CS189–Fall 2016 — Homework 2 Solutions

Yide Shentu, SID 25745510

1. Visualizing Eigenvectors of Gaussian Covariance Matrix

```
→  hw2_code python q1.py
The mean of X1 is 4.00924797466
The mean of X2 is 5.12837044422
the cov matrix
[[ 3.79711518  1.35483109]
 [ 1.35483109  9.10028968]]
The eigenvalues and the eigenvectors
(array([ 3.47103855,  9.42636632]), array([[-0.97223774, -0.23399523],
       [ 0.23399523, -0.97223774]]))
```

Figure 1: Output of problem1

a)

From figure 1, mean of X1 is 4.01 and mean of X2 is 5.13

b)

From figure 1,

$$\begin{vmatrix} 3.80 & 1.35 \\ 1.35 & 9.1 \end{vmatrix}$$

c)

From figure 1, those eigenvalue, eigenvector pairs is:
$\lambda_1 = 3.47 -> [-0.97, 0.23]^T$
$\lambda_2 = 9.43 -> [-0.23, 0.97]^T$

d)

Indicate by red dots and two black vector arrows

e)

Indicate by blue dots

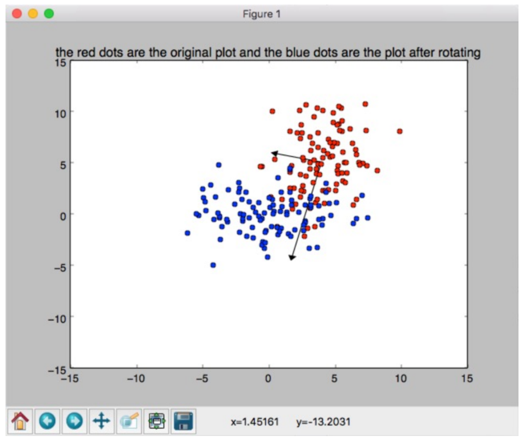

Figure 2: for part d and e

2.

a) $X = AZ + b$ $\quad Z = \begin{vmatrix} N(0,1) \\ N(0,1) \\ N(0,1) \\ \vdots \end{vmatrix}$

$$\Sigma_{x_{ij}} = \mathbb{E}[(X_i - b_i)(X_j - b_j)] \quad \text{since } b = \bar{X}$$
$$= \mathbb{E}\left[\left(\sum_{k=1}^{n} A_{ik} Z_k\right)\left(\sum_{p=1}^{n} A_{jp} Z_p\right)\right]$$
$$= \sum A_{ik} \sum A_{jp} \underline{\mathbb{E}(Z_k Z_p)}$$
$$\hookrightarrow = \delta_{kj} = \begin{cases} 0 & k \neq j \\ 1 & k = j \end{cases}$$

$\Sigma_x = AA^T$ invertable $\qquad = (AA^T)_{ij}$

iff AA^T invertiable

$\det(AA^T) = \det(A)\det(A^T) \Leftrightarrow A$ invertiable

Σ_x^{-1} not exist $\Leftrightarrow$ A not invertiable

$X = AZ + b \Rightarrow X_1\ X_2\ X_3 \cdots X_n$ Linearly dependant.

We can get A from X, $\sqrt{\Sigma_x} = A$. Σ_x semipositive therefore $\sqrt{\Sigma_x}$ exist

then we do row reduce of A get $A' = \begin{vmatrix} - & - \\ - & - \\ - & - \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{vmatrix}$ n row all zero

then we cut the last n terms in X

convert x to x'. Since the last n are linearly dependent term we do not lose information.

b). Σ symmetric and positive defined.

Σ^{-1} also symmetric and positive defined.

