CS 189: Introduction to Machine Learning

Fall 2017

•

Homework 10

•

Solutions by

Jinhong Du

3033483677

Question 1

(a)

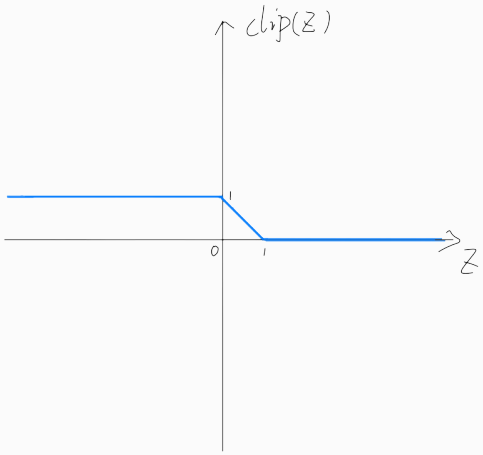
Jinhong Du
jaydu@berkeley.edu

(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.

Question 2

(a)



It is not convex since choosing $x_1 = -0.5, x_2 = 0.5$, then

$$\begin{aligned}\frac{clip(-0.5) + clip(0.5)}{2} &= \frac{1}{2} \\ &\leqslant 1 = clip\left(\frac{-0.5 + 0.5}{2}\right)\end{aligned}$$

(b)

When $z \geqslant 1$, it means that
(1) $w^T x$ and y have the same sign;
(2) $w^T x \geqslant 1$, i.e. it is very posibile to predict x into class y.
So the $clip(z) = 0$ at this case.
When $z < 0$, it means that $w^T x$ and y have different signs, i.e., the predict label for x is different from the true label y. So the loss function has the biggest loss, $clip(z) = 1$ at this case.
When $0 \leqslant z < 1$, even though $w^T x$ and y have the same sign. Since $w^T x < 1$, we are not pretty sure that we classify x at the right label. So it's better to punish this with $clip(z) = 1 - z$, the less uncertainty of prediction($w^T x$ closer to 1) will cause less loss.
It is reasonable to look at $yw^T x$ since classifying x into a right label leads to $yw^T x > 0$ and otherwise $yw^T x \leqslant 0$. So we can use $yw^T x$ to describle how accurately the model predicts. The bigger $w^T x$ means the better accuracy.

(c)

$\because$

$$loss(z) = clip(z) \geqslant 0$$

Solution (cont.)

$\therefore$

$$R_S[w] = \frac{1}{n}\sum_{i=1}^{n} loss(w^T x_i y_i) \geqslant 0$$

$\therefore$ $R_S[w] = 0$ only when $\forall\ i \in \{1, 2, \cdots, n\}$, $loss(w^T x_i y_i) = 0$

i.e.

$$w^T x_i y_i \geqslant 1$$

$\because$ $y_i = \pm 1$

$\therefore$ the data with different labels lie in $w^T x \geqslant 1$ and $w^T x \leqslant -1$ respectively

$\because$ $\|w\|_2^2 < 1$

$\therefore$ the hyperplanes define by w has margin $\dfrac{|1-(-1)|}{2\|w\|_2} \geqslant 1$

(d)

Suppose that the population random variable is $(X, Y) \sim \mathscr{D}$. Then $\mathbb{E}x_i = \mathbb{E}X, Var(x_i) = Var(X), \mathbb{E}y_i = \mathbb{E}Y, Var(y_i) = Var(Y)$, and therefore $\mathbb{E}[x_i y_i] = \mathbb{E}[XY]$.

$\because$ $clip(z)$ is a linear function of z

$\therefore$

$$\begin{aligned} R[w] &= \mathbb{E}_{\mathscr{D}}[loss(w^T x, y)] \\ &= \mathbb{E}[clip(w^T XY)] \\ &= clip(\mathbb{E}[w^T XY]) \\ &= clip(w^T \mathbb{E}[XY]) \end{aligned}$$

$\because$

$$\begin{aligned} \mathbb{E}_S[R_S[w]] &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}[loss(w^T x_i, y_i)] \\ &= \frac{1}{n}\sum_{i=1}^{n} clip(\mathbb{E}[w^T x_i y_i]) \\ &= \frac{1}{n}\sum_{i=1}^{n} clip(w^T \mathbb{E}[x_i y_i]) \\ &= \frac{1}{n}\sum_{i=1}^{n} clip(w^T \mathbb{E}[XY]) \\ &= clip(w^T \mathbb{E}[XY]) \end{aligned}$$

$\therefore$

$$\mathbb{E}_S[R_S[w]] = R[w]$$

(e)

$\because$ $\forall\ (x, y) \in S$, $loss(w^T x, y) = clip(w^T xy) \in [0, 1]$

$\therefore$ $\mathbb{E}[clip(w^T xy)]^2 \leqslant 1$

Solution (cont.)

$\because$ (x_i, y_i) and (x_j, y_j) $(i \neq j)$ are independnet

$\therefore$ $clip(w^T x_i y_i)$ and $clip(w^T x_j y_j)$ are independent, i.e. $Cov[clip(w^T x_i y_i), clip(w^T x_j y_j)] = 0$ $(i \neq j)$

$$
\begin{aligned}
Var(R_S[s]) &= \frac{1}{n^2}\sum_{i=1}^{n} Var[clip(w^T x_i y_i)] + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} Cov[clip(w^T x_i y_i), clip(w^T x_j y_j)] \\
&= \frac{1}{n^2}\sum_{i=1}^{n} Var[clip(w^T x_i y_i)] \\
&= \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{E}[clip(w^T x_i y_i)]^2 - \frac{1}{n^2}\sum_{i=1}^{n} \{\mathbb{E}[clip(w^T x_i y_i)]\}^2 \\
&\leqslant \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{E}[clip(w^T x_i y_i)]^2 \\
&\leqslant \frac{1}{n^2} \cdot n \\
&= \frac{1}{n}
\end{aligned}
$$

(f)

Let $w = 2$, $S = \{(1,1), (-1,-1)\}$

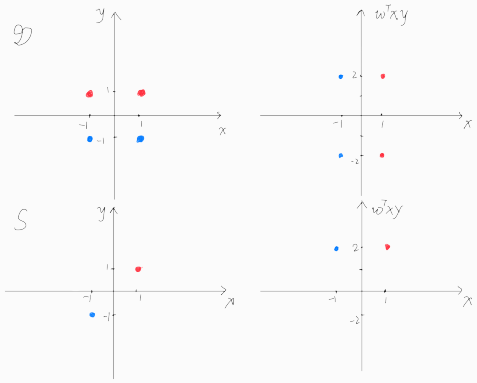
Let $\mathscr{D}$ be defined by $f(x,y) = \begin{cases} \frac{1}{4} & , x = -1, y = -1 \\ \frac{1}{4} & , x = -1, y = 1 \\ \frac{1}{4} & , x = 1, y = -1 \\ \frac{1}{4} & , x = 1, y = 1 \end{cases}$

