HW0-1

a)

Names	Email Address
Qingyang Zhao	jhun0324@berkeley.edu

Description of Team: Best Group Ever

How did I work?

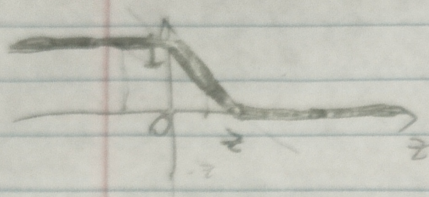
Comments:

b) I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Qingyang Zhao

HW10-02

a) Clip Loss Function



no this is not convex.

when $z_1 < 0$, $0 < z_2 \leq z$

$\lambda \text{clip}(z_1) + (1-\lambda)\text{clip}(z_2) \leq \text{clip}(\lambda z_1 + (1-\lambda) z_2)$

b) when y_i, $w^T x$ has different sign, there's loss of 1

when y_i, $w^T x$ has the same sign if $|w^T x| > 1$, pretty sure it is from "+1" class, there's no loss.

if $|w^T x| < 1$, there's loss smaller than 1, which means we want to maximize the margin

c) $R_S[w] = \frac{1}{n} \sum_{i=1}^{n} \text{loss}(w^T x_i, y_i)$

$= \frac{1}{n} \sum_{i=1}^{n} \text{clip}(w^T x_i, y_i)$

$|w^T x_i| > 1 \Rightarrow -1 < w^T x < 1$

which means $w^T x$ has a margin of 1

d) $E[R_S[w]] = E_D\left[\frac{1}{n} \sum_{i=1}^{n} \text{loss}(w^T x_i, y_i)\right]$

$= \frac{1}{n} \sum_{i=1}^{n} E_D[\text{loss}(w^T x_i, y_i)]$ ← i.i.d

$= \frac{1}{n} \cdot n \, E_D[\text{loss}(w^T x, y)]$

$= R[w]$

e) $\mathrm{Var}[R_S[w]] \le \frac{1}{n}$

$\mathrm{var}[\mathrm{loss}(w^T x, y)]$

$\mathrm{var}\left[\frac{1}{n}\sum_{i=1}^{n} \mathrm{loss}(w^T x_i, y_i)\right]$

$= E[(\mathrm{loss}(w^T x, y) - E[\mathrm{loss}(w^T x, y)])^2]$

$= \frac{1}{n^2}\sum_{i=1}^{n} \mathrm{var}[\mathrm{loss}(w^T x_i, y_i)]$

Since $\mathrm{loss}(w^T x, y) - E[\mathrm{loss}(w^T x, y)] \le 1$

$= \frac{1}{n^2}\sum_{i=1}^{n} \mathrm{var}[\mathrm{loss}(w^T x, y)] < n$

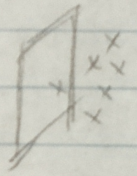
$\mathrm{var}[\mathrm{loss}(w^T x, y)] \le 1$

$\le \frac{1}{n^2} \cdot n \cdot 1$

$= \frac{1}{n}$

f) $R[w] = E_D[\mathrm{loss}(w^T x, y)]$

$= 1 \cdot \mathbb{P}(z < 0) + (1 - z)\,\mathbb{P}(0 \le z < 1) \to 0$



$R[w] = 0$ when $\mathbb{P}(z < 0) = 0 \wedge (1 - z) = 0$

There's no outliers and all points on $w^T x = 1$ Hyperplane beyond the hyperplane,

Thus when all points are $|w^T x| \ge 1$ and $w^T x y > 0$

which means $R_S[w] = 0$, but $R[w] > 0$

HW10-03

b) Stochastic converge much quickly.

c) coordinate Gradient Descent with sgd converge the fastest
c-sgd > (faster than) c-fls > (faster than) c-fgd

d) Kaczmar SGD can converge with in 10 points (In most cases) usually within first three points, uses the smallest # of sample points

e) SGD and Kaczmar SGD still performance the best
Coordinate SGD and FLS are similar
Batch GD convergence much slower
Coordinate GD doesn't converge

