Section 05: Solutions

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

```python
# Given dataset of 1000-by-50 feature matrix X, and 1000-by-1 labels vector
import numpy as np

X = np.random.random((1000,50))
y = np.random.random((1000,))

def fit(Xin, Yin, lbda):
    mu = np.mean(Xin, axis=0)
    Xin = Xin - mu
    w = np.linalg.solve(np.dot(Xin.T, Xin) + lbda, np.dot(Xin.T, Yin))
    b = np.mean(Yin) - np.dot(w, mu)
    return w, b


def predict(w, b, Xin):
    return np.dot(Xin, w) + b


# Note: X, y are all the data and labels for the entire experiments
# We first split the data into the training set and test set.
N_SAMPLES = X.shape[0]
idx = np.random.permutation(N_SAMPLES)
K_FOLD = 5

# We use an array of randomized indices to slice the data into the training and test sets.
NON_TEST = idx[0: 9 * N_SAMPLES // 10]
N_PER_FOLD = len(NON_TEST) // K_FOLD
TEST = idx[9 * N_SAMPLES // 10::]

# regularization coefficient candidates to choose from
lbdas = [0.1, 0.2, 0.3]
err = np.zeros(len(lbdas))


for lbda_idx, lbda in enumerate(lbdas):
    for i in range(K_FOLD):
        # CRUCIAL: we use slicing to calculate the indices the training set and validation set should use!
        # Using the ith fold as the validation set
        VAL = NON_TEST[i * N_PER_FOLD:(i+1) * N_PER_FOLD]
        # Using the rest as the train set
        TRAIN = np.concatenate((NON_TEST[:i * N_PER_FOLD], NON_TEST[(i + 1) * N_PER_FOLD:]))

        ytrain = y[TRAIN]
        Xtrain = X[TRAIN]
        yval = y[VAL]
        Xval = X[VAL]

        w, b = fit(Xtrain, ytrain, lbda)
        yval_hat = predict(w, b, Xval)
        # accumulate error from this fold of validation set
        err[lbda_idx] += np.mean((yval_hat - yval)**2)

    # calculate the error for the k-fold validation
    err[lbda_idx] /= K_FOLD
```

```
    # After trying all candidates for the regularization coefficient, we select the best lambda.
    lbda_best = lbdas[np.argmin(err)]

    # Fit the model again using all training data from CV.
    Xtot = np.concatenate((Xtrain, Xval), axis=0)
    ytot = np.concatenate((ytrain, yval), axis=0)

    w, b = fit(Xtot, ytot, lbda_best)

    ytest = y[TEST]
    Xtest = X[TEST]

    # Predict values using model fit on entire training set and the separate test set, and report error.
    ytot_hat = predict(w, b, Xtot)
    train_error = np.mean((ytot_hat - ytot) ** 2)
    ytest_hat = predict(w, b, Xtest)
    test_error = np.mean((ytest_hat - ytest) ** 2)

    print('Best choice of lambda = ', lbda_best)
    print('Train error = ', train_error)
    print('Test error = ', test_error)
```

2. Lasso and CV (Demonstrative Code)

```
import numpy as np

LR = 0.01
NUM_ITERATIONS = 500

# NOTE: here, X and Y represent only the training data, not the overall dataset (train + test).
X = np.random.random((1000, 50))
Y = np.random.random((1000,))

def predict(w, b, Xin):
   return np.dot(Xin, w) + b

def fit(Xin, Yin, l1_penalty) :
   # no_of_training_examples, no_of_features
   m, n = Xin.shape

   # weight initialization
   w = np.zeros(n)
   b = 0

   # gradient descent learning
   for i in range(NUM_ITERATIONS) :
      w, b = update_weights(w, b, Xin, Yin, l1_penalty)

   return w, b

def update_weights(w, b, Xin, Yin, l1_penalty) :
   m, n = Xin.shape
   Y_pred = predict(w, b, Xin)

   # calculate gradients
   dW = np.zeros(n)
   for j in range(n) :
      if w[j] > 0 :
         dW[j] = ( - ( 2 * ( Xin[:, j] ).dot(Yin - Y_pred))
              + l1_penalty ) / m
      else :
         dW[j] = ( - ( 2 * ( Xin[:, j] ).dot(Yin - Y_pred))
              - l1_penalty ) / m

   db = - 2 * np.sum(Yin - Y_pred) / m

   # update weights
   w = w - LR * dW
   b = b - LR * db

   return w, b

def rmse_lasso(w, b, Xin, Yin):
    Y_pred = predict(w, b, Xin)
    return rmse(Yin, Y_pred)

def rmse(a, b):
    return np.sqrt(np.mean(np.square(a - b)))

# candidate values for l1 penalty
l1_penalties = 10 ** np.linspace(-5, -1)
err = np.zeros(len(l1_penalties))

```