Then

$$R[w] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} > 0$$

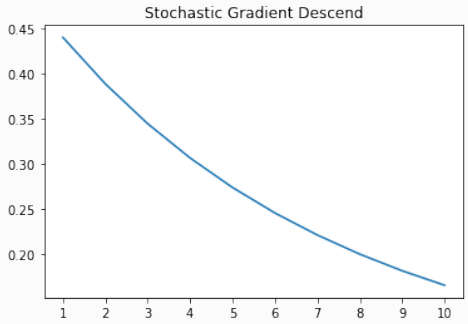
and

$$R_S[w] = 0$$



Question 3

(a)

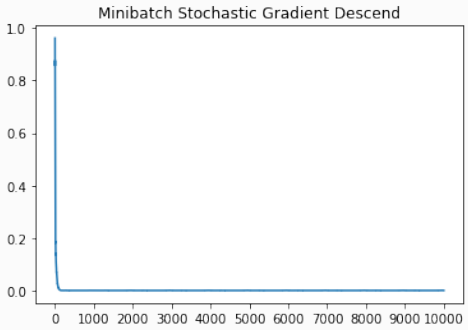



```python
class SGD(object):
    def __init__(self,
                 total_step_count = 10,
                 learning_rate = 0.05,
                 err = 0):
        self.total_step_count = total_step_count
        self.learning_rate = learning_rate
        self.err = err
        self.MSE = np.zeros((total_step_count,1))

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w):
        W = [np.array(w)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step(x, y, W[-1], k)
            self.MSE[k] = np.mean(np.linalg.norm(np.dot(x,new)-y,
              axis=1)**2)/2
            if np.linalg.norm(W[-1]-new)<self.err:
                break
            else:
                W.append(new)
        return np.array(W)

```

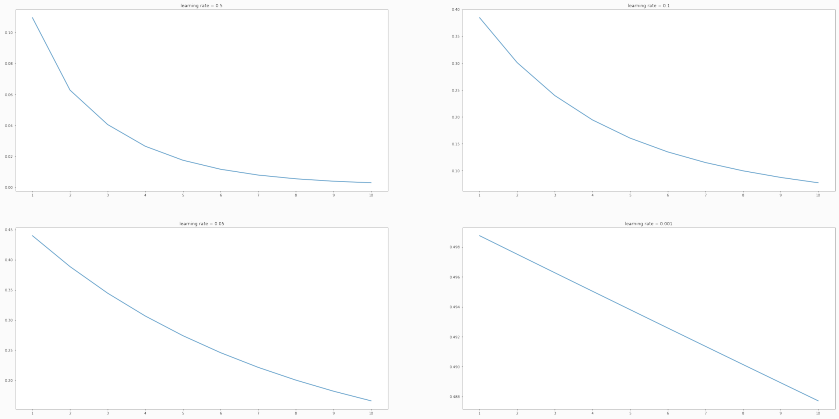
Solution (cont.)

```
31        def train(self,x,y):
32            _,m = np.shape(x)
33            _,n = np.shape(y)
34            w = np.zeros((m,n))
35            self.w = self.gradient_descend(x, y, w)
36    sgd = SGD()
37    sgd.train(train_x,train_y)
```

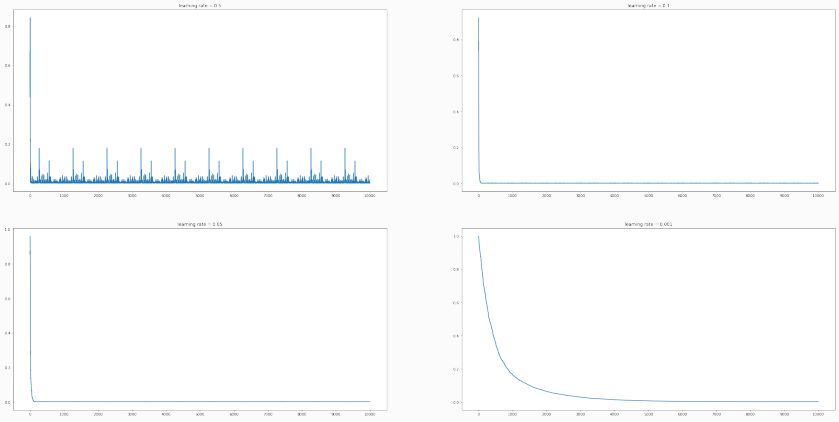
(b)



MSE of SGD vs Learning Rates



MSE of BSGD vs Learning Rates



Solution (cont.)

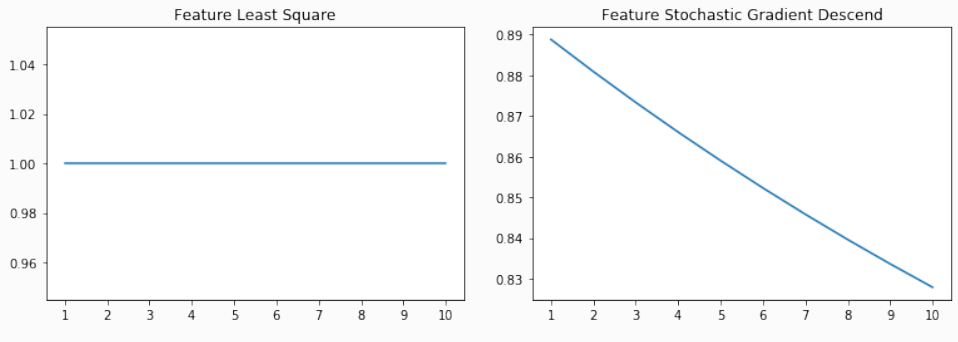
Given learning rate to be in $\{0.5, 0.1, 0.05, 0.01\}$.
For SGD, increasing the learning rates will benifit the training. 0.5 is the best learning rate for SGD.
For BSGD, increasing the learning rates will benifit the training at first and then lower training effect. 0.1 is the best learning rate for BSGD.

```
class BSGD(object):
    def __init__(self,
                 total_step_count = 10,
                 learning_rate = 0.05,
                 err = 0):
        self.total_step_count = total_step_count
        self.learning_rate = learning_rate
        self.err = err
        self.MSE = []

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w, batchSize):
        W = [np.array(w)]
        for k in range(self.total_step_count):
            for i in range(0,len(y),batchSize):
                end = min(i+batchSize,len(y))
                batchX = x[i:end,:]
                batchY = y[i:end,:]

                new = self.gradient_descend_step(batchX, batchY,
                   W[-1], k)
                self.MSE.append(np.mean(np.linalg.norm(np.dot(x,new)
                  -y,axis=1)**2))
                if np.linalg.norm(W[-1]-new)<self.err:
                    break
                else:
                    W.append(new)
        return np.array(W)

    def train(self, x, y, batchSize):
        _,m = np.shape(x)
        _,n = np.shape(y)
```

