Machine Learning HW01

Ben Zhang, bz957

February 4, 2018

2.1 Feature Normalization

```
def dataset_minmax(train, test):
    train = train[:,np.any(~np.isnan(train),axis=0)]
    test = test[:,np.any(~np.isnan(test),axis=0)]
    train_normalized = (train - train.min(axis=0)) /
                       (train.max(axis=0) - train.min(axis=0))
    test_normalized = (test - test.min(axis=0)) /
                      (test.max(axis=0) - test.min(axis=0))
    return train_normalized, test_normalized
```

2.2 Gradient Descent Setup

2.2.1

2.2.1 $J(\theta) = \frac{1}{m} \| X\theta - y \|_2^2, \ X \in R^{m \times d}, \ \theta, y \in R^m$

2.2.2

2.2.2 $\nabla J(\theta) = \frac{2}{m} X^T (X\theta - y)$, $X \in R^{m \times d}$, $\theta, y \in R^m$

2.2.3

2.2.3. $J(\theta+\eta h)-J(\theta)$

$$= \frac{1}{m}\left[\eta\left(2\theta^T X^T X - 2y^T X\right)h + \eta^2 h^T X^T X \cdot h\right]$$

$$= \frac{1}{m}\cdot \eta \|h\| \cdot \nabla J(\theta)$$

2.2.4

2.2.4 $\theta \leftarrow \theta - \eta \cdot \nabla J(\theta)$

$$\theta \leftarrow \theta - \eta \cdot \frac{2}{m} \cdot X^T (X\theta - y)$$

2.2.5

```
def compute_square_loss(X, y, theta):
    SL = np.dot((np.dot(X,theta[:, np.newaxis])-y[:, np.newaxis]).T,
                np.dot(X,theta[:, np.newaxis])-y[:, np.newaxis]) / X.shape[0]

    return SL[0]
```

2.2.6

```
def compute_square_loss_gradient(X, y, theta):
    SL_gradient = (2 * np.dot(X.T,np.dot(X,theta[:, np.newaxis])-y[:, np.newaxis]) /
                  X.shape[0]).T[0]
    return SL_gradient
```

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
    e = np.identity(num_features, dtype = int)
    for i in range(num_features):
        theta_add = theta + epsilon * e[i]
        theta_minus = theta - epsilon * e[i]
        approx_grad[i] = (np.dot((np.dot(X,theta_add[:, np.newaxis])
                         -y[:, np.newaxis]).T,
                         np.dot(X,theta_add[:, np.newaxis])-y[:, np.newaxis]) -
                         np.dot((np.dot(X,theta_minus[:, np.newaxis])
                         -y[:, np.newaxis]).T,
                         np.dot(X,theta_minus[:, np.newaxis])
                         -y[:, np.newaxis]))/ X.shape[0]*2*epsilon
    dist = np.sqrt(np.sum(np.square(true_gradient - approx_grad)))
    return dist <= tolerance
```

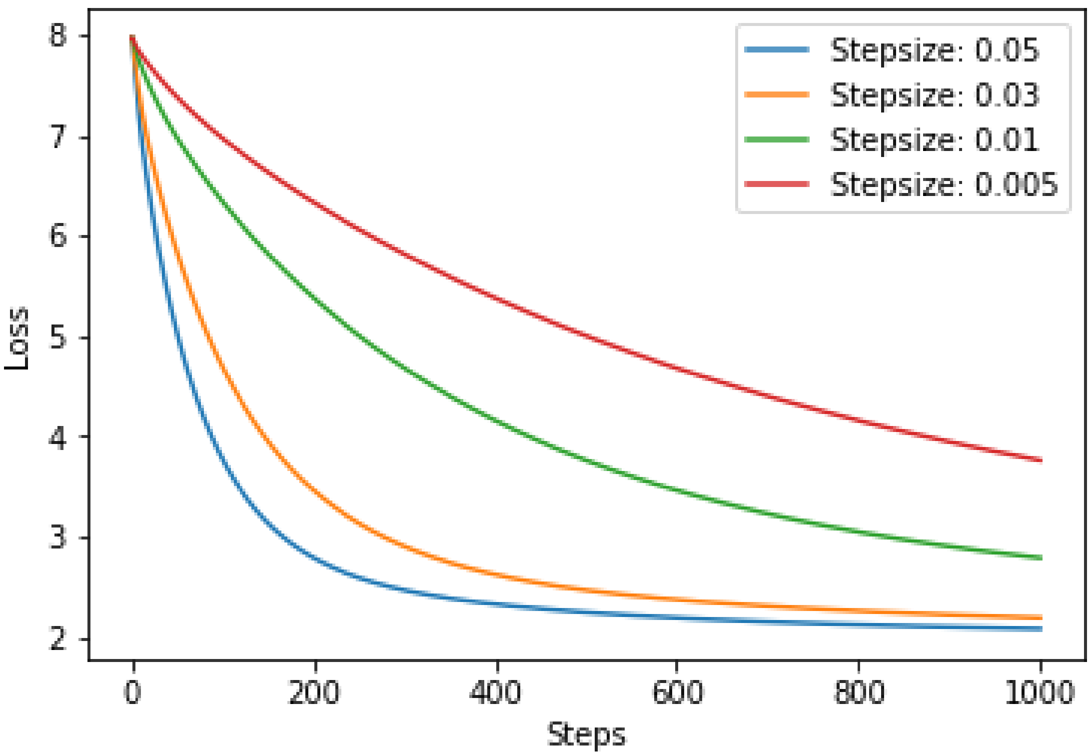
2.4 Batch Gradient Descent3

2.4.1

