School of Data and Computer Science, Sun Yat-sen University

RandomForest——集成学习实现人格分类

17341111 Xuehai Liu

2020 年 6 月 10 日

目录

1 实验内容

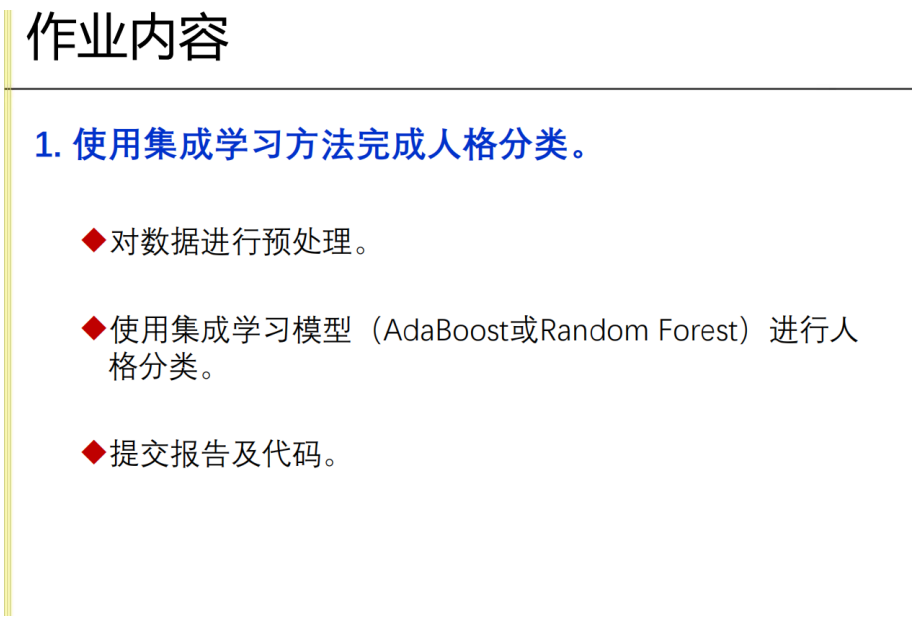

图 1:

本次实验需要使用集成学习的方法根据用户的发言文档记录对其人格进行分类。本文采用的方法是随机森林 (bagging) 的模型，首先对数据进行预处理，然后进行模型搭建，最后进行实验结果的验证和计算 acc、F1 score 以评估模型。

2 数据预处理——去噪与文档向量化

2.1 去噪

首先看到数据的形式为一列 label 分别对应一行用户的文档，其中用户的文档掺杂了比较多的无关信息（如网站等），需要先进行一定的去噪处理。此外，用户发言中的很多词语与模型预测的结果无关，需要去除 (这里包含了停用词以及 think、people 等所有类型的人格都常常会用到的词语)。我使用了 nltk 的停用词集并对其进行扩充。

2.2 文档向量化

得到去噪后的文档后，要将文档放入决策树进行训练首先需要将文档进行特征提取并向量化。向量化采用词袋模型，并在其基础上使用 tfdif 进行文档关键特征的提取，使用 SVD 将庞大的特征降维到 10 维从而加快后续训练决策树的速度。其中，Tf 代表词频（term-frequency），而 tf-idf 代表词频乘以逆向文档频率（inverse document-frequency）。

词袋表征（The Bag of Words representation）是采用将文档文件转化为数值特征的一般过程的相关策略。将文档表示为向量的过程称为向量化。通过将每个词表示为一个整型，统计每个词出现的