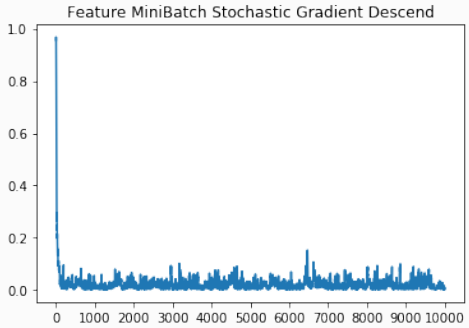
Solution (cont.)

```
            self.w = np.zeros((m,n))
            self.w = self.gradient_descend(x,y,self.w,batchSize)
bsgd = BSGD()
bsgd.train(train_x,train_y,1)

learning_rate = [0.1,0.75,0.05,0.001]
sgd_group = [SGD(learning_rate=learning_rate[i]) for i in range(
  len(learning_rate))]
plt.figure(figsize=(40,20))
for i in range(len(learning_rate)):
    plt.subplot(2,2,i+1)
    sgd_group[i].train(train_x,train_y)
    plt.plot(np.arange(1,11),sgd_group[i].MSE)
    plt.xticks(np.arange(1,11),np.arange(1,11))
    plt.title('learning rate = '+str(learning_rate[i]))
plt.show()

learning_rate = [0.1,0.75,0.05,0.001]
bsgd_group = [BSGD(learning_rate= learning_rate[i]) for i in range(
  len(learning_rate))]
plt.figure(figsize=(40,20))
for i in range(len(learning_rate)):
    plt.subplot(2,2,i+1)
    bsgd_group[i].train(train_x,train_y,1)
    plt.plot(np.arange(1,10001),bsgd_group[i].MSE)
    plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
    plt.title('learning rate = '+str(learning_rate[i]))
plt.show()
```

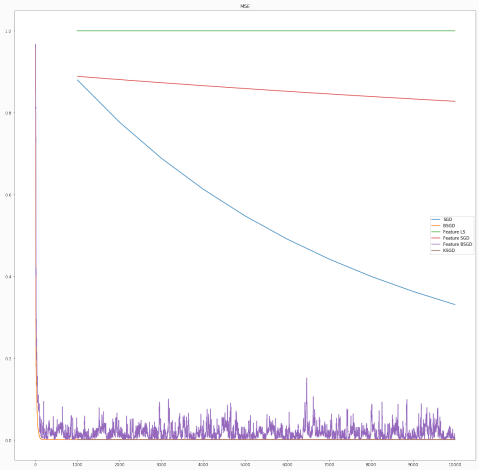
(c)



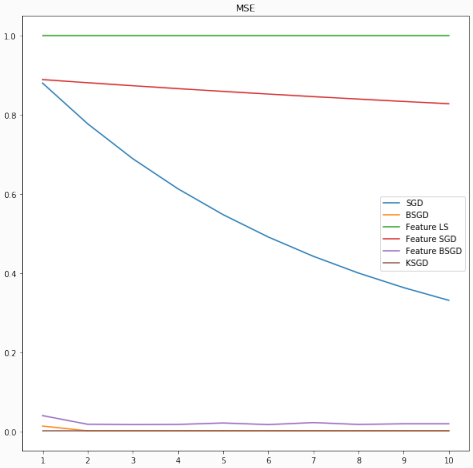
Solution (cont.)



MSE



smooth MSE



Both the Feature SGD and Feature BSGD performs worse than SGD and BSGD respectively.

```python
class FLS(object):
    def __init__(self,
                 total_step_count = 10,
                 err = 0):
        self.total_step_count = total_step_count
        self.err = err
        self.MSE = []

    def LS(self,x,y):
        return np.sum(x*y)/np.sum(x*x)

    def train(self, x, y):
        _,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        for i in range(self.total_step_count):
            index = np.random.randint(0,m)
            self.w[index,:] = self.LS(x[:,index],y)
            self.MSE.append(np.mean(np.linalg.norm(np.dot(x,self.w)
```

Solution (cont.)

```
                -y,axis=1)**2))
np.random.seed(0)
fls = FLS()
fls.train(train_x,train_y)
class FSGD(object):
    def __init__(self,
                 total_step_count = 10,
                 learning_rate = 0.05):
        self.total_step_count = total_step_count
        self.learning_rate = learning_rate
        self.MSE = np.zeros((total_step_count,1))

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w):
        x = x[:,np.newaxis]
        w = w[np.newaxis,:]
        W = [np.array(w)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step(x, y, W[-1], k)
            self.MSE[k] = np.mean(np.linalg.norm(np.dot(x,new)-y,
              axis=1)**2)
            W.append(new)
        return W[-1]

    def train(self,x,y):
        _,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        for i in range(self.total_step_count):
            index = np.random.randint(0,m)
            self.w[index,:] = self.gradient_descend(x[:,index],y,
              self.w[index,:])
np.random.seed(0)
fsgd = FSGD()
fsgd.train(train_x,train_y)
class FBSGD(object):
```

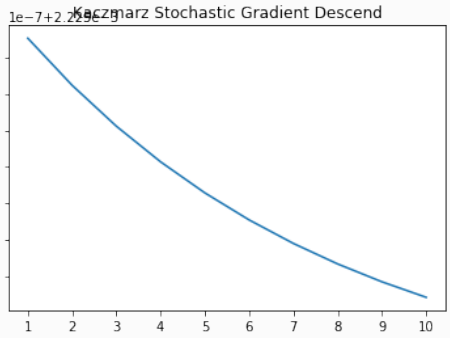
Solution (cont.)

```
63      def __init__(self,
64                   total_step_count = 10,
65                   learning_rate = 0.05):
66          self.total_step_count = total_step_count
67          self.learning_rate = learning_rate
68          self.MSE = []
69
70      def gradient(self, x, y, w):
71          return np.dot(x.T,np.dot(x,w)-y)/len(y)
72
73
74      def gradient_descend_step(self,x, y, w, step_count):
75          return w - self.learning_rate * self.gradient(x, y, w)
76
77      def train(self, x, y, batchSize):
78          _,m = np.shape(x)
79          _,n = np.shape(y)
80          self.w = np.zeros((m,n))
81          for k in range(self.total_step_count):
82              for i in range(0,len(y),batchSize):
83                  index = np.random.randint(0,m)
84                  end = min(i+batchSize,len(y))
85                  batchX = x[i:end,index]
86                  batchY = y[i:end,:]
87                  batchX = batchX[:,np.newaxis]
88                  w = self.w[index,:]
89                  w = w[np.newaxis,:]
90                  self.w[index,:] = self.gradient_descend_step(
91                    batchX, batchY, w, k)
92                  self.MSE.append(np.mean(np.linalg.norm(
93                    np.dot(x,self.w)-y,axis=1)**2))
94  np.random.seed(0)
95  fbsgd = FBSGD()
96  fbsgd.train(train_x,train_y,1)
```

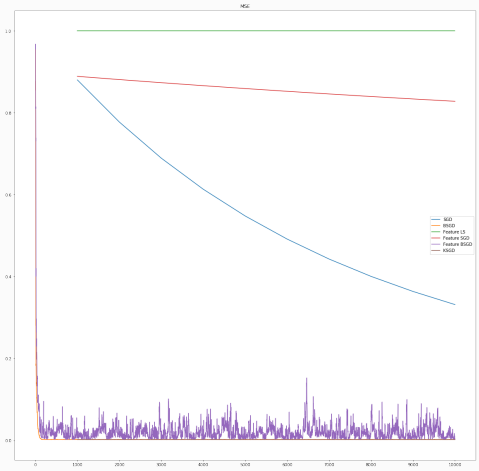
(d)

As we can see, the Kaczmarz SGD performs best and BSG ranks second. And both Feature SGD nad Feature BSGD performs worse than SGD and BSGD since they use less features to train at one time.