$\Sigma^{-1} = U \Lambda U^T \qquad \Lambda$ positive $\Lambda = \sqrt{\Lambda} \cdot \sqrt{\Lambda}$

$X^T U \sqrt{\Lambda} \sqrt{\Lambda} U^T X = \| \sqrt{\Lambda} U^T X \|_2^2$

$A = \sqrt{\Lambda} U^T \qquad \Lambda$ = diagonal matrix after diagonalizing Σ^{-1}

c). sample from

convert X Back to ~~Entirely~~ ~~Independent~~ $Z = \begin{vmatrix} N(0,1) \\ N(0,1) \\ N(0,1) \\ \vdots \end{vmatrix}$

$\Sigma = VV^T$ $\quad X = VZ + b \to b=0$

$\Sigma^{-1} = V^{T-1}V^{-1}$ $\quad = VZ$

$x^T\Sigma^{-1}x = x^TV^{T-1}V^{-1}x$ $\quad x = Vz + \cancel{b}$. $\quad x, z$ are samples from X, Z

$= z^T \underbrace{V^T V^{T-1}}_{1} \underbrace{V^{-1} V}_{1} z = z^Tz = \|Ax\|_2^2$

change sample space.

d). $\|Ax\|_2^2 = x^T\Sigma^{-1}x = x^TU\Lambda U^Tx$. $\quad U^T$ change ~~base~~ base

therefore $\quad \|U^Tx\|_2 = 1$

Max = Maximum entry of Λ

Min = ~~Max entry~~ Minimum entry of Λ

entries of Λ are eigenvalues of Σ^{-1}

if $X_i \perp\!\!\!\perp X_j$ then Σ diagonal Matrix entries correspond to variance of X_i

Maximum of $\|Ax\|_2^2 = \frac{1}{a}$ $\quad a$ = Minimum variance among X_i

Minimum of $\|Ax\|_2^2 = \frac{1}{b}$ $\quad b$ = Maximum variance among X_i

choose X_i makes $\|Ax\|_2^2 = \frac{1}{b}$ $\quad b$ = Max variance

X_i is the ~~normalized~~ ~~eigenvector~~ variable with Largest variance

3.