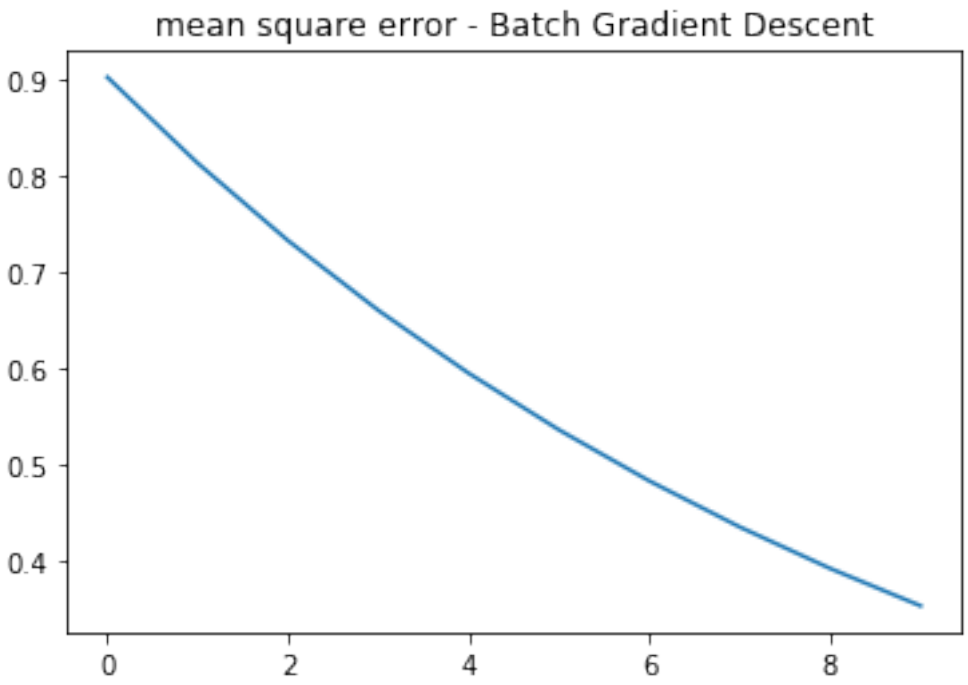
hw10

November 6, 2017

```
In [1]: import numpy as np
        import scipy.io as sio
        import matplotlib.pyplot as plt

In [2]: gd_data = sio.loadmat('gradient_descent_data.mat')

In [3]: gd_x = gd_data['x']
        gd_y = gd_data['y']

In [4]: def mean_square_error (w, gd_x, gd_y):
            return np.sum(np.power(np.dot(gd_x, w) - gd_y, 2))/np.size(gd_y)

        def feat_normalize(gd_x):
            mu = np.mean(gd_x, axis = 0)
            sigma = np.ones((1, np.size(gd_x, axis = 1)),dtype = np.float)
            for i in range(np.size(gd_x, axis = 1)):
                sigma[0][i] = np.sqrt(np.sum(np.power(gd_x[:,i] - mu[i], 2))/np.size(gd_x, axis
                gd_x[:,i] = (gd_x[:,i] - mu[i])/sigma[0][i]
            return gd_x
```

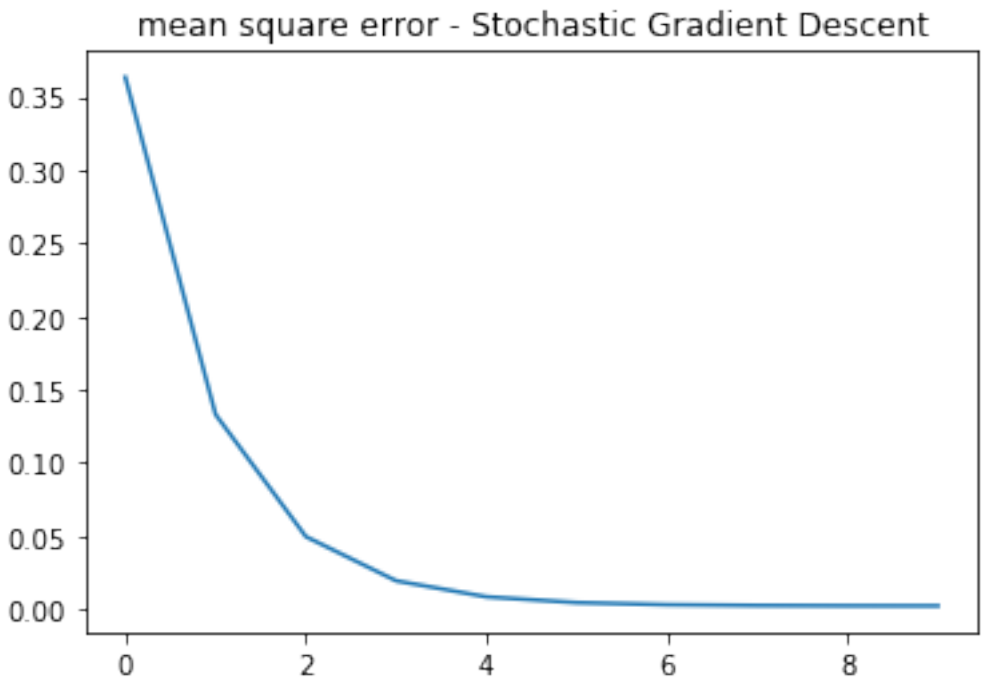
1 a) Batch Gradient Descent

```
In [5]: def batch_gd (gd_x, gd_y, init_w, lr = 0.05, ite = 10):
            w = init_w
            error = []
            gd_x = feat_normalize(gd_x)
            gd_y = feat_normalize(gd_y)
            for i in range(ite):
                w = w - lr * (np.dot(np.dot(gd_x.T, gd_x), w) - np.dot(gd_x.T, gd_y))/gd_y.shap
                error.append(mean_square_error(w, gd_x, gd_y))
            plt.plot(range(ite), error)
            plt.title('mean square error - Batch Gradient Descent')
            plt.show()
            return w
        w = batch_gd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), lr = 0.05, ite = 10)
```