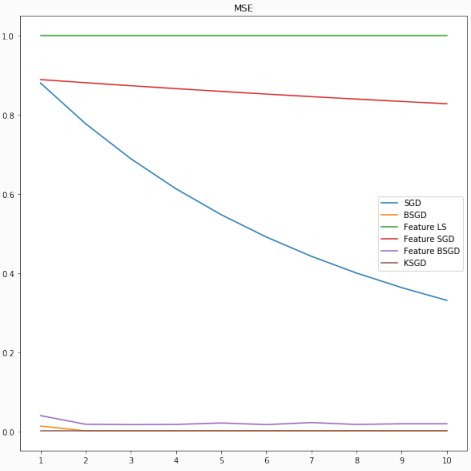
Solution (cont.)



MSE



smooth MSE



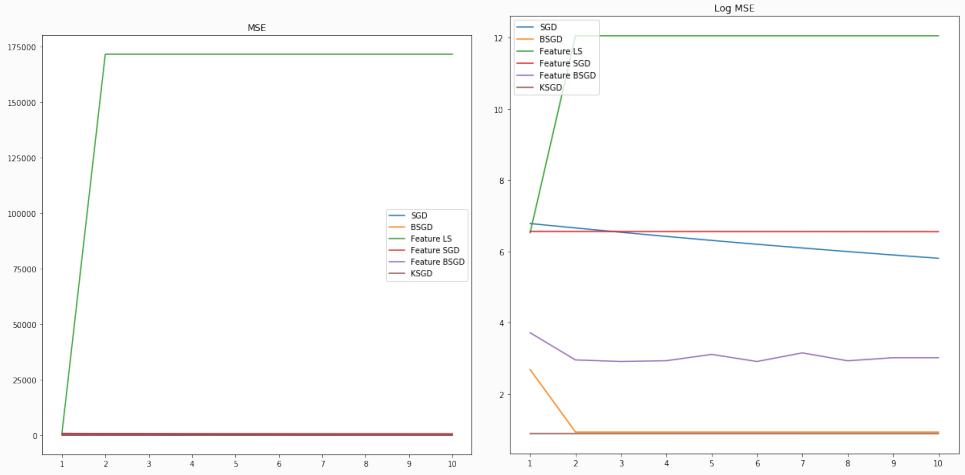

```python
class KSGD(object):
    def __init__(self,
                 total_step_count = 10):
        self.total_step_count = total_step_count
        self.MSE = np.zeros((total_step_count,1))

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w):
        W = [np.array(w)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step(x, y, W[-1], k)
            self.MSE[k] = np.mean(np.linalg.norm(
              np.dot(x,new)-y,axis=1)**2)
```

Solution (cont.)

```
                W.append(new)
            return W[-1]

        def train(self,x,y):
            r,m = np.shape(x)
            _,n = np.shape(y)
            self.w = np.zeros((m,n))
            for i in range(self.total_step_count):
                J = [np.linalg.norm(x[i,:])**2 for i in range(r)]
                j = np.argmax(J/np.linalg.norm(x)**2)
                self.learning_rate = 1/J[j]
                self.w = self.gradient_descend(x,y,self.w)
    ksgd = KSGD()
    ksgd.train(train_x,train_y)
```

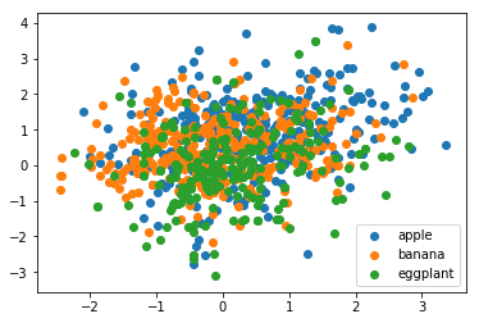
(e)

After adding some nonlinearity, Feature LS didn't work well while the others works well. And both Feature SGD and Feature BSGD perform worse than SGD and BSGD. And Kaczmarz SGD performs best.



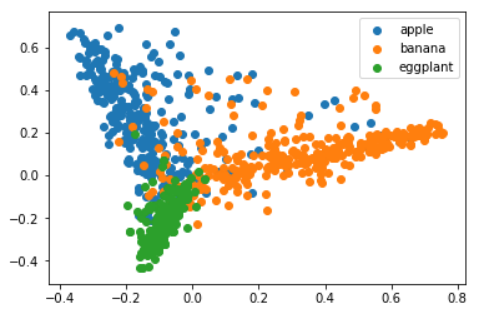
Question 4

(a)



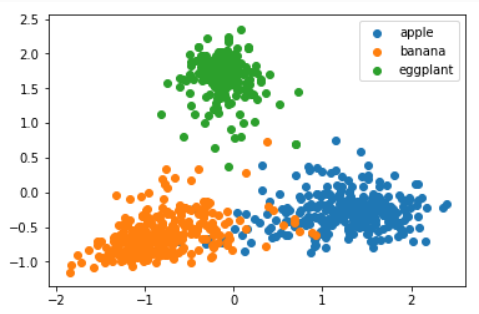

```
def get_random_proj(self):
        '''
        Return A which is size 2 by 729
        '''
    return np.random.normal(0,1,(2,729))
```

(b)



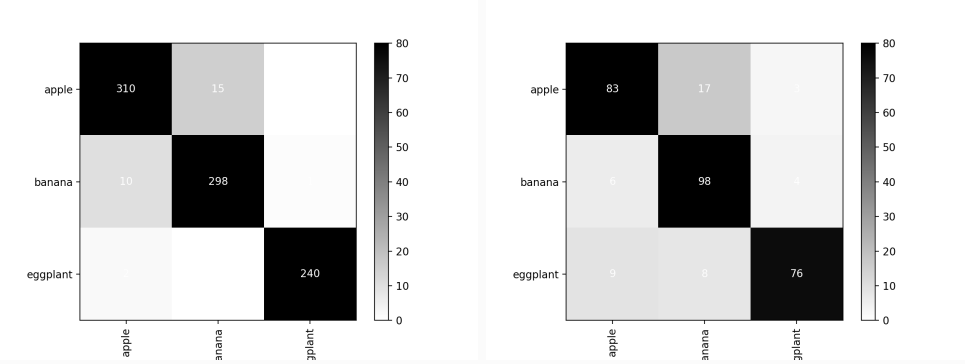

```
def pca_projection(self,X,Y):

        '''
        Return U_2^T
        '''
        X2,Y2 = subtract_mean_from_data(X,Y)
        SigmaXX = compute_covariance_matrix(X2,X2)
        u,s,vT = svd(SigmaXX)
        return u[:,:2].T
```