```python
# We will perform 10-fold CV. Here, we will create the training and validation sets by
# creating an indices array with randomized index values to use when slicing our training data.
k_fold = 10
num_samples = len(X) // k_fold
indices = np.random.permutation(len(X))

for idx, l1_penalty in enumerate(l1_penalties):
    for k in range(k_fold): #10-fold CV
        # slice larger training set into validation and training sets for each fold
        VAL = indices[k * num_samples : (k + 1) * num_samples]
        TRAIN = np.concatenate((indices[: k * num_samples], indices[(k + 1) * num_samples:]))

        x_train_fold = X[TRAIN]
        y_train_fold = Y[TRAIN]

        x_val_fold = X[VAL]
        y_val_fold = Y[VAL]

        w, b = fit(x_train_fold, y_train_fold, l1_penalty)

        # accumulate error from this fold of validation set
        err[idx] += rmse_lasso(w, b, x_val_fold, y_val_fold)

    #calculate error for kth fold
    err[idx]/=k_fold

l1_penalty_best = l1_penalties[np.argmin(err)]

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 $f : D \longrightarrow \mathbb{R}$ at $x \in D \subseteq \mathbb{R}^d$ if*

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

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^*.

4. Convexity

Convexity is defined for both sets and functions. For today we'll focus on discussing the convexity of functions.

Definition 2 (convex functions). *A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** on a set A if for all $x, y \in A$ and $\lambda \in [0, 1]$:*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y)$$

When this definition holds with the inequality being reversed, then f is said to be concave. From the definition, it is clear that a function f is convex if and only if $-f$ is concave.

(a) Why do we care whether a function is convex or not?

Solution:

Many numerical methods or algorithms were developed for finding local minima while in machine learning we are typically interested in finding the global minimum. Convex functions are useful because local minima are always the global minimum. Here is a short proof of this result: let x^* is a local minimizer for a convex function f and suppose $f(x_0) < f(x^*)$. Now note that there exists a $\lambda \in (0, 1)$ such that $y = \lambda x^* + (1 - \lambda)x_0$ and $f(y) \geq f(x^*)$ (i.e., y is close to x^* enough). We now have a contradiction:

$$f(y) \leq \lambda f(x^*) + (1 - \lambda)f(x_0) < f(x^*) \leq f(y).$$

In words, a line segment between any arbitrary point x_0 and a local minimizer x^* should be entirely above the function by definition of convexity, ensuring that $f(x_0) < f(x^*)$ cannot happen.

(b) Which of the following functions are convex? (Hint: draw a picture!)

(i) $|x|$

(ii) $\cos(x)$

(iii) $x^T x$

Solution:

$|x|$ and $x^T x$ are both convex. $\cos(x)$ is not convex since we can draw a line at two points (from say $\frac{\pi}{2}$ to $2\pi + \frac{\pi}{2}$) that is not entirely above the function.

Proof that $|x|$ is convex:

$$\begin{aligned} f(\lambda x + (1 - \lambda)y) &= |\lambda x + (1 - \lambda)y| \\ &\leq \lambda|x| + (1 - \lambda)|y| \end{aligned}$$

Proof that $x^T x$ is convex:
We begin by examining the definition: whenever $\lambda \in [0, 1]$, we have

$$\begin{aligned} (\lambda x + (1 - \lambda)y)^T(\lambda x + (1 - \lambda)y) &= \lambda^2 x^T x + (1 - \lambda)^2 y^T y + 2\lambda(1 - \lambda)x^T y \\ &= \lambda x^T x + (1 - \lambda)y^T y - \lambda(1 - \lambda)(x^T x - 2x^T y + y^T y) \\ &= \lambda x^T x + (1 - \lambda)y^T y - \lambda(1 - \lambda)(x - y)^T(x - y) \\ &\leq \lambda x^T x + (1 - \lambda)y^T y, \end{aligned}$$

where the inequality holds because $(x - y)^T(x - y) = ||x - y||_2^2 \geq 0$. So our function is convex.

