ROB313: Assignment 2

Sidhanth Moolayil
1004101679
March 3, 2020

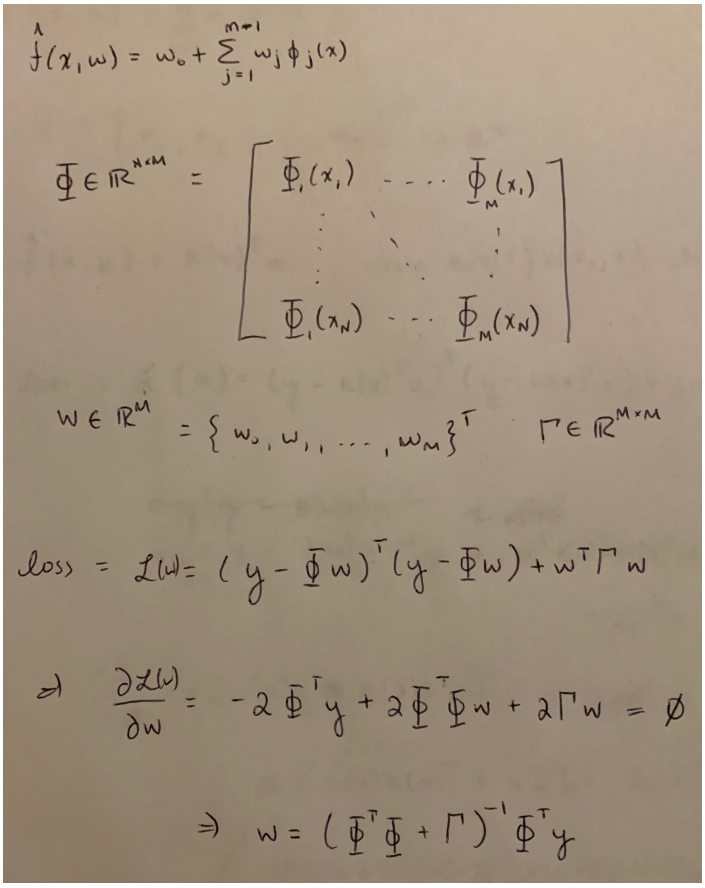
1. Objectives

The purpose of this assignment is to introduce generalized linear models, and the different methods to solve for them computationally. This includes formulating optimization problems by deriving a closed form expression of the weights to be solved, using the radial basis function model to computationally build a prediction model with the Gaussian Kernel, and designing the basis functions for a model to select and be capable of using to accurately predict unseen data. In addition, the practice of tuning hyperparameters such as the shape parameter and regularization parameter was also explored.

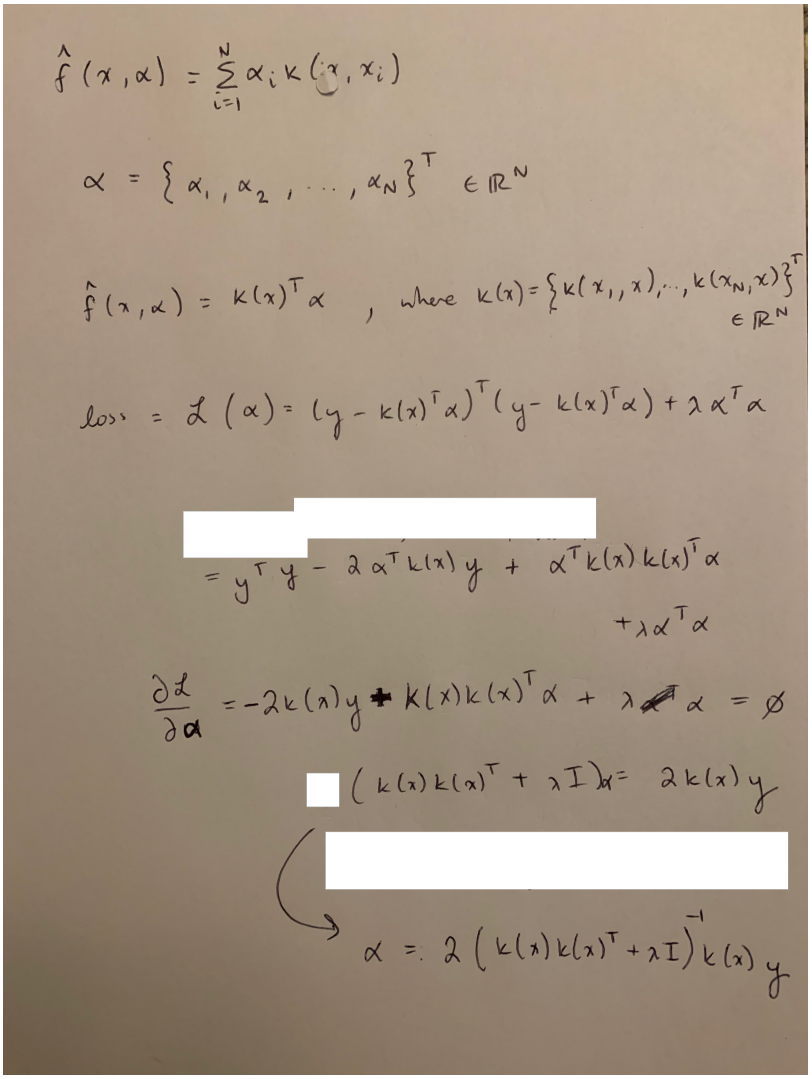
2. Code Structure and Employed Strategies

