

```

1 import numpy as np
2
3 def fun(x):
4     """
5         Compute a simple quadratic function  $f: \mathbb{R}^5 \rightarrow \mathbb{R}$ .
6
7     """
8
9     return 1.3 + 2.*x[0] - 1.1*x[1] + 0.7*x[2] + 1.2*x[3] + \
10         0.4* (x[0]**2) -1.5 *x[1]*x[3] - 0.7 * (x[4]**2)
11
12 def postfix(N,d,sigma):
13     """Converts parameters into a handy string, to be appended to
14     file names."""
15     return "_N_%d_d_%d" % (N,d) \
16         + "_sig_%s" % str(sigma).replace(".", "_")
17
18 if __name__ == "__main__":
19     # Set random seed for reproducibility
20     np.random.seed(42)
21
22
23     # Number of samples
24     N = 1000
25
26     # Noise variance
27     sigma = 0.01
28
29
30     # Feature dimension
31     d = 40
32
33     print("Generating dataset with N = %d,  $\sigma$  = %f, d = %d..." % (
34         N,sigma,d), end="")
35
36     # Generate random features
37     X = np.random.randn(N, d)
38
39     # Generate pure labels
40     y = []
41
42     for i in range(N):
43         y.append(
44             fun(X[i,:])

```

```
44         )
45     y = np.array(y)
46
47     # Add noise to labels
48     err = np.random.normal(scale = sigma, size = N)
49     y = y + err
50
51     print(" done")
52
53     psfx = postfix(N,d,sigma)
54
55     print("Saving X and y... ",end="")
56     np.save("X" + psfx,X) #saves X with name, X N d sigma
57     np.save("y" + psfx,y) #saves y with name, y N d sigma
58     print(" done")
59
60
61
```

```

1 import numpy as np
2 from sklearn.model_selection import KFold, train_test_split,
  cross_val_score
3 from sklearn.linear_model import Lasso
4 from sklearn.metrics import root_mean_squared_error as rmse
5 import matplotlib.pyplot as plt
6 from data_generator import postfix
7
8
9 # Number of samples
10 N = 1000
11
12 # Noise variance
13 sigma = 0.01
14
15
16 # Feature dimension
17 d = 40
18
19 psfx = postfix(N,d,sigma)
20
21 # function lift = liftDataset
22 def lift(x_initial):
23     x_prime = []
24
25     xi = len(x_initial)
26     for row in range(xi):
27         x_expand = []
28         x_0 = x_initial[row]
29
30         l = len(x_0)
31         for i in range(l):
32             x_expand.append(x_0[i])
33             for j in range(i,l):
34                 x_expand.append(x_0[j]*x_0[i])
35
36         x_prime.append(x_expand)
37
38     return np.array(x_prime)
39
40
41 X = np.load("X"+psfx+".npy")
42 y = np.load("y"+psfx+".npy")
43 X = lift(X)
44

```

```

45 print("Dataset has n=%d samples, each with d=%d features," % X.
      shape,"as well as %d labels." % y.shape[0])
46
47 X_train, X_test, y_train, y_test = train_test_split(
48     X, y, test_size=0.30, random_state=42)
49
50 print("Randomly split dataset to %d training and %d test samples"
      % (X_train.shape[0],X_test.shape[0]))
51
52
53 # alpha = 0.1
54 #
55 # model = Lasso(alpha = alpha)
56 #
57 # cv = KFold(
58 #     n_splits=5,
59 #     random_state=42,
60 #     shuffle=True
61 # )
62 #
63 #
64 #
65 # scores = cross_val_score(
66 #     model, X_train, y_train, cv=cv, scoring="
67 #     neg_root_mean_squared_error")
68 #
69 # print("Cross-validation RMSE for  $\alpha$ =%f : %f  $\pm$  %f" % (alpha,-np.
70 #     mean(scores),np.std(scores)) )
71
72 # === New code: scan alpha values and pick the best one ===
73
74 alphas = 2.0 ** np.arange(-10, 11) # from  $2^{-10}$  to  $2^{10}$ 
75 cv = KFold(n_splits=5, random_state=42, shuffle=True)
76
77 mean_rmse = []
78 std_rmse = []
79
80 print("Performing 5-fold CV for multiple  $\alpha$  values...")
81
82 for alpha in alphas:
83     model = Lasso(alpha=alpha, max_iter=10000, random_state=42)
84     scores = cross_val_score(
85         model, X_train, y_train,
86         cv=cv,

```

```

86         scoring="neg_root_mean_squared_error"
87     )
88
89     mean_rmse.append(-np.mean(scores))
90     std_rmse.append(np.std(scores))
91     print("α = %8.5f -> CV RMSE = %.6f ± %.6f" % (alpha, -np.
mean(scores), np.std(scores)))
92
93 # Find best alpha (lowest mean RMSE)
94 best_idx = np.argmin(mean_rmse)
95 best_alpha = alphas[best_idx]
96
97 print("\nBest α found: %f (CV RMSE = %.6f)" % (best_alpha,
mean_rmse[best_idx]))
98
99 model = Lasso(alpha=best_alpha, max_iter=10000, random_state=42)
100 print("Fitting linear model over entire training set...",end="")
101 model.fit(X_train, y_train)
102 print(" done")
103
104
105 # Compute RMSE
106 rmse_train = rmse(y_train,model.predict(X_train))
107 rmse_test = rmse(y_test,model.predict(X_test))
108
109 print("Train RMSE = %f, Test RMSE = %f" % (rmse_train,rmse_test
))
110
111
112
113 # Plot CV mean RMSE as a function of alpha with error bars
114 plt.figure(figsize=(8, 5))
115 plt.errorbar(alphas, mean_rmse, yerr=std_rmse, fmt='-o', capsize
=4, label='CV RMSE')
116 plt.xlabel('Alpha')
117 plt.ylabel('Mean Cross-Validation RMSE')
118 plt.title('RMSE vs Alpha')
119 plt.grid(True, which='both', linestyle='--', alpha=0.6)
120 plt.legend()
121 plt.show()
122
123
124
125 print("Model parameters:")
126 print("\t Intercept: %3.5f" % model.intercept_,end="")