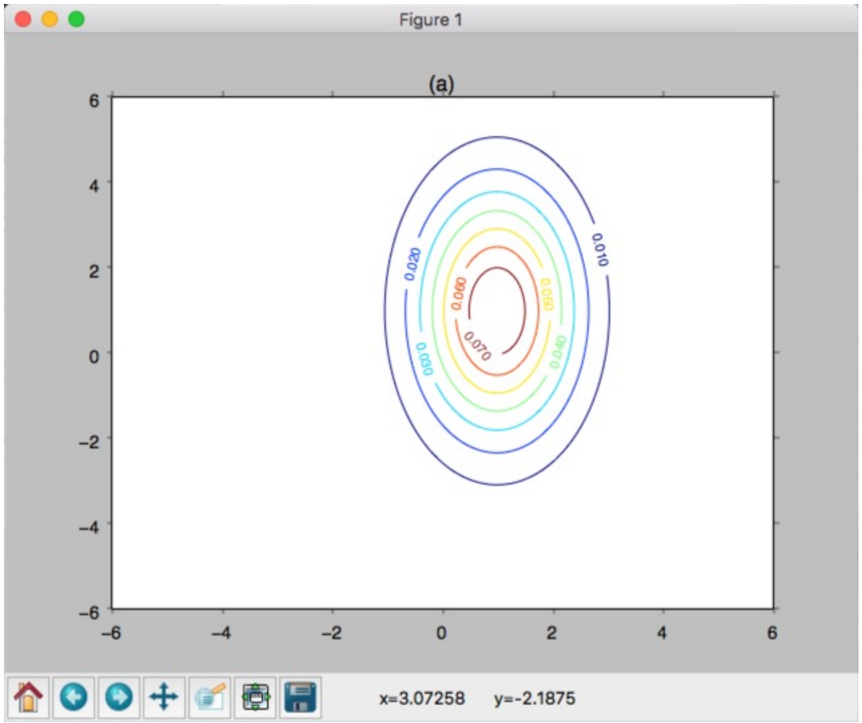

Figure 3: a

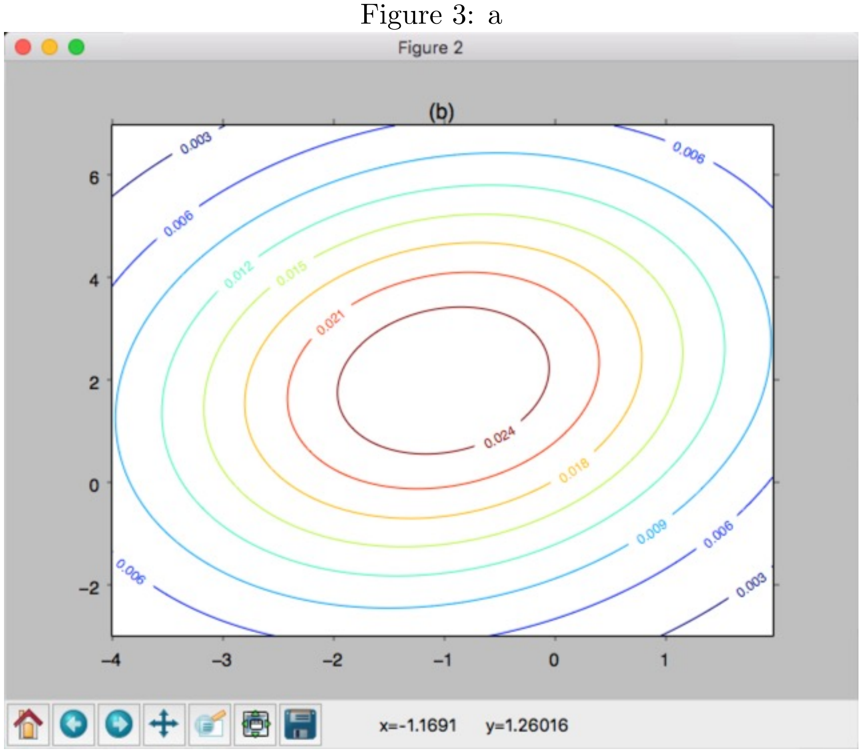

Figure 4: b

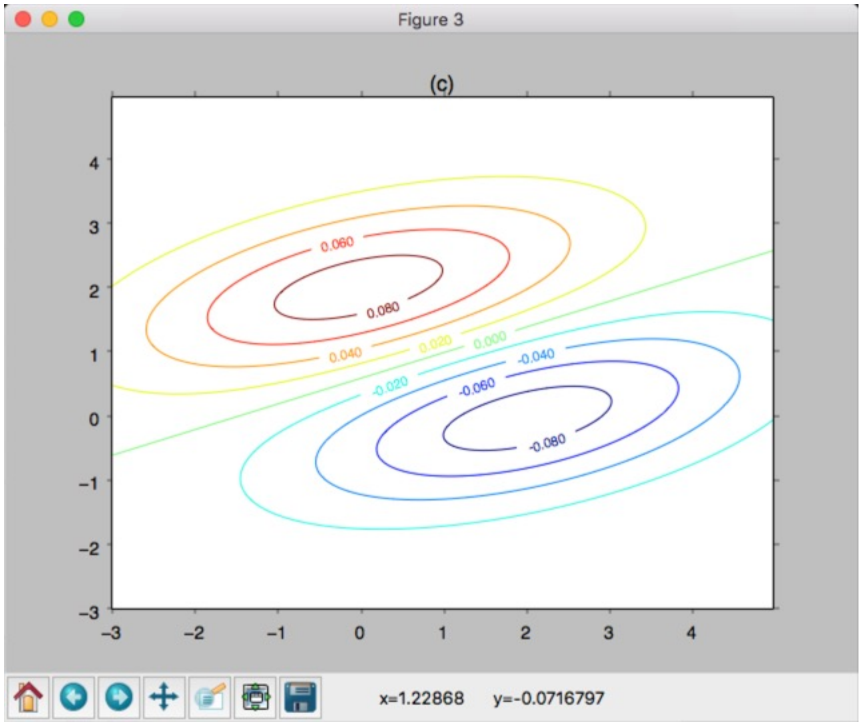

Figure 5: c

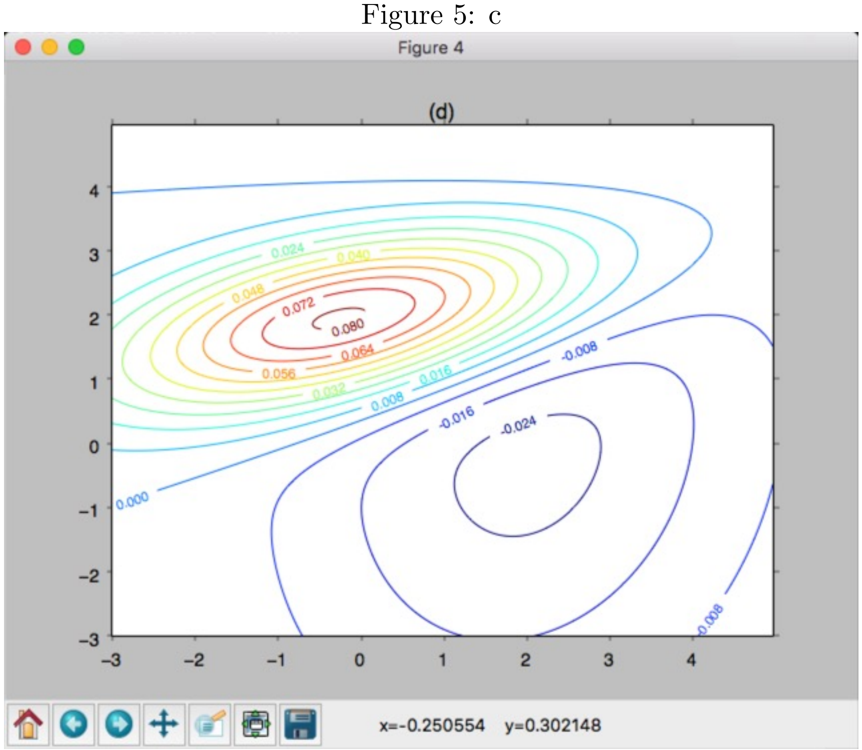

