Python机器学习实战案例

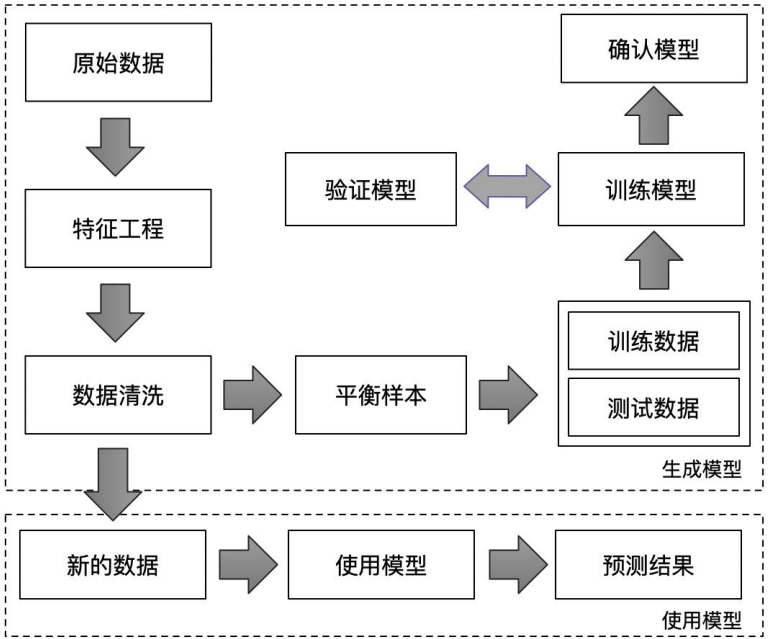
——基于分类算法的学习失败预警

复旦大学 **赵卫东**

业务背景分析

- 通过对学习者的学习过程、学习日志、学习结果进行多维分析，综合判断学生的学习情况，能够提前发现存在失败风险的学生，对其进行系统干预和人工干预。
- 学生失败风险需要分析学生历史学习情况，分析学生在班级中的学习情况，分析学生和标准学习过程的偏离，从横向纵向多维度进行分析。

学习失败风险预测流程


原始数据
特征工程
数据清洗
平衡样本
训练数据
测试数据
生成模型
验证模型
训练模型
确认模型
新的数据
使用模型
预测结果
使用模型

数据收集

- 数据量较多且种类较全，包括学生信息、课程信息、教师信息等常见的教务信息数据，除此之外还包括日常通用日志、课程学习日志、教学设计日志、学习成绩、教学活动记录等。
- 对上述数据进行梳理汇总，从学生、行为、成绩三个维度统计学生的基础信息，并细化为学生基本数据、网络浏览日志数据、课程学习行为数据、形考成绩数据、论坛行为数据、网络课堂表现数据等6个方面的信息。
- 由于系统中数据量很大，这里只选择其中某一门课的学生学习和日志数据，为分析方便，将其从数据库中抽取出来之后，另存为csv文件，进行后续分析。

数据预处理（1）

- 数据探查及特征选择：使用pandas将用户相关的特征读取进来转化为DataFrame对象，并将样本数量、字段数量、数据格式、非空值数量等详细信息进行显示。

```
df = pd.read_csv('uwide.csv')
df.info()
```

运行后，其结果输出如下：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2006 entries, 0 to 2005
Data columns (total 52 columns):
USERID                          2006 non-null object
BROWSER_COUNT                   1950 non-null float64
COURSE_COUNT                    2006 non-null int64
COURSE_LAST_ACCESS              1950 non-null float64
COURSE_SUM_VIEW                 1327 non-null float64
COURSE_AVG_SCORE                228 non-null float64
EXAM_AH_SCORE                   2006 non-null float64
EXAM_WRITEN_SCORE               2006 non-null float64
EXAM_MIDDLE_SCORE               2006 non-null float64
EXAM_LAB                        1468 non-null float64
......
FACESTUDYSCORE                  2006 non-null float64
ONLINESTUDYSCORE                2006 non-null float64
dtypes: float64(29), int64(13), object(10)
memory usage: 815.0+ KB
```

数据预处理（2）

- 为了方便对前述各项指标进行质量检查，包括对数据的分布情况、数据类型、最大值、最小值、均值、标准差等，另外重点查看变量的有效记录数的比例、完成百分比、有效记录数等，作为字段筛选的依据。