The structure of the code is formatted with several helper functions answering individual question parts, with the main code below where the user can control which questions should be run, as well as the parameters to be used by the model.

3.1. Question 1



3.2. Question 2



$$\hat{f}(x,\alpha) = \sum_{i=1}^{N} \alpha_i k(x, x_i)$$

$$\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}^T \in \mathbb{R}^N$$

$$\hat{f}(x,\alpha) = k(x)^T \alpha \quad , \text{ where } k(x) = \{k(x_1, x), \cdots, k(x_N, x)\}^T \in \mathbb{R}^N$$

$$\text{loss} = \mathcal{L}(\alpha) = (y - k(x)^T\alpha)^T (y - k(x)^T\alpha) + \lambda \alpha^T \alpha$$

$$= y^T y - 2\alpha^T k(x) y + \alpha^T k(x) k(x)^T \alpha + \lambda \alpha^T \alpha$$

$$\frac{\partial \mathcal{L}}{\partial \alpha} = -2k(x)y + k(x)k(x)^T\alpha + \lambda\alpha = \emptyset$$

$$(k(x)k(x)^T + \lambda I)\alpha = 2k(x)y$$

$$\alpha = 2\left(k(x)k(x)^T + \lambda I\right)^{-1} k(x) y$$

Compared to the expression found in class, this expression for the weights appears different.

3.3. Question 4

Data Table 1: Test Data Results for Radial Basis Function Model

Dataset	Testing RMSE
mauna_loa	0.1497733877195412
rosenbrock	0.1481244275513203
	Testing Accuracy
iris	0.8709677419354839

3.4. Question 5

Upon observing the mauna_loa dataset, two characteristics can be observed. First, an almost linear increase in y over x, as well as periodic oscillations. To fit this behavior, the 502 basis functions include 250 sine and 250 cosine functions of varying frequencies to model the regular oscillation, and a linear and quadratic basis function to model the general increase.

