

Section 05: Solutions

1. K-fold Cross-Validation (Demonstrative code)

```
1 # Given dataset of 1000-by-50 feature matrix X, and 1000-by-1 labels vector
2 import numpy as np
3
4 X = np.random.random((1000,50))
5 y = np.random.random((1000,))
6
7 def fit(Xin, Yin, lbda):
8     mu = np.mean(Xin, axis=0)
9     Xin = Xin - mu
10    w = np.linalg.solve(np.dot(Xin.T, Xin) + lbda, np.dot(Xin.T, Yin))
11    b = np.mean(Yin) - np.dot(w, mu)
12    return w, b
13
14
15 def predict(w, b, Xin):
16     return np.dot(Xin, w) + b
17
18
19 # Note: X, y are all the data and labels for the entire experiments
20 # We first split the data into the training set and test set.
21 N_SAMPLES = X.shape[0]
22 idx = np.random.permutation(N_SAMPLES)
23 K_FOLD = 5
24
25 # We use an array of randomized indices to slice the data into the training and test sets.
26 NON_TEST = idx[0: 9 * N_SAMPLES // 10]
27 N_PER_FOLD = len(NON_TEST) // K_FOLD
28 TEST = idx[9 * N_SAMPLES // 10::]
29
30 # regularization coefficient candidates to choose from
31 lbdas = [0.1, 0.2, 0.3]
32 err = np.zeros(len(lbdas))
33
34
35 for lbda_idx, lbda in enumerate(lbdas):
36     for i in range(K_FOLD):
37         # CRUCIAL: we use slicing to calculate the indices the training set and validation set should use!
38         # Using the ith fold as the validation set
39         VAL = NON_TEST[i * N_PER_FOLD:(i+1) * N_PER_FOLD]
40         # Using the rest as the train set
41         TRAIN = np.concatenate((NON_TEST[:i * N_PER_FOLD], NON_TEST[(i + 1) * N_PER_FOLD:]))
42
43         ytrain = y[TRAIN]
44         Xtrain = X[TRAIN]
45         yval = y[VAL]
46         Xval = X[VAL]
47
48         w, b = fit(Xtrain, ytrain, lbda)
49         yval_hat = predict(w, b, Xval)
50         # accumulate error from this fold of validation set
51         err[lbda_idx] += np.mean((yval_hat - yval)**2)
52
53 # calculate the error for the k-fold validation
54 err[lbda_idx] /= K_FOLD
```

```
55
56 # After trying all candidates for the regularization coefficient, we select the best lambda.
57 lbda_best = lbdas[np.argmin(err)]
58
59 # Fit the model again using all training data from CV.
60 Xtot = np.concatenate((Xtrain, Xval), axis=0)
61 ytot = np.concatenate((ytrain, yval), axis=0)
62
63 w, b = fit(Xtot, ytot, lbda_best)
64
65 ytest = y[TEST]
66 Xtest = X[TEST]
67
68 # Predict values using model fit on entire training set and the separate test set, and report error.
69 ytot_hat = predict(w, b, Xtot)
70 train_error = np.mean((ytot_hat - ytot) ** 2)
71 ytest_hat = predict(w, b, Xtest)
72 test_error = np.mean((ytest_hat - ytest) ** 2)
73
74 print('Best choice of lambda = ', lbda_best)
75 print('Train error = ', train_error)
76 print('Test error = ', test_error)
```

