Introduction to Machine Learning

Homework Solution - 2

Instructor: Prof Chinmay Hegde *TA:* Devansh Bisla, Maryam Majzoubi

Problem 1
In class we derived the optimal linear predictor for scalar data, and *wrote down* the optimal lnear predictor for vector data (without proof). Here, let us *derive a proof* for the optimal linear predictor in the vector case. Suppose that $\{x_1, x_2, \ldots, x_n\}$ denote training data samples, where each $x_i \in \mathbb{R}^d$. Suppose $\{y_1, y_2, \ldots, y_n\}$ denote corresponding (scalar) labels.

(a) Show that the mean squared-error loss function for multivariate linear regression can be written in the following form:

$$MSE(w) = \|y - Xw\|_2^2$$

where X is an $n \times (d+1)$ matrix and where the first column of X is all-ones. What is the dimension of w and y? What do the coordinates of w represent?

(b) Theoretically prove that the optimal linear regression weights are given by:

$$\hat{w} = (X^T X)^{-1} X^T y?$$

What algebraic assumptions on X did you make in order to derive the closed form?

(Solution)

(a) Let $v_i^T = [1, x_i^T]$, then MSE is defined as,

$$MSE = \frac{1}{N} \sum_{i=1}^{n} (y_i - v_i^T w)^2$$

Using the definition of 2-norm we can rewrite the equation as,

$$MSE = \frac{1}{N} ||y - Xw||_2^2$$

Where,

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}, X = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}_{n \times (d+1)}, w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d+1} \end{bmatrix}_{(d+1) \times 1}$$

The first co-ordinate of w i.e w_1 represent bias while $[w_2, w_3, \cdots, w_{d+1}]^T$ represent d-dimensional weight vector.

(b) From problem (a) we know that,

$$\begin{aligned} MSE &= \frac{1}{N}||y - Xw||_2^2 \\ &= \frac{1}{N}(y - Xw)^T(y - Xw) \end{aligned}$$

w^* minimizes MSE therefore, gradient of MSE at $w = w^*$ is equal to 0 ($\nabla_w \text{MSE}\Big|_{w=w^*}$=0) i.e

$$\begin{aligned} \nabla_w MSE\Big|_{w=w^*} &= \frac{2}{N}(X^TXw^* - X^Ty) = 0 \qquad \text{[From matrix differentiation rules]} \\ &\implies w^* = (X^TX)^{-1}X^Ty \end{aligned}$$

We assume that X^TX is an invertible matrix.

Problem 2
In class, we argue that convexity together with smoothness is *a sufficient condition* for minimizing a loss function using gradient descent. Is convexity also a *necessary condition* for gradient descent to successfully train a model? Argue why or why not. You can intuitively explain your answer in words and/or draw a figure in support of your explanation. (Hint: What about $f(x) = x^2 + 3\sin^2 x$?)

(Solution) No, convexity is not a necessary condition to use gradient decent method. Because first of all, there are functions where they are not convex but only have one global minimum and no other local minima. For instance, function $f(x) = x^2 + 3\sin^2 x$ has only one point where its gradient is equal to zero and that is its global minimum at $x = 0$. Therefore, the gradient decent method will converge to this point.

Moreover, it is true that theoretically, non-convex functions and having local minima can create a significant issue, as it can lead to a sub-optimal trained model. However, we know that deep learning models usually result in a large number of local minima which perform similarly to each other and to the global minima. So using gradient decent even when we have multiple local minima might not be problematic in practice for such scenarios.

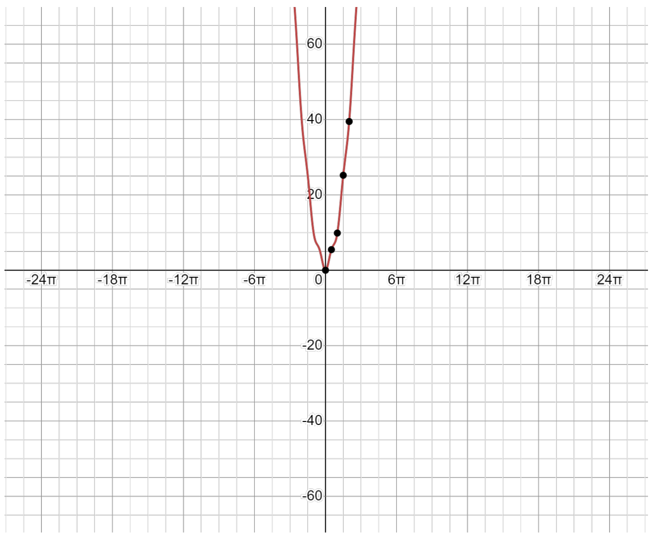

Figure 1: $f(x) = x^2 + 3\sin^2 x$

Problem 3
The goal of this problem is to implement multivariate linear regression from scratch using gradient descent and validate it.