```
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

    loss_hist[0] = compute_square_loss(X, y, theta)

    for i in range(num_iter):
        theta = theta - alpha*compute_square_loss_gradient(X, y, theta)
        loss_hist[i+1] = compute_square_loss(X, y, theta_hist[i])
        theta_hist[i+1,:] = theta
    return theta_hist, loss_hist
```

2.4.2

```
df = pd.read_csv('data.csv', delimiter=',')
X = df.values[:,:-1]
y = df.values[:,-1]

print('Split into Train and Test')
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =100,
                                        random_state=10)

print("Scaling all to [0, 1]")
X_train, X_test = dataset_minmax(X_train, X_test)
X_train = np.hstack((X_train, np.ones((X_train.shape[0], 1))))  # Add bias term
X_test = np.hstack((X_test, np.ones((X_test.shape[0], 1)))) # Add bias term
X_train.shape, X_test, y_train.shape, y_test

list_alpha=[0.05, 0.03, 0.01, 0.005]
for alpha in list_alpha:
    theta_hist, loss_hist = batch_grad_descent(X_train, y_train, alpha,
                                num_iter=1000, check_gradient=False)
    plt.plot(loss_hist,label="Stepsize: "+ str(alpha))

plt.legend()
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.show()
```



Findings: Among the stepsizes which don't lead to diverge(here is less than 0.1), the larger stepsize is, the faster the loss decrease.

2.5 Ridge Regression (i.e. Linear Regression with ‘2 regularization)

2.5.1

2.5.1 $J(\theta) = (\|X\theta - y\|^2 + \lambda \|\theta\|^2) \cdot \frac{1}{m}$

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

$$\theta \leftarrow \theta - \eta \cdot \nabla J(\theta)$$

$$\theta - \eta \cdot \frac{2}{m} \cdot [X^T(X\theta - y) + \lambda\theta]$$

2.5.2

```
def compute_regularized_square_loss_gradient(X, y, theta, lambda_reg):
    grad_theta = 2 *(np.dot(X.T, np.dot(X, theta) - y) + lambda_reg*theta)/
                (X.shape[0])
    return grad_theta
```

2.5.3