2 b) Stochastic Gradient Descent

```
In [6]: def stochastic_gd(gd_x, gd_y, init_w, mini_batch = 1, lr = 0.005, epochs = 10):
            w = init_w.copy()
            gd_x = feat_normalize(gd_x)
            gd_y = feat_normalize(gd_y)
            error = []
            for i in range(epochs):
                for j in range(0, gd_y.shape[0], mini_batch):
                    end  = min(j + mini_batch, gd_y.shape[0])
                    w = w - lr * (np.dot(np.dot(gd_x[j:end,:].T, gd_x[j:end,:]), w)
                                  - np.dot(gd_x[j:end,:].T, gd_y[j:end,:]))/mini_batch
                error.append(mean_square_error(w, gd_x, gd_y))
            plt.plot(range(np.size(error)),error)
            plt.title('mean square error - Stochastic Gradient Descent')
            plt.show()
            return w

In [7]: w = stochastic_gd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), mini_batch = 1, lr = 0.000
```

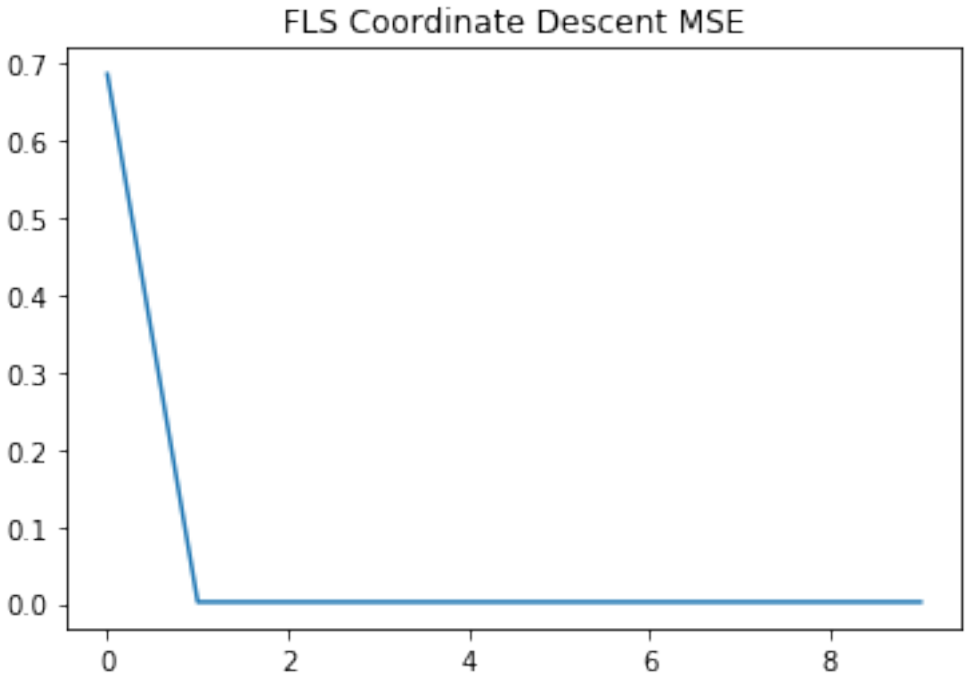


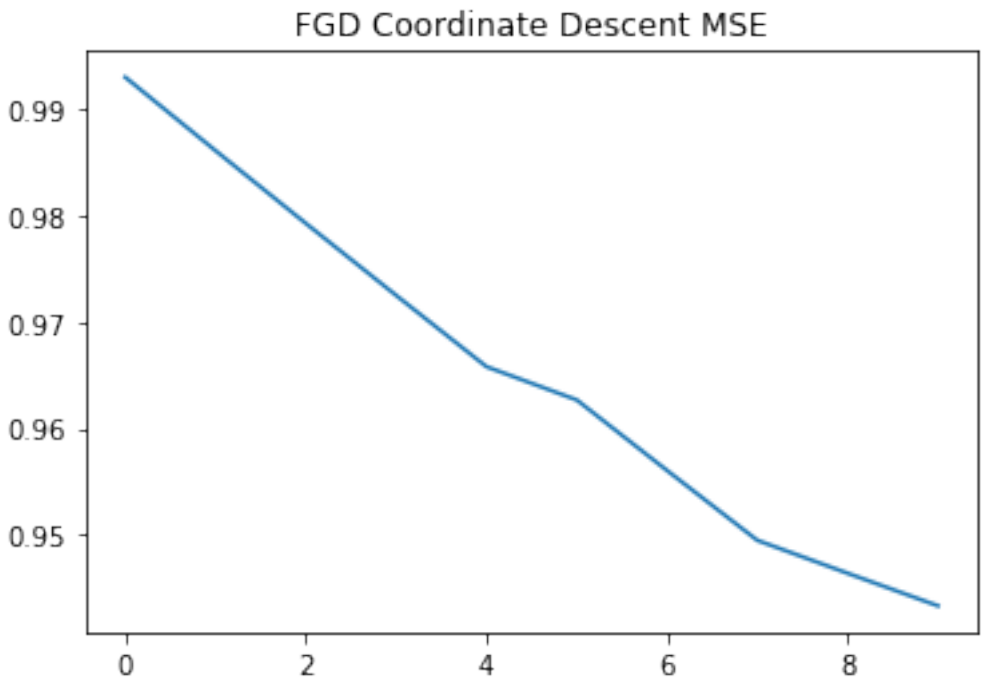
3 c) Coordinate Gradient Descent

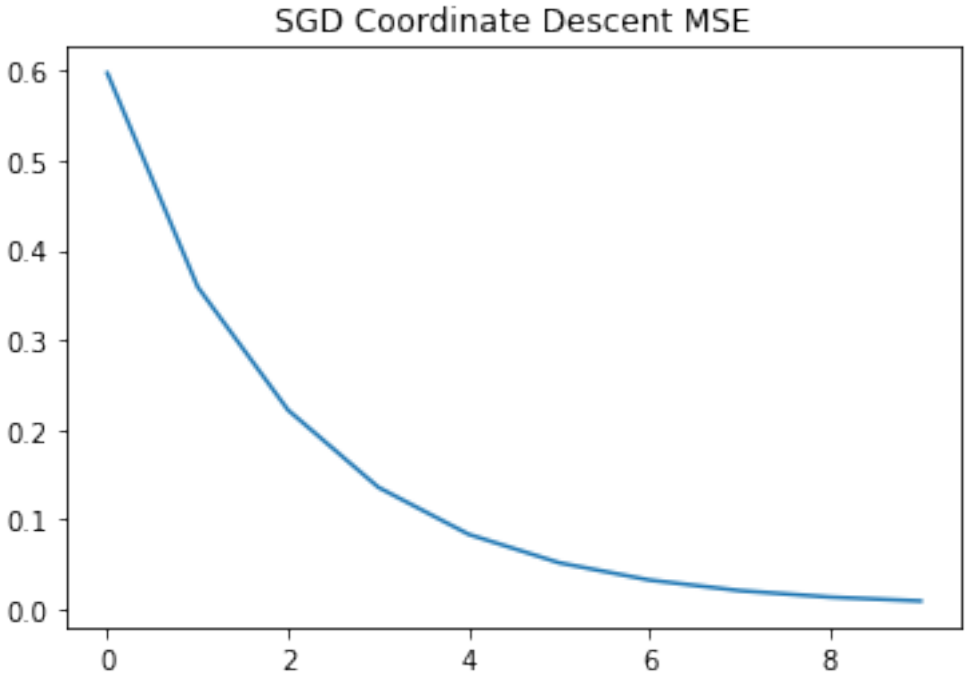
```
In [8]: def coordinate_gd(gd_x, gd_y, init_w, mini_batch = 1, lr = 0.005, ite = 100, algo = 'F
            w = init_w.copy()
            gd_x = feat_normalize(gd_x)
            gd_y = feat_normalize(gd_y)
            error = []
            if (algo == 'FLS'):
                for i in range(ite):
                    coord = np.random.randint(0,gd_x.shape[1])
                    w[coord] = np.dot(gd_x[:, coord].T,gd_y)/np.dot(gd_x[:,coord].T,gd_x[:,coo