(a) Implement a function for learning the parameters of a linear model for a given tranining data with user-specified learning rate η and number of epochs T. Note: you *cannot* use existing libraries such as sklearn; you need to write it out yourself.

(b) Validate your algorithm on the glucose dataset discussed in Lecture 2. Confirm that the model obtained by running your code gets similar R^2 values as the one produced by sklearn.

(Solution)

HW2_Q3

February 22, 2020

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets

def grad_descent(w, X, y, lr, itr):
    # Author:
    #       Devansh Bisla
    #
    # Function to run gradient descent
    # input:
    #       w       - initial w
    #       X       - Data vector X
    #       y       - Target value (label)
    #      lr       - Learning rate
    #     itr       - No. of iterations
    # output:
    #      w_star   - final solution
    for _ in range(itr):
        # Tip:
        # We can quickly compute matrix multiplication in
        # python using '@' - operator
        #
        # Transpose can be quickly computed with '.T' - operator
        #
        y_hat = X@w
        grad = (X.T)@(yhat-y)  # compute gradient

        w = w - lr*grad       # update

        # Break condition:
        # we check if the gradient is smaller than some threshold
        # if true, we break the loop and return the solution
        # otherwise we keep iterating
        if np.linalg.norm(grad) < 1e-4:
            break

        # print grad norm after some iterations
        if _ % 1000 ==0:
            print("itr: {:5d},  ||grad||: {:0.6E}".format(_, np.linalg.
↪norm(grad)))

    return w


# Load diabetes dataset
data = datasets.load_diabetes()
X, Y = data.data, data.target.reshape(-1,1)

# splitting dataset in (80%) training and (20%)testing
total_size = X.shape[0]
split_idx = int(0.80*total_size)
X_train, Y_train = X[0:split_idx,:], Y[0:split_idx]
X_test, Y_test = X[split_idx:,:], Y[split_idx:]

# Adding a column of ones
X_train = np.concatenate([np.ones((X_train.shape[0],1)), X_train], axis=1)
X_test = np.concatenate([np.ones((X_test.shape[0],1)), X_test], axis=1)
```

[2]:
```
# check if the column of 1's is added
X_train[:,0:3]
```

[2]:
```
array([[ 1.        ,  0.03807591,  0.05068012],
       [ 1.        , -0.00188202, -0.04464164],
       [ 1.        ,  0.08529891,  0.05068012],
       …,
       [ 1.        , -0.02730979,  0.05068012],
       [ 1.        , -0.0854304 ,  0.05068012],
       [ 1.        ,  0.01264814,  0.05068012]])
```

[3]:
```
# define random w
w0 = np.random.normal(size=(11,1))
lr = 0.003
itr = 10000

# compute w_star using gradient descent
w_star = grad_descent(w0, X_train, Y_train, lr, itr)
```

```
itr:     0,  ||grad||: 5.362739E+04
itr:  1000,  ||grad||: 5.886447E+01
itr:  2000,  ||grad||: 1.073139E+01
itr:  3000,  ||grad||: 5.021051E+00
itr:  4000,  ||grad||: 4.049052E+00
itr:  5000,  ||grad||: 3.754679E+00
itr:  6000,  ||grad||: 3.586009E+00
```

```
itr:  7000,  ||grad||: 3.458540E+00
itr:  8000,  ||grad||: 3.353792E+00
itr:  9000,  ||grad||: 3.264292E+00
```

[4]:
```
#compute RSS on test data
y_pred = X_test@w_star
RSS = np.mean((y_pred-Y_test)**2)/(np.std(Y_test)**2)
Rsq = 1 - RSS
print(Rsq)
```