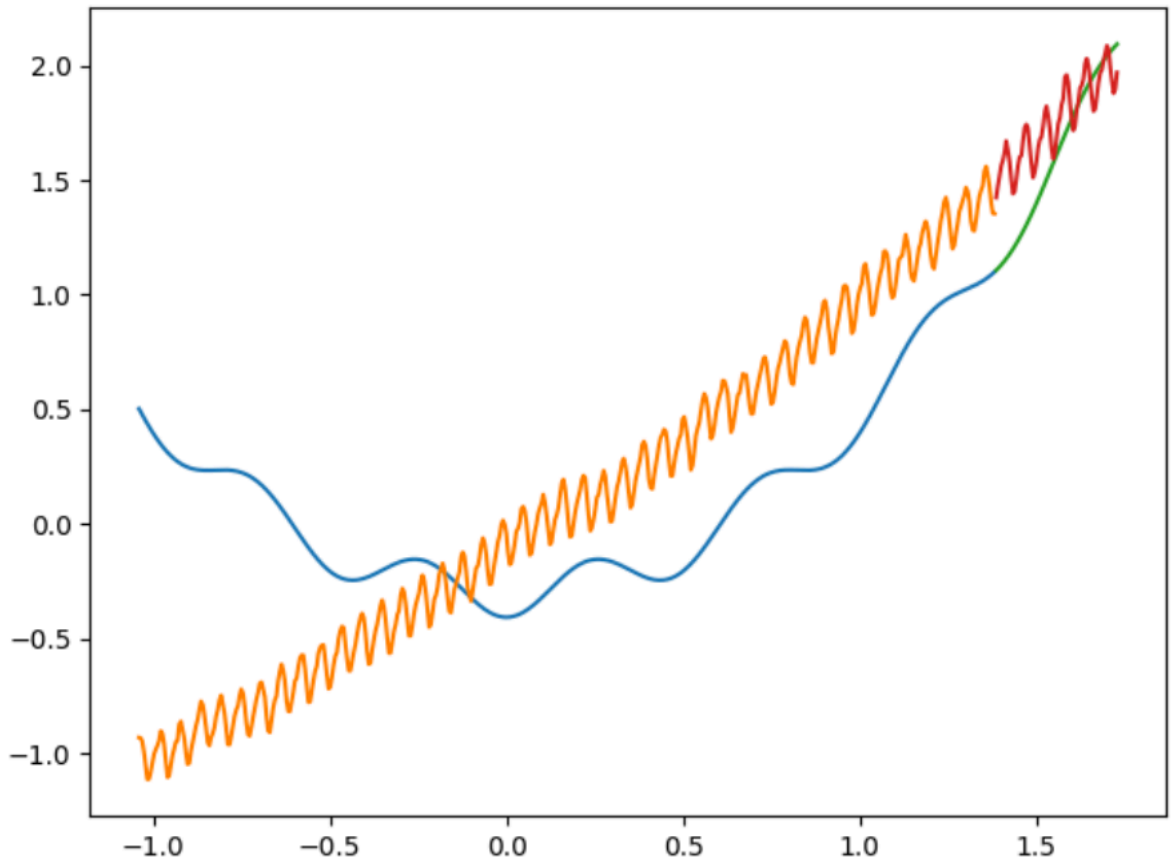


Figure 1. Model Prediction (blue) vs mauna loa data. Testing RMSE: 0.24142689375920817, 3 basis functions used

Appendix A: Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import cho_factor, cho_solve, svd
from data_utils import load_dataset

def Gaussian_Kernel(x, z, theta):
    x = np.expand_dims(x, axis=1)
    z = np.expand_dims(z, axis=0)
    return np.exp(-np.sum(np.square(x - z) / theta, axis=2, keepdims=False))

def RBFregression(dataset, shape_params, reg_params):

    # load the dataset
    if dataset == 'rosenbrock':
        x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset(dataset,
d=2, n_train=1000)
    else:
        x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset(dataset)

    #initialization of minimum loss
    min_valid_loss = 1.0
```

```
    for lam in reg_params:
        for theta in shape_params:

            # dual parameters
            C = cho_factor(Gaussian_Kernel(x_train, x_train, theta) +
lam*np.identity(x_train.shape[0]))
            alpha = cho_solve(C, y_train)

            # prediction on validation data
            y_valid_pred = Gaussian_Kernel(x_valid, x_train, theta).dot(alpha)

            # evalate on the validation set to get best theta and lambda values
            valid_loss = np.linalg.norm(y_valid_pred-
y_valid)/np.sqrt(y_valid.shape[0])
            if valid_loss < min_valid_loss:
                min_valid_loss = valid_loss
                min_theta = theta
                min_lambda = lam

    # combine the validation set with training
    x_train_val = np.vstack([x_valid, x_train])
    y_train_val = np.vstack([y_valid, y_train])

    # get the dual parameters for the final model
    C = cho_factor(Gaussian_Kernel(x_train_val, x_train_val, min_theta) + min_lambda *
np.identity(x_train_val.shape[0]))
    alpha = cho_solve(C, y_train_val)

    # evalate on the test set
    y_pred = Gaussian_Kernel(x_test, x_train_val, min_theta).dot(alpha)

    test_loss = np.linalg.norm(y_pred - y_test) / np.sqrt(y_test.shape[0])
    print(dataset)
    #print('theta = ', min_theta, ' lambda = ', min_lambda, ' RMSE = ',
min_valid_loss)
    print('Test RMSE: ', test_loss)

def RBFclassification(dataset, shape_params, reg_params):

    # load the dataset
    x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset(dataset)

    #initialization of maximum accuracy
    max_accuracy = 0

    for lam in reg_params:
        for theta in shape_params:

            # dual parameters
            C = cho_factor(Gaussian_Kernel(x_train, x_train, theta) +
lam*np.identity(x_train.shape[0]))
            alpha = cho_solve(C, y_train)