                    error.append(mean_square_error(w, gd_x, gd_y))
                plt.plot(range(ite),error)
                plt.title('FLS Coordinate Descent MSE')
            elif (algo == 'FGD'):
                for i in range(ite):
                    coord = np.random.randint(0,gd_x.shape[1])
                    grad = np.dot((np.dot(gd_x, w) - gd_y).T, gd_x[:,coord])/gd_y.shape[0]
                    w[coord] = w[coord] - lr * grad
                    error.append(mean_square_error(w, gd_x, gd_y))
                plt.plot(range(ite),error)
                plt.title('FGD Coordinate Descent MSE')
            elif (algo == 'SGD'):
                for i in range(ite):
```

```
                for j in range(0, gd_y.shape[0], mini_batch):
                    coord = np.random.randint(0,gd_x.shape[1])
                    end = min(j + mini_batch, gd_y.shape[0])
                    grad = np.dot((np.dot(gd_x[j:end], w) - gd_y[j:end]).T, gd_x[j:end, co
                    w[coord] = w[coord] - lr * grad
                error.append(mean_square_error(w, gd_x, gd_y))
            plt.plot(range(ite),error)
            plt.title('SGD Coordinate Descent MSE')
        plt.show()
        return w
```

```
In [9]: w_FLS = coordinate_gd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), ite = 10, algo = 'FLS
        w_FGD = coordinate_gd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), lr = 0.005, ite = 10,
        w_SGD = coordinate_gd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), mini_batch = 1, lr = (
```



