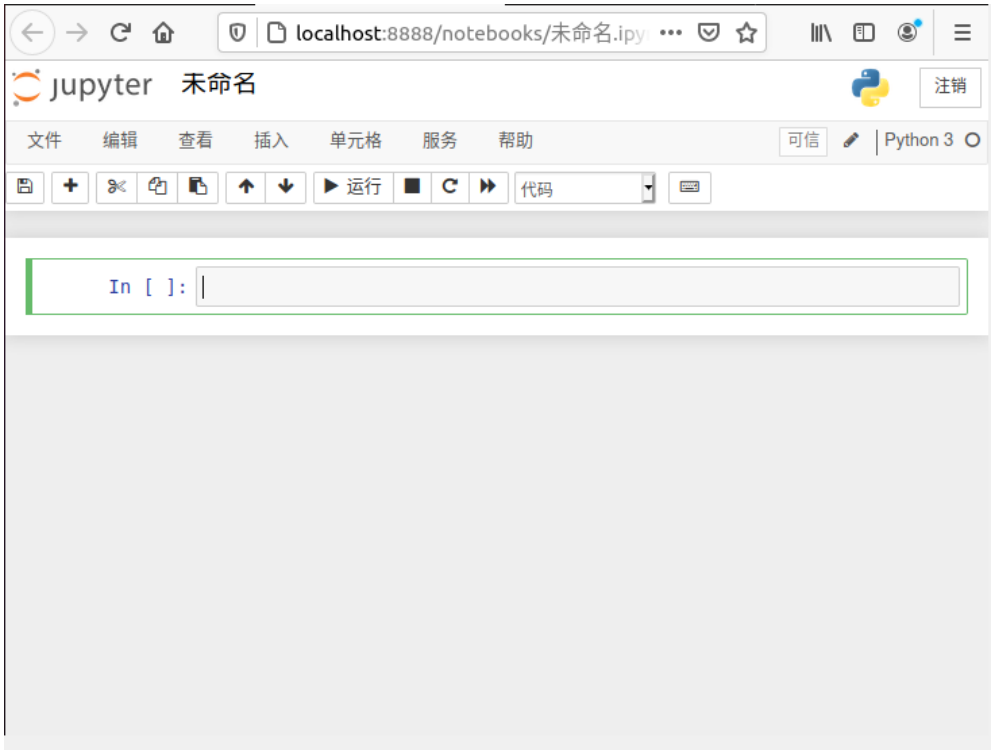
Jupyter notebook

## 4. 实验步骤

1)打开终端，然后输入 jupyter notebook，出现如下界面




2)选定特定文件夹，新建 ipynb 文件，在未命名出可重命名文件




**5. 实操**

Step 1:数据预处理

1. 导入库
2. 导入数据集
3. 数据切分

```
#导入库
import pandas as pd
import numpy as np
import re
import pydot
from sklearn.tree import export_graphviz
from sklearn import metrics
import matplotlib.pylab as plt

%matplotlib inline
from matplotlib.font_manager import FontProperties
font_set = FontProperties()

#导入数据集
df_data = pd.read_excel('D:/jupyter_notebook/ml_full/5.6randomforest/da
ta/实验数据.xlsx', engine='openpyxl')
df_data['自变量 7'] = 0
k = 0
for (index,i) in zip(df_data.index,df_data['时间顺序']):
    if i == 1:
        k += 1
    df_data['自变量 7'][index] = k
    if isinstance(df_data['自变量 4'][index],str):
        x4 = df_data['自变量 4'][index]
        x4 = re.split("°|′",x4)
        x4[-1] = x4[-1][:-1]
        x4_f = float(x4[0]) + float(x4[1])/60 + float(x4[2])/3600
        df_data['自变量 4'][index] = x4_f
    else:
        df_data['自变量 4'][index] = float(df_data['自变量 4'][index])
df_data.drop(['时间顺序'],axis=1,inplace=True,)
df_data['自变量 1'][366] = df_data['自变量 1'][367]
df_data.columns = ['道路纵坡','曲线半径','曲线长度','路线转角','出口线形
', '初始速度','因变量','道路编号']
df_data

labels = np.array(df_data['因变量'],df_data['道路编号'])
features = df_data.drop(['因变量','道路编号'], axis=1)
feature_list = list(features)
features = np.array(features)

# 数据切分
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
```

```
train_features, test_features, train_labels, test_labels = \
train_test_split(features,labels, test_size = 0.15,random_state = 42) #
测试集比例 15%

print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

Step 2:随机森林模型

```
# 建模，训练

# Import the model we are using
from sklearn.ensemble import RandomForestRegressor

# Instantiate model
# n_estimators 森林中树的树量
# max_depth 树的最大深度
rf = RandomForestRegressor(n_estimators=1000, max_depth = 10, random_st
ate=42)
# rf = xgb.XGBRegressor(max_depth=5, learning_rate=0.1, n_estimators=16
0, silent=False)

# Train the model on training data
rf.fit(train_features, train_labels)
```

Step 3:模型测试

```
# Use the forest's predict method on the test data
predictions = rf.predict(test_features)

# Calculate the absolute errors
errors = abs(predictions - test_labels)

# Print out the mean absolute error (mae)
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Calculate mean absolute percentage error (MAPE)
mape = 100 * (errors / test_labels)

# Calculate and display accuracy
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
fig,ax = plt.subplots(figsize=(10,5))
ax.plot(test_labels,'-',label='True',linewidth=2)
ax.plot(predictions,'-', label='Predict',linewidth=2)
plt.legend(loc='upper right')
fig.savefig('Predict.png')
```