棕榈学院

7 天 Python 进阶训练营讲义

第四讲

2019 年 1 月 14 日-2019 年 1 月 20 日

目录

在上节课中我们提到了 OneHotEncoder，对变量进行 0 或 1 的编码，也对数据进行了划分，以 8:2 的比例对训练集和训练集进行了划分。并且对逻辑回归、决策树、随机森林这三个逻辑回归模型进行了简单的介绍。这节课进入训练模型这一部分。

一、训练模型

1. 从 sklearn 中调取三个包裹。

```
In [332]: #从sklearn中调用逻辑回归、决策树，随机森林包
          from sklearn.linear_model import LogisticRegression
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.ensemble import RandomForestClassifier
```

可以看到：

逻辑回归模型在 linear_model 中，是线性模型；

决策树是树的模型；

随机森林虽然也是树的模型，但同时也是 ensemble 的模型，ensemble 是指训练了很多个不同的决策树，把结果整合在一起的到了最终的结果。这个理念接下来还会用到。

2. 初始化模型

```
In [333]: #初始化模型，分别存入logr,dtree,rf变量中，并将变量置于名为models的列表中
          logr = LogisticRegression()
          dtree = DecisionTreeClassifier()
          rf = RandomForestClassifier()
          models = [logr,dtree,rf]
```

学院君提示小诀窍：

在使用 Python 的时候如果已经导入，那么在打字时输出前几个字母就可以出现所有的可能性，选取所需要的即可，非常方便～

这三个就是接下来要运用的模型。

将这三个模型放到 models 的列表中，将来可以一起使用。

3. 逻辑回归模型

```
In [334]: #这个函数调用逻辑回归模型来学习训练集和测试集
          logr.fit(xTrain,yTrain)

Out[334]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, max_iter=100, multi_class='warn',
                             n_jobs=None, penalty='l2', random_state=None, solver='warn',
                             tol=0.0001, verbose=0, warm_start=False)
```

直接训练这个模型，用 fit 训练数据。

可以看到逻辑回归中有很多参数，比如 class_weight，是指在两种错误预测的结果

但造成影响不同的情况下，会对 class_weight 这一参数进行调整；random_state 上节课也提到过，是指如果我们定义了一个 random_state 且使用同样数据的话，所有得到的结果都会保持一致；等等。有些参数并不是非常重要，在这里并不作一一展开，接下来会对模型以及重要参数进行更深入的解析。

```
In [335]: #根据自变量xTest,xTrain分别利用训练好的模型预测出因变量
          y_pred_test = logr.predict(xTest)
          y_pred_train = logr.predict(xTrain)
```

这里看两个结果：Train 和 Test 的结果。
和前面不同，前面用 fit 和 transform，是把数据进行了一些变化，这里不一样，只是进行预测，所以 fit 后进行 predict 即可。
用 Text 的 x 预测 Text 的 y，Train 的 x 预测 Train 的 y，得到的维度都是一样的。

```
In [336]: #算出测试集精确度（求均值）
          np.mean(y_pred_test == yTest)
Out[336]: 0.7821229050279329

In [337]: #算出训练集精确度
          np.mean(y_pred_train == yTrain)
Out[337]: 0.824438202247191
```

可以直接看一下结果，做了这么多准备后精确度到底是怎样。
mean 是均值，如果相等直接会返回 True，不相等返回 False。
算平均值，就相当于训练集数据中最重的精确度，即正确率。
同样的方法看测试集。
相对来说测试集比训练集要低一点，这一结论非常正常，因为用的创建这个模型的数据是训练集，模型更偏向于这个数据。
这样几乎已经得到模型训练的结果。这些测试假设成未知的，正确率大概在 78%。

4. 对每一个模型，分别测试训练集和测试集的精确度

```
In [338]: #对每一个模型，分别测试训练集和测试集的精确度
          for model in models:
              print ('\nThe current model is', model)
              model.fit(xTrain, yTrain)
              print ('\nTraining accuracy is',np.mean(model.predict(xTrain) == yTrain))
              print ('\nTesting accuracy is',np.mean(model.predict(xTest) == yTest))
```

```
The current model is LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

Training accuracy is 0.824438202247191

Testing accuracy is 0.7821229050279329

The current model is DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

Training accuracy is 0.9873595505617978

Testing accuracy is 0.7877094972067039

The current model is RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

Training accuracy is 0.9620786516853933

Testing accuracy is 0.8212290502793296
```

可以看一下其他模型是否会有所不同。用到 For 循环。

在决策树模型中：
训练集精确度在 99%，相当高；
测试集只有 77%，甚至不如逻辑回归。
在这种情况下这一 model 为过度拟合，这一模型过度拟合于已有的训练的数据，所以对于测试集非常不友好，测试集的拟合度就会非常低。过度拟合的问题下节课会详细讲解如何处理，在这里不做赘述。