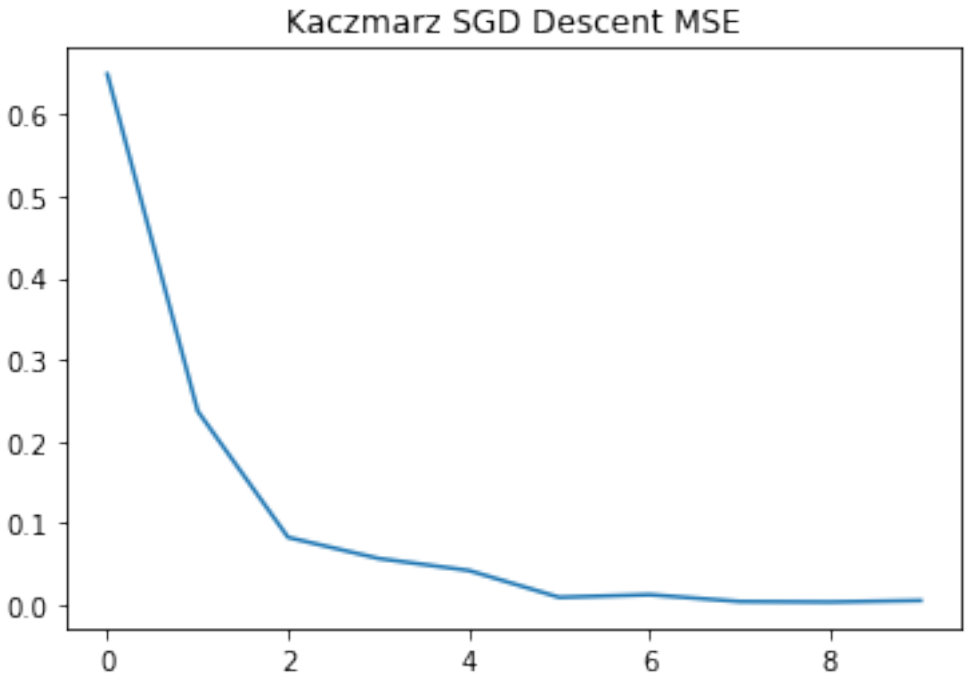

FGD Coordinate Descent MSE
0.99
0.98
0.97
0.96
0.95
0
2
4
6
8


SGD Coordinate Descent MSE
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
2
4
6
8

4 d) Kaczmarz Stochastic Gradient Descent

```
In [130]: def kaczmarz_sgd(gd_x, gd_y, init_w, ite = 10):
              # initialize probability for learning rate
              prob = []
              for i in range(gd_y.shape[0]):
                  prob.append((np.linalg.norm(gd_x[i,:],2)/np.linalg.norm(gd_x,2))**2)
              w = init_w.copy()
              gd_x = feat_normalize(gd_x)
              gd_y = feat_normalize(gd_y)
              error = []
              for j in range(1):
                  for k in range(ite):
                      # get the random picked index
                      ind = np.random.choice(range(gd_y.shape[0]), 1, prob)
                      grad = (np.dot(gd_x[ind], w) - gd_y[ind])*gd_x[ind]
                      lr = 1.0/(np.linalg.norm(gd_x[ind,:],2)**2)
                      w = w - lr * grad.T
                      error.append(mean_square_error(w, gd_x, gd_y))
              plt.plot(range(ite),error)
              plt.title('Kaczmarz SGD Descent MSE')
              plt.show()
              return w
```

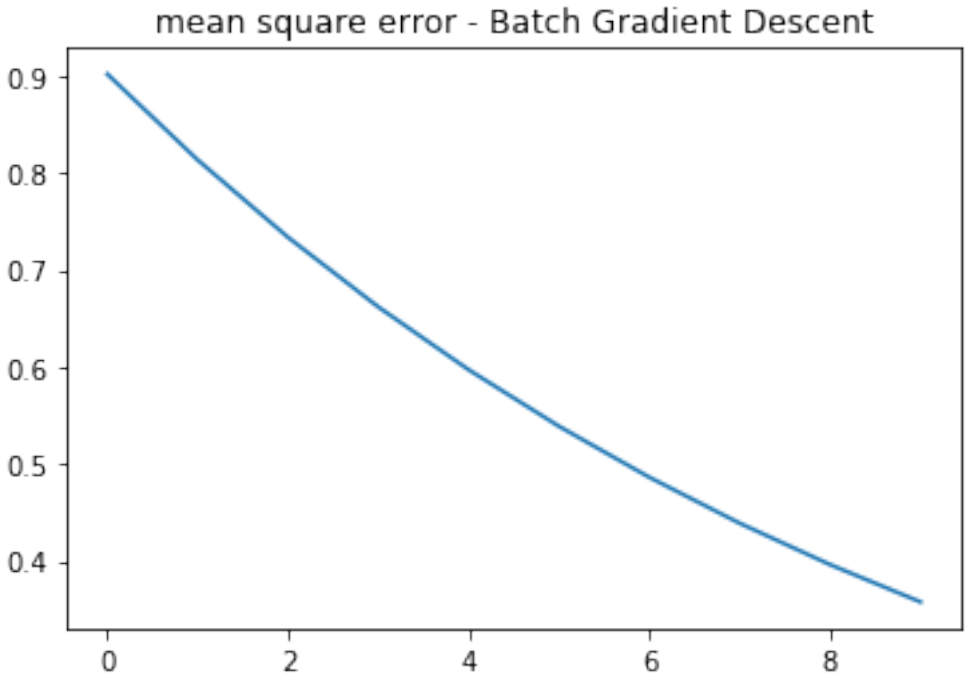
```
In [160]: w = kaczmarz_sgd(gd_x.copy(), gd_y.copy(), np.zeros((2,1)), ite = 10)
          print(w)
```