```
df.describe()
```

	USERID	BROWSER_COUNT	COURSE_COUNT	COURSE_LAST_ACCESS	COURSE_SUM_VIEW	COURSE_AVG_SCORE	EXAM_AH_SCORE	EXAM_WR
count	2.006000e+03	1950.000000	2006.000000	1.950000e+03	1327.000000	228.000000	2006.000000	
mean	-3.925864e+14	266.023077	1369.385344	1.513165e+09	477.848950	0.443196	85.837506	
std	5.297723e+18	252.247323	4224.334880	1.393563e+06	2961.198469	0.218687	11.497755	
min	-9.222340e+18	3.000000	1.000000	1.503993e+09	-0.015417	0.000000	18.333333	
25%	-4.673419e+18	105.000000	13.000000	1.512395e+09	0.023900	0.302937	83.000000	
50%	-6.692481e+16	190.000000	63.000000	1.513543e+09	0.400961	0.435340	88.516667	
75%	4.704848e+18	334.750000	495.000000	1.514146e+09	2.282650	0.550064	93.062500	
max	9.211461e+18	2182.000000	71043.000000	1.515064e+09	61494.619641	1.000000	99.666667	

数据预处理（3）

- 将性别等中文字段进行因子化（Factorize）处理为数字型变量，查看样本中的空值情况，对所有特征值为空的样本以0填充。

```
factor = pd.factorize(df['SEX'])
df.SEX = factor[0]

null_columns=df.columns[df.isnull().any()]
print(df[df.isnull().any(axis=1)][null_columns].head())

df=df.fillna(0)
```

	BROWSER_COUNT	COURSE_LAST_ACCESS	COURSE_SUM_VIEW	COURSE_AVG_SCORE	\
0	334.0	1.513620e+09	NaN	0.0	
2	17.0	1.505817e+09	NaN	NaN	
3	154.0	1.513881e+09	0.021609	NaN	
4	50.0	1.514159e+09	NaN	NaN	
5	138.0	1.514115e+09	0.000174	NaN	

	EXAM_LAB	NODEBB_LAST_POST	NODEBB_CHANNEL_COUNT	NODEBB_TOPIC_COUNT	\
0	30.583333	1.510339e+09	2.0	2.0	
2	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	

数据预处理（4）

- 生成标签列，按照当前课程的总分（TOTALSCORE）作为标准，以60分为界将学生划分为及格和不及格两个类别标签。对学习这一门之后成绩低于60分的作为失败，即标记为1，反之标记为0，将其存入一个列名为SState的字段，作为学习失败与否的标签。
- 将数据质量较差的字段移除，同时去掉USERID（学号）、GRADE（年级）、专业编号（MAJORID）,(CLASSID)()等无意义变量，这类字段无助于分类算法的改进。

```
cols = df.columns.tolist()
df = df[[
 'BROWSER_COUNT',
 'COURSE_COUNT',
 'COURSE_SUM_VIEW',
 'COURSE_AVG_SCORE',
 'EXAM_AH_SCORE',
 'EXAM_WRITEN_SCORE',
 'EXAM_MIDDLE_SCORE',
 'EXAM_LAB',
 'EXAM_PROGRESS',
 'EXAM_GROUP_SCORE',
......
 'COURSE_WORKCOUNT',
 'SState'
]]
print(df.columns.tolist())
```

数据预处理（5）

- 总共有29列，其中BROWSER_COUNT(浏览次数)、COURSE_COUNT（已上课数量）、COURSE_SUM_VIEW（总浏览时间）、EXAM_AH_SCORE（形考成绩）等28个变量作为输入特征，SState作为预测的目标变量。

	BROWSER_COUNT	COURSE_COUNT	COURSE_SUM_VIEW	COURSE_AVG_SCORE	EXAM_AH_SCORE	EXAM_WRITEN_SCORE	EXAM_MIDDLE_SCORE	EXAM_
0	334.0	11	0.000000	0.000000	94.000000	2.727273	1.727273	30.58
1	744.0	407	0.718819	0.416819	91.454545	2.727273	1.727273	21.35
2	17.0	9	0.000000	0.000000	54.166667	6.166667	7.500000	0.00
3	154.0	133	0.021609	0.000000	81.000000	0.000000	17.000000	0.00
4	50.0	1	0.000000	0.000000	67.000000	0.000000	17.000000	0.00
5	138.0	45	0.000174	0.000000	77.400000	7.400000	10.000000	0.00
6	54.0	1	0.000000	0.000000	50.000000	0.000000	17.000000	0.00
7	80.0	7	0.000000	0.000000	87.500000	3.000000	6.500000	0.00
8	227.0	8	0.000000	0.000000	89.375000	2.250000	4.875000	0.00
9	238.0	1368	0.306042	0.000000	71.733333	9.066667	5.533333	26.16

