2024 春 数据分析及实践
实验五: 数据挖掘方法实现分类预测

2024 年 5 月 21 日

马天开

tiankaima@mail.ustc.edu.cn

ID: 3 / PB2100030

数据集 1: 「威斯康辛州乳腺癌数据集」

数据信息和预处理

读取 data.csv 文件，查看数据集的基本信息:

- 列名:

```
['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst']
```

- 大小:

  $(560, 32)$

- 缺失值:

```
smoothness_mean           1
fractal_dimension_mean    2
texture_se                2
compactness_se            1
concavity_se              1
fractal_dimension_se      1
radius_worst              1
smoothness_worst          1
```

- 数据类型:

  除 diagnosis 为 str 外，其余均为 float64 / int64

- 预处理:

  移除缺失值，将 diagnosis 转换为整数类型:

```
# drop missing values:
data = data.dropna()
# convert diagnosis to binary:
data['diagnosis'] = data['diagnosis'].map({'M': 1, 'B': 0})
```

数据集划分

采用 k-fold 交叉验证, 不单独设置验证集. 首先将数据集划分为训练集和测试集:

```
from sklearn.model_selection import train_test_split

X = data.drop(columns=['id', 'diagnosis'])
y = data['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(
  X, y, test_size=0.2, random_state=42
)
```

其中参数:

- `X`: 特征
- `y`: 标签
- `test_size`: 测试集占比
- `random_state`: 随机种子

分类算法模型、主实验、参数实验

由于已经划分了训练集、验证集, 我们把参数实验和主实验放在一起, 通过 `GridSearchCV` 来寻找最优的超参数.

随机森林

随机森林可以视作多个决策树的集成, 通过投票的方式来决定最终的分类结果. 在 sklearn 中, 我们使用 `RandomForestClassifier` 类, 我们先看看默认参数下的效果:

```
from sklearn.ensemble import RandomForestClassifier

# random forest classifier:
rf_clf = RandomForestClassifier()
scores = cross_val_score(rf_clf, X_train, y_train, cv=5)
print("Cross validation scores: ", scores)
print("Mean score: ", scores.mean())

rf_clf.fit(X_train, y_train)
test_score = rf_clf.score(X_test, y_test)
print("Test set score: ", test_score)
```

结果:

```
Cross validation scores: [0.96666667 0.97777778 0.92222222 0.97752809 0.98876404]
Mean score:  0.9665917602996255
Test set score:  0.9464285714285714
```

接下来按照 `GridSearchCV` 的方法调整超参数:

```python
# use grid search to find the best hyperparameters:
from sklearn.model_selection import GridSearchCV

param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [3, 5, 7],
    "max_features": [5, 10, 15],
}

grid_search = GridSearchCV(rf_clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best parameters: ", grid_search.best_params_)
print("Best cross-validation score: ", grid_search.best_score_)
print("Test set score: ", grid_search.score(X_test, y_test))
```

结果:

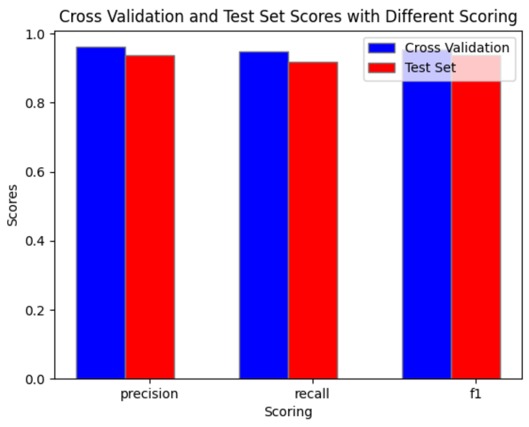
```
Best parameters:  {'max_depth': 5, 'max_features': 5, 'n_estimators': 100}
Best cross-validation score:  0.9688139825218476
Test set score:  0.9464285714285714
```

在此基础上替换评测指标, 分别尝试 `precision`, `recall`, `f1`:

```python
# find if different measure matters. try precision, recall, f1 score:
def grid_search_with_scoring(scoring):
    grid_search = GridSearchCV(rf_clf, param_grid, cv=5, scoring=scoring)
    grid_search.fit(X_train, y_train)
    print("Best parameters: ", grid_search.best_params_)
    print("Best cross-validation score: ", grid_search.best_score_)
    print("Test set score: ", grid_search.score(X_test, y_test))

for scoring in ["precision", "recall", "f1"]:
    print(f"Scoring with {scoring}:")
    grid_search_with_scoring(scoring)
```

结果:



使用 `precision` 和 `f1` 作为评测指标时, (在最优超参数下) 的效果最好.

