RegularizationRegression

November 5, 2024

[95]:
```
from Dataset import Dataset
import numpy as np
import sklearn
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.model_selection import train_test_split, KFold
import sklearn.feature_selection
from matplotlib.pylab import (
    figure,
    grid,
    legend,
    loglog,
    semilogx,
    show,
    subplot,
    title,
    xlabel,
    ylabel,
)

from dtuimldmtools import bmplot, feature_selector_lr, rlr_validate
```

[96]:
```
dataset = Dataset(original_data=False)
for i, n in enumerate(dataset.attributeNames):
    print(i, n)
attribute_to_predict = 3 # weight
y = dataset.X_mean_std[:, attribute_to_predict]  # we want to predcit the weight
X = np.delete(dataset.X_mean_std, attribute_to_predict, 1)  # delete fourth␣
↪column out of matrix
# X = dataset.X_mean_std[:, 2] # height i think
attributeNames = np.delete(dataset.attributeNames, attribute_to_predict)
N, M = X.shape
```

```
0 Gender
1 Age
2 Height
3 Weight
4 family_history_with_overweight
5 NCP
```

```
6 CH2O
7 FAF
8 TUE
9 CALC
10 MTRANS
11 NObeyesdad
```

[97]:
```python
# Add offset attribute
X = np.concatenate((np.ones((X.shape[0], 1)), X), 1)
attributeNames = np.concatenate([np.array(["Offset"]), attributeNames])
M = M + 1
```

[98]:
```python
## Crossvalidation
# Create crossvalidation partition for evaluation
K = 10
CV = KFold(K, shuffle=True)

# Values of lambda
lambdas = np.power(10.0, np.arange(-1, 10, 1))

# Initialize variables
Error_train = np.empty((K, 1))
Error_test = np.empty((K, 1))
Error_train_rlr = np.empty((K, 1))
Error_test_rlr = np.empty((K, 1))
Error_train_nofeatures = np.empty((K, 1))
Error_test_nofeatures = np.empty((K, 1))

w_rlr = np.empty((M, K))
mu = np.empty((K, M - 1))
sigma = np.empty((K, M - 1))
w_noreg = np.empty((M, K))
```

[99]:
```python
k = 0
for train_index, test_index in CV.split(X, y):
    # extract training and test set for current CV fold
    X_train = X[train_index]
    y_train = y[train_index]
    X_test = X[test_index]
    y_test = y[test_index]
    internal_cross_validation = 10

    (
        opt_val_err,
        opt_lambda,
        mean_w_vs_lambda,
        train_err_vs_lambda,
```

```
        test_err_vs_lambda,
    ) = rlr_validate(X_train, y_train, lambdas, internal_cross_validation)
    print(f"Optimal lambda: {opt_lambda}")

    # Standardize outer fold based on training set, and save the mean and standard
    # deviations since they're part of the model (they would be needed for
    # making new predictions) - for brevity we won't always store these in the scripts
    mu[k, :] = np.mean(X_train[:, 1:], 0)
    sigma[k, :] = np.std(X_train[:, 1:], 0)

    X_train[:, 1:] = (X_train[:, 1:] - mu[k, :]) / sigma[k, :]
    X_test[:, 1:] = (X_test[:, 1:] - mu[k, :]) / sigma[k, :]

    Xty = X_train.T @ y_train
    XtX = X_train.T @ X_train

    # Compute mean squared error without using the input data at all
    Error_train_nofeatures[k] = (
        np.square(y_train - y_train.mean()).sum(axis=0) / y_train.shape[0]
    )
    Error_test_nofeatures[k] = (
        np.square(y_test - y_test.mean()).sum(axis=0) / y_test.shape[0]
    )

    # Estimate weights for the optimal value of lambda, on entire training set
    lambdaI = opt_lambda * np.eye(M)
    lambdaI[0, 0] = 0  # Do no regularize the bias term
    w_rlr[:, k] = np.linalg.solve(XtX + lambdaI, Xty).squeeze()
    # Compute mean squared error with regularization with optimal lambda
    Error_train_rlr[k] = (
        np.square(y_train - X_train @ w_rlr[:, k]).sum(axis=0) / y_train.shape[0]
    )
    Error_test_rlr[k] = (
        np.square(y_test - X_test @ w_rlr[:, k]).sum(axis=0) / y_test.shape[0]
    )

    # OR ALTERNATIVELY: you can use sklearn.linear_model module for linear regression:
    m = LinearRegression().fit(X_train, y_train)
    Error_train[k] = np.square(y_train-m.predict(X_train)).sum()/y_train.shape[0]
    Error_test[k] = np.square(y_test-m.predict(X_test)).sum()/y_test.shape[0]

    # Display the results for the last cross-validation fold
```