```
df.SState.value_counts()
```

输出结果如下：

```
0    1801
1     205
Name: SState, dtype: int64
```

数据集划分及不平衡样本处理

- 对两类标签的样本进行再平衡，基本原理是，当某一类别下的样本数量超过另一类别的样本量8倍，则对其进行降采样，将降采样之后样本与另一类进行合并（concat）,组成一个新的DataFrame对象。
- 将整体数据集按照8:2的比例随机划分为训练集和测试集两部分，训练集样本数量为1476条，测试集样本数量为369条。

```
df_majority = df[df.SState==0]
df_minority = df[df.SState==1]
count_times = 8
df_majority_downsampled = df_majority
if len(df_majority)>len(df_minority) * count_times:
    new_major_count = len(df_minority) * count_times
    df_majority_downsampled=resample(df_majority,replace=False,n_samples =
new_major_count, random_state=123)
df = pd.concat([df_majority_downsampled, df_minority])
df.SState.value_counts()
```

```
X = df.iloc[:,0:len(df.columns.tolist())-1].values
y = df.iloc[:,len(df.columns.tolist())-1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
random_state = 21)
print("train count:",len(X_train),"test count:",len(X_test))
```

样本生成及准准化处理

- 在训练集中，及格和不及格学生数分别为1327和149条，为了提高模型性能，采用SMOTE技术对训练集中不及格学生样本进行重采样，即通过对小样本数据进行学习以合成新样本。
- 对所有输入变量采用sklearn中的StandardScaler标准化方法转换。

```
from collections import Counter
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)#
print('Original dataset shape %s' % Counter(y_train))
X_res, y_res = sm.fit_resample(X_train, y_train)
print('Resampled dataset shape %s' % Counter(y_res))
#新生成的样本覆盖训练集
X_train = X_res
y_train = y_res
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

随机森林算法

- 随机森林算法利用多棵子树对样本进行训练，通过引入投票机制实现多棵决策树预测结果的汇总，对于多维特征的分析效果较优，并且可以用其进行特征的重要性分析，运行效率和准确率方面均比较高。
- 模型采用随机森林算法，并使用网格搜索（grid search）进行优化。

```
param_grid = {
    'min_samples_split': range(2, 10),
    'n_estimators' : [10,50,100,150],
    'max_depth': [5, 10,15,20],
    'max_features': [5, 10, 20]
}
scorers = {
    'precision_score': make_scorer(precision_score),
    'recall_score': make_scorer(recall_score),
    'accuracy_score': make_scorer(accuracy_score)
}

classifier = RandomForestClassifier(criterion = 'entropy',
oob_score=True,random_state = 42)
refit_score='precision_score'
skf = StratifiedKFold(n_splits=3)
grid_search = GridSearchCV(classifier, param_grid, refit=refit_score,cv=skf,
return_train_score=True,scoring=scorers, n_jobs=-1)
grid_search.fit(X_train, y_train)
```

检测结果

- 训练完成后，使用测试集（X_test）进行验证，并将结果输出。
- 将测试集的预测结果与实际结果合起来，应用confusion_matrix构建混淆矩。
- 可以看到最大树深度为20，最多特征数量为10个，最小样本分拆量为4个，子树数量为10个。 使用测试集样本对模型进行验证，在测试集的569名学生中，及格人数为313，不及格人数为56，经过模型预测，得到混淆矩阵

```
y_pred = grid_search.predict(X_test)
print(grid_search.best_params_)
print(pd.DataFrame(confusion_matrix(y_test, y_pred),columns=['pred_neg', 'pred_pos'], index=['neg', 'pos']))
```

