2023-2-8

MLP

参数

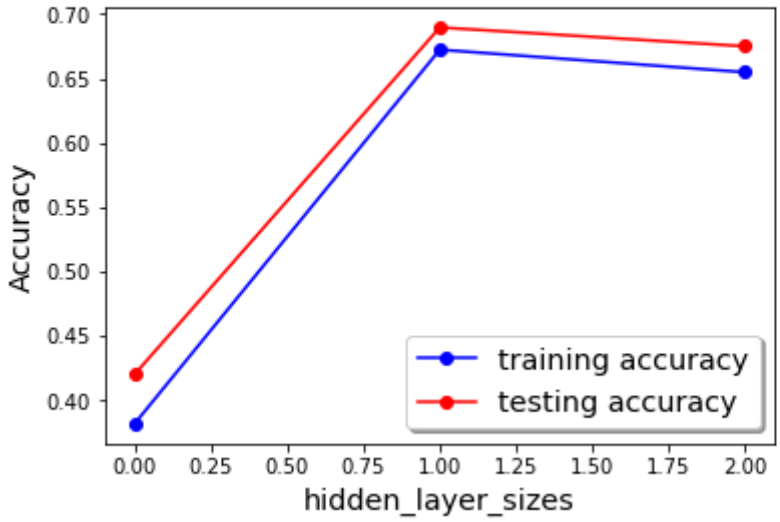
- **activation**：激活函数
- **hidden_layer_sizes**：隐藏层的size
- **max_iter**：最大迭代次数
- **solver**：求解器

单变量调参

隐藏层

- 隐藏层的size的不同

```
train_acc: 0.6547924849519939
 test_acc: 0.6750052290995516
(50, 50)
```



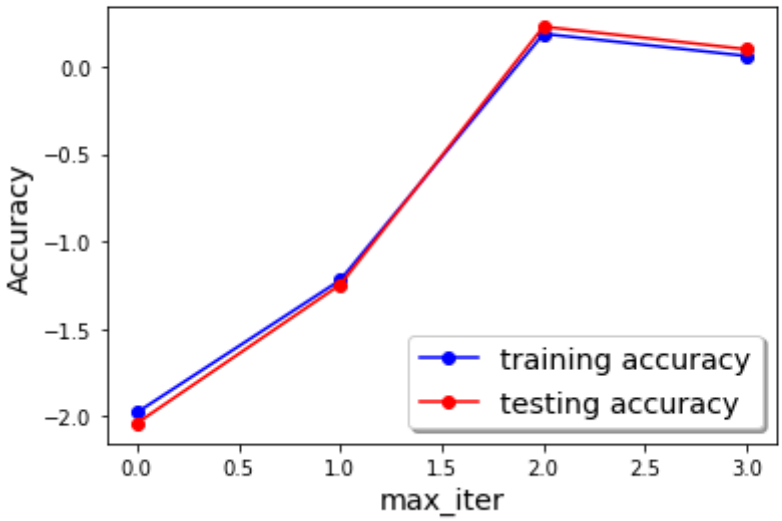
- 代码截图

```
#hidden_layer_sizes
hidden_layer_sizes = [(20,), (30, 20, 10), (50, 50)]
train_accuracy = []
test_accuracy = []

for d in hidden_layer_sizes:
    mlp_reg = neural_network.MLPRegressor(activation='relu', hidden_layer_sizes=d, max_iter=2000, solver='adam')
    mlp_reg.fit(X_train, y_train)
    train_accuracy.append(mlp_reg.score(X_train, y_train))
    test_accuracy.append(mlp_reg.score(X_test, y_test))
print('train_acc:', mlp_reg.score(X_train, y_train),
      '\n test_acc:', mlp_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(3), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(3), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('hidden_layer_sizes', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

最大迭代次数

- 迭代次数的不同

```
train_acc: 0.058984773766912135
 test_acc: 0.09729192113419705
2000
```



- 代码截图