(c)




```
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 COVARIANCE MAT
        reg = 1e-5
        n = len(np.unique(Y))
        onehot_Y = create_one_hot_label(Y,n)
        X2,Y2 = subtract_mean_from_data(X,onehot_Y)
        SigmaXX = compute_covariance_matrix(X2,X2)
        SigmaXY = compute_covariance_matrix(X2,Y2)
        SigmaYY = compute_covariance_matrix(Y2,Y2)
        SigmaXX_2 = inv(sqrtm(SigmaXX+reg*np.eye(max(X[0].shape))))
        u,s,vT = svd(SigmaXX_2.dot(SigmaXY).dot(sqrtm(SigmaYY+reg*np.eye(max(Y2[0].
        return u[:,:k].T,SigmaXX_2
```

(d)

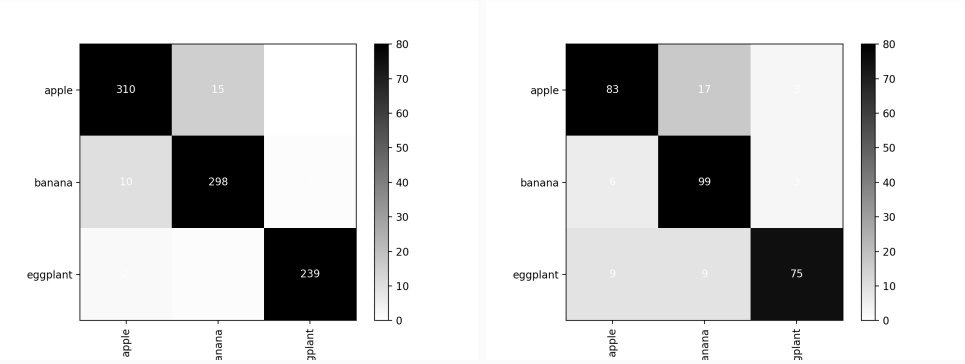



```
class Ridge_Model():

        def __init__(self,class_labels):
```

Solution (cont.)

```
                ###RIDGE HYPERPARAMETER
                self.lmda = 1.0

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

                '''
                n = len(np.unique(Y))
                onehot_Y = create_one_hot_label(Y,n)
                self.clf = Ridge(self.lmda)
                self.clf.fit(X,onehot_Y)

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

(e)

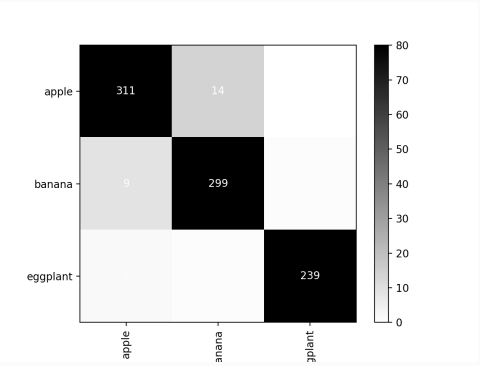


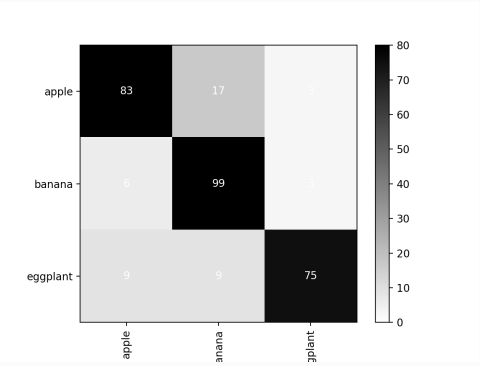

```
class LDA_Model():

        def __init__(self,class_labels):
                self.reg_cov = 0.001
                self.NUM_CLASSES = len(class_labels)

```

Solution (cont.)

```
        def train_model(self,X,Y):
                '''
                FILL IN CODE TO TRAIN MODEL
                MAKE SURE TO ADD HYPERPARAMTER TO MODEL
                '''
                X2, _ = subtract_mean_from_data(X,Y)
                self.Sigma = compute_covariance_matrix(X2,X2)
                xgroup = [[] for _ in range(self.NUM_CLASSES)]
                for i in range(len(Y)):
                        xgroup[Y[i]].append(X[i])
                self.mean = []
                for i in range(self.NUM_CLASSES):
                        xi = np.array(xgroup[i])
                        self.mean.append(np.mean(xi,axis=0))

        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([-(x-self.mean[j]).T.dot(
    inv(self.Sigma+self.reg_cov
      *np.eye(len(self.Sigma)))).dot(x-self.mean[j])
                                for j in range(self.NUM_CLASSES)])
```

(f)



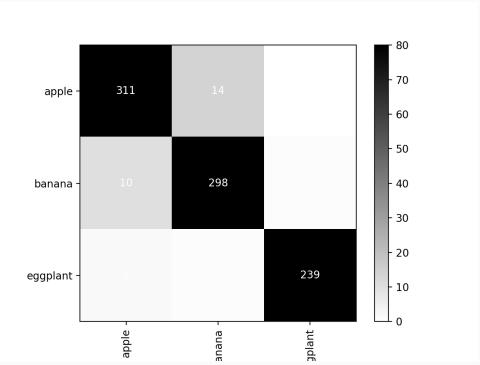


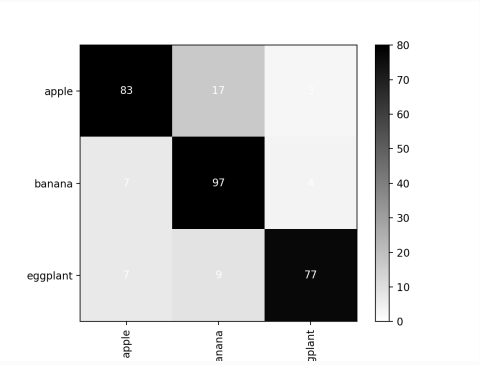

```
class QDA_Model():

        def __init__(self,class_labels):
                self.reg_cov = 0.01
```

Solution (cont.)