	预测及格	预测不及格
实际及格	300	13
实际不及格	7	49

```
import sklearn
print("accuracy:",accuracy_score(y_test, y_pred))
print("recall:",recall_score(y_test, y_pred))
print("roc_auc:",sklearn.metrics.roc_auc_score(y_test,grid_search.predict_pro
ba(X_test)[:,1]))
print("f1:",sklearn.metrics.f1_score(y_test, y_pred))
```

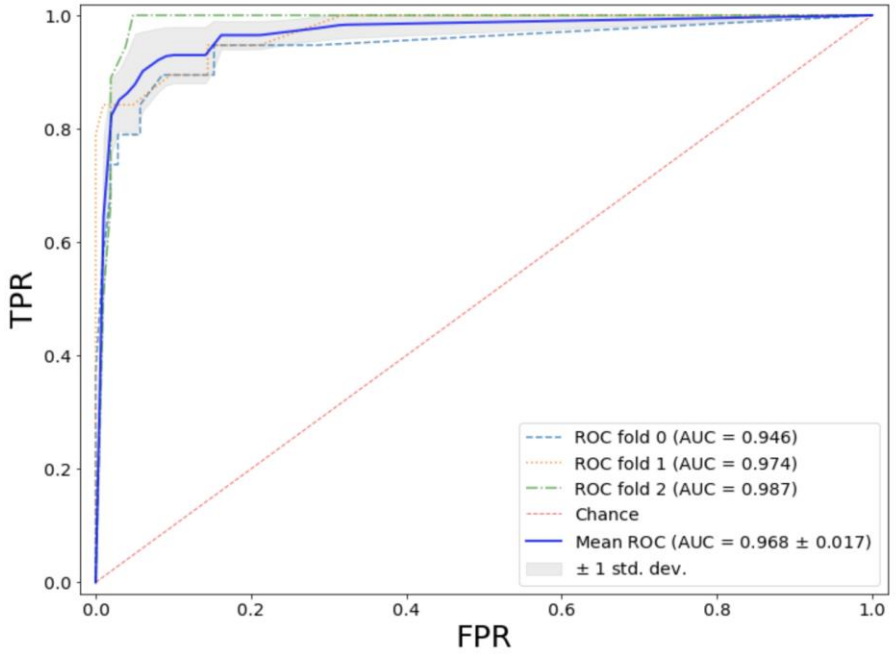
上述代码运行之后的输入结果如下：

```
accuracy: 0.94579945799458
recall: 0.875
roc_auc: 0.9681652213601096
F1: 0.8305084745762712
```

结果分析与可视化(1)

- 实现本模型的ROC曲线绘制，将测试集划分为3部分，即fold 0、fold 1、fold 2, 绘制其ROC曲线，并分别计算它们的AUC值。

```
from scipy import interp

params = {'legend.fontsize': 'x-large',
          'figure.figsize': (12, 9),
         'axes.labelsize': 'x-large',
         'axes.titlesize':'x-large',
         'xtick.labelsize':'x-large',
         'ytick.labelsize':'x-large'}
plt.rcParams.update(params)
```

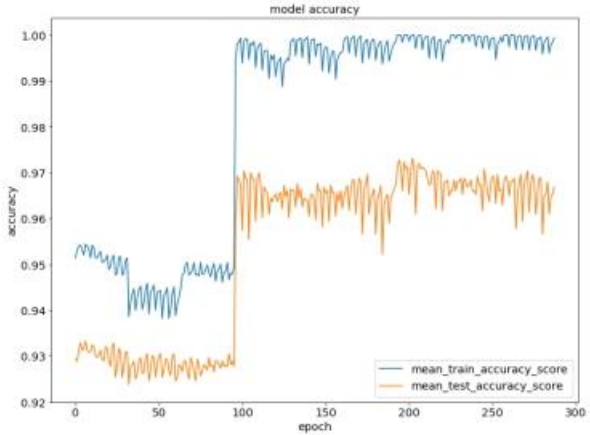


结果分析与可视化(2)