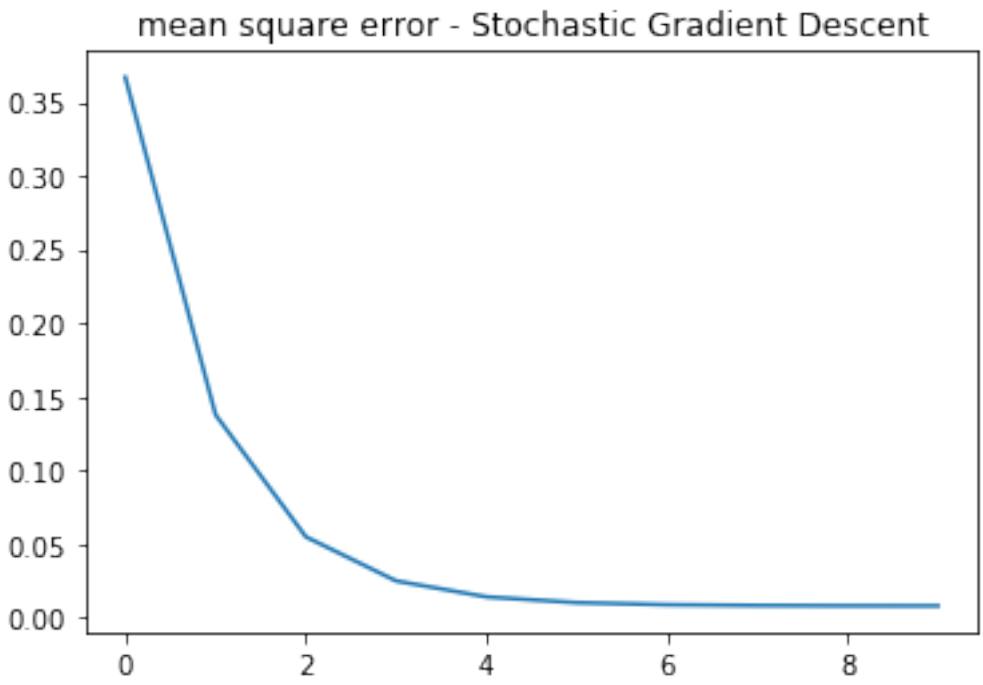


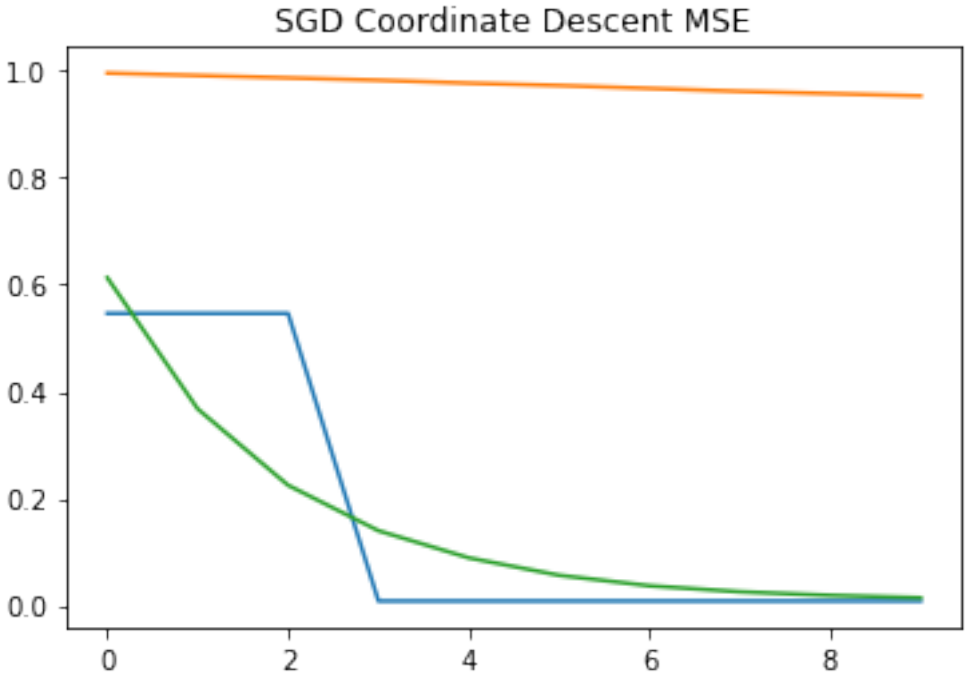
```
[[ 0.81994384]
 [ 0.49208224]]
```

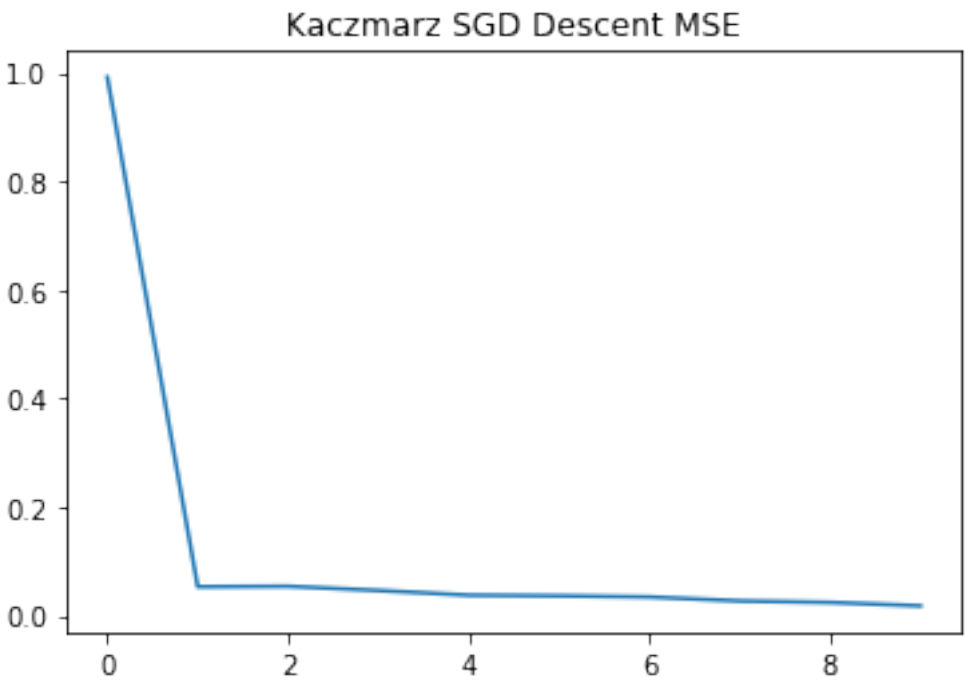
5 e) Redo nonlinear data

```
In [152]: gd_y_non = gd_y + np.asarray([np.power(gd_x[:,1], 3)*0.1]).T
          w_gd_non = batch_gd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), lr = 0.05, ite =
          w_sgd_non = stochastic_gd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), mini_batch =
          w_FLS_non = coordinate_gd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), ite = 10, al
          w_FGD_non = coordinate_gd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), lr = 0.005,
          w_SGD_non = coordinate_gd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), mini_batch =