```
                self.NUM_CLASSES = len(class_labels)

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

                '''
                xgroup = [[] for _ in range(self.NUM_CLASSES)]
                for i in range(len(Y)):
                        xgroup[Y[i]].append(X[i])
                self.mean = []
                self.Sigma = []
                for i in range(self.NUM_CLASSES):
                        xi = np.array(xgroup[i])
                        self.mean.append(np.mean(xi,axis=0))
                        xiz,_ = subtract_mean_from_data(xi,Y)
                        self.Sigma.append(compute_covariance_matrix(
    xiz,xiz))


        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([-(x-self.mean[j]).T.dot(
    inv(self.Sigma[j]+self.reg_cov
      *np.eye(len(self.Sigma[j])))).dot(
          x-self.mean[j])-np.log(det(self.Sigma[j]))
                                for j in range(self.NUM_CLASSES)])
```

(g)



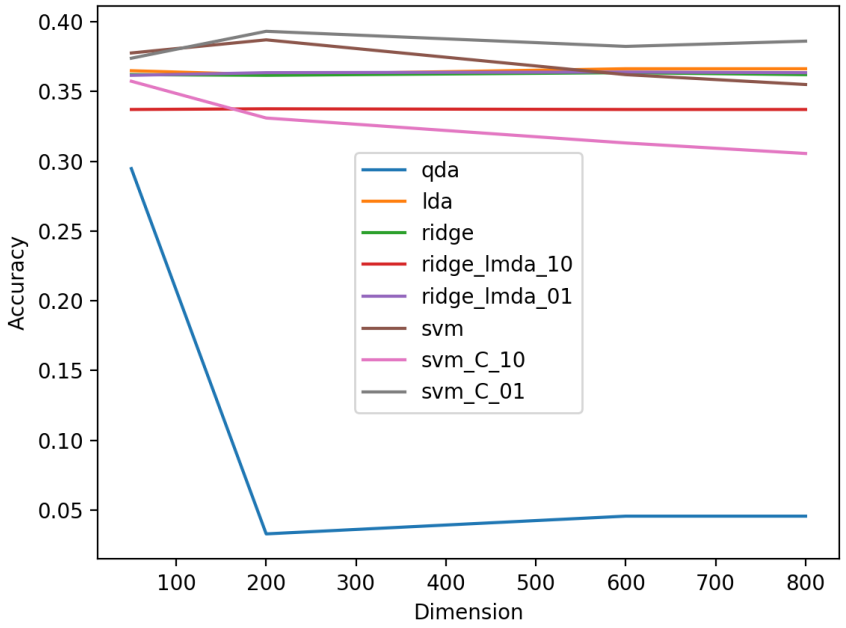


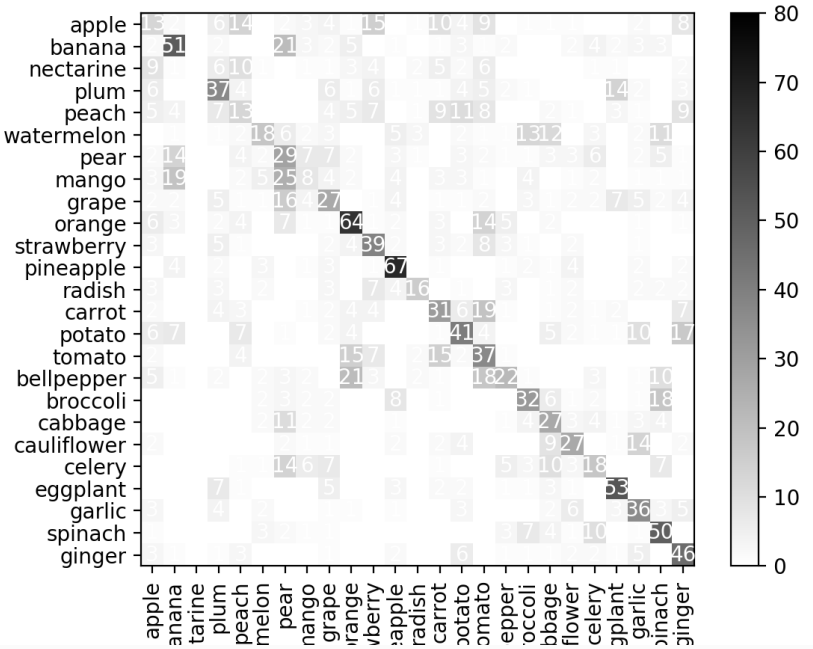

```python
class SVM_Model():

        def __init__(self,class_labels,projection=None):

                ###SLACK HYPERPARAMETER
                self.C = 1.0

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

                '''
                self.clf = LinearSVC(random_state=0,     C=self.C)
                self.clf.fit(X,Y)

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

(h)

The best model is SVM with weight $C = 0.1$ of slack variables and dimension 50. And QDA doesn't perform well in high dimension.

Solution (cont.)





Question 5

Question How to use SVM with multiclass?

Solution Suppose that we have n labels.

(1) One-against-one: to train $\frac{n(n-1)}{2}$ classifiers.

(2) One-vs-rest: to train n pair-wise classifiers with one of the voting schemes: Majority voting; Pairwise coupling.

(3) Extending the formulation of SVM to support the n class problem: By considering all classes at once; By considering each class with only the training data points belonging to that particular class. Like Crammer-Singer Multi-Class SVM. While Crammer-Singer is interesting from a theoretical perspective as it is consistent, it is seldom used in practice as it rarely leads to better accuracy and is more expensive to compute.

HW10

November 4, 2017

1 Question 3

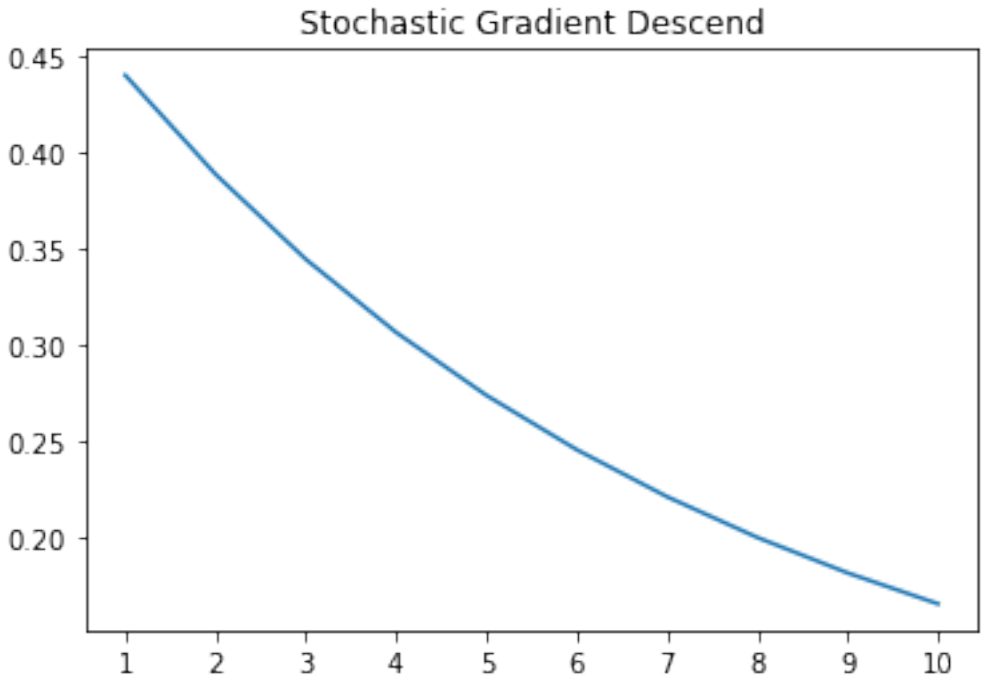
1.1 (a)