Figure 6: d

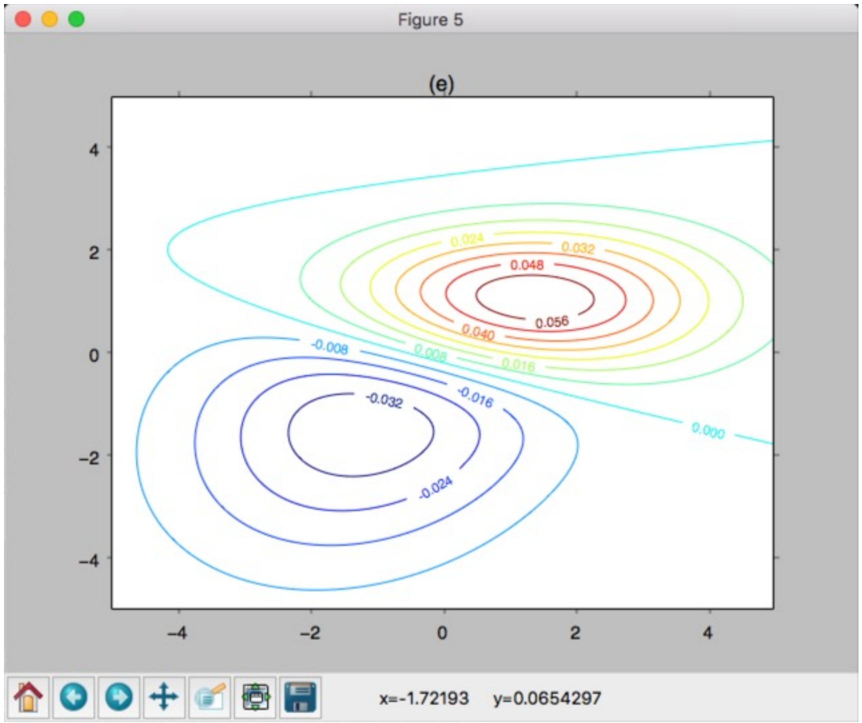
Figure 5
(e)
0.024
0.032
0.048
0.056
0.040
0.008
0.016
0.000
-0.008
-0.016
-0.032
-0.024
4
2
0
−2
−4
−4
−2
0
2
4
x=-1.72193 y=0.0654297

