DS-GA 1003 HW1

Yiyan Chen

January 2018

1 Introduction

2 Linear Regression

2.1 Feature Normalization

```
def feature_normalization(train, test):
    train_normalized = np.zeros((train.shape[0],train.shape[1]))
    test_normalized = np.zeros((test.shape[0],test.shape[1]))
    for idx in range(train.shape[1]):
        mean_val = np.mean(train[:,idx], axis = 0)
        max_val = np.max(train[:,idx], axis = 0)
        min_val = np.min(train[:,idx], axis = 0)

        if max_val != min_val:
            train_normalized[:,idx] = (train[:,idx] - mean_val)/(max_val - min_val)
            test_normalized[:,idx] = (test[:,idx] - mean_val)/(max_val - min_val)
        else:
            train_normalized[:,idx] = train[:,idx]
            test_normalized[:,idx] = test[:,idx]

    return (train_normalized, test_normalized)
```

2.2 Gradient Descent Setup

2.2.1

$$
\begin{aligned}
e_\theta &= y - X\theta \\
J(\theta) &= \frac{1}{m}\sum e_i^2 \\
&= \frac{1}{m} e_\theta^T e_\theta \\
&= \frac{1}{m}(y - X\theta)^T (y - X\theta) \\
&= \frac{1}{m}(y^T y - y^T X\theta - \theta^T X^T y + \theta^T X^T X\theta) \\
&= \frac{1}{m}(y^T y - 2y^T X\theta + \theta^T X^T X\theta)
\end{aligned}
$$

2.2.2

$$\nabla_\theta J(\theta) = \frac{2}{m}(-X^T y + X^T X\theta)$$

2.2.3

$$
\begin{aligned}
J(\theta+\eta h) - J(\theta) &= \frac{1}{m}[y^T y - 2y^T X(\theta+\eta h) + (\theta+\eta h)^T X^T X(\theta+\eta h) - y^T y + 2y^T X\theta - \theta^T X^T X\theta] \\
&= \frac{1}{m}[-2\eta y^T Xh + 2\eta h^T X^T X\theta + \eta^2 h^T X^T Xh] \\
&= \eta h^T \nabla_\theta J(\theta) + \frac{\eta^2 h^T X^T Xh}{m} \\
&\approx \eta h^T \nabla_\theta J(\theta)
\end{aligned}
$$

2.2.4

$$\begin{aligned}\theta_{k+1} &= \theta_k - \eta \nabla_\theta J(\theta_k) \\ &= \theta_k - \frac{2\eta}{m}(-X^T y + X^T X\theta)\end{aligned}$$

2.2.5

```
def compute_square_loss(X, y, theta):
    loss = 0 #initialize the square_loss
    #TODO
    m = X.shape[0]
    return 1/m*(np.matmul(y.transpose(),y)-2*np.matmul(np.matmul(y.transpose(),X),theta)
            +np.matmul(theta.transpose(),np.matmul(X.transpose(), np.matmul(X, theta))))
```

2.2.6

```python
def compute_square_loss_gradient(X, y, theta):
    m = X.shape[0]
    return 2/m*(-np.matmul(y.transpose(), X) + np.matmul(X.transpose(),np.matmul(X,theta)))
```

2.3 (OPTIONAL)Gradient Checker

```
def grad_checker(X, y, theta, epsilon=0.01, tolerance=1e-4):
    true_gradient = compute_square_loss_gradient(X, y, theta) #the true gradient
    num_features = theta.shape[0]
    approx_grad = np.zeros(num_features) #Initialize the gradient we approximate
    #TODO
    for i in range(num_features):
        e_i = np.zeros(num_features)
        e_i[i] = 1
        approx_grad[i] = (compute_square_loss(X, y, theta+epsilon*e_i)
                         -compute_square_loss(X, y, theta-epsilon*e_i))/(2*epsilon)
    distance = np.linalg.norm(approx_grad-true_gradient.transpose())
    if distance > tolerance:
        return False
    return True
```