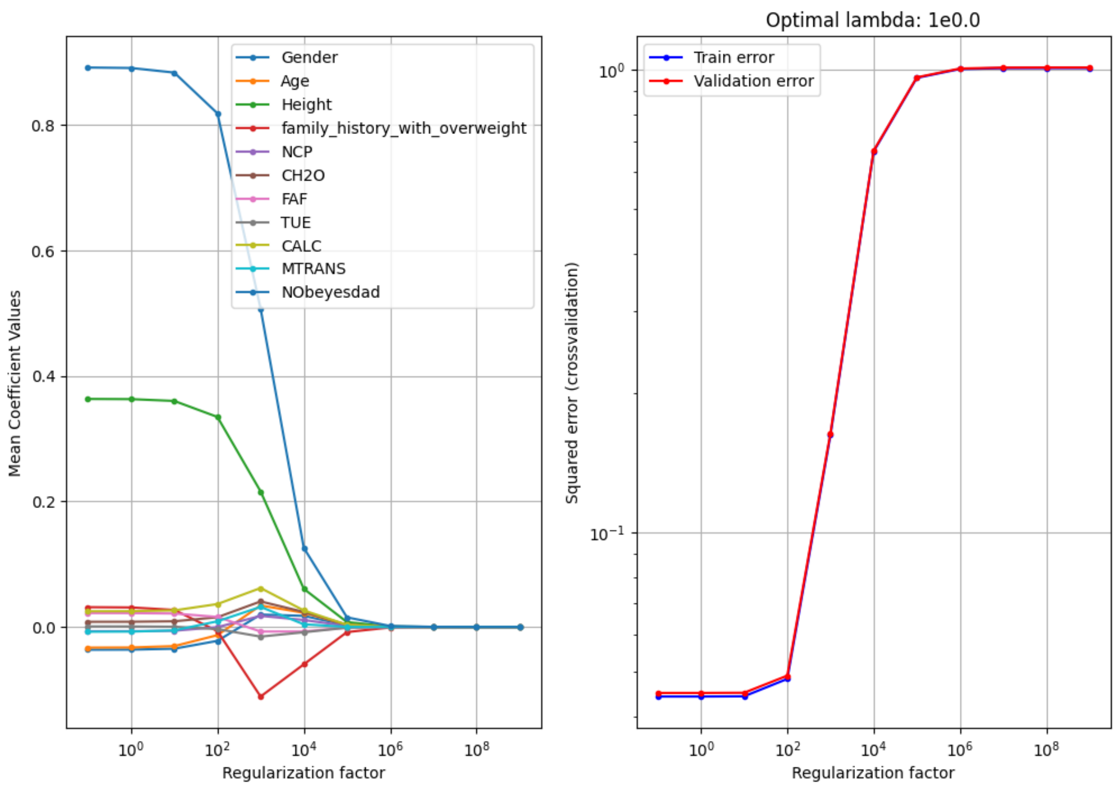
```python
    if k == K - 1:
        figure(k, figsize=(12, 8))
        subplot(1, 2, 1)
        semilogx(lambdas, mean_w_vs_lambda.T[:, 1:], ".-")  # Don't plot the␣
↪bias term
        xlabel("Regularization factor")
        ylabel("Mean Coefficient Values")
        grid()
        # You can choose to display the legend, but it's omitted for a cleaner
        # plot, since there are many attributes
        legend(attributeNames[1:], loc='best')

        subplot(1, 2, 2)
        title("Optimal lambda: 1e{0}".format(np.log10(opt_lambda)))
        loglog(
            lambdas, train_err_vs_lambda.T, "b.-", lambdas, test_err_vs_lambda.
↪T, "r.-"
        )
        xlabel("Regularization factor")
        ylabel("Squared error (crossvalidation)")
        legend(["Train error", "Validation error"])
        grid()

    # To inspect the used indices, use these print statements
    # print('Cross validation fold {0}/{1}:'.format(k+1,K))
    # print('Train indices: {0}'.format(train_index))
    # print('Test indices: {0}\n'.format(test_index))

    k += 1
```

```
Optimal lambda: 1.0
Optimal lambda: 0.1
Optimal lambda: 0.1
Optimal lambda: 1.0
Optimal lambda: 0.1
Optimal lambda: 0.1
Optimal lambda: 1.0
Optimal lambda: 1.0
Optimal lambda: 1.0
Optimal lambda: 1.0
```



[100]:
```python
# Display results
print("Linear regression without regularization:")
print("- Training error: {0}".format(Error_train.mean()))
print("- Test error:     {0}".format(Error_test.mean()))
print(
    "- R^2 train:     {0}".format(
        (Error_train_nofeatures.sum() - Error_train.sum())
        / Error_train_nofeatures.sum()
    )
)
print(
    "- R^2 test:     {0}\n".format(
        (Error_test_nofeatures.sum() - Error_test.sum()) / Error_test_nofeatures.sum()
    )
)
print("Regularized linear regression:")
print("- Training error: {0}".format(Error_train_rlr.mean()))
print("- Test error:     {0}".format(Error_test_rlr.mean()))
print(
    "- R^2 train:     {0}".format(
        (Error_train_nofeatures.sum() - Error_train_rlr.sum())
```

```python
        / Error_train_nofeatures.sum()
    )
)
print(
    "- R^2 test:     {0}\n".format(
        (Error_test_nofeatures.sum() - Error_test_rlr.sum())
        / Error_test_nofeatures.sum()
    )
)

print("Weights in last fold:")
for m in range(M):
    print("{:>15} {:>15}".format(attributeNames[m], np.round(w_rlr[m, -1], 2)))
```

```
Linear regression without regularization:
- Training error: 0.04338168448524744
- Test error:     0.04404886219103793
- R^2 train:     0.9565968367373117
- R^2 test:     0.9558546463266406

Regularized linear regression:
- Training error: 0.043381935808718866
- Test error:     0.0440492900277003
- R^2 train:     0.9565965852894072
- R^2 test:     0.9558542175527784

Weights in last fold:
         Offset           -0.01
         Gender           -0.04
            Age           -0.03
         Height            0.36
family_history_with_overweight            0.03
            NCP           -0.01
           CH2O            0.01
            FAF            0.02
            TUE             0.0
           CALC            0.03
         MTRANS           -0.01
     NObeyesdad            0.89
```

[ ]:

[ ]:

[ ]: