Assignment 1 (Linear Regression)

CS584 - Machine Learning

Amit Nikam (A20470263)

In [1]:
```
# load required library
from sklearn.datasets import load_diabetes
import matplotlib.pyplot as plt
import numpy as np

# Assignment 1 - Linear Regression
# Package Prereqs: scikit-learn, matplotlib, numpy
# Implement what is asked at the TODO section.
# !! Please notice no library versions of linear regression are allowed.


def load_dataset():
    '''
    ** Do not modify this function. **
    Load diabetes dataset. We only use one feature and 60 instances.
    '''

    X, y = load_diabetes(return_X_y=True)
    return X[:60, 2], y[:60]
```

In [2]:
```
def plot_data(X, y):
    '''
    Draw scatter plot using raw data.
    '''

    ###############################################################################
    # Full Mark: 10                                                               #
    # TODO:                                                                       #
    # 1. make a scatter plot of the raw data                                      #
    # 2. set title for the plot                                                   #
    # 3. set label for X,y axis                                                   #
    # e.g.,                                                                       #
    #https://matplotlib.org/3.2.0/api/_as_gen/matplotlib.pyplot.scatter.html#
    ###############################################################################

    plt.scatter(X, y)
    plt.title("Scatter Plot of X Feature v/s y - Target")
    plt.xlabel("X Feature")
    plt.ylabel("y - Target")

    ###############################################################################
    #                        END OF YOUR CODE                                     #
    ###############################################################################

    # return the plt object
    return plt
```

In [3]:
```
def train_test_split(X, y):
    '''
    Randomly split data into train and test set.
    Two thirds of the raw data will be the train set and one third of the raw data will be the test set.
    '''

    ###############################################################################################
    # Full Mark: 10                                                                               #
    # TODO:                                                                                       #
    # 1. shuffle the indices of data first.                                                       #
    # (Hint: use numpy.arange and numpy.random.shuffle)                                           #
    # 2. select two thirds of the data as train set, the rest of data as test set.                #
    ###############################################################################################

    # dynamically find number of 2/3 train records ~ and 1/3 test
    train_len = int((2/3) * len(X))

    # merge the feature and target columns and shuffle
    mix = np.column_stack((X,y))
    np.random.shuffle(mix)

    # slice the shuffled matrix
    X_train, y_train = mix[:train_len,:2], mix[:train_len,2:]
    X_test, y_test = mix[train_len:,:2], mix[train_len:,2:]

    ###############################################################################
    #                        END OF YOUR CODE                                     #
    ###############################################################################

    return X_train, X_test, y_train, y_test
```

In [4]:
```
def cost_function(weights, X, y):
    '''
    Define the cost function.
    '''

    ###########################################################################
    # Full Mark: 25                                                           #
    # TODO:                                                                   #
    # Implement the Mean Square Error function:                               #
    # https://en.wikipedia.org/wiki/Mean_squared_error#Mean                   #
    #                                                                         #
    # (Hint: Use numpy functions)                                             #
    ###########################################################################

    # find y predicted
    y_hat = np.dot(X, weights)
    y_hat = y_hat[..., np.newaxis]

    # calculate cost from difference between y and y predicted
    y_diff = y_hat - y
    cost = np.sum(y_diff ** 2) / (2 * len(X))

    ###########################################################################
    #                         END OF YOUR CODE                                #
    ###########################################################################

    # return cost
    return cost
```

In [5]:
```
def gradient_descent(weights, X, y):
    '''
    Update weights using gradient descent algorithm.
    '''

    # define your learning_rate and epoch
    lr = 0.1
    epoch = 20000

    # define cost
    cost_list = []

    # for loop
    for i in range(epoch):
        ###########################################################################
        # Full Mark: 25                                                           #
        # TODO:                                                                   #
        # 1. update weights with learning rate lr                                 #
        # 2. append the updated cost to cost list                                 #
        # (Hint: Use numpy functions)                                             #
        ###########################################################################

        # find y predicted
        y_hat = np.dot(X, weights)
        y_hat = y_hat[..., np.newaxis]
        y_diff = y_hat - y

        # gradient and update weights
        grad = np.dot(y_diff.T,X) / len(X)
        weights[0] = weights[0] - lr * grad[0][0]
        weights[1] = weights[1] - lr * grad[0][1]

        # find cost
        cost_list.append(cost_function(weights,X,y))

        ###########################################################################
        #                         END OF YOUR CODE                                #
        ###########################################################################

    # return updated weights and cost list
    return weights, cost_list
```

In [6]:
```
def plot_iteration(cost, epoch=20000):
    '''
    Plot the cost for each iteration.
    '''

    ###########################################################################
    # Full Mark: 10                                                           #
    # TODO:                                                                   #
    # 1. plot the cost for each iteration                                     #
    # 2. set title and labels for the plot                                    #
    # (Hint: Use plt.plot function to plot and range(n))                      #
    ###########################################################################

    plt.scatter(np.arange(epoch), cost, marker='.')
    plt.title("Plot of the cost for each iteration")
    plt.xlabel("iteration")
```

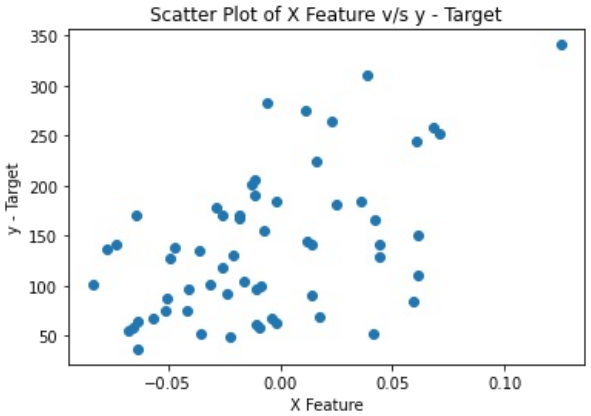
```
    plt.ylabel("cost")

    ##########################################################################
    #                        END OF YOUR CODE                                #
    ##########################################################################

    # show plot
    plt.show()
```

In [7]:
```
def plot_final(weights, X, y):
    '''
    Draw the simple linear regression model.
    '''

    # draw the raw data first
    model_plot = plot_data(X, y)

    ##########################################################################
    # Full Mark: 10                                                          #
    # TODO:                                                                  #
    # 1. create a series of x coordinates in proper range.                   #
    # (Hint: use numpy.arange)                                               #
    # 2. calculate y coordinates:                                            #
    #                           y = w * X + b                                #
    # 3. plot the curve and set title                                        #
    ##########################################################################

    # model
    y_hat = (weights[1] * X) + weights[0]
    y_hat = y_hat[..., np.newaxis]

    # make figure
    model_plot.plot(X, y_hat, 'r--', label='Model')
    model_plot.title("Model with respect to raw data")
    model_plot.xlabel("X Feature")
    model_plot.ylabel("y_hat - predicted target")

    ##########################################################################
    #                        END OF YOUR CODE                                #
    ##########################################################################

    # show plot
    model_plot.show()
```

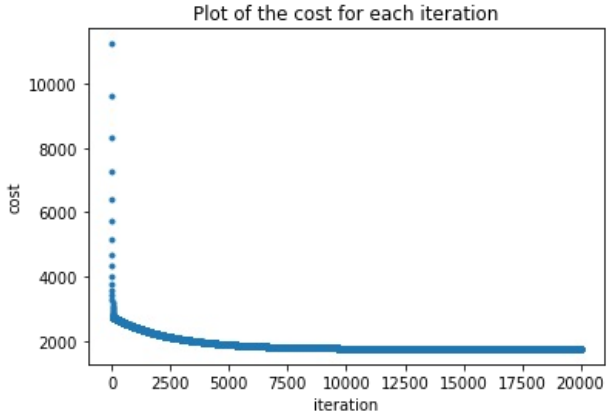
In [8]:
```
def print_test_error(weights, X, y_true):
    '''
    Use optimized weights to predict y, and print test error.
    '''

    ##########################################################################
    # Full Mark: 10                                                          #
    # TODO:                                                                  #
    # 1. predict the target value y of X:                                    #
    #                              y = w * X + b                             #
    # 2. calculate the Mean Square Error using true y and predicted y        #
    ##########################################################################

    # predicted y
    y_hat = (weights[1] * X) + weights[0]
    y_hat = y_hat[..., np.newaxis]

    # error
    y_diff = y_hat - y_true
    error = np.sum(y_diff ** 2) / (2 * len(X))

    ##########################################################################
    #                        END OF YOUR CODE                                #
    ##########################################################################

    # print test error
    print("Test error: %.4f" % error)
    return error
```

In [9]:
```
def main():
    '''
    ** Do not modify this function. **
    '''

    # Plot raw data points
    X, y = load_dataset()
    plot = plot_data(X, y)
    plot.show()

    # Split train and test set
    X = np.c_[np.ones(X.size), X]
    X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
    # initialize weight
    weights = np.ones(X_train.shape[1])

    # calculate training cost
    init_cost = cost_function(weights, X_train, y_train)
    print("Initial cost: %.4f" % init_cost)

    # gradient descent to find the optimal fit
    weights, cost_list = gradient_descent(weights, X_train, y_train)

    # draw the cost change for iterations
    plot_iteration(cost_list)

    # draw the final linear model
    # it is shown as a red line, you can change the color anyway
    plot_final(weights, X_train[:, 1], y_train)

    # Print test error
    print_test_error(weights, X_test[:, 1], y_test)


if __name__ == '__main__':
    main()
```




```
Initial cost: 13228.3690
```




```
Test error: 2197.9936
```