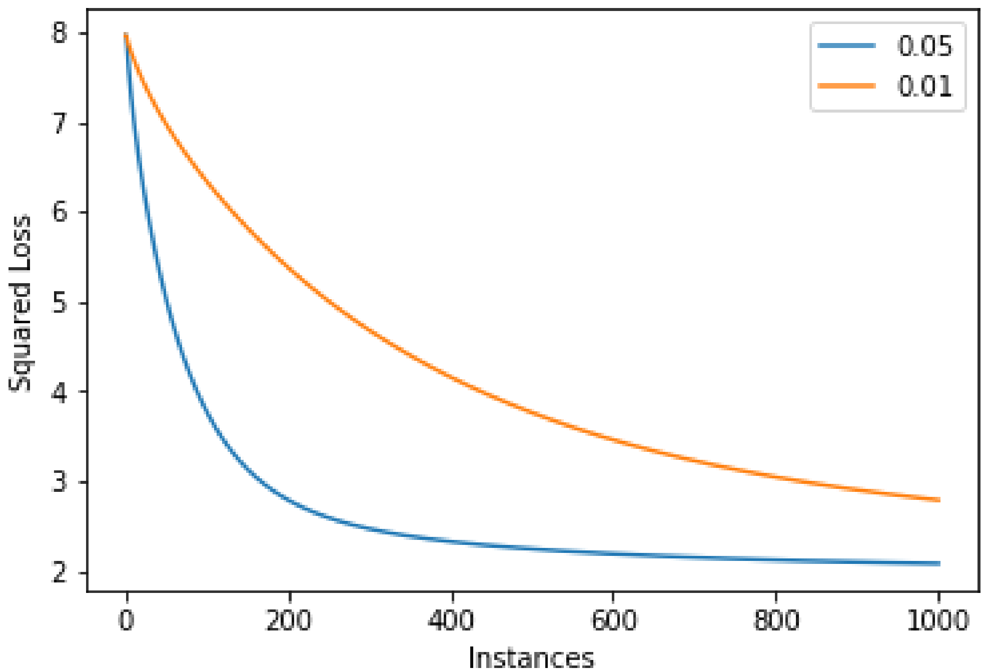
```
def generic_gradient_checker(X, y, theta, objective_func, gradient_func, epsilon=0.01, tolerance=1e-4):
    #TODO
    true_gradient = gradient_func(X,y,theta)
    num_features = theta.shape[0]
    approx_grad = np.zeros(num_features) #Initialize the gradient we approximate
    for i in range(num_features):
        e_i = np.zeros(num_features)
        e_i[i] = 1
        approx_grad[i] = (objective_func(X, y, theta+epsilon*e_i)
                         -objective_func(X, y, theta-epsilon*e_i))/(2*epsilon)
    distance = np.linalg.norm(approx_grad-true_gradient.transpose())
    if distance > tolerance:
        return False
    return True
```

2.4 Batch Gradient Descent

2.4.1

```python
def batch_grad_descent(X, y, alpha=0.1, num_iter=1000, check_gradient=False):
    num_instances, num_features = X.shape[0], X.shape[1]
    theta_hist = np.zeros((num_iter+1, num_features))  #Initialize theta_hist
    loss_hist = np.zeros(num_iter+1) #initialize loss_hist
    theta = np.zeros(num_features) #initialize theta
    #TODO
    theta_hist[0] = theta
    loss_hist[0] = compute_square_loss(X,y,theta)
    for i in range(num_iter):
        if check_gradient and (not grad_checker(X,y,theta_hist[i])):
            return False
        elif (not check_gradient) or (check_gradient and grad_checker(X,y,theta_hist[i])):
            grad = compute_square_loss_gradient(X,y,theta_hist[i])
            loss = compute_square_loss(X,y,theta_hist[i])
            theta_hist[i+1] = theta_hist[i]-alpha*grad
            loss_hist[i+1] = loss
    if check_gradient and (not grad_checker(X,y,theta_hist[num_iter])):
        return False
    return (theta_hist, loss_hist)
```

2.4.2



2.4.3 OPTIONAL

2.5 Ridge Regression

2.5.1

$$\nabla_\theta J(\theta) = \frac{2}{m}(-X^T y + X^T X\theta) + 2\lambda\theta$$

$$\begin{aligned}\theta_{k+1} &= \theta_k - \eta\nabla_\theta J(\theta) \\ &= \theta_k - \frac{2\eta}{m}(-X^T y + X^T X\theta) - 2\eta\lambda\theta\end{aligned}$$

2.5.2

```python
def compute_regularized_square_loss_gradient(X, y, theta, lambda_reg=1):
    m = X.shape[0]
    return 2/m*(-np.matmul(X.transpose(), y)
            + np.matmul(X.transpose(), np.matmul(X,theta)))+2*lambda_reg*theta
```

2.5.3

```python
def regularized_grad_descent(X, y, alpha=0.1, lambda_reg=1, num_iter=1000):
    (num_instances, num_features) = X.shape
    theta = np.zeros(num_features) #Initialize theta
    theta_hist = np.zeros((num_iter+1, num_features))  #Initialize theta_hist
    loss_hist = np.zeros(num_iter+1) #Initialize loss_hist
    #TODO
    theta_hist[0] = theta
    loss_hist[0] = compute_square_loss(X,y,theta)
    for i in range(num_iter):
        grad = compute_regularized_square_loss_gradient(X, y, theta_hist[i], lambda_reg)
        theta_hist[i+1] = theta_hist[i]-alpha*grad
        loss = compute_square_loss(X,y,theta_hist[i+1])
        loss_hist[i+1] = loss
    return (theta_hist, loss_hist)
```

2.5.4

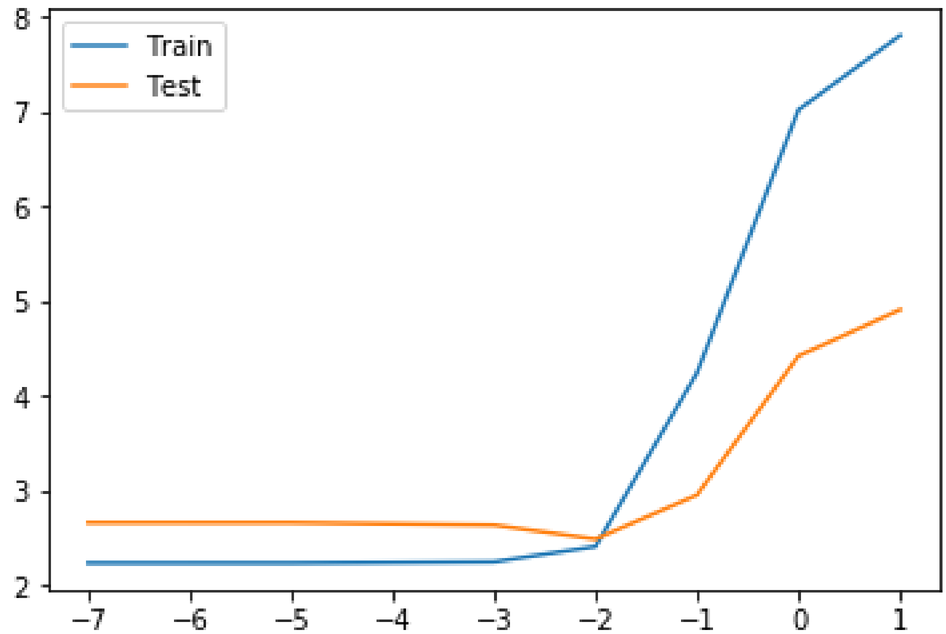
$$\theta_{k+1} = \theta_k + \frac{2\eta}{m} X^T y - (\frac{2\eta}{m} X^T X\theta + 2\eta\lambda)\theta$$

Making B larger means the θ is replaced by $B\theta$. When the $B\theta$ increases, the new θ gets small. The bigger $B\theta$, the smaller new θ is. The effective regularization on the bias term, i.e. θ decreases. We make regularization weaker when B is larger. But B shouldn't be too large to avoid the convergence of loss function.