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

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

In [3]: [x ,y] = data['x'], data['y']

In [4]: train_x = (x - np.average(x))/ np.std(x)
        train_y = (y - np.average(y))/ np.std(y)

In [150]: class SGD(object):
              def __init__(self,
                           total_step_count = 10,
                           learning_rate = 0.05,
                           err = 0):
                  self.total_step_count = total_step_count
                  self.learning_rate = learning_rate
                  self.err = err
                  self.MSE = np.zeros((total_step_count,1))

              def gradient(self, x, y, w):
                  return np.dot(x.T,np.dot(x,w)-y)/len(y)


              def gradient_descend_step(self,x, y, w, step_count):
                  return w - self.learning_rate * self.gradient(x, y, w)


              def gradient_descend(self, x, y, w):
                  W = [np.array(w)]
                  for k in range(self.total_step_count):
                      new = self.gradient_descend_step(x, y, W[-1], k)
                      self.MSE[k] = np.mean(np.linalg.norm(np.dot(x,new)-y,axis=1)**2)/2
```

```
                if np.linalg.norm(W[-1]-new)<self.err:
                    break
                else:
                    W.append(new)
            return np.array(W)

        def train(self,x,y):
            _,m = np.shape(x)
            _,n = np.shape(y)
            w = np.zeros((m,n))
            self.w = self.gradient_descend(x, y, w)
```

In [151]:
```
sgd = SGD()
sgd.train(train_x,train_y)
```

In [152]:
```
plt.figure()
plt.plot(np.arange(1,11),sgd.MSE)
plt.title('Stochastic Gradient Descend')
plt.xticks(np.arange(1,11),np.arange(1,11))
plt.show()
```



In [153]:
```
sgd.MSE
```

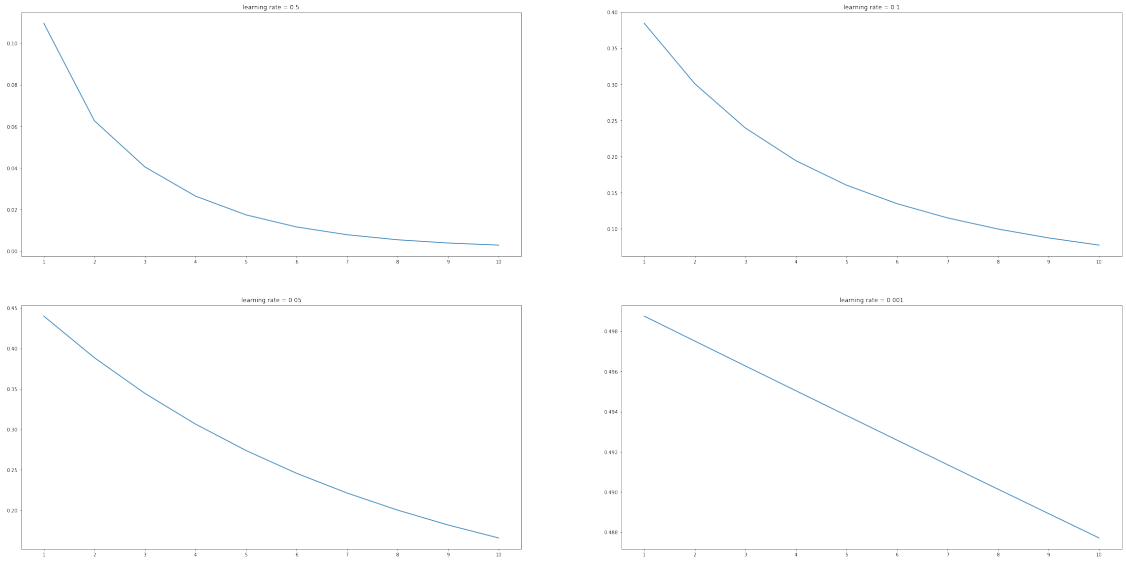
Out[153]:
```
array([[ 0.44003054],
       [ 0.38864365],
```

```
                [ 0.34454226],
                [ 0.30662823],
                [ 0.2739717 ],
                [ 0.24578521],
                [ 0.22140176],
                [ 0.20025633],
                [ 0.18187014],
                [ 0.16583741]])
```

```
In [154]: sgd.w[-1]
```

```
Out[154]: array([[ 0.37367638],
                 [ 0.15982575]])
```

```
In [180]: learning_rate = [0.5,0.1,0.05,0.001]
          sgd_group = [SGD(learning_rate=learning_rate[i]) for i in range(len(learning_rate))]
          plt.figure(figsize=(40,20))
          for i in range(len(learning_rate)):
              plt.subplot(2,2,i+1)
              sgd_group[i].train(train_x,train_y)
              plt.plot(np.arange(1,11),sgd_group[i].MSE)
              plt.xticks(np.arange(1,11),np.arange(1,11))
              plt.title('learning rate = '+str(learning_rate[i]))
          plt.show()
```



1.2 (b)

```
In [159]: class BSGD(object):
              def __init__(self,
```

```
                    total_step_count = 10,
                    learning_rate = 0.05,
                    err = 0):
        self.total_step_count = total_step_count
        self.learning_rate = learning_rate
        self.err = err
        self.MSE = []

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w, batchSize):
        W = [np.array(w)]
        for k in range(self.total_step_count):
            for i in range(0,len(y),batchSize):
                end = min(i+batchSize,len(y))
                batchX = x[i:end,:]
                batchY = y[i:end,:]

                new = self.gradient_descend_step(batchX, batchY, W[-1], k)
                self.MSE.append(np.mean(np.linalg.norm(np.dot(x,new)-y,axis=1)**2))
                if np.linalg.norm(W[-1]-new)<self.err:
                    break
                else:
                    W.append(new)
        return np.array(W)

