Section 05: Convexity and Gradient Descent

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?

(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)

(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?

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

(i) $|x|$

(ii) $\cos(x)$

(iii) $x^T x$

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

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.

(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.

(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.

(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 $||z||_1 = \sum_i |z_i|$.

(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).

(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.

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