频率，可以将一个文档标识为一个向量。由于大多数文档只用到了所有可能用词的一个子集，因此文本向量化产生的矩阵大多为稀疏矩阵。

以下是预处理阶段的全部代码，这个预处理得到的最终数据集是一个 dataframe 形式的 8675*11 的数据集，其中 10 列为之前降维得到的特征，而 11 列为对应的 labels。(labels 已进行离散化)

```
# Data Aggregate
import numpy as np
import pandas as pd
import sklearn
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer
from sklearn.pipeline import make_pipeline

#扩充停用词以进行去噪
stopwords = nltk.corpus.stopwords.words('english')
stopwords.append("think")
stopwords.append("people")
stopwords.append("thing")
stopwords.append("http")
stopwords.append("https")
stopwords.append("youtube")

labeldict = {}

#创建数据集，同时将labels离散化
def createDataset(path):
    data = pd.read_csv(path, header=None)
    data = data.values.tolist()
    labels = []
    dataset = []
    i = 0
    for line in data[1:]:
        if(line[0] not in labeldict.keys()):
            labeldict[line[0]] = i
            labels.append(i)
            i += 1
        else:
            labels.append(labeldict[line[0]])
        dataset.append(line[1])
    return dataset,labels

mydataset, labels = createDataset("mbti_1.csv")
#X = vectorizer.fit_transform(mydataset)
```

```
#使用tfidf进行去噪并提取关键特征
tfidf_vectorizer = TfidfVectorizer(max_features=1000, stop_words=stopwords, use_idf=True
                                                    )
X = tfidf_vectorizer.fit_transform(mydataset)

#使用SVD将特征降维到10维
svd = TruncatedSVD(10)
normalizer = Normalizer(copy=False)
lsa = make_pipeline(svd, normalizer)
mydataset = lsa.fit_transform(X)

#得到最终的dataframe形式的dataset，其最后一列为labels
mydataset = pd.DataFrame(mydataset)
mydataset['labels'] = labels
```

3 Random forest 模型搭建与训练测试

本 section 进行随机森林模型的搭建。

构建随机森林大概分为五个步骤：

- 1、从原始训练集中使用 Bootstraping 方法随机有放回采样选出 m 个样本，进行 n_tree 次采样，生成 n_tree 个训练集
- 2、对于 n_tree 个训练集，我们分别训练 n_tree 个决策树模型
- 3、对于单个决策树模型，假设训练样本特征的个数为 n，那么每次分裂时根据基尼指数选择最好的特征进行分裂
- 4、每棵树都一直这样分裂下去，直到该节点的所有训练样例都属于同一类。在决策树的分裂过程中不需要剪枝
- 5、将生成的多棵决策树组成随机森林。对于分类问题，按多棵树分类器投票决定最终分类结果；对于回归问题，由多棵树预测值的均值决定最终预测结果

本次实验中，我手动实现了决策树模型，并在 randomforest 中实现了随机抽样，每次抽取一个和原样本相同数量，但是只随机选取一部分特征的新样本来进行训练一个决策树。最后，采用 bagging 的思想对各个决策树所得的结果进行投票来决定最终的预测结果。以下是随机森林的模型和训练代码。

```
# coding:utf-8
"""
功能：随机森林，RandomForestClassification，wine数据集[1,2]二分类
"""
from __future__ import division
```

```
import pandas as pd
import copy
import random
import math
from preDeal import mydataset


# 如果最后一个属性还不能将样本完全分开，此时数量最多的label被选为最终类别
def majorClass(classList):
    classDict = {}
    for cls in classList:
        classDict[cls] = classDict.get(cls, 0) + 1
    sortClass = sorted(classDict.items(), key=lambda item: item[1])
    return sortClass[-1][0]


# 计算基尼系数
def calcGini(dataSet):
    labelCounts = {}
    # 给所有可能分类创建字典
    for dt in dataSet:
        currentLabel = dt[-1]
        labelCounts[currentLabel] = labelCounts.get(currentLabel, 0) + 1
    Gini = 1
    for key in labelCounts:
        prob = labelCounts[key] / len(dataSet)
        Gini -= prob * prob
    return Gini


# 对连续变量划分数据集
def splitDataSet(dataSet, featIndex, value):
    leftData, rightData = [], []
    for dt in dataSet:
        if dt[featIndex] <= value:
            leftData.append(dt)
        else:
            rightData.append(dt)
    return leftData, rightData


# 选择最好的数据集划分方式
def chooseBestFeature(dataSet):
    bestGini = 1
    bestFeatureIndex = -1
    bestSplitValue = None
    # 第i个特征
```

```
    for i in range(len(dataSet[0]) - 1):
        featList = [dt[i] for dt in dataSet]
        # 产生候选划分点
        sortfeatList = sorted(list(set(featList)))
        splitList = []
        for j in range(len(sortfeatList) - 1):
            splitList.append((sortfeatList[j] + sortfeatList[j + 1]) / 2)

        # 第j个候选划分点，记录最佳划分点
        for splitValue in splitList:
            newGini = 0
            subDataSet0, subDataSet1 = splitDataSet(dataSet, i, splitValue)
            newGini += len(subDataSet0) / len(dataSet) * calcGini(subDataSet0)
            newGini += len(subDataSet1) / len(dataSet) * calcGini(subDataSet1)
            if newGini < bestGini:
                bestGini = newGini
                bestFeatureIndex = i
                bestSplitValue = splitValue
    return bestFeatureIndex, bestSplitValue


# 去掉第i个属性，生成新的数据集
def splitData(dataSet, featIndex, features, value):
    newFeatures = copy.deepcopy(features)
    newFeatures.remove(features[featIndex])
    leftData, rightData = [], []
    for dt in dataSet:
        temp = []
        temp.extend(dt[:featIndex])
        temp.extend(dt[featIndex + 1:])
        if dt[featIndex] <= value:
            leftData.append(temp)
        else:
            rightData.append(temp)
    return newFeatures, leftData, rightData


# 建立决策树
def createTree(dataSet, features):
    classList = [dt[-1] for dt in dataSet]
    # label一样，全部分到一边
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    # 最后一个特征还不能把所有样本分到一边，则选数量最多的label
    if len(features) == 1:
        return majorClass(classList)
    bestFeatureIndex, bestSplitValue = chooseBestFeature(dataSet)
```

```
    bestFeature = features[bestFeatureIndex]
    # 生成新的去掉bestFeature特征的数据集
    newFeatures, leftData, rightData = splitData(dataSet, bestFeatureIndex, features,
                                                 bestSplitValue)
    # 左右两颗子树，左边小于等于最佳划分点，右边大于最佳划分点
    myTree = {bestFeature: {'<' + str(bestSplitValue): {}, '>' + str(bestSplitValue): {}
                                                 }}
    myTree[bestFeature]['<' + str(bestSplitValue)] = createTree(leftData, newFeatures)
    myTree[bestFeature]['>' + str(bestSplitValue)] = createTree(rightData, newFeatures)
    return myTree


# 用生成的决策树对测试样本进行分类
def treeClassify(decisionTree, featureLabel, testDataSet):
    firstFeature = list(decisionTree.keys())[0]
    secondFeatDict = decisionTree[firstFeature]
    splitValue = float(list(secondFeatDict.keys())[0][1:])
    featureIndex = featureLabel.index(firstFeature)
    if testDataSet[featureIndex] <= splitValue:
        valueOfFeat = secondFeatDict['<' + str(splitValue)]
    else:
        valueOfFeat = secondFeatDict['>' + str(splitValue)]
    if isinstance(valueOfFeat, dict):
        pred_label = treeClassify(valueOfFeat, featureLabel, testDataSet)
    else:
        pred_label = valueOfFeat
    return pred_label


# 随机抽取样本，样本数量与原训练样本集一样，维度为sqrt(m-1)
def baggingDataSet(dataSet):
    n, m = dataSet.shape
    features = random.sample(list(dataSet.columns.values[:-1]), int(math.sqrt(m - 1)))
    features.append(dataSet.columns.values[-1])
    rows = [random.randint(0, n - 1) for _ in range(n)]
    trainData = dataSet.iloc[rows][features]
    return trainData.values.tolist(), features

from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score

def main():
    df = mydataset
    df_train, df_test = train_test_split(df, test_size=0.2)
    labels = df.columns.values.tolist()
    #df = df[df[labels[-1]] != 3]
    # 生成多棵决策树，放到一个list里边
```

```
    treeCounts = 10
    treeList = []
    for i in range(treeCounts):
        baggingData, bagginglabels = baggingDataSet(df_train)
        decisionTree = createTree(baggingData, bagginglabels)
        treeList.append(decisionTree)
        print("finish building a tree")
    # 对测试样本分类
    acc = 0
    y_true = df_test['labels'].values.tolist()
    y_pred = []
    for i in range(df_test.shape[0]):
        testData = df_test.iloc[i].values.tolist()[:-1]
        ans = df_test.iloc[i].values.tolist()[-1]
        labelPred = []
        for tree in treeList:
            label = treeClassify(tree, labels[:-1], testData)
            labelPred.append(label)
        # 投票选择最终类别
        labelDict = {}
        for label in labelPred:
            labelDict[label] = labelDict.get(label, 0) + 1
        sortClass = sorted(labelDict.items(), key=lambda item: item[1])
        y_pred.append(sortClass[-1][0])
        acc += (sortClass[-1][0] == ans)
    acc = acc /df_test.shape[0]
    print("acc: {}".format(acc))
    print("F1 score: {}".format(f1_score(y_true, y_pred, average='micro')))
    print("F1 score: {}".format(f1_score(y_true, y_pred, average='macro')))

main()
```

3.1 训练与预测

分别令 n_tree(即随机森林中决策树的数量) = 1 / 5 / 10，计算预测结果的 acc 与 F1 的整体评价。所得结果如图。图中从上到下显示了 acc、F1 score(average = micro) 、F1 score(average = macro)。

```
D:\python\python.exe "C:/Users/OMEN/Desktop/大学/机器学习/机器学习大作业 文本分类/randomforest.py"
finish building a tree
acc: 0.26628242074927955
F1 score: 0.26628242074927955
F1 score: 0.07205555429054686
D:\python\lib\site-packages\sklearn\metrics\classification.py:1437: UndefinedMetricWarning: F-score is ill-
  'precision', 'predicted', average, warn_for)

Process finished with exit code 0
```

图 2: n_tree = 1

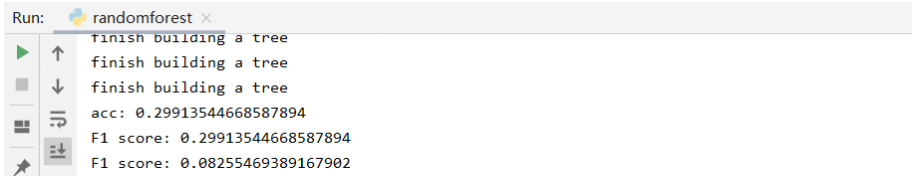

图 3: n_tree = 5

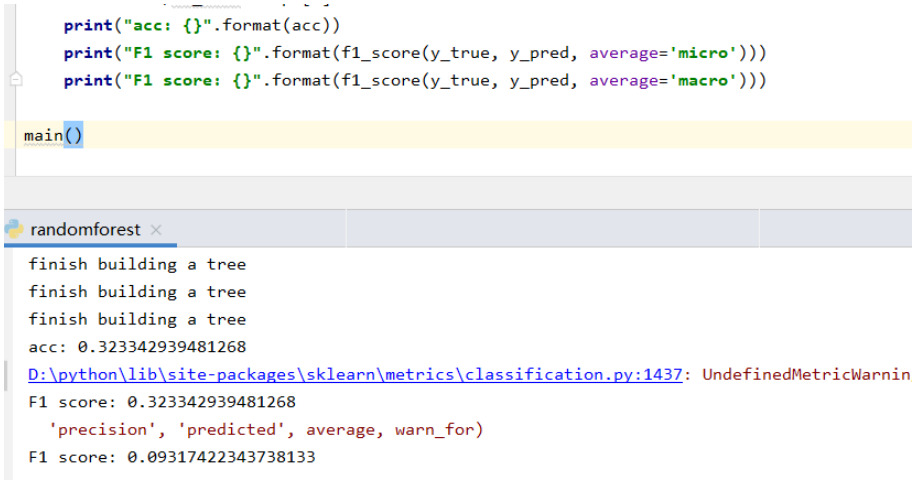

图 4: n_tree = 10

可以看出，随机森林的准确率会随着决策树的数量上升而上升，但是也会随之带来更加复杂的模型结构以及更长的训练时间。此外，我也进行了只使用决策树来进行分类的对比试验，所得 acc = 0.28，可见随机森林的方法在拥有一定数量的决策树之后能够取得比决策树更优的结果.