```
#max_iter
max_iter = [500, 1000, 1500, 2000]
train_accuracy = []
test_accuracy = []

for d in max_iter:
    mlp_reg = neural_network.MLPRegressor(activation='relu', hidden_layer_sizes=(10,), max_iter=d, solver='adam')
    mlp_reg.fit(X_train, y_train)
    train_accuracy.append(mlp_reg.score(X_train, y_train))
    test_accuracy.append(mlp_reg.score(X_test, y_test))
print('train_acc:', mlp_reg.score(X_train, y_train),
      '\n test_acc:', mlp_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(4), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(4), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('max_iter', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

自动调参

最优参数

```
{'activation': 'relu', 'hidden_layer_sizes': (50, 50), 'max_iter': 2000, 'solver': 'adam'}
MLPRegressor(hidden_layer_sizes=(50, 50), max_iter=2000)
```

代码截图

```
mlp_reg = neural_network.MLPRegressor()

param_grid = {"activation" : ['relu'],
              "hidden_layer_sizes" : [(10,), (15,), (20,), (30, 20, 10), (50, 50), (100,)],
              "max_iter" : [500, 1000, 1500, 2000],
              "solver" : ['adam', 'sgd']}

mlp_reg = GridSearchCV(mlp_reg, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)

mlp_reg.fit(X_train, y_train)
```

GridSearchCV
estimator: MLPRegressor
MLPRegressor

XGBoost

参数

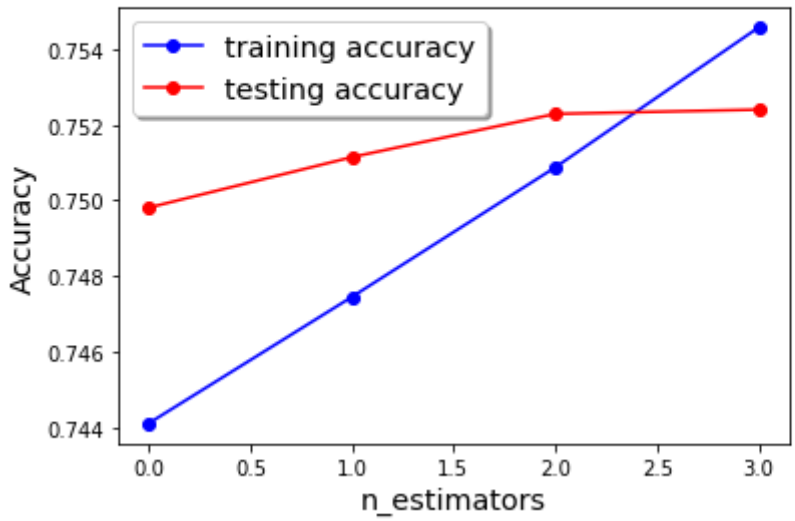
- **n_estimators**：树的数量
- **eta**：学习率
- **max_depth**：树的深度
- **colsample_bytree**：每棵树随机抽样出的特征占所有特征的比例
- **min_child_weight**：叶子节点上所需要的最小样本权重

单变量调参

最大迭代次数

- 最大迭代次数

```
train_acc: 0.7545567332208292
 test_acc: 0.7523903245247908
200
```



