Pattern & Anomaly Detection Lab

Experiment 8

Nested Cross Validation with Hyperparameter Tuning

Submitted By:

Dhruv Singhal

500075346

R177219074

AIML B3

Submitted To:

Dr. Gopal Phartiyal

Asst. Professor

SOCS

UPES

Hyperparameter Tuning and Nested Cross validation on Linear Regression

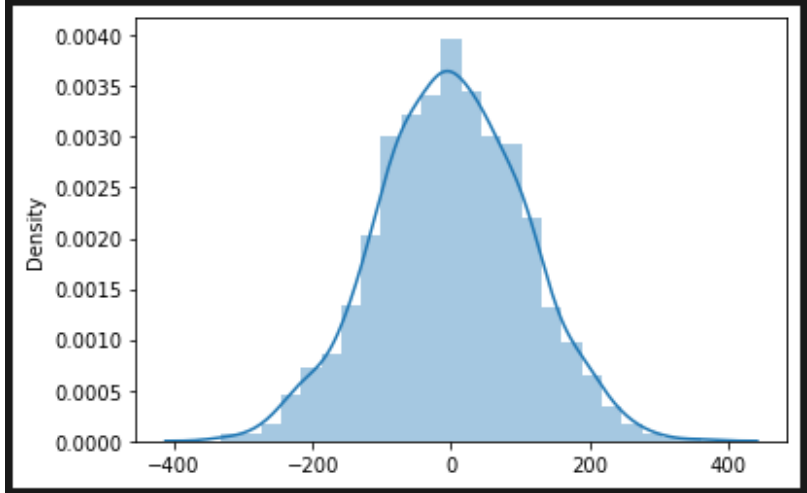
CODE:

```
import numpy as np
import matplotlib.pyplot  as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#%%
from sklearn.datasets import make_regression
x,y=make_regression(n_samples=1000,n_features=5,noise=20)
sns.distplot(y)

#%%
plt.hist(x)
#%%
x_train = x
y_train =y

#%%
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
NUM_TRIALS = 30


tuned_parameters = [{'solver' : ['svd', 'lsqr'],'fit_intercept': ['True'],'normalize': ['False']},
                    {'solver' : ['sag', 'cholesky'],'fit_intercept': ['False'],'normalize': ['true']}]

score = 'r2'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)

# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LinearRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = Ridge(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= Ridge(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()


score_difference = non_nested_scores - nested_scores

print("Average difference of {:6f} with std. dev. of {:6f}."
      .format(score_difference.mean(), score_difference.std()))
```

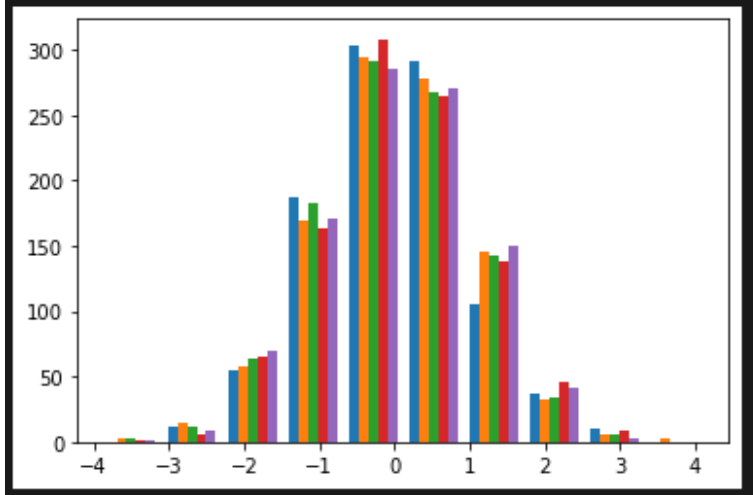
OUTPUT:

Console 3/A

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp9.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp9.py')
```




```
In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/exp9.py')
```




```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/exp9.py')

In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/exp9.py')
Average difference of -0.003284 with std. dev. of 0.001567.
```

Hyperparameter Tuning and Nested Cross validation on Logistic Regression

CODE:

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn import linear_model
#from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
#%%
# Generating Data
X, y = make_classification(n_samples = 1000, n_features = 5, n_classes = 2)
x_train = X
y_train = y
#%%
NUM_TRIALS = 30

tuned_parameters = [{ 'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
 'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']
   }]

score = 'accuracy'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LogisticRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = linear_model.LogisticRegression(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    print(model.best_params_)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= linear_model.LogisticRegression(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()
score_difference = non_nested_scores - nested_scores
print("Average difference of {:6f} with std. dev. of {:6f}.".format(score_difference.mean(), score_difference.std()))
```

OUTPUT:

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.23357214690901212, 'penalty': 'l1', 'solver': 'liblinear'}
Average difference of 0.001900 with std. dev. of 0.001719.
```

Hyperparameter Tuning and Nested Cross validation on Logistic Regression (Breast Cancer Dataset)

CODE:

```python
import numpy as np
#from sklearn.datasets import make_classification
from sklearn import linear_model
#from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
#%%
# Generating Data
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()

X = data.data

y = data.target
x_train = X
y_train = y
#%%
NUM_TRIALS = 30


tuned_parameters = [{ 'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']
   }]
score = 'accuracy'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LogisticRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = linear_model.LogisticRegression(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    print(model.best_params_)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= linear_model.LogisticRegression(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()
score_difference = non_nested_scores - nested_scores
print("Average difference of {:6f} with std. dev. of {:6f}.".format(score_difference.mean(), score_difference.std()))
```

OUTPUT:

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
Average difference of 0.009804 with std. dev. of 0.006161.
```