Figure 7: e

#4. a) code
and use form solution from HW1.
$$w = (\lambda I + X^T X)^{-1} X^T y$$

b) $W_{n+1} = W_n - \alpha(2XX^T W_n - 2XY^T + 2\lambda W_n)$.

c) $W_{t+1} = W_t - \alpha(2x_i x_i^T W_t - 2x_i y_i^T + 2\lambda W_t)$.
In order to help the iterative algorithm converge. α should decrease generally.
can use exp or $(1 - \frac{i}{\#iteration})$ ~~

d) gradient descent: α = default value reg = 0.1 <default>.

Stochastic gradient descent: α = default reg = 0.1
and $\alpha_i = \alpha * (1 - \frac{i}{\#\ iteration})$

From the Graph we can see that the error rate of gradient descent is always decreasing and the curve is very smooth. Because we compute the real derivative and with a proper α the update direction is always correct.

The graph of stochastic graident descent is a little different. Although it trends to decrease, we can see a lot of noise during the process
Because we randomly choose vector to compute the "expected" value of derivative, some times the update direction is not correct.

e) My score is 0.94460

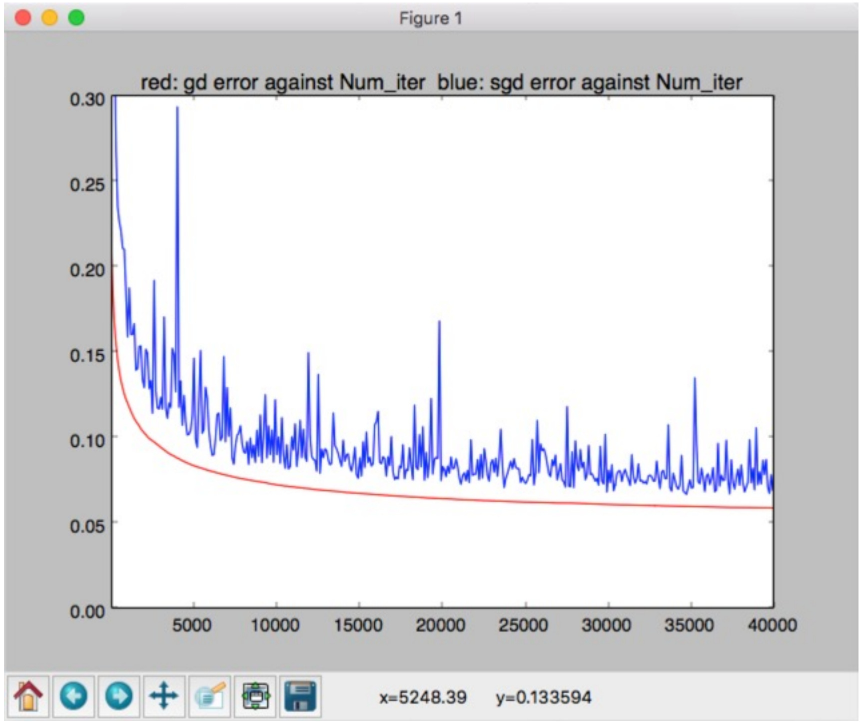

Figure 8: SGD with liear learning rate

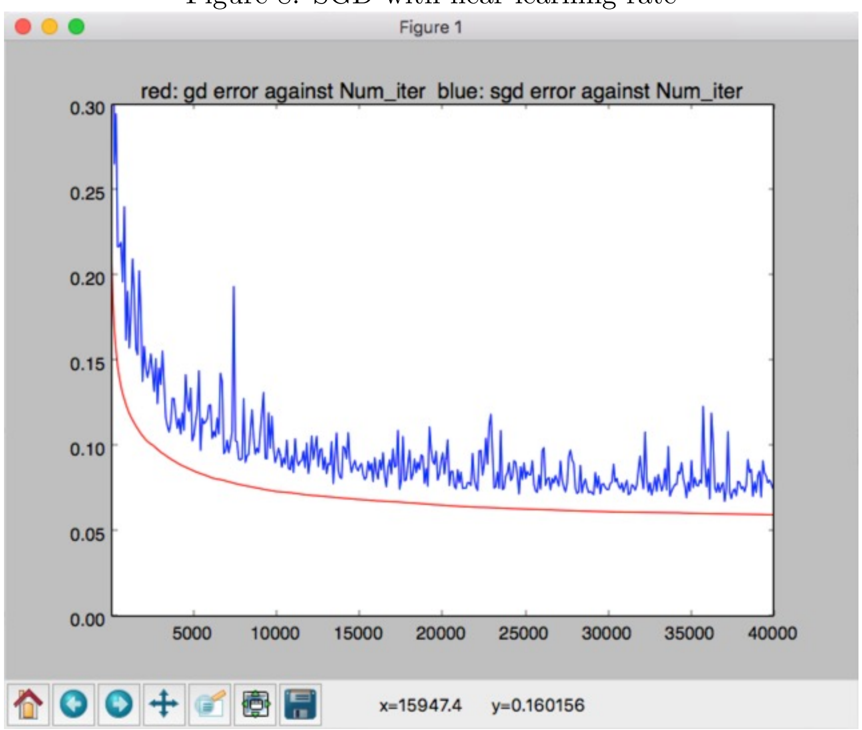

Figure 9: SGD with linear annealing learning rate

5

#5.

a). $A(A^TA+\mu I)=(AA^T+\mu I)A \Leftarrow A A^TA+\mu IA = AA^TA + A\mu I.$

and invertiable $\Leftarrow$ Both positive

$$(AA^T+\mu I)^{-1}\underbrace{A(A^TA+\mu I)(A^TA+\mu I)^{-1}}_{X}=\underbrace{(AA^T+\mu I)^{-1}(AA^T+\mu I)}_{X}A(A^TA+\mu I)^{-1}$$

$$\Updownarrow$$

$$(AA^T+\mu I)^{-1}A=A(A^TA+\mu I)^{-1} \quad \square$$