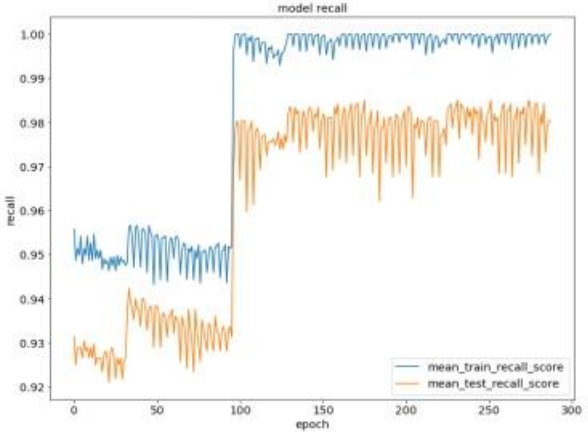
- 为了将迭代计算随机森林评估器的准确率和查合率，将3份数据的平均准确率和平均查全率变化趋势可视化。

```
results = grid_search.cv_results_
plt.plot(results['mean_train_accuracy_score'])
plt.plot(results['mean_test_accuracy_score'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['mean_train_accuracy_score','mean_test_accuracy_score'],
loc='lower right')
plt.show()
```

（1）准确率



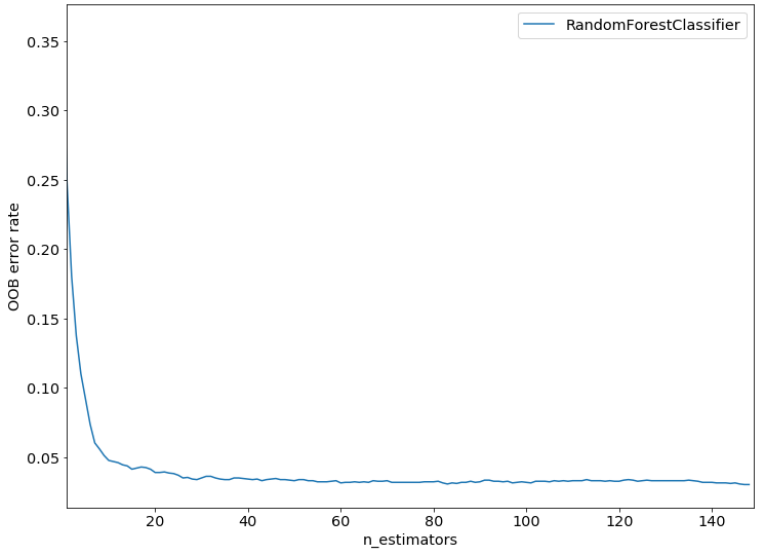
（2）查全率



结果分析与可视化(3)

- 为了评估随机森林的模型性能，绘制其袋外误差（OOB error rate）曲线。
- 随着评估器的增加，模型的误差逐渐下降，在超过100个评估器之后，基本达到最低值，其后下降趋势放缓，并趋于稳定。

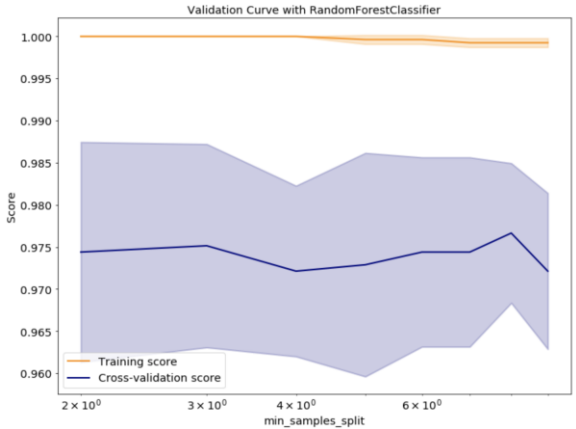
```
min_estimators = 1
max_estimators = 149
clf = grid_search.best_estimator_
errs = []
for i in range(min_estimators, max_estimators + 1):
    clf.set_params(n_estimators=i)
    clf.fit(X_train, y_train)
    oob_error = 1 - clf.oob_score_
    errs.append(oob_error)


plt.plot(errs,label='RandomForestClassifier')
plt.xlim(min_estimators, max_estimators)
plt.xlabel("n_estimators")
plt.ylabel("OOB error rate")
plt.legend(loc="upper right")
plt.show()
```