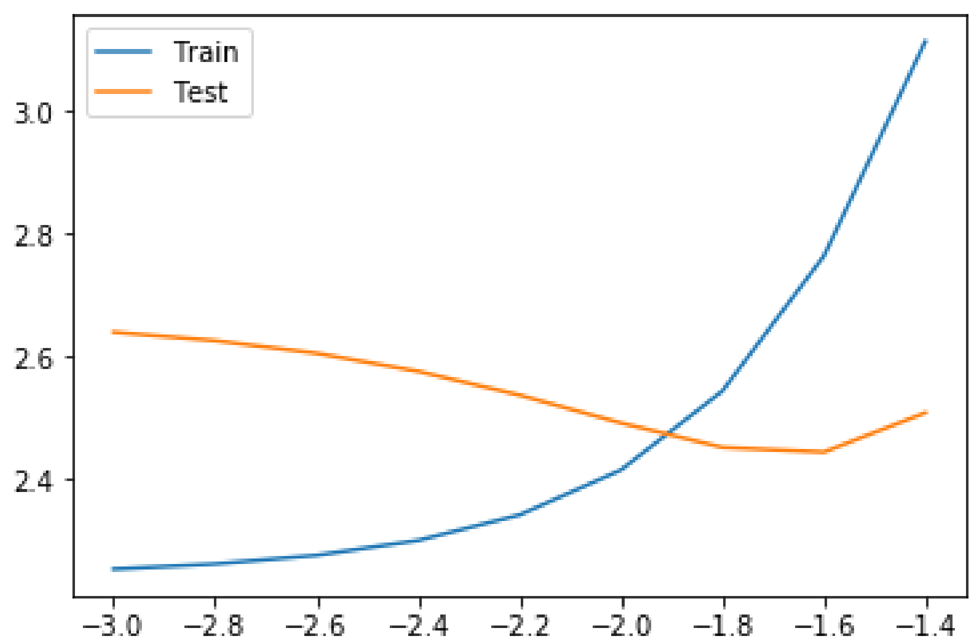
2.5.5 OPTIONAL

2.5.6 OPTIONAL

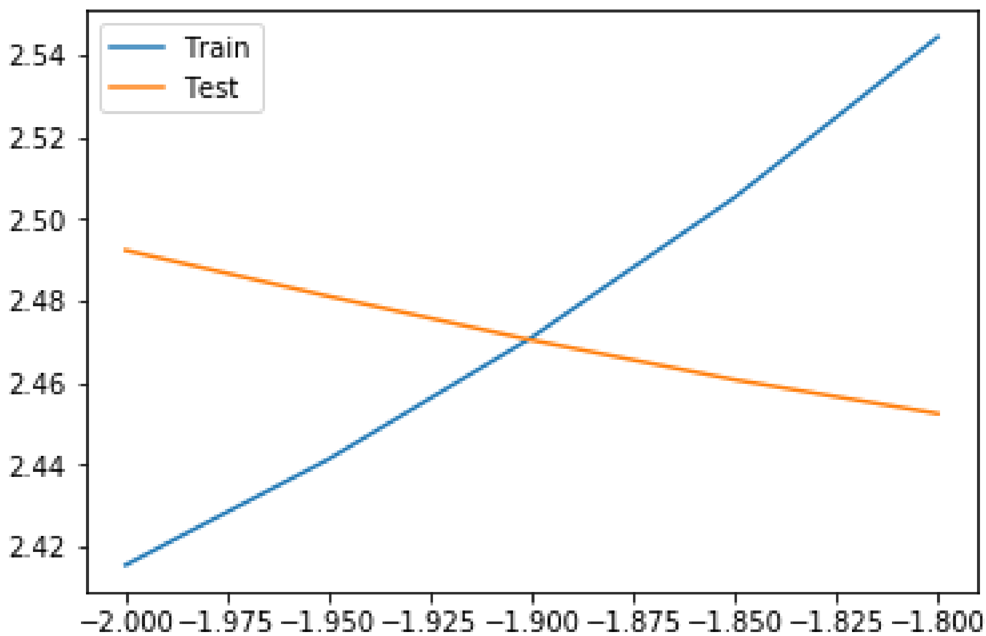
2.5.7



Train
Test
8
7
6
5
4
3
2
−7
−6
−5
−4
−3
−2
−1
0
1



Train
Test
3.0
2.8
2.6
2.4
−3.0
−2.8
−2.6
−2.4
−2.2
−2.0
−1.8
−1.6
−1.4


Train
Test
2.54
2.52
2.50
2.48
2.46
2.44
2.42
−2.000
−1.975
−1.950
−1.925
−1.900
−1.875
−1.850
−1.825
−1.800

2.5.8

The θ should be $10^{-1.9}$ which is the intersection of train and test square loss.
Reason: it minimizes the test error

2.6 Stochastic Gradient Descent

2.6.1

$$
\begin{aligned}
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta \\
&= \frac{1}{m}[\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + m\lambda\theta^T\theta] \\
&= \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta]
\end{aligned}
$$

$$
f_i(\theta) = (h_\theta(x_i) - y_i)^2 + \lambda\theta^T\theta
$$

2.6.2

$$\begin{aligned} E(\nabla f_i(\theta)) &= \sum(\nabla f_i(\theta)p(x_i)) \\ &= \sum(\nabla f_i(\theta)\frac{1}{m}) \\ &= \frac{1}{m}\sum \nabla f_i(\theta) \\ &= \nabla J(\theta) \end{aligned}$$

2.6.3

$$
\begin{aligned}
\theta_{k+1} &= \theta_k - \eta \nabla_\theta f_k(\theta) \\
&= \theta_k - \eta \frac{d((h_\theta(x_k) - y_k)^2 + \lambda \theta_k^T \theta_k)}{d\theta_k} \\
&= \theta_k - \eta \frac{d((\theta_k^T x_k - y_k)^2 + \lambda \theta_k^T \theta_k)}{d\theta_k} \\
&= \theta_k - 2\eta((\theta_k^T x_k - y_k)x_i + \lambda \theta_k)
\end{aligned}
$$

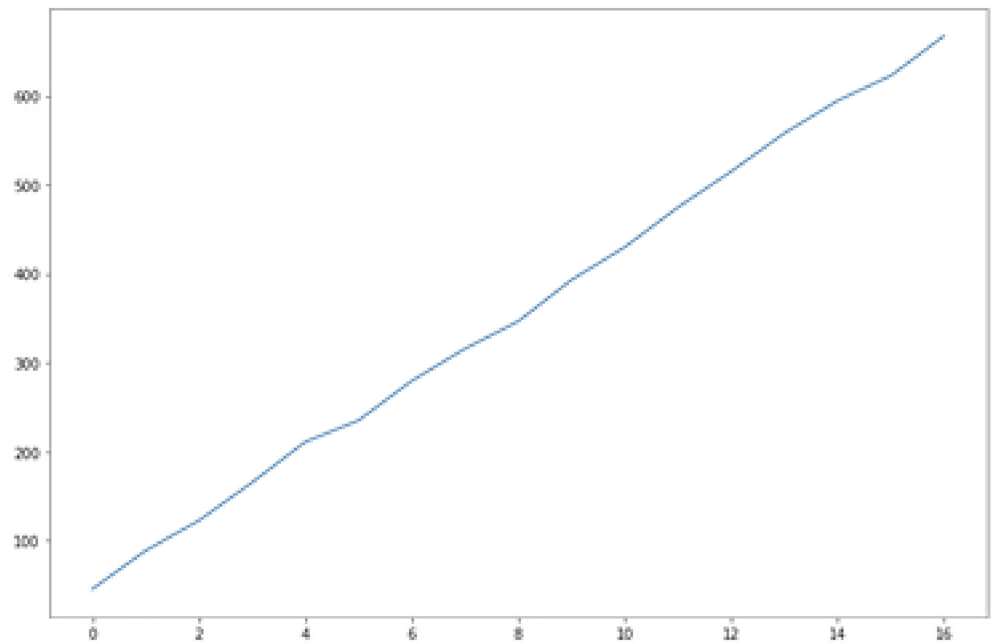
2.6.4

```python
def compute_sgd_loss(X, y, theta, lambda_reg):
    return (np.dot(theta,X)-y)**2 + lambda_reg*np.dot(theta,theta)
def compute_sgd_gradient(X, y, theta, lambda_reg):
    return 2*(np.dot(theta, X)-y)*X + 2 * lambda_reg*theta
```

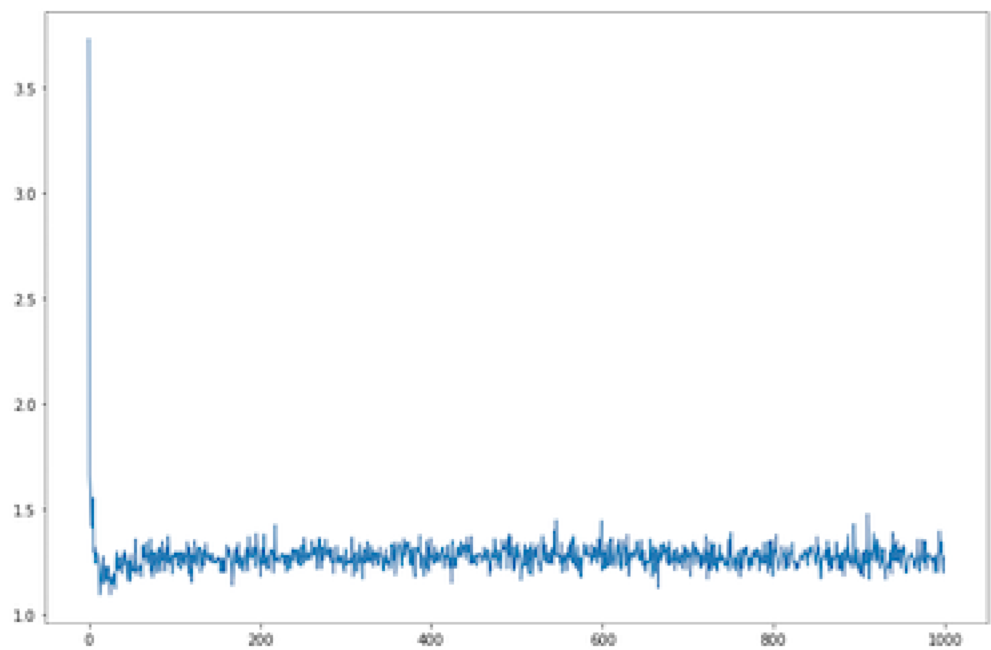
```python
def stochastic_grad_descent(X, y, alpha=0.1, lambda_reg=1, num_iter=1000):
    num_instances, num_features = X.shape[0], X.shape[1]
    theta = np.ones(num_features) #Initialize theta


    theta_hist = np.zeros((num_iter, num_instances, num_features))  #Initialize theta_hist
    loss_hist = np.zeros((num_iter, num_instances)) #Initialize loss_hist
    #TODO
#      grad0 = compute_square_loss_gradient(X[row], y, theta_hist[i][row]) + 2*lambda_reg*theta_hist[i][row]
    step_size = 10

    for i in range(num_iter):
        change_idx = np.arange(num_instances)
        np.random.shuffle(change_idx)
        for row in change_idx:
            theta_hist[i][row] = theta
            loss_hist[i][row] = compute_sgd_loss(X[row], y[row], theta, lambda_reg)
            grad = compute_sgd_gradient(X[row], y[row], theta, lambda_reg)
            if isinstance(alpha, float):
                theta = theta - alpha*grad
            elif alpha == '1/sqrt(t)':
                theta = theta - 1./np.sqrt(step_size)*grad
                step_size += 1
            elif alpha == '1/t':
                theta = theta - 1./step_size*grad
                step_size += 1
            else:
                theta = theta - theta/(1+step_size*lambda_reg*theta)*grad
                step_size += 1
    return (theta_hist, loss_hist)
```

2.6.5

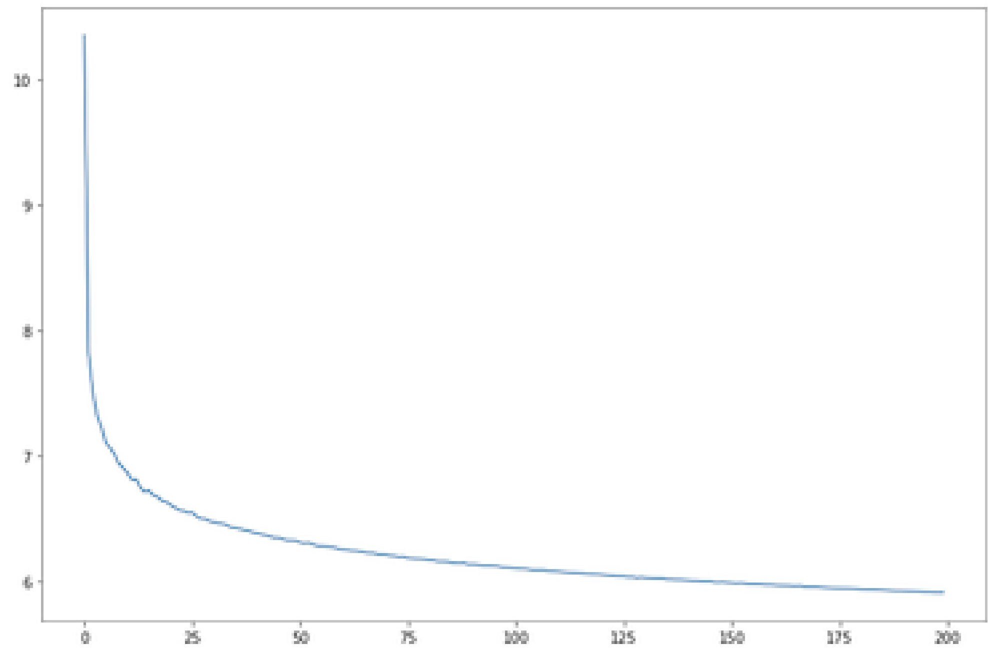
The stochastic gradient descent plot with fixed step size as 0.05:



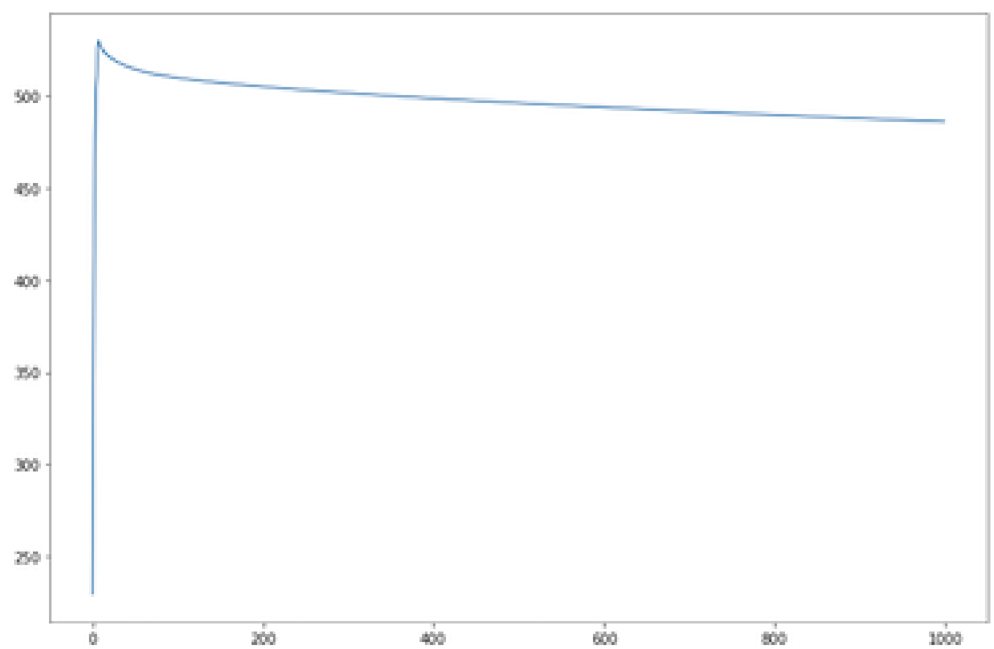
The stochastic gradient descent plot with fixed step size as 0.005:



When the step size is 0.05, the stochastic gradient descent cannot converge.
The stochastic gradient descent plot with step size as 1/t:



The stochastic gradient descent plot with step size as 1/sqrt(t):



When step size is 1/sqrt(t), the stochastic gradient descent cannot converge.

2.6.6 OPTIONAL

3 Risk Minimization

3.1 Square Loss

3.1.1

$$\text{Risk } = R(a) = E[(a-y)^2]$$

$$\begin{aligned}
E[(a-y)^2] &= E[(a-E(y)+E(y)-y^2)^2] \\
&= E[(a-E(y)^2 + 2(a-E(y))(E(y)-y) + (E(y)-y)^2]
\end{aligned}$$

$$\begin{aligned}
E[2(a-E(y))(E(y)-y)] &= 2E[aE(y) - ya - E(y)^2 + yE(y)] \\
&= 2[E(a)E(y) - E(y)E(a) - E(E(y))^2 + E(y)E(y)] \\
&= 2[E(a)E(y) - E(y)E(a) - E(y)^2 + E(y)E(y)] \\
&= 0
\end{aligned}$$

$$E[(a-y)^2] = E[a-E(y)^2] + E[(E(y)-y)^2]$$

Since the second term in the expectation function is independent of a:

$$a^* = E(y)$$

Bayes risk:

$$\begin{aligned}
R(a^*) &= E[(a^*-y)^2] \\
&= E[(E(y)-y)^2] \\
&= E(y^2 - 2yE(y) + (E(y))^2 \\
&= E(y^2) - 2E(y)E(y) + (E(y))^2 \\
&= E(y^2) - (E(y))^2 \\
&= Var(y)
\end{aligned}$$

3.1.2 (a)

$$a^* = E(y|x)$$

3.1.3 (b)

$$\begin{aligned} E[(a-y)^2|x] &= E[(a-E(y|x)+E(y|x)-y^2)^2|x] \\ &= E(a-E(y|x)^2 + 2E[(a-E(y|x))(E(y|x)-y)|x] + E(E(y|x)-y)^2|x) \end{aligned}$$

$$\begin{aligned} E[(a-E(y|x))(E(y|x)-y)|x] &= (a-E(y|x))(E(y|x)-E(y|x)) \\ &= 0 \end{aligned}$$

$$\begin{aligned} E[(a-y)^2|x] &= E[a-E(y|x)^2|x] + E[(E(y|x)-y)^2|x] \\ &= E[a-E(y|x)^2|x] + E[(a^*-y)^2|x] \\ &= E[f(x)-E(y|x)^2|x] + E[(f(x)^*-y)^2|x] \\ E[(f(x)-y)^2|x] &= E[f(x)-E(y|x)^2|x] + E[(f(x)^*-y)^2|x] \end{aligned}$$

$$E[(f(x)^*-y)^2|x] \leq E[(f(x)-y)^2|x]$$

$$\begin{aligned} E[E[(f(x)^*-y)^2|x]] &\leq E[E[(f(x)-y)^2|x]] \\ E[(f(x)^*-y)^2] &\leq E[(f(x)-y)^2] \end{aligned}$$

3.2 OPTIONAL