b). From Hw1 we have $W_* = W = (\lambda I + X^TX)^{-1}X^Ty$

from part a we write as $= X^T(XX^T+\mu I)^{-1}y$

$= X^TK \qquad K=(XX^T+\lambda I)^{-1}y$

$= \sum_{i=1}^{n} a_ix_i$

$$a_i = \left[(XX^T+\mu I)^{-1}y\right]_i$$

~~and which can also indicate that a_i is unique~~

~~since $(XX^T+\mu I)^{-1}$ is unique~~

W_* is unique since $W_* = (\lambda I+X^TX)^{-1}X^Ty$

and $(\lambda I + X^TX)$ invertiable then W_* uniquely defined.

c). $W = W_* + W_\perp \qquad W_* = \sum a_ix_i$ and $W_\perp$ from $Null(X)$

$XW_\perp = 0. \quad X_iW_\perp = 0.$

$$\underbrace{\frac{1}{n}\sum_{i=1}^{n} Loss(W^Tx_i, y_i)}_{A} + \underbrace{\mu\|W\|_2^2}_{} \leftarrow B$$

takes parameter W^Tx_i and y_i

$$W^Tx_i = (W_*^T + W_\perp^T)x_i = W_*^Tx_i$$

therefore W^Tx_i's value not depends on $W_\perp$

the value of part A not depends on $W_\perp$

~~c) $\min_{w \in \mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} loss(w^T x_i, y_i) + \mu\|w\|_2^2$~~

~~Loss function convex for $\forall k_1 < k_2$~~

~~$\lambda\, Loss(k_1, y_i) + (1-\lambda)\, loss(k_2, y_i) \geq loss(\lambda k_1 + (1-\lambda) k_2, y_i)$.~~

For Part B $= \mu\|w\|_2^2 = \mu(w_*^T + w_\perp^T)(w_* + w_\perp)$

$= \mu(w_*^T w_* + w_\perp^T w_\perp)$

if fix w_*, has minimum value when $w_\perp = 0$

therefore to ensure $\frac{1}{n}\sum_{i=1}^{n} Loss(w^T x_i, y_i) + \mu\|w\|_2^2$ has minimum value, $w_\perp$ should be zero

$w_* = \sum_{i=1}^{n} x_i a_i$

We never use the condition "Loss function is convex"

therefore the proof above is a general proof.