2. Lasso and CV (Demonstrative Code)

```
1 import numpy as np
2
3 LR = 0.01
4 NUM_ITERATIONS = 500
5
6 # NOTE: here, X and Y represent only the training data, not the overall dataset (train + test).
7 X = np.random.random((1000, 50))
8 Y = np.random.random((1000,))
9
10 def predict(w, b, Xin):
11     return np.dot(Xin, w) + b
12
13 def fit(Xin, Yin, l1_penalty) :
14     # no_of_training_examples, no_of_features
15     m, n = Xin.shape
16
17     # weight initialization
18     w = np.zeros(n)
19     b = 0
20
21     # gradient descent learning
22     for i in range(NUM_ITERATIONS) :
23         w, b = update_weights(w, b, Xin, Yin, l1_penalty)
24
25     return w, b
26
27 def update_weights(w, b, Xin, Yin, l1_penalty) :
28     m, n = Xin.shape
29     Y_pred = predict(w, b, Xin)
30
31     # calculate gradients
32     dW = np.zeros(n)
33     for j in range(n) :
34         if w[j] > 0 :
35             dW[j] = ( - ( 2 * ( Xin[:, j] ).dot(Yin - Y_pred))
36                     + l1_penalty ) / m
37         else :
38             dW[j] = ( - ( 2 * ( Xin[:, j] ).dot(Yin - Y_pred))
39                     - l1_penalty ) / m
40
41     db = - 2 * np.sum(Yin - Y_pred) / m
42
43     # update weights
44     w = w - LR * dW
45     b = b - LR * db
46
47     return w, b
48
49 def rmse_lasso(w, b, Xin, Yin):
50     Y_pred = predict(w, b, Xin)
51     return rmse(Yin, Y_pred)
52
53 def rmse(a, b):
54     return np.sqrt(np.mean(np.square(a - b)))
55
56 # candidate values for l1 penalty
57 l1_penalties = 10 ** np.linspace(-5, -1)
58 err = np.zeros(len(l1_penalties))
59
```

```

60 # We will perform 10-fold CV. Here, we will create the training and validation sets by
61 # creating an indices array with randomized index values to use when slicing our training data.
62 k_fold = 10
63 num_samples = len(X) // k_fold
64 indices = np.random.permutation(len(X))
65
66 for idx, l1_penalty in enumerate(l1_penalties):
67     for k in range(k_fold): #10-fold CV
68         # slice larger training set into validation and training sets for each fold
69         VAL = indices[k * num_samples : (k + 1) * num_samples]
70         TRAIN = np.concatenate((indices[: k * num_samples], indices[(k + 1) * num_samples:]))
71
72         x_train_fold = X[TRAIN]
73         y_train_fold = Y[TRAIN]
74
75         x_val_fold = X[VAL]
76         y_val_fold = Y[VAL]
77
78         w, b = fit(x_train_fold, y_train_fold, l1_penalty)
79
80         # accumulate error from this fold of validation set
81         err[idx] += rmse_lasso(w, b, x_val_fold, y_val_fold)
82
83         #calculate error for kth fold
84         err[idx]/=k_fold
85
86 l1_penalty_best = l1_penalties[np.argmin(err)]
87
88 print('Best choice of l1_penalty = ', l1_penalty_best)

```

3. Subgradients

We start with the definition of subgradients before discussing the motivation and its usefulness.

Definition 1 (subgradients). A vector $g \in \mathbb{R}^d$ is a subgradient of a convex function $f : D \rightarrow \mathbb{R}$ at $x \in D \subseteq \mathbb{R}^d$ if

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y \in D.$$

One interpretation of subgradient g is that the affine function (of y) $f(x) + g^T(y - x)$ is a global underestimator of f . Note that if a convex function f is differentiable at x (i.e., $\nabla f(x)$ exists), then $f(y) \geq f(x) + \nabla f(x)^T(y - x)$ is true for all $y \in D$, meaning that $\nabla f(x)$ is a subgradient of f at x . But a subgradient can exist even when f is not differentiable at x .

- (a) Why are subgradients useful in optimization? If $g = 0$ is a subgradient of a function f at x^* , what does it imply? **Solution:**

Consider the problem of minimizing a function f . If f is differentiable, we know that $\nabla f(x) = 0$ is a necessary condition for x to be a local extremum. Together with the convexity of f , $\nabla f(x) = 0$ becomes a sufficient condition for x to be a local minimizer, and hence global minimizer. If solving for $\nabla f(x) = 0$ analytically is difficult or infeasible, we have numerical methods such as gradient descent to obtain the solution(s). These results are very useful in minimizing a convex differentiable function and subgradients can be treated as a generalization to situations when the underlying function (to be minimized) is nondifferentiable. In the analytical aspect, if $g = 0$ is a subgradient of f at x^* , then from the definition above, we have

$$f(y) \geq f(x^*) + g^T(y - x^*) = f(x^*) \quad \text{for all } y \in D,$$

indicating that $f(x^*)$ is a global minimum. To obtain the solution(s) numerically, we can consider subgradient descent.

- (b) What are the subgradients of $f(x) = \max(x, x^2)$ at 0, with $x \in \mathbb{R}$? (Hint: draw a picture and note that subgradients at a point might not be unique)

Solution:

From definition, a scalar g is a subgradient of f at $x = 0$ if $\max(y, y^2) = f(y) \geq f(0) + g(y - 0) = gy$ for any $y \in \mathbb{R}$. Solving $\max(y, y^2) \geq gy$ yields that $g \in [0, 1]$. That is, for any $g \in [0, 1]$, g is a subgradient of f at 0.

- (c) Some important results about subgradients are

- If f is convex, then a subgradient of f at $x \in \text{int}(D)$ (interior of the domain of f) always exists.
- If f is convex, then f is differentiable at x if and only if $\nabla f(x)$ is the only subgradient of f at x .
- A point x^* is a global minimizer of a function f (not necessarily convex) if and only if $g = 0$ is a subgradient of f at x^* .