          w_KSGD_non = kaczmarz_sgd(gd_x.copy(), gd_y_non.copy(), np.zeros((2,1)), ite = 10)
          plt.show()
```



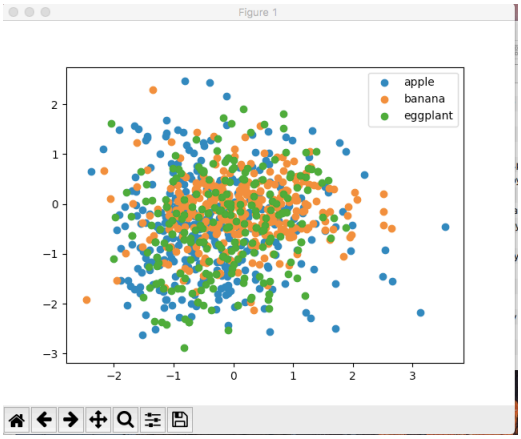

mean square error - Stochastic Gradient Descent
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
2
4
6
8


SGD Coordinate Descent MSE
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8


Kaczmarz SGD Descent MSE
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

HW10 - 4

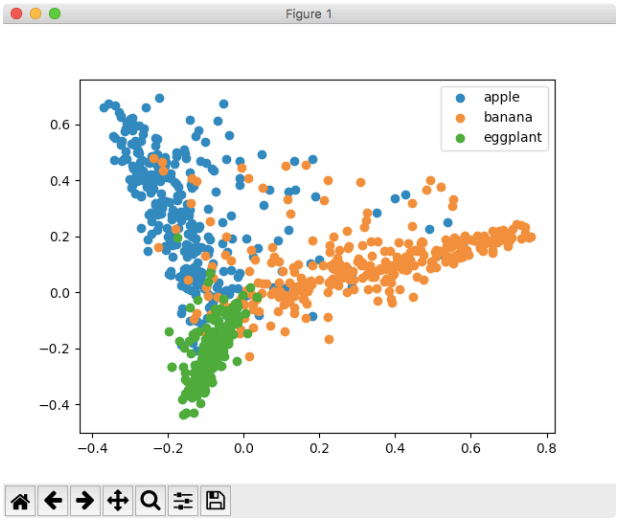
a) Random projection



code:

```
def get_random_proj(self):
    '''
    Return A which is size 2 by 729
    '''
    A = np.random.normal(loc = 0.0, scale = 1.0, size = (2, 729))
    return A
```