结果分析与可视化(4)

- 网格搜索过程中，使用sklearn中model_selection的validation_curve方法实现对节点再划分所需最小样本数（min_samples_split）的参数的变化情况进行分析。

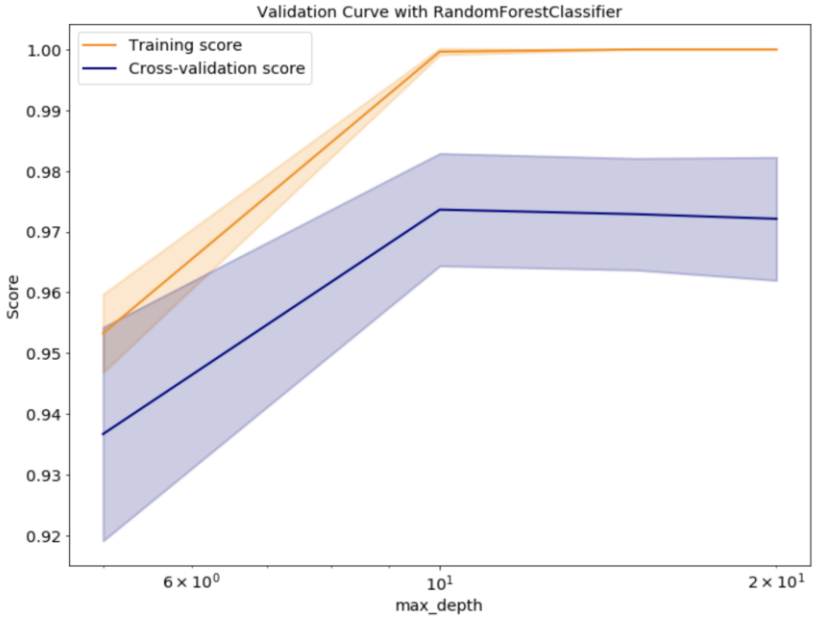
```
from sklearn.model_selection import validation_curve
param_range = range(2,10)
train_scores, test_scores =
validation_curve(grid_search.best_estimator_, X_train,
y_train,'min_samples_split',param_range,cv=3,scoring="r
ecall", n_jobs=-1)
```




```
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.title("验证曲线")
plt.xlabel("min_samples_split")
plt.ylabel("Score")
lw = 2
plt.semilogx(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")
plt.show()
```

结果分析与可视化(5)

- 将validation_curve方法的min_samples_split参数改为max_depth参数，则会对应生成权的最大深度参数在不同的取值时模型查全率的变化图表，节点再划分所需最小样本数的参数在取值为5时其交叉验证评分值最高，而树的最大深度参数在取值为10时，模型的评分最高。