在随机森林模型中：
得到的结果比决策树要好，因为采取了决策树的模型的优点，再进行整合后得到的结果。

这就是三个模型训练后得到的结果。

5. 用 OneHotEncoder 的数据进行同样的处理。

In [339]:
```
#对第二组数据进行相同的操作，每一个模型，分别测试训练集和测试集的精确度
for model in models:
    print ('\nThe current model is', model)
    model.fit(x2Train, y2Train)
    print ('\nTraining accuracy is',np.mean(model.predict(x2Train) == y2Train))
    print ('\nTesting accuracy is',np.mean(model.predict(x2Test) == y2Test))
```

```
The current model is LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

Training accuracy is 0.8300561797752809

Testing accuracy is 0.8156424581005587

The current model is DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

Training accuracy is 0.9873595505617978

Testing accuracy is 0.7821229050279329

The current model is RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

Training accuracy is 0.9789325842696629

Testing accuracy is 0.8100558659217877
```

可以发现：精确度都略有提高。

到这里已经完成了比较完整的机器学习的整个步骤：得到数据——分析数据——整理创建数据——选择模型——得到结果。
接下来，该如何比较模型的好坏？哪个模型才是最好的选择呢？在这里引入交叉验证。

二、交叉验证

在上节课做 train_text_split 的时候只做了一次，很容易数据会有一定的偏差，为了避免偏差，要引入交叉验证。

交叉验证所做的事情，举例来说，即为：比如有 5 份数据，每份都是 20%，我们会用其中 4 份当作 Train，余下来的 1 份当作 Text，然后进行循环，每 20%都会成为 Text，每个数据都会用过 Train 和 Text，这样得到的结果会更加均衡。

1. 调用 k 次交叉验证包

In [340]:
```
#交叉验证是机器学习领域常用的验证模型是否优秀的方法；简而言之就是把数据切分成几个部分然后在训练集和测试集中交换使用
#比如这次在训练集中用到的数据，下一次会放进测试集来使用，因此被称为 交叉
#简单的交叉验证会直接按百分比来划分训练集和测试集，更难一些的方法是K-Fold，我们会使用这个方法来进行交叉验证
#K-Fold其实就是把数据集切成K份，将每一份儿小数据集分别做一次验证集，每次剩下的K-1组份儿数据作为训练集，得到K个模型
#
#从sklearn中调用k次交叉验证包
from sklearn.model_selection import KFold
```

2. 定义 k 次交叉验证函数

In [359]:
```
#定义k次交叉验证函数
def CVKFold(k, X, y, Model):
```

k 即为把模型分成几份，分得越细模型训练会更完整；

X 是变量；

y 是最终结果；

Model 是需要使用的模型。

```
    # Random seed: reproducibility
    np.random.seed(1)

    # accuracy score
    train_accuracy = [0 for i in range(k)]
    test_accuracy = [0 for i in range(k)]

    # index
    idx = 0

    # CV loop
    kf = KFold(n_splits = k, shuffle = True)
```

random. seed 指随便一个数字在其中结果都会一样。

因为将其分成了 k 份，所以要进行 k 次的训练，

划分时候分两种：

一种是直接划分成 k 份，固定进行训练和测试；

另一种是 shuffle，每次会选取（k-1）/k 进行训练，1/k 进行测试。这种相当于更加随机，数据越随机越好，越随机越不会出现过度拟合的情况。

```
# Generate the sets
for train_index, test_index in kf.split(X):
    # Iteration number
    #print(train_index,len(train_index))
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Calling the function/model

    if Model == "Logit":
        clf = LogisticRegression(random_state = 0)

    if Model == "RForest":
        clf = RandomForestClassifier(random_state = 0)
```

iloc——index 的 location

每一次循环都会得到新的 X_train, X_test, y_train, y_text.

```
    # Fit the model
    clf = clf.fit(X_train, y_train)
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)
```

注意！模型定义好之后不要忘记对其进行 fit 和 predict。

```
    train_accuracy[idx] = np.mean(y_train_pred == y_train)
    test_accuracy[idx] = np.mean(y_test_pred == y_test)
    idx += 1

print (train_accuracy)
print (test_accuracy)
return train_accuracy, test_accuracy
```

对 train_accuracy 和 test_accuracy 进行记录。

注意！idx 要加 1，这样才会依次向下记录。

作为一个 function，可以直接 print 和 return。

学院君小提示⏰：

print 和 return 的位置要和上列 function 对齐，否则无法进行操作。

3. 将 k 分为 10 份，得到 10 个不同的结果

In [362]:
```
#应用逻辑回归模型，将数据分为10份，每次取一个样本作为验证数据，剩下k-1个样本作为训练数据
train_acc,test_acc = CVKFold(10,all_x,y,"Logit")
```

```
[0.8202247191011236, 0.8017456359102244, 0.8154613466334164, 0.8216957605985037, 0.8054862842892768, 0.8104738154613467, 0.8042394014962594, 0.8104738154613467, 0.8092269326683291, 0.816708229426434]
[0.7555555555555555, 0.8539325842696629, 0.7752808988764045, 0.7415730337078652, 0.8539325842696629, 0.7865168539325843, 0.8876404494382022, 0.797752808988764, 0.8314606741573034, 0.797752808988764]
```