(c) Can a function be both convex and concave on the same set? If so, give an example. If not, describe why not.

Solution:

Linear functions (i.e. functions such that $f(\lambda x + (1 - \lambda)y) = \lambda f(x) + (1 - \lambda)f(y)$) are both convex and concave.

5. Practical Methods for Checking Convexity

Using the definition to check whether a function is convex or not can be a tedious task in many situations. Some basic methods that can help us achieve the task in an efficient way are introduced below:

- for differentiable function, examine $f(y) \geq f(x) + \nabla f(x)^T(y - x)$ for any x, y in the domain of f.
- for twice differentiable functions, examine $\nabla^2 f(x) \succeq 0$ (i.e., the Hessian matrix is positive semidefinite).
- nonnegative weighted sum
- composition with affine function
- pointwise maximum and supremum

Note: there are even more such methods, which are covered in a convex optimization course or textbook.

(a) If f is differentiable, then f is convex if and only if $f(y) \geq f(x) + \nabla f(x)^T(y - x)$ for any x, y in the domain of f. A geometric interpretation of this characterization is that any tangent plane of a convex function f must lie entirely below f. One interesting application of this characterization is one of the most important inequalities in probability and statistics: the Jensen's inequality, which states that $\mathbb{E}f(X) \geq f(\mathbb{E}(X))$ when f is convex. Prove Jensen's inequality using the other inequality mentioned here.

Solution:

Let $\mu = \mathbb{E}(X)$, then since f is convex, we have

$$f(X) \geq f(\mu) + \nabla f(\mu)^T(X - \mu)$$

with probability 1. This means that taking expectation on both sides preserves the inequality: $\mathbb{E}f(X) \geq f(\mu) = f(\mathbb{E}X)$.

(b) If f is twice differentiable with convex domain, then f is convex if and only if

$$\nabla^2 f(x) \succeq 0,$$

for any x in the domain of f. Use this method to show that the objective function in linear regression is convex.

Solution:

Let $f(w) = (Y - Xw)^T(Y - Xw)$, then

$$\nabla^2 f(w) = 2(X^T X),$$

which is clearly a positive semidefinite matrix.

(c) Let $\alpha \geq 0$ and $\beta \geq 0$, and if f and g are convex, then αf, $f + g$, $\alpha f + \beta g$ are all convex. One application: When a (possibly complicated) objective function can be expressed as a sum (e.g., the negative log-likelihood function), then showing the convexity of each individual term is typically easier.

Solution:

From definition, αf and $f + g$ are easily proved convex. To show that $\alpha f + \beta g$ is convex, first note that αf and βg are both convex, hence their sum is convex as well.

(d) Suppose $f(\cdot)$ is convex, then $g(x) := f(Ax + b)$ is convex. Use this method to show that $||Ax + b||_1$ is convex (in x), where $||z||_1 = \sum_i |z_i|$.

Solution:

With this method, we only need to show the convexity of $||x||_1$. This is true from definition by observing that

$$||\lambda x + (1-\lambda)y||_1 = \sum_i |\lambda x_i + (1-\lambda)y_i| \leq \sum_i \lambda|x_i| + (1-\lambda)|y_i| = \lambda||x||_1 + (1-\lambda)||y||_1,$$

where the inequality holds because of triangular inequality.

(e) Suppose you know that f_1 and f_2 are convex functions on a set A. The function $g(x) := \max\{f_1(x), f_2(x)\}$ is also convex on A. In general, if $f(x, y)$ is convex in x for each y, then $g(x) := \sup_y f(x, y)$ is convex. Use this method to show that the largest eigenvalue of a matrix X, $\lambda_{\text{Max}}(X)$, is convex in X (Using the definition of convexity would make this question quite difficult).

Solution:

Consider $f(v, X) := v^T X v$, then for each v, we have

$$f(v, \lambda X + (1-\lambda)Y) = \lambda f(v, X) + (1-\lambda) f(v, Y),$$

suggesting that $f(v, X)$ is convex in X for each v. Then $g(X) := \lambda_{\text{Max}}(X) = \sup_{||v||_2=1} f(v, X)$ is convex in X using this method.

(f) Does the same result hold for $h(x) := \min\{f_1(x), f_2(x)\}$? If so, give a proof. If not, provide convex functions f_1, f_2 such that h is not convex.