```
0.5352841636123449
```

Problem 4
In this lab, we will illustrate the use of multiple linear regression for calibrating robot control. The robot data for the lab is taken from TU Dortmund's Multiple Link Robot Arms Project. We will focus on predicting the current drawn into one of the joints as a function of the robot motion. Such models are essential in predicting the overall robot power consumption.

1. Read in the data in the attached exp_train.csv file; check that the data that you read actually corresponds to the data in the .csv file. In Python, you can use the commands given at the end of this document.

2. Create the training data:

   - Labels *y*: A vector of all the samples in the 'I2' column

   - Data *X*: A matrix of the data with the columns: ['q2','dq2','eps21', 'eps22', 'eps31', 'eps32','ddq2']

3. Fit a linear model between X and y (using sklearn, or any other library of your choice). Report the MSE of your model.

4. Using the linear model that you trained above, report the MSE on the test data contained in the attached exp_test.csv file.

(Solution)

4 a)

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

names =[
't', # Time (secs)
'q1', 'q2', 'q3', # Joint angle
'dq1', 'dq2', 'dq3', # Joint velocity
'I1', 'I2', 'I3', # Motor current (A)
'eps21', 'eps22', 'eps31', 'eps32', # Strain measurements
'ddq1', 'ddq2', 'ddq3' # Joint accelerations
]
df = pd.read_csv('exp_train.csv', header=None,sep=',',names=names, index_col=0)
df.head(6)
```

Out[1]:

	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	eps21	eps22	eps31	eps3
t													
0.00	-0.000007	2.4958	-1.1345	-7.882100e-21	-4.940656e-321	3.913100e-29	-0.081623	-0.40812	-0.30609	-269.25	-113.20	3.5918	1.578
0.01	-0.000007	2.4958	-1.1345	-2.258200e-21	-4.940656e-321	2.626200e-31	-0.037411	-0.37241	-0.26698	-270.91	-116.05	1.4585	-1.739
0.02	-0.000007	2.4958	-1.1345	-6.469800e-22	-4.940656e-321	1.762500e-33	-0.066319	-0.40302	-0.31459	-269.25	-112.97	3.5918	0.867
0.03	-0.000007	2.4958	-1.1345	-1.853600e-22	-4.940656e-321	1.182800e-35	-0.068020	-0.43703	-0.28398	-269.97	-114.39	1.6956	-0.080
0.04	-0.000007	2.4958	-1.1345	-5.310600e-23	-4.940656e-321	-5.270900e-03	-0.052715	-0.40472	-0.30779	-269.97	-114.15	3.1177	0.867
0.05	-0.000007	2.4958	-1.1345	-1.521500e-23	-4.940656e-321	3.252600e-04	-0.088425	-0.42342	-0.29589	-269.25	-114.15	2.4066	-0.080

4 b)

In [2]:

```
Xtrain = np.array(df[['q2', 'dq2', 'eps21', 'eps22', 'eps31', 'eps32', 'ddq2']])
ytrain = np.array(df['I2'])
```

4 c)

In [3]:

```
regr = linear_model.LinearRegression()
regr.fit(Xtrain, ytrain)
ytrain_pred = regr.predict(Xtrain)
print('Mean squared error of the model: %.2f'% mean_squared_error(ytrain, ytrain_pred))
```

```
Mean squared error of the model: 0.01
```

4 d)

In [4]:

```
df = pd.read_csv("exp_test.csv", header=None, sep=',', names=names, index_col=0)
df.head(6)
```

Out[4]:

	q1	q2	q3	dq1	dq2	dq3	I1	I2	I3	eps21	eps22	eps31	eps3
t													
0.00	-0.000007	1.9024	0.26063	-0.000364	4.940656e-321	0.012596	-0.096928	-0.15134	-0.017005	-130.83	-41.856	-6.3635	5.1341
0.01	0.000013	1.9024	0.26073	0.000739	4.940656e-321	0.012095	-0.028908	-0.11903	-0.020406	-138.18	-51.100	-14.6590	-5.0582
0.02	-0.000007	1.9024	0.26086	-0.000580	4.940656e-321	0.011596	-0.059517	-0.13944	-0.047614	-139.36	-51.812	-14.6590	-5.2952
0.03	0.000013	1.9024	0.26099	0.001409	4.940656e-321	0.013933	-0.079923	-0.15304	-0.023807	-135.57	-48.019	-11.3410	-0.7916
0.04	-0.000007	1.9024	0.26110	-0.001273	4.940656e-321	0.010793	-0.025507	-0.12924	-0.006802	-135.81	-49.204	-12.0520	-2.2139
0.05	-0.000007	1.9024	0.26124	0.001928	4.940656e-321	0.011915	-0.083324	-0.14964	-0.034010	-139.60	-53.471	-16.0820	-6.9545

In [5]:

```
Xtest = np.array(df[['q2', 'dq2', 'eps21', 'eps22', 'eps31', 'eps32', 'ddq2']])
ytest = np.array(df['I2'])

ytest_pred = regr.predict(Xtest)
print('Mean squared error of the test data: %.2f'% mean_squared_error(ytest, ytest_pred))
```

```
Mean squared error of the test data: 0.01
```