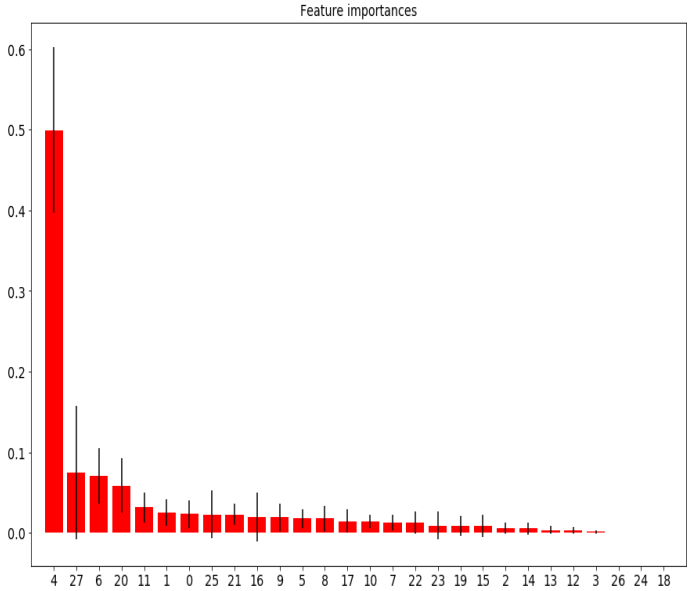


特征重要性分析

- 模型的目标是找到不及格学生，即标识存在学习失败风险的学生，所以在网格搜索过程中，优化指标选择为查全率（Recall）作为参数确认依据，使得训练之后得到分类模型尽可能识别不及格学生的特征。
- classifier是随机森林的最优子树，其feature_importances_属性表示重要的特征列表，按列（X.shape[1]）遍历输入特征,输出其特征列号及重要度的值。

```
classifier = grid_search.best_estimator_
importances = classifier.feature_importances_
indices = np.argsort(importances)[::-1]
for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f],
importances[indices[f]]))
```

```
plt.figure()
plt.title("Feature importances")
std = np.std([tree.feature_importances_ for tree in classifier.estimators_],
axis=0)
plt.bar(range(X.shape[1]),importances[indices],color="r",yerr=std[indices],al
ign="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```



与其它算法比较（1）

- 分别应用scikit-learn中的支持向量机（SVM）、逻辑回归（Logistic Regression）、AdaBoost算法进行对比实验。
- 与支持向量机的比较

```
import sklearn.svm
import sklearn.metrics
from matplotlib import pyplot as plt
clf = sklearn.svm.LinearSVC().fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted']))
print("accuracy_score:",accuracy_score(y_test, y_pred))
print("recall_score:",recall_score(y_test, y_pred))
print("roc_auc_score:",roc_auc_score(y_test, y_pred))
print("f1_score:",f1_score(y_test, y_pred))
```

```
Predicted    0   1
Actual
0          276  37
1            7  49
accuracy_score: 0.8807588075880759
recall_score: 0.875
roc_auc_score: 0.8783945686900958
f1_score: 0.6901408450704225
```

与其它算法比较（2）

- 与逻辑回归算法的性能比较：在逻辑回归算法中，参数C的取值为1.0,最大迭代次数（max_iter）为100，采用L2作为正则化项。

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
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)
y_pred = model.predict(X_test)
print("accuracy_score:",accuracy_score(y_test, y_pred))
print("recall_score:",recall_score(y_test, y_pred))
print("roc_auc_score:",sklearn.metrics.roc_auc_score(y_test, y_pred))
print("f1_score:",sklearn.metrics.f1_score(y_test, y_pred))
```

输出结果如下：

```
accuracy_score: 0.8807588075880759
recall_score: 0.8928571428571429
roc_auc: 0.9370721131903241
f1_score: 0.6944444444444445
```

与其它算法比较（3）

- 在AdaBoost算法中，评估器（base_estimator）采用决策树分类器（DecisionTreeClassifier），树的最大深度为3。子树即评估器数量（n_estimators）为500，学习率为0.5,采用算法（algorithm）为SAMME。

```
from sklearn.externals.six.moves import zip
import matplotlib.pyplot as plt
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
bdt_discrete = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=3),
    n_estimators=500,
    learning_rate=.5,
    algorithm="SAMME")
bdt_discrete.fit(X_train, y_train)
```

运行之后，可以看到模型的各项缺省参数信息，其结果如下：

```
AdaBoostClassifier(algorithm='SAMME',
          base_estimator=DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=3,
            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'),
          learning_rate=0.5, n_estimators=500, random_state=None)
```

与其它算法比较（4）

- 计算不同算法在准确率、查全率、F1值、AUC值指标。
- 可以看到随机森林算法在查全率上仅次于逻辑回归算法，但是其准确率和F1值最高，达到83.05%,具有较强的实用价值，所以最终采用随机森林作为最佳模型。

算法	准确率	查全率	F1值	AUC值
SVM	0.8807	0.875	0.6901	0.8783
逻辑回归	0.8807	0.8928	0.6944	0.9370
AdaBoost	0.9430	0.8214	0.8141	0.9771
随机森林	0.9457	0.875	0.8305	0.9681

总结

- 应用Python对数据进行探查和预处理，实现了样本划分、再平衡和特征的标准化处理等常规操作，重点分析了随机森林、支持向量机、逻辑回归、AdaBoost等算法的使用方法，并结合实例对网络搜索参数调优进行详细阐述，最后对比了各算法在准确率、查全率、AUC值及F1值等指标上性能，选择了与业务相匹配的最优算法。