Loss not convex, optimal solution still has the form

$w_* = \sum_{i=1}^{n} a_i x_i$

#6. P(get those data)

$\prod P(y_i|x_i)$

$$= \prod N(w_0 + w_1 x_i, \sigma^2)(y_i)$$

$$= \prod \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - (w_0 + w_1 x_i))^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{[(y_1 - (w_0 + w_1 x_1))^2 + (y_2 - (w_0 + w_1 x_2))^2 - \cdots]}{2\sigma^2}}$$

Want to Maximize $\prod P(y_i|x_i)$

then Maximize $\sum_{i=1}^{n} (y_i - (w_0 + w_1 x_i))^2 = f(w_0, w_1)$

$f_{w_0} = \sum 2(y_i - (w_0 + w_1 x_i)) \cdot = 0$

$f_{w_1} = \sum -2(y_i - (w_0 + w_1 x_i))(-x_i) = 0$

$$\frac{f_{w_0}}{2} = -\sum_{i=1}^{n} y_i \; \cancel{+\sum_{i=1}^{n} w_0} + w_1 \sum_{i=1}^{n} x_i = 0.$$

$+ n w_0$

$$w_0 = \bar{y} - w_1 \bar{x}$$

plug in w_0.

$\cancel{\frac{\sum y_i - n w_0 - w_1 \sum x_i}{n} = 0}$

$\cancel{\bar{y} - \bar{y}}$

$f_{w_1} = 0$

$$= -\sum y_i x_i + w_0 \sum x_i + w_1 \sum x_i^2 = 0.$$

$$-\sum x_i y_i + \bar{y} \sum x_i - w_1 \bar{x} \sum x_i + w_1 \sum x_i^2 = 0$$

$$w_1 = \frac{\sum_i x_i y_i - \bar{y} \sum x_i}{\sum x_i^2 - \bar{x} \sum x_i} = \frac{\sum_i x_i y_i - \bar{y}\bar{x} \cdot n}{\sum_i x_i^2 - n\bar{x}^2}$$

#7. a). (0, 1) (0, -1) (-1, 0) (1, 0) all $\frac{1}{4}$

$Cov(X, Y) = E(X \cdot Y) = 0$ uncorrelated

But $P(X=0, Y=0) = 0 \neq P(X=0) \cdot P(Y=0)$ not independent.

b). consider X, Y.

$P(X=1) = \frac{1}{2}$ $P(Y=1) = \frac{1}{2}$ $P(X=1, Y=0) = \frac{1}{4} = P(X=1) \cdot P(Y=0)$

for all other cases $P(X, Y) = P(X) P(Y)$

for XZ YZ pair due to ~~symmetricity~~ symmetric propority they are all ~~mutually~~ pairwise independent