```
def compute_regularized_square_loss(X, y, theta, lambda_reg):
    loss = np.dot(X, theta) - y
    return (np.sum(loss ** 2) +lambda_reg*np.sum(np.dot(theta,theta)))/ (X.shape[0])
```

2.5.4

2.5.4. $J(\theta) = \frac{1}{2}\left[\sum_{i=1}^{m} \|y_i - \tilde{y}_i\|^2 + \frac{\lambda}{2}\hat{\theta}^T\hat{\theta}\right) \cdot \frac{1}{m}$

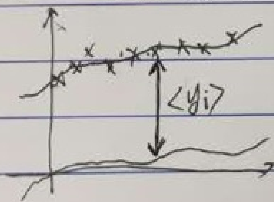
$$\hat{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} B \\ x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad \hat{\theta} = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}, \quad \hat{x}, \hat{\theta} \in R^{d+1}$$

$$\tilde{y} = \hat{\theta}^T \hat{x} = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

we want to minimize $J(\theta)$, $\theta_0 x_0 = \theta_0 B = \langle y_i \rangle$, $\theta_0 = \frac{\langle y_i \rangle}{B}$

~~but we don't want to squeeze $\langle y_i \rangle$~~, we wanna θ_0 be small.

2-D. e.g.



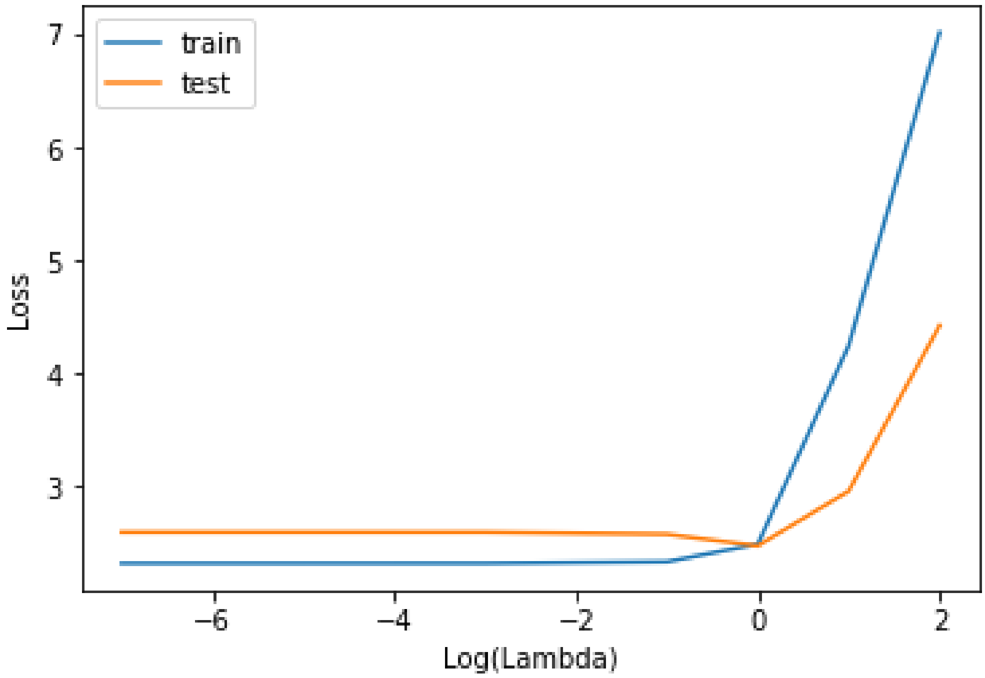
in order to get good regression, ~~we want to keep λ~~ and ~~keep~~ avoid overfitting, we will increase λ.

in order ~~to~~ trade off the affection of λ, we can increase B (greater than 1) then $\lambda \cdot \theta_0$ will ~~keep same almost same~~ decrease, $J(\theta)$ will be small.

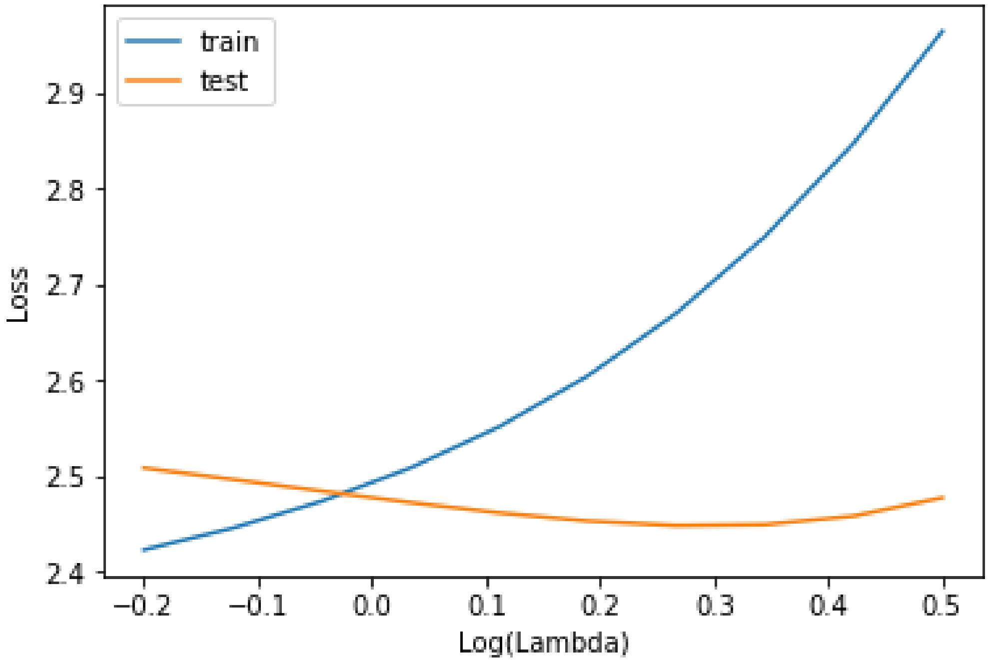
let ~~$B = \frac{y_0}{\theta_0}$~~ ~~$B \gg$~~ $B \gg \langle y_i \rangle$, $\theta_0 \ll 1$.

2.5.7

```
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

    loss_hist[0] = compute_regularized_square_loss(X, y, theta.T, lambda_reg)

    for i in range(num_iter):
        grad = compute_regularized_square_loss_gradient(X, y, theta.T, lambda_reg)
        theta = theta - alpha * grad.T
        loss_hist[i+1] = compute_regularized_square_loss(X, y, theta.T, lambda_reg)
        theta_hist[i+1,:] = theta
    return theta_hist, loss_hist


list_lambda=[1e-7,1e-5,1e-3,1e-1,1,10,100]
llist_train = []
losslist_train = []
llist_test = []
losslist_test = []
for l in list_lambda:
    theta_hist, loss_hist = regularized_grad_descent(X_train, y_train, lambda_reg=l,
                                alpha=0.02, num_iter=1000)
    theta_train=theta_hist[-1]
    train_loss = compute_square_loss(X_train, y_train, theta_train)
    test_loss = compute_square_loss(X_test, y_test, theta_train)
    llist_train.append(math.log10(l))
    losslist_train.append(train_loss)
    llist_test.append(math.log10(l))
    losslist_test.append(test_loss)
    print(train_loss,test_loss)

plt.plot(llist_train,losslist_train,label="train")
plt.plot(llist_test,losslist_test,label="test")
plt.legend()
plt.xlabel('Log(Lambda)')
plt.ylabel('Loss')
plt.show()
```

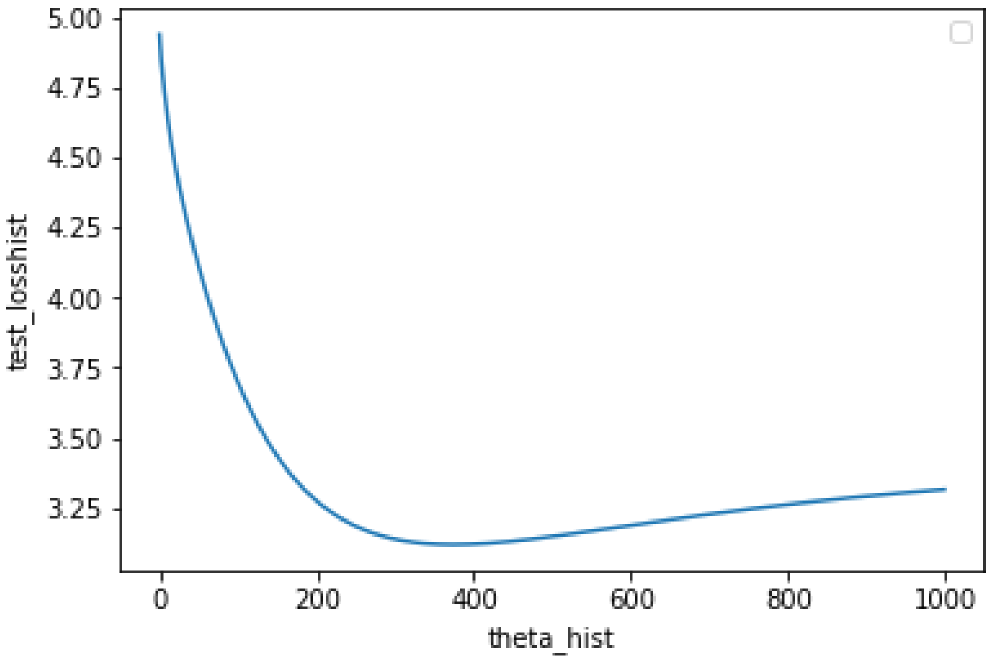



```
list_lambda=[10**x for x in np.linspace(-0.2,0.5,10)]
llist_train = []
losslist_train = []
llist_test = []
losslist_test = []
for l in list_lambda:
    theta_hist, loss_hist = regularized_grad_descent(X_train, y_train, lambda_reg=l,
                                                     alpha=0.02, num_iter=1000)
    theta_train=theta_hist[-1]
    train_loss = compute_square_loss(X_train, y_train, theta_train)
    test_loss = compute_square_loss(X_test, y_test, theta_train)
    llist_train.append(math.log10(l))
    losslist_train.append(train_loss)
    llist_test.append(math.log10(l))
    losslist_test.append(test_loss)
    print(train_loss,test_loss)

plt.plot(llist_train,losslist_train,label="train")
plt.plot(llist_test,losslist_test,label="test")
plt.legend()
plt.xlabel('Log(Lambda)')
plt.ylabel('Loss')
plt.show()
```


train
test
Loss
2.4
2.5
2.6
2.7
2.8
2.9
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
Log(Lambda)

2.5.8

```
theta_hist_train, loss_hist_train = regularized_grad_descent(X_train, y_train,
                                        lambda_reg=2, alpha=0.02, num_iter=1000)
(num_instances, num_features) = X_test.shape
theta_hist_train=theta_hist_train[1:]
num_iter=1000
test_losshist = np.zeros(num_iter) #Initialize loss_hist

for i in range(num_iter):
    test_losshist[i] = compute_regularized_square_loss(X_test, y_test,
                    theta_hist_train[i].T, lambda_reg=2)

x_theta=range(1000)
plt.plot(x_theta, test_losshist)
plt.legend()
plt.xlabel('theta_hist')
plt.ylabel('test_losshist')
plt.show()
print ("Lowest loss is",np.ndarray.min(test_losshist),'\n',
        "Best theta which gives the lowest loss is",
            theta_hist[np.argmin(test_losshist)])
```



Lowest loss is 3.11793016157, Best theta which gives the lowest loss is [-9.00402125e-01 3.85685744e-01 1.08183750e+00 1.39942161e+00 -9.01466487e-01 -6.28496325e-01 -5.68473284e-01 -5.68473284e-01 4.02634667e-01 1.08412967e+00 1.44352293e+00 -1.19457981e-01 -1.34616559e+00 -2.52978755e+00 1.08157548e+00 1.61254062e+00 1.17188746e+00 2.71238695e-01 -7.87481984e-02 -7.87481984e-02 -7.87481984e-02 -6.36017399e-03 -6.36017399e-03 -6.36017399e-03 2.18325496e-02 2.18325496e-02 2.18325496e-02 3.53277417e-02 3.53277417e-02 3.53277417e-02 4.29522384e-02 4.29522384e-02 4.29522384e-02 -9.60892835e-04 -9.60892835e-04 -9.60892835e-04 1.09965622e-01 1.09965622e-01 1.09965622e-01 8.95927569e-02 8.95927569e-02 8.95927569e-02 8.02491636e-02 8.02491636e-02 8.02491636e-02 7.50896218e-02 7.50896218e-02 7.50896218e-02 -1.13514056e+00]

Best theta which gives the lowest loss is It isn't the last theta of the from the $theta_hist$ generated from train dataset. Because after we find the best theta, and we continue to calculate, we will make over fit problem in training set. So when we use test set to validate the theta list, we will find the loss goes down first, when reach the minimum, it starts to increase because of the over fit.

2.6 Stochastic Gradient Descent

2.6.1

2.6.1
$$
\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m} f_i(\theta) &= \frac{1}{m}\sum_{i=1}^{n} [(h_\theta(x_i)-y_i)^2+\lambda\theta^T\theta)] \\
&= \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x_i)-y_i)^2+\frac{1}{m}\cdot m\cdot\lambda\theta^T\theta \\
&= \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x_i)-y_i)^2 \\
&= J(\theta)
\end{aligned}
$$

2.6.2

2.6.2 $E(\nabla f_i(\theta)] = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\theta)$

$$= \frac{1}{m}\sum_{i=1}^{m} \frac{f_i(\theta+h) - f_i(\theta)}{h}$$

$$= \frac{\frac{1}{m}\sum_{i=1}^{m} f_i(\theta+h)\cancel{\theta} - \frac{1}{m}\sum_{i=1}^{m} f_i(\theta)}{h}$$

$$= \frac{J(\theta+h) - J(\theta)}{h}$$

$$= \nabla J(\theta).$$

2.6.3

2.6.3. $f_i(\theta)=(\|X_i\theta-y_i\|^2+\lambda\|\theta\|^2)$

$\nabla f_i(\theta)=2X_i^T(X_i\theta-y)+2\lambda\theta$

$\theta \leftarrow \theta-\eta\nabla f_i(\theta)$

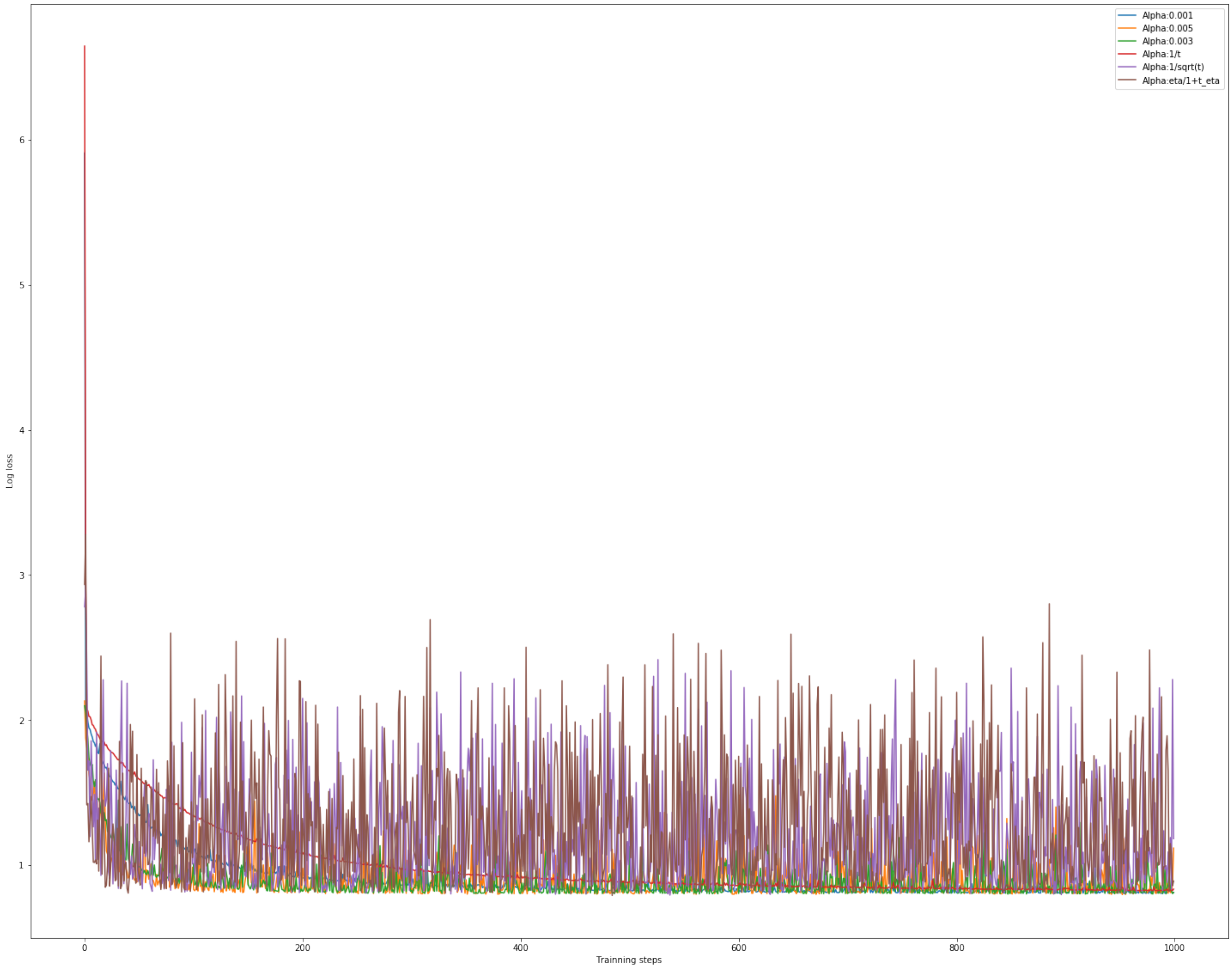
$\theta-\eta\cdot 2[X_i^T(X_i\theta-y)+\lambda\theta]$

2.6.4

```
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))
    loss_hist = np.zeros((num_iter, num_instances))
    l=list(range(num_instances))
    for i in range(num_iter):
        np.random.shuffle(l)
        for j in l:
            if alpha == '1/t':
                alpha = 1/(i+2000)
            elif alpha=='1/sqrt(t)':
                alpha = 1/np.sqrt(i+5000)
            elif alpha == 'eta/1+t_eta':
                alpha = 0.1/(1+0.1*lambda_reg*(i+5000))
            a=j
            grad_theta = 2*np.dot(X[a].T, np.dot(X[a], theta.T) - y[a])
                        + 2 * lambda_reg*theta

            theta = theta - alpha*grad_theta
            loss_hist[i,j]=compute_square_loss(X,y,theta)
            theta_hist[i,j] = theta
    return theta_hist, loss_hist
```

2.6.4

```
def Finding_SDG(X,y):
    alphas =[0.001, 0.005, 0.003, '1/t', '1/sqrt(t)', 'eta/1+t_eta']
    plt.figure(figsize=(25,20))
    for alpha in alphas:
        thetas, losses = stochastic_grad_descent(X, y, alpha, lambda_reg=1e-2,
                                             num_iter=1000)
        plt.plot(np.log(losses[:, -1]), label = "Alpha:" + str(alpha))
    plt.legend()
    plt.xlabel("Trainning steps")
    plt.ylabel("Log loss")
    plt.show()
Finding_SDG(X_train, y_train)
```



3.1 Square Loss

3.1.1

3.1.1.
$$\begin{aligned}
E[(a-y)^2] &= E(a^2-2ay+y^2) \\
&= E(a^2) - 2E(ay) + E(y^2) \\
&= a^2 - 2aE(y) + E(y^2) \\
&= a^2 - 2aE(y) + E(y)^2 - E(y)^2 + E(y^2) \\
&= [a - E(y)]^2 + Var(y) \\
&= Var(y).
\end{aligned}$$

$$a^* = \operatorname{argmin}_a E(a-y)^2 \Rightarrow a^* = Ey.$$

3.1.2

3.1.2. a). $\hat{a}^* = E(y)$

$\Rightarrow a^* = \underset{a}{\arg\min}\ E((a-y)^2)$

$\because a^* = \underset{a}{\arg\min}\ E(l(a^*, y)|x)$

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

$\hat{a}^* = E(y|x)$

$\Rightarrow f(x) = E(y|x)$

3.1.3

(b) $E_{xy}[l(x,y)] = \int_x \int_y f(x,y)\, l(x,y)\, dx\, dy$

$$= \int_x \int_y f(y|x) \cdot f(x) \cdot l(x,y)\, dy\, dx$$

$\therefore E_x[E_y(f^*(x)-y)^2|x]] \le E_x[E_y[f(x)-y)^2|x]]$

$\Rightarrow E[(f^*(x)-y)^2] \le E[(f(x)-y)^2]$