Solution:

No, consider $f_1(x) = x^2$, $f_2(x) = (x-1)^2$. Then $h(0) = h(1) = 0$, but $h(0.5) = 0.25$, so $h(0.5*0+0.5*1) = 0.25 > 0 = 0.5 * h(0) + 0.5 * h(1)$. So the minimum of two convex functions is not convex in general.

6. Gradient Descent

We will now examine gradient descent algorithm and study the effect of learning rate α on the convergence of the algorithm. Recall from lecture that Gradient Descent takes on the form of $x_{t+1} = x_t - \alpha \nabla f$

(a) Assume that $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable, and additionally,

$$||\nabla f(x) - \nabla f(y)|| \leq L||x - y|| \text{ for any } x, y$$

I.e., ∇f is Lipschitz continuous with constant $L > 0$
Show that:
Gradient descent with fixed step size $\eta \leq \frac{1}{L}$ satisfies

$$f(x^{(k)}) - f(x^*) \leq \frac{||x^{(0)} - x^*||^2}{2\eta k}$$

I.e., gradient descent has convergence rate $O(\frac{1}{k})$

Hints:

(i) ∇f is Lipschitz continuous with constant $L > 0 \rightarrow f(y) \leq f(x) + \nabla f(x)(y - x) + \frac{L}{2}||y - x||^2$ for all x, y.

(ii) f is convex $\rightarrow f(x) \leq f(x^*) + \nabla f(x)(x - x^*)$, where x^* is the local minima that the gradient descent algorithm is converging to.

(iii) $2\eta\nabla f(x)(x - x^*) - \eta^2||\nabla f(x)||^2 = ||x - x^*||^2 - ||x - \eta\nabla f(x) - x^*||^2$

Solution:

Proof:
For any positive integer k, $x^{(k)} = x^{(k-1)} - \eta\nabla f(x)$, according to the gradient descent algorithm.
By hint(1), we have
$f(x^{(k)}) \leq f(x^{(k-1)}) + \nabla f(x^{(k-1)})(x^{(k)} - x^{(k-1)}) + \frac{L}{2}||(x^{(k)} - x^{(k-1)}||^2$
$= f(x^{(k-1)}) - \eta\nabla f(x^{(k-1)})^2 + \frac{L}{2}\eta^2\nabla f(x^{(k-1)})^2$
$\leq f(x^{(k-1)}) + (-\eta + \frac{\eta}{2})\nabla f(x^{(k-1)})^2$ (Since $\eta \leq \frac{1}{L}$)
$= f(x^{(k-1)}) - \frac{\eta}{2}\nabla f(x^{(k-1)})^2$
$\leq f(x^*) + \nabla f(x^{(k-1)})(x^{(k-1)} - x^*) - \frac{\eta}{2}\nabla f(x^{(k-1)})^2$ (By hint(2))
$= f(x^*) + \frac{1}{2\eta}(2\eta\nabla f(x^{(k-1)})(x^{(k-1)} - x^*) - \eta^2\nabla f(x^{(k-1)})^2)$
$\leq f(x^*) + \frac{1}{2\eta}(||x^{(k-1)} - x^*||^2 - ||x^{(k-1)} - \eta\nabla f(x^{(k-1)}) - x^*||^2)$ (By hint(3))
$= f(x^*) + \frac{1}{2\eta}(||x^{(k-1)} - x^*||^2 - ||x^{(k)} - x^*||^2)$

Hence, we have

$$f(x^{(k)}) - f(x^*) \leq \frac{1}{2\eta}(||x^{(k-1)} - x^*||^2 - ||x^{(k)} - x^*||^2)$$

Adding up from 1 to k:
$\Sigma_{i=1}^{k}[f(x^{(i)}) - f(x^*)] \leq \Sigma_{i=1}^{k}\frac{1}{2\eta}(||x^{(i-1)} - x^*||^2 - ||x^{(i)} - x^*||^2)$
$\Sigma_{i=1}^{k}f(x^{(i)}) - kf(x^*) \leq \frac{1}{2\eta}(||x^{(0)} - x^*||^2 - ||x^{(k)} - x^*||^2) \leq \frac{1}{2\eta}(||x^{(0)} - x^*||^2)$
Since $f(x^{(k)}) \leq f(x^{(k-1)})$, $f(x^{(k)}) \leq \frac{1}{k}\Sigma_{i=1}^{k}f(x^{(i)})$

Hence,

$$f(x^{(k)}) - f(x^*) \leq \frac{1}{2k\eta}(||x^{(0)} - x^*||^2)$$