But they are not mutually independent

$P(X=1, Y=1, Z=1) = 0 \neq \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2}$

```python
from mnist import MNIST
import sklearn.metrics as metrics
import numpy as np
import scipy
import pdb
import time
from numpy.linalg import inv
from numpy.linalg import solve
import matplotlib.pyplot as plt

NUM_CLASSES = 10
d = 1000 # the raisen dimension
G_transpose = np.random.normal(scale = 0.1, size = (d, 784)) #the transpose of G, dim matched
b = np.random.random((d,1))*6.2832
def load_dataset():
    mndata = MNIST('./data/')
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0
    return (X_train, labels_train), (X_test, labels_test)


def train(X_train, y_train, reg=0):
    ''' Build a model from X_train -> y_train ''' #dim of X_train is 5000,600000
    a = np.dot(np.matrix.transpose(X_train), X_train) + reg*np.identity(d)
    y = np.dot(np.transpose(X_train), y_train)
    w = solve(a,y)
    return w

def train_gd(X_train, y_train, alpha=0.1, reg=0, num_iter=10000):
    ''' Build a model from X_train -> y_train using batch gradient descent '''
    #initalize a W
    alpha = alpha/X_train.shape[0]
    W = np.zeros((d,10))
    help1 = np.dot(np.transpose(X_train), X_train)
    help2 = np.dot(np.transpose(X_train), y_train)
    Wlist = []
    for i in range(num_iter):
        # if (i%100 == 0):
        #     pdb.set_trace()
        l = reg*W + np.dot(help1, W) - help2
        W = W - l*alpha
        if ((i+1) % 100 == 0):
            Wlist.append(W)
    return Wlist
    return W
def train_sgd(X_train, y_train, alpha=0.1, reg=0, num_iter=10000):
    ''' Build a  from X_train -> y_train using stochastic gradient descent '''
    W = np.zeros((d,10))
    Wlist = []#for plotting data
    for i in range(num_iter):
        index = np.random.randint(low = 0, high = 60000-1)
        vector = X_train[index].T
        yvector = y_train[index]
        derivative = reg*W + np.dot(vector[:, None], (np.dot(vector, W) - yvector)[None,:])
        W = W - derivative*alpha*(1-i/(num_iter)) #linear
        if ((i+1) % 100 == 0):
```

```python
            if ((i+1) % 100 == 0):
                Wlist.append(W)
    return Wlist
    return W

def one_hot(labels_train):
    '''Convert categorical labels 0,1,2,....9 to standard basis vectors in R^{10} ''' #return 60000*784
    return np.array([[1 if i == labels_train[k] else 0 for i in range(10)] for k in range(len(labels_train))])

def predict(model, X):
    ''' From model and data points, output prediction vectors '''
    result = np.dot(np.matrix.transpose(model), np.transpose(X)) #get a vector
    return [np.argmax(i) for i in np.matrix.transpose(result)]

def phi(X):
    ''' Featurize the inputs using random Fourier features '''
    X = np.cos(np.dot(G_transpose, np.transpose(X)) + b) #dim of X is 5000,60000
    return np.transpose(X) #60000,5000



if __name__ == "__main__":
    print("The data has been raisen to dimension {0}".format(d))
    (X_train, labels_train), (X_test, labels_test) = load_dataset()
    y_train = one_hot(labels_train)
    y_test = one_hot(labels_test)
    X_train, X_test = phi(X_train), phi(X_test)
    start_time = time.time()
    model = train(X_train, y_train, reg=0.1)
    print("Training through closed form solution takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Closed form solution")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))

    start_time = time.time()
    model = train_gd(X_train, y_train, alpha=1e-3, reg=0.1, num_iter=20000)[-1]
    print("Training though batch gradient descent takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Batch gradient descent")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))



    model = train_sgd(X_train, y_train, alpha=1e-3, reg=0.1, num_iter=500000)[-1]
    print("Training though stochastic gradient descent takes :{0}".format(time.time() - start_time))
    pred_labels_train = predict(model, X_train)
    pred_labels_test = predict(model, X_test)
    print("Stochastic gradient descent")
    print("Train accuracy: {0}".format(metrics.accuracy_score(labels_train, pred_labels_train)))
    print("Test accuracy: {0}".format(metrics.accuracy_score(labels_test, pred_labels_test)))
```

```python
#plot the error against the num_iter
WgdList = train_gd(X_train, y_train, alpha=1e-3, reg=0, num_iter=40000)
WsdList = train_sgd(X_train, y_train, alpha=1e-3, reg=0, num_iter=40000)
WgdErrorList = [1-metrics.accuracy_score(labels_train, predict(model, X_train)) for model in WgdList]
WsdErrorList = [1-metrics.accuracy_score(labels_train, predict(model, X_train)) for model in WsdList]
plt.figure()
x = np.arange(100,40000 + 1, 100)
print(x)
print(WgdErrorList)
print(WsdErrorList)

plt.plot(x, WgdErrorList, "r")
plt.plot(x, WsdErrorList, "b")
plt.axis([100,40000,0,0.3])
plt.title("red: gd error against Num_iter  blue: sgd error against Num_iter")
plt.show()
```