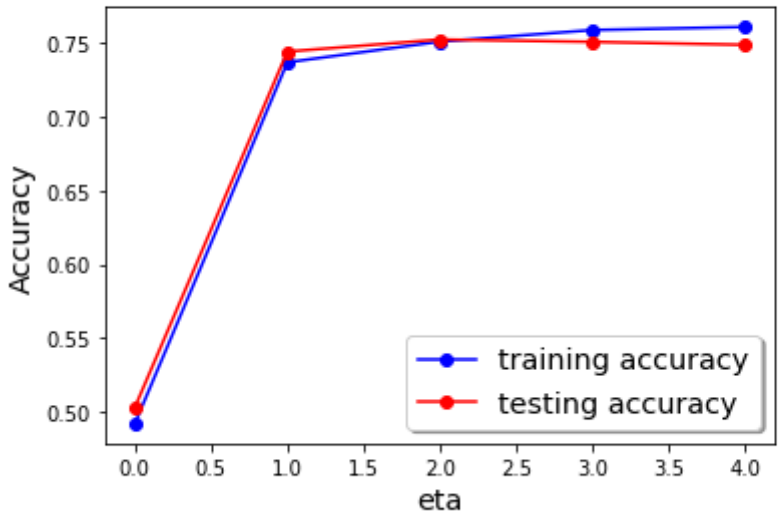
- 代码截图

```
#n_estimators
n_estimators = [100, 120, 150, 200]
train_accuracy = []
test_accuracy = []

for d in n_estimators:
    xgb_reg = xgboost.XGBRegressor(n_estimators=d, eta=0.1, max_depth=5, colsample_bytree=1, min_child_weight=4)
    xgb_reg.fit(X_train, y_train)
    train_accuracy.append(xgb_reg.score(X_train, y_train))
    test_accuracy.append(xgb_reg.score(X_test, y_test))
print('train_acc:', xgb_reg.score(X_train, y_train),
      '\n test_acc:', xgb_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(4), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(4), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('n_estimators', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

学习率

- 学习率的不同

```
train_acc: 0.7607963407485161
 test_acc: 0.7486729207059308
0.3
```



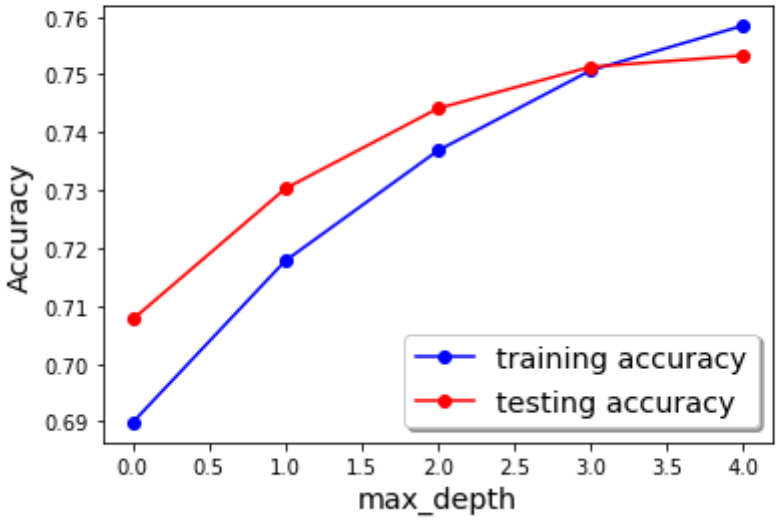
- 代码截图

```
#eta
eta = [0.01, 0.05, 0.1, 0.2, 0.3]
train_accuracy = []
test_accuracy = []

for d in eta:
    xgb_reg = xgboost.XGBRegressor(n_estimators=150, eta=d, max_depth=5, colsample_bytree=1, min_child_weight=4)
    xgb_reg.fit(X_train, y_train)
    train_accuracy.append(xgb_reg.score(X_train, y_train))
    test_accuracy.append(xgb_reg.score(X_test, y_test))
print('train_acc:', xgb_reg.score(X_train, y_train),
      '\n test_acc:', xgb_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(5), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(5), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('eta', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