            # prediction on validation data
            y_valid_pred = Gaussian_Kernel(x_valid, x_train, theta).dot(alpha)

            # evalate on the validation set to get best theta and lambda values
            accuracy = np.mean(np.argmax(y_valid_pred, axis=1) == np.argmax(y_valid,
axis=1))
            if accuracy > max_accuracy:
```

```
                max_accuracy = accuracy
                min_theta = theta
                min_lambda = lam

    # combine the validation set with training
    x_train_val = np.vstack([x_valid, x_train])
    y_train_val = np.vstack([y_valid, y_train])

    # get the dual parameters for the final model
    C = cho_factor(Gaussian_Kernel(x_train_val, x_train_val, min_theta) + min_lambda *
np.identity(x_train_val.shape[0]))
    alpha = cho_solve(C, y_train_val)
    # evalate on the test set
    y_pred = Gaussian_Kernel(x_test, x_train_val, min_theta).dot(alpha)

    test_accuracy = np.mean(np.argmax(y_pred, axis=1) == np.argmax(y_test, axis=1))

    print(dataset)
    #print('theta = ', min_theta, ' lambda = ', min_lambda, ' Accuracy = ',
max_accuracy)
    print('Test accuracy: ', test_accuracy)




def MDL(residual, k):
    loss = np.mean(np.square(residual))
    return (residual.size/2)*np.log(loss) + k/2*np.log(residual.size)


def phi_builder(selection, data):
    # custom basis functions
    Phi = data.copy()

    # sine, cosine, linear, and parabolic basis functions considered
    for p in selection:
        if p < 5:
            Phi = np.hstack([Phi, np.power(data, p-1)])
        elif p < 255:
            Phi = np.hstack([Phi, np.sin(2*np.pi * data * 1/(p*0.001))])
        else:
            Phi = np.hstack([Phi, np.cos(2*np.pi * data * 1/(p*0.001))])


    Phi = Phi[:, 1:]
    return Phi

def greedy_regression(x_train, y_train):

    # initialization
    weights = np.zeros((0,1))
    residual = y_train.copy()
    mdl_last = MDL(residual, k=weights.size)

    i_dict = range(1, 503)
    i_selected = np.zeros((0,x_train.shape[1]))

    while True:
        # phi dictionary
        phi_dict = phi_builder(i_dict, x_train)
```

```
        # select and add basis
        [idx_selected] = np.argmax(np.abs(phi_dict.T.dot(residual)), axis=0)
        i_selected_new = np.vstack([i_selected, i_dict[idx_selected-1]])

        Phi = phi_builder(i_selected_new, x_train)

        # find weights for chosen basis functions
        u, s, vh = np.linalg.svd(Phi)
        sigma = np.diag(s)
        sigma_inv = np.linalg.pinv(np.vstack([sigma, np.zeros((len(x_train) - len(s),
len(s)))]))
        weights_new = np.dot(np.transpose(vh), np.dot(sigma_inv,
np.dot(np.transpose(u), y_train)))

        # new residual
        residual_new = Phi.dot(weights_new) - y_train

        # check if to terminate
        mdl_new = MDL(residual_new, k=weights_new.size)
        if mdl_new < mdl_last:
            # remove added basis function from dictionary
            i_dict = np.delete(i_dict, idx_selected, axis=0)
            # update values
            weights, mdl_last, i_selected, residual = weights_new, mdl_new,
i_selected_new, residual_new
        else:
            break
    Phi = phi_builder(i_selected, x_train)
    y_pred = Phi.dot(weights)
    plt.plot(x_train, y_pred)
    plt.plot(x_train, y_train)

    return i_selected, weights



#Question 4
print('QUESTION 4')
shape_params = [0.05, 0.1, 0.5, 1, 2]
reg_params = [0.001, 0.01, 0.1, 1]

RBFregression('mauna_loa', shape_params, reg_params)
RBFregression('rosenbrock', shape_params, reg_params)
RBFclassification('iris', shape_params, reg_params)

#Question 5
print('QUESTION 5')
# load the dataset
x_train, x_valid, x_test, y_train, y_valid, y_test = load_dataset('mauna_loa')
# train the greedy regression model
i_selected, weights = greedy_regression(x_train, y_train)

print((x_train[-1] - x_train[0]))
Phi = phi_builder(i_selected, x_test)


y_pred = Phi.dot(weights)
plt.plot(x_test, y_pred)
plt.plot(x_test, y_test)
plt.show()
```

```
test_rmse = np.linalg.norm(y_pred - y_test)/np.sqrt(y_test.shape[0])

print('test RMSE:', test_rmse, ' Number of Basis Functions: ', weights.size)
```