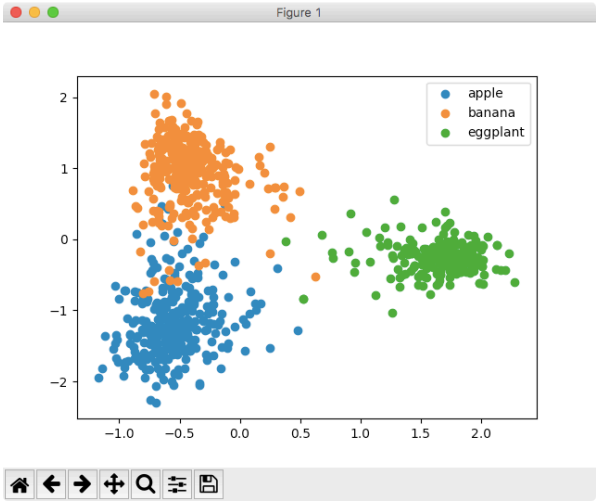
b) PCA



code:

```
def pca_projection(self,X,Y):

    '''
    Return U_2^T
    '''
    x = X.copy()
    y = Y.copy()

    x, x = subtract_mean_from_data(x, x)
    x_cov = compute_covariance_matrix(x, x)
    U, _, _ = svd(x_cov)
    return U[:, 0:2].T
```

c) CCA

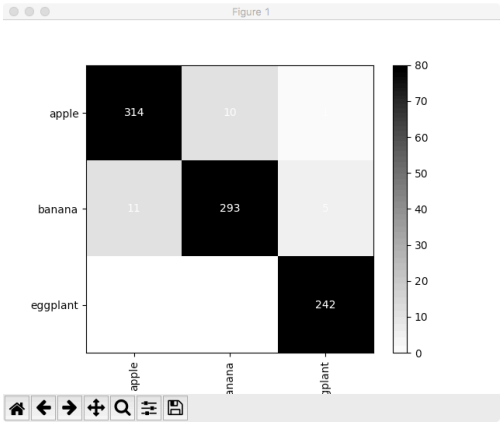


Code:

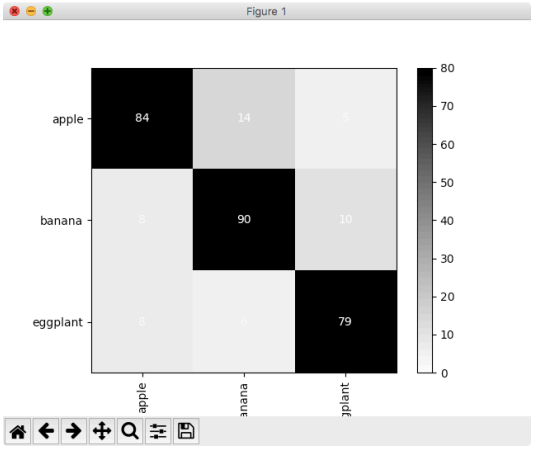
```
def cca_projection(self,X,Y,k=2):

    '''
    Return U_K^T, \Simgma_{XX}^{-1/2}
    '''

    ###SCALE AN IDENTITY MATRIX BY THIS TERM AND ADD TO COMPUTED COVA
    reg = 1e-5
    x = X.copy()
    y = Y.copy()

    # y to one-hot
    y_onehot = create_one_hot_label(y, self.NUM_CLASSES)

    x, y_onehot = subtract_mean_from_data(x, y_onehot)
    x_cov = compute_covariance_matrix(x, x)
    y_cov = compute_covariance_matrix(y_onehot, y_onehot)
    xy_cov = compute_covariance_matrix(x, y_onehot)
    x_cov = x_cov + np.eye(self.d_x) * reg

    u_x, diag_x, v_x = svd(x_cov)
    diag_x = 1/np.sqrt(diag_x)
    u_y, diag_y, v_y = svd(y_cov)
    diag_y = 1/np.sqrt(diag_y)
    x_cov_mh = np.dot(np.dot(u_x, np.diag(diag_x)), v_x)
    y_cov_mh = np.dot(np.dot(u_y, np.diag(diag_y)), v_y)
    cc_mat = np.dot(np.dot(x_cov_mh, xy_cov), y_cov_mh)
    #cc_mat, x_cov_mh, _, _ = self.cc_matrix(x, y_onehot, reg)

    U, _, _ = svd(cc_mat)
    return U[:, 0:k].T, x_cov_mh
```

d) Ridge ConfusionMatrix

Training:



test:



code:

```python
class Ridge_Model():

    def __init__(self,class_labels):

        ###RIDGE HYPERPARAMETER
        self.lmda = 1.0
        self.num_classes = np.size(class_labels)


    def train_model(self,X,Y):
        '''
        FILL IN CODE TO TRAIN MODEL
        MAKE SURE TO ADD HYPERPARAMTER TO MODEL

        '''
        y = Y.copy()
        x = np.asarray(X.copy())
        y_onehot = create_one_hot_label(y, self.num_classes)

        self.w = LA.solve(np.dot(x.T, x) + np.eye(x.shape[1]) * self.lmda, np.dot(x.T, y_onehot))


    def eval(self,x):
        '''
        Fill in code to evaluate model and return a prediction
        Prediction should be an integer specifying a class
        '''
        return np.argmax(np.dot(x, self.w))
```