最大深度

- 最大深度的不同

```
train_acc: 0.7584261274102954
 test_acc: 0.7532576711282377
7
```



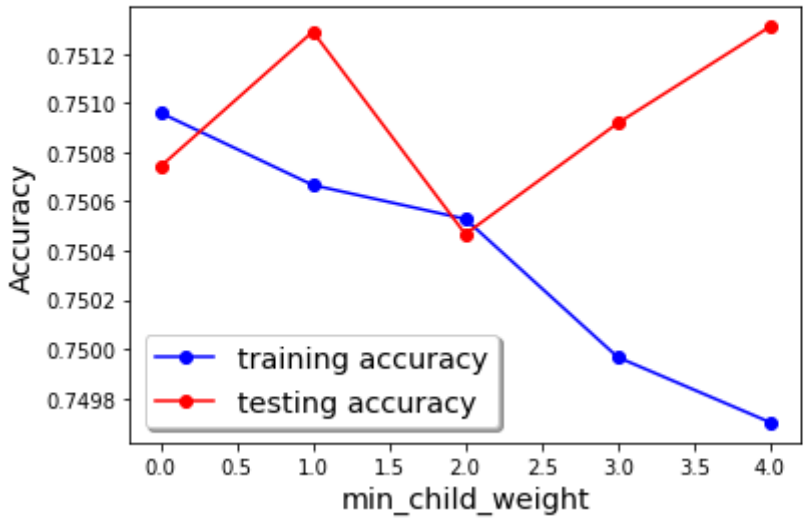
- 代码截图

```
#max_depth
max_depth = [3, 4, 5, 6, 7]
train_accuracy = []
test_accuracy = []

for d in max_depth:
    xgb_reg = xgboost.XGBRegressor(n_estimators=150, eta=0.05, max_depth=d, colsample_bytree=1, min_child_weight=4)
    xgb_reg.fit(X_train, y_train)
    train_accuracy.append(xgb_reg.score(X_train, y_train))
    test_accuracy.append(xgb_reg.score(X_test, y_test))
print('train_acc:', xgb_reg.score(X_train, y_train),
      '\n test_acc:', xgb_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(5), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(5), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('max_depth', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

叶子节点最小权重

- 最小权重的不同

```
train_acc: 0.7497012513753881
 test_acc: 0.7513103430784404
7
```



- 代码截图

```python
#min_child_weight
min_child_weight = [1, 2, 3, 4, 5]
train_accuracy = []
test_accuracy = []

for d in max_depth:
    xgb_reg = xgboost.XGBRegressor(n_estimators=150, eta=0.05, max_depth=6, colsample_bytree=1, min_child_weight=d)
    xgb_reg.fit(X_train, y_train)
    train_accuracy.append(xgb_reg.score(X_train, y_train))
    test_accuracy.append(xgb_reg.score(X_test, y_test))
print('train_acc:', xgb_reg.score(X_train, y_train),
      '\n test_acc:', xgb_reg.score(X_test, y_test))
print(d)
#Plot the train & test accuracy
plt.plot(range(5), train_accuracy, 'bo-', label ='training accuracy')
plt.plot(range(5), test_accuracy, 'ro-', label = 'testing accuracy')

plt.xlabel('min_child_weight', fontsize='x-large')
plt.ylabel('Accuracy', fontsize='x-large')

plt.legend(loc='best', shadow=True, fontsize='x-large')
plt.show()
```

自动调参

最优参数

```python
print(gs_reg.best_score_)
print(gs_reg.best_params_)
print(gs_reg.best_estimator_)
```

```
-93228448801.14752
{'colsample_bytree': 1, 'eta': 0.1, 'max_depth': 5, 'min_child_weight': 4, 'n_estimators': 300}
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
             early_stopping_rounds=None, enable_categorical=False, eta=0.1,
             eval_metric=None, feature_types=None, gamma=None, gpu_id=None,
             grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=4, missing=nan, monotone_constraints=None,
             n_estimators=300, n_jobs=None, num_parallel_tree=None,
             predictor=None, ...)
```

代码截图

```
import xgboost
xgb_reg = xgboost.XGBRegressor()

param_grid = {
        "n_estimators": [50,100,150,200,300],
        "eta": [0.05, 0.1, 0,2, 0.3],
        "max_depth": [3,4,5,6,7],
        "colsample_bytree": [0.4,0.6,0.8,1],
        "min_child_weight": [1,2,3,4]
    }

gs_reg = GridSearchCV(xgb_reg, param_grid, n_jobs=-1, scoring='neg_mean_squared_error', cv=5)
gs_reg.fit(X_train, y_train)
```