K-近邻

K-近邻算法是一种基于实例的学习方法, 通过计算待分类样本与训练集中各个样本的距离, 选取距离最近的 k 个样本, 通过投票的方式来决定最终的分类结果. 在 sklearn 中, 我们使用 `KNeighborsClassifier` 类, 我们先看看默认参数下的效果:

```python
from sklearn.neighbors import KNeighborsClassifier

# k-nearest neighbors classifier:
knn_clf = KNeighborsClassifier()
scores = cross_val_score(knn_clf, X_train, y_train, cv=5)
print("Cross validation scores: ", scores)
print("Mean score: ", scores.mean())

knn_clf.fit(X_train, y_train)
test_score = knn_clf.score(X_test, y_test)
print("Test set score: ", test_score)
```

结果:

```
Cross validation scores: [0.73333333 0.81111111 0.65555556 0.76404494 0.71910112]
Mean score:  0.736629213483146
Test set score:  0.7142857142857143
```

接下来按照 `GridSearchCV` 的方法调整超参数:

```python
param_grid = {
    "n_neighbors": [3, 5, 7, 9],
    "weights": ["uniform", "distance"],
    "p": [1, 2],
}

grid_search = GridSearchCV(knn_clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

print("Best parameters: ", grid_search.best_params_)
print("Best cross-validation score: ", grid_search.best_score_)
print("Test set score: ", grid_search.score(X_test, y_test))
```

结果:

```
Best parameters:  {'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
Best cross-validation score:  0.8057677902621723
Test set score:  0.7857142857142857
```

分类成绩并不算理想, 我们考虑把只保留部分特征, 以减少维度:

```
from sklearn.feature_selection import SelectKBest, f_classif

# select best features:
selector = SelectKBest(f_classif, k=10)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

grid_search.fit(X_train_selected, y_train)

print("Best parameters: ", grid_search.best_params_)
print("Best cross-validation score: ", grid_search.best_score_)
print("Test set score: ", grid_search.score(X_test_selected, y_test))
```

带来的收益提升了一些:

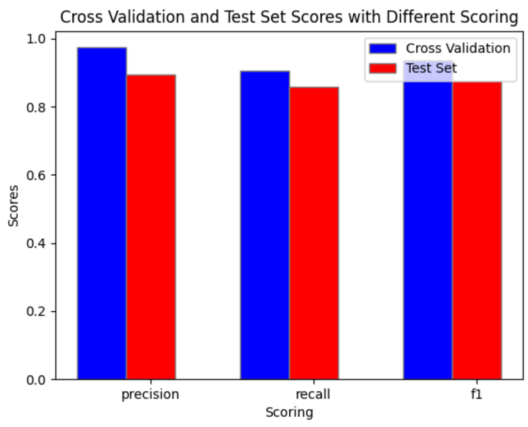
```
Best parameters:  {'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
Best cross-validation score:  0.957578027465668
Test set score:  0.8928571428571429
```

跟上面一样, 我们考虑使用不同的评测指标:

```
def grid_search_with_scoring(scoring):
    grid_search = GridSearchCV(knn_clf, param_grid, cv=5, scoring=scoring)
    grid_search.fit(X_train_selected, y_train)
    print("Best parameters: ", grid_search.best_params_)
    print("Best cross-validation score: ", grid_search.best_score_)
    print("Test set score: ", grid_search.score(X_test_selected, y_test))

for scoring in ["precision", "recall", "f1"]:
    print(f"Scoring with {scoring}:")
    grid_search_with_scoring(scoring)
```

结果:



使用 `precision` 作为评测指标时, (在最优超参数下) 的效果最好.

数据集 2: 「乳腺癌数据」

数据信息和预处理

- 和 Homework 4 一致的处理方法, 不再赘述.
- 大小: $(277, 11)$

数据集划分

值得一提的事, 使用 `random_state=42` 划分数据集可以得到一个 `score=1.0` 的逆天结果. 我们另外选取一个 `random_state=0` 来观察不同的结果.

分类算法模型、主实验、参数实验

随机森林

- 默认参数结果:

```
Cross validation scores: [1. 1. 1. 1. 0.97727273]
Mean score:  0.9954545454545455
Test set score:  0.9821428571428571
```

- GridSearchCV 结果:

```
Best parameters:  {'max_depth': 3, 'max_features': 5, 'n_estimators': 50}
Best cross-validation score:  1.0
Test set score:  0.9821428571428571
```

(默认参数下) 和 (最优超参数下) 的效果一致.

- 不同评测指标结果:

```
Scoring with precision:
Best cross-validation score:  1.0
Test set score:  1.0
Scoring with recall:
Best cross-validation score:  1.0
Test set score:  0.9444444444444444
Scoring with f1:
Best cross-validation score:  1.0
Test set score:  0.9714285714285714
```

`precision` 再一次表现最好.

K-近邻

- 默认参数结果:

```
Cross validation scores: [1. 1. 1. 1. 0.97727273]
Mean score:  0.9954545454545455
Test set score:  0.9821428571428571
```

- GridSearchCV 结果:

```
Best parameters:  {'n_neighbors': 5, 'p': 2, 'weights': 'distance'}
Best cross-validation score:  1.0
Test set score:  0.9821428571428571
```

(默认参数下) 和 (最优超参数下) 的效果一致.

- 减少维度结果:

```
Best parameters:  {'n_neighbors': 5, 'p': 2, 'weights': 'distance'}
Best cross-validation score:  1.0
Test set score:  0.9821428571428571
```

减少维度后, 效果没有提升.

- 后续内容我们 skip 掉, 通过这样的方法计算的结果与随机森林完全一致.

总结

通过实验, 我们发现

- 在第一个数据集上, 随机森林的效果要好于 K-近邻算法.
- 在第二个数据集上, 两种算法的效果一致, 这主要是得益于数据本身的特性.
- 在评测指标上, 对大部分数据, `precision` 作为评测指标时, 效果最好.
- 通过 `GridSearchCV` 来寻找最优的超参数, 可以提升模型的效果.
- 适当减少维度, 可以提升 K-近邻算法的效果.
- 适当的数据预处理和特征选择, 可以提升模型的效果.