```

```
127 for i,val in enumerate(model.coef_):
128     print(",  $\beta$ %d: %3.5f" % (i,val), end="")
129 print("\n")
130
131
132
```

```

1 import numpy as np
2 from mpmath.matrices.eigen_symmetric import c_he_tridiag_0
3 from sklearn.model_selection import train_test_split
4 from sklearn.linear_model import LinearRegression
5 import matplotlib.pyplot as plt
6 from sklearn.metrics import root_mean_squared_error as rmse
7 from data_generator import postfix
8
9
10
11 # Number of samples
12 N = 1000
13
14 # Noise variance
15 sigma = 0.01
16
17 # Feature dimension
18 d = 40
19
20
21 psfx = postfix(N,d,sigma)
22
23 # function lift = liftDataset
24 def lift(x_initial):
25     x_prime = []
26
27     xi = len(x_initial)
28     for row in range(xi):
29         x_expand = []
30         x_0 = x_initial[row]
31
32         l = len(x_0)
33         for i in range(l):
34             x_expand.append(x_0[i])
35             for j in range(i,l):
36                 x_expand.append(x_0[j]*x_0[i])
37
38         x_prime.append(x_expand)
39
40     return np.array(x_prime)
41 # test lift function
42 # x_initials = [[1,2,3,4,5],
43 #                [2,4,6,8,10]]
44 # new_x_i =lift(x_initials)
45 # print(new_x_i)

```

```

46
47
48
49 X = np.load("X"+psfx+".npy")
50 y = np.load("y"+psfx+".npy")
51
52
53
54 print("Dataset has n=%d samples, each with d=%d features," % X.
      shape,"as well as %d labels." % y.shape[0])
55 frac = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
56 rmse_train_array = []
57 rmse_test_array = []
58 training_length_array = []
59 X = lift(X)
60
61 for i, fr in enumerate(frac):
62     # ... use both i and frac
63     print(f"Training with {fr*100}% of the training data")
64     #Split data: test set made up of fraction of N data (ex 0.30*
1000 = 300)
65     testSize = 0.30
66     X_train, X_test, y_train, y_test = train_test_split(
67         X, y, test_size=testSize, random_state=42)
68
69     print("Randomly split dataset to %d training and %d test
samples" % (X_train.shape[0],X_test.shape[0]))
70     print("Training with number of n samples = %d" % int(len(
X_train) * fr))
71
72     ### select only a fraction (ex:10%) of the test data (X_train
y_train)
73     #fr = 0.1 # fraction of test set (ex 10%)
74     # Calculate the number of elements to select (ex:10% of 1000)
75     num_elements_to_select = int(len(X_train) * fr)
76
77     # Generate random indices
78     random_indices = np.random.choice(len(X_train),
num_elements_to_select, replace=False)
79
80     #print("Original X array:", X_train)
81     #print("Original y array:", y_train)
82
83     # Select the elements using the random indices
84     X_train = X_train[random_indices]

```

```

85     y_train = y_train[random_indices]
86
87     #print(f"Selected {fr} of X data:", X_train)
88     #print(f"Selected {fr} of y data:", y_train)
89
90     ###
91
92
93     model = LinearRegression()
94
95     print("Fitting linear model...",end="")
96     model.fit(X_train, y_train)
97     print(" done")
98
99
100    # Compute RMSE on train and test sets
101    rmse_train = rmse(y_train,model.predict(X_train))
102    rmse_test = rmse(y_test,model.predict(X_test))
103    rmse_train_array.append(rmse_train)
104    rmse_test_array.append(rmse_test)
105    training_length_array.append(len(y_train))
106
107    print("Train RMSE = %f, Test RMSE = %f" % (rmse_train,
rmse_test))
108
109
110    print("Model parameters:")
111    print("\t Intercept: %3.5f" % model.intercept_,end="")
112    for i,val in enumerate(model.coef_):
113        print(",  $\beta$ d: %3.5f" % (i,val), end="")
114    print("\n")
115    #print(f'x1',X_test)
116
117    #Plot the train and test RMSE as a function of the number of
training samples given to model
118
119    plt.figure(figsize=(8, 6))
120    plt.plot(training_length_array, rmse_train_array, marker='o',
label='Train RMSE')
121    #plt.plot(training_length_array, rmse_test_array, marker='s',
label='Test RMSE')
122    plt.xlabel("Training Set Size: number of training samples")
123    plt.ylabel("RMSE")
124    plt.title(f"Training and Test RMSE vs Training Set Size (Data
from {psfx})")

```

```
125 plt.legend()  
126 plt.grid(True)  
127 plt.show()  
128  
129  
130  
131
```