4. 验证训练数据和测试数据的精确度

In [363]:
```
#验证训练数据和测试数据的精确度
np.mean(train_acc),np.mean(test_acc)
```

Out[363]: (0.811573594104626, 0.8081398252184769)

可以发现：结果相比于之前的一次性的用 train 和 test 更加稳定，因为已经把所有的数据都当作 train 过，都当作 test 过。

如果只用一次，不用交叉验证，得到的结果非常随机。只有在经过交叉验证之后才可以知道这个模型究竟有多好。

5. 应用随机森林模型

```
In [364]: #应用随机森林模型，将数据分为10份，每次取一个样本作为验证数据，剩下k-1个样本作为训练数据
          train_acc,test_acc = CVKFold(10,all_x,y,"RForest")

[0.9650436953807741, 0.9688279301745636, 0.9763092269326683, 0.9763092269326683, 0.972568578553616, 0.9738154613466334, 0.97132169
57605985, 0.972568578553616, 0.9750623441396509, 0.9738154613466334]
[0.7555555555555555, 0.7865168539325843, 0.7191011235955056, 0.7865168539325843, 0.8426966292134831, 0.8089887640449438, 0.8314606
741573034, 0.8202247191011236, 0.8876404494382022, 0.797752808988764]
```

在这里必须要注意!!! 由于在这里定义的名字均为 train_acc， text_acc，所以在运行的时候必须三个模型依次来做，否则下一模型得出的结果会是上一模型的结果。但如果三个模型中定义为不同名字，则不会存在此问题。

这节课我们把三个模型都跑了一遍，介绍了交叉验证的重要知识点，也看到了过度拟合的问题，但我们只得到了最简单的 accuracy 的数字，在下节课我们会想办法怎样去克服这一问题，对结果也会有不同的展示，看怎样去更好地理解这些模型，更多的去理解其中重要的参数，敬请期待～

三、作业

请思考在我们运用 Kfold 的交叉验证时，k 的值是否越大越好？k 变大有什么好处或坏处？最大 k 可以设为多少？

四、第三讲作业答案

1. 输入代码：xTrain.describe()
2. 输入代码：xTest.describe()
3. 分别输出结果如图：

```
In [119]: xTrain.describe()
```

Out[119]:

	Pclass	Age	SibSp	Parch	Fare	Title	Family Size	IsAlone	Sex_Code	Embarked_Code	AgeBin_Code	FareBin_Cc
count	712.000000	712.000000	712.000000	712.000000	712.00000	712.000000	712.000000	712.000000	712.000000	712.000000	712.000000	712.000(
mean	2.317416	29.613652	0.529494	0.391854	31.86295	1.700843	1.921348	0.603933	0.653090	1.543539	1.358146	1.501
std	0.833767	14.433668	1.140842	0.821368	50.13531	0.998789	1.664385	0.489423	0.476321	0.783668	0.895113	1.120(
min	1.000000	0.420000	0.000000	0.000000	0.00000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000(
25%	2.000000	21.000000	0.000000	0.000000	7.91770	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.750(
50%	3.000000	29.000000	0.000000	0.000000	14.45625	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.500(
75%	3.000000	38.000000	1.000000	0.000000	31.27500	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000	3.000(
max	3.000000	80.000000	8.000000	6.000000	512.32920	5.000000	11.000000	1.000000	1.000000	2.000000	4.000000	3.000(

```
In [120]: xTest.describe()
```

Out[120]:

	Pclass	Age	SibSp	Parch	Fare	Title	Family Size	IsAlone	Sex_Code	Embarked_Code	AgeBin_Code	FareBin_C
count	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.000000	179.00(
mean	2.273743	29.565196	0.497207	0.340782	33.561615	1.837989	1.837989	0.597765	0.625698	1.491620	1.363128	1.48(
std	0.846628	14.453567	0.938408	0.742802	48.008223	1.142395	1.394817	0.491724	0.485300	0.830383	0.878573	1.11:
min	1.000000	0.420000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.00(
25%	1.000000	19.000000	0.000000	0.000000	7.972900	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.50(
50%	3.000000	27.000000	0.000000	0.000000	14.000000	1.000000	1.000000	1.000000	1.000000	2.000000	1.000000	1.00(
75%	3.000000	39.500000	1.000000	0.000000	30.500000	2.000000	2.000000	1.000000	1.000000	2.000000	2.000000	2.00(
max	3.000000	64.000000	5.000000	5.000000	263.000000	5.000000	8.000000	1.000000	1.000000	2.000000	3.000000	3.00(

4. 比较两个数据集的以下几个指标，自己分析看有什么不同哦（开放性答案不做要求）

（1） 均值
（2） 男女比例
（3） 三个 embark 的比例
（4） pClass 的比例
（5） 中值
（6） 标准差