    def train(self, x, y, batchSize):
        _,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        self.w = self.gradient_descend(x,y,self.w,batchSize)
```

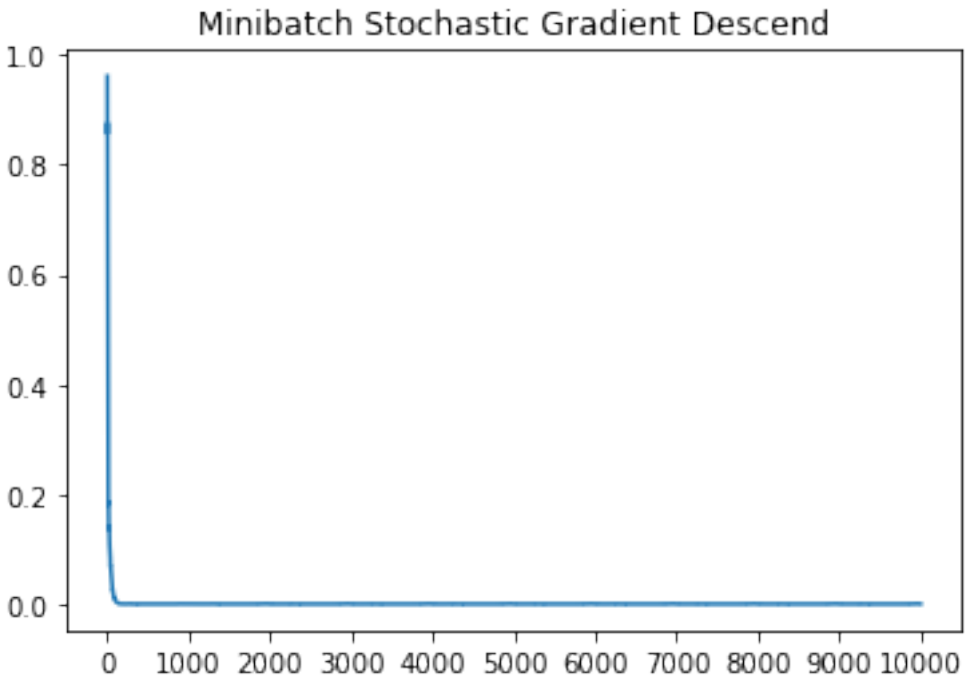
In [160]:
```
bsgd = BSGD()
bsgd.train(train_x,train_y,1)
```

In [161]:
```
np.shape(bsgd.MSE)
```

Out[161]: (10000,)

In [162]:
```
plt.figure()
plt.plot(np.arange(1,10001),bsgd.MSE)
plt.title('Minibatch Stochastic Gradient Descend')
```

```
plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
plt.show()
```



In [163]:
```
bsgd.MSE
```

Out[163]:
```
[0.96048411140890133,
 0.8555981571525012,
 0.87581871838886893,
 0.79706060810879398,
 0.69971726087624098,
 0.60759041722730578,
 0.60226083182528012,
 0.46206637286648639,
 0.28921516040932921,
 0.28087341196482307,
 0.29398787403183496,
 0.27382846578716796,
 0.20045796181462208,
 0.18284724723734389,
 0.18678387868231278,
 0.1861729780879233,
 0.16194576845114547,
 0.15884854065141099,
 0.15845422603993239,
 0.13826630591597944,
```

```
0.13883866113491558,
0.14068914115929768,
0.14148778012901611,
0.12563906818137857,
0.12492453015060516,
0.12421834819182424,
0.1235093897112705,
0.11012422471755698,
0.10799727040843331,
0.10313796278083222,
0.10458985958834151,
0.10446992803667567,
0.10425746208593853,
0.09761141050107483,
0.094045675613782367,
0.086369391628453834,
0.084519326935837497,
0.072273680310499341,
0.069528197316830145,
0.070358535795560104,
0.070252569652204569,
0.070403472302121059,
0.068130497723925362,
0.067961549723279446,
0.069026713608680151,
0.066350967597144442,
0.066432502668288124,
0.061074498639085102,
0.048491035826656342,
0.044661266805943338,
0.044244628141220266,
0.039823474949555411,
0.039499996674662342,
0.036285683588117498,
0.035992957681086839,
0.03479446030746218,
0.032478483675992625,
0.03199811202636145,
0.030960763004674399,
0.029911952361502011,
0.028799562587791667,
0.029052948366884147,
0.028752304452101397,
0.027715417434194343,
0.028221997873625112,
0.028259125939272316,
0.026672047313340268,
0.024032200847261619,
```

```
0.024030591785593223,
0.02402695106746542,
0.021366449804172898,
0.021062854410317692,
0.02114318212341253,
0.020308245797380116,
0.02002216889375311,
0.019513913268786674,
0.018456129040979725,
0.018229840605838064,
0.017960075331342345,
0.015531766944790075,
0.01532888467593127,
0.01532276736039203,
0.01522730378785069,
0.014782980833002998,
0.014375250724405779,
0.013019686885990665,
0.012727018013917927,
0.012706648783024351,
0.012698279134996942,
0.012803942423107362,
0.012851620632796301,
0.012811865716053276,
0.012205172647911631,
0.011905461089395494,
0.012478431014973751,
0.008497991855026598,
0.0082027421463517838,
0.008198844785848744,
0.0080884805948368381,
0.0079502595117603939,
0.0076354673568997276,
0.0076356448756519212,
0.0074588755830575478,
0.0073998098117380217,
0.0066747778246442555,
0.0064727974092315338,
0.0064411956007048125,
0.0064377438870045047,
0.0063323026182658107,
0.0059614021361984416,
0.0058246651182057097,
0.0056735352910651619,
0.0058381638383195374,
0.0059584573913365694,
0.0057605539644810634,
0.0048193094622574918,
```

```
0.0048072117440829416,
0.0047890099198344685,
0.0047971740386769882,
0.0048079163643270882,
0.004653159132533554,
0.0046271812655899001,
0.0045432325917351317,
0.0044132472844115754,
0.0044160448144589031,
0.0043135778947139972,
0.0043390032602319464,
0.0040689001527821692,
0.0039739227863932807,
0.0036378785129623213,
0.0036039398493150718,
0.0034727966946686956,
0.0033273928247910395,
0.0032854865326010186,
0.0032498525427395882,
0.0032474790706720977,
0.0032539551859529321,
0.0034647461462813728,
0.003268810871886337,
0.0029922209608856673,
0.00301315082542519,
0.0029824786498849304,
0.0028691451383084272,
0.0027241037182919804,
0.0028032064826389248,
0.0026523270985477172,
0.0025637023428888186,
0.0026468861854872772,
0.0026382268208945897,
0.0026115692737305337,
0.0026450594966897605,
0.0026458204735160451,
0.002692584659711744,
0.0025182519758832462,
0.0025087258889230435,
0.0025099260030981229,
0.0025061665316274463,
0.0023886518801972166,
0.0023623018921389029,
0.0023606290169717594,
0.0024511531692170577,
0.002411709436144059,
0.0023621833383116711,
0.0023545942422169581,
```

0.0024006859011341279,
0.0023872570299053432,
0.0024182417727741179,
0.0025785377165248391,
0.0025951711026440112,
0.0025790921193792875,
0.0026178905767607974,
0.0026112178186558851,
0.0026767298056624266,
0.0026620574971051728,
0.0024591677563857361,
0.0024568558417765816,
0.0024785259111835341,
0.0023450249916894695,
0.0023723475611935364,
0.0023052525925372543,
0.0023059714685857032,
0.0023130365969276116,
0.0023845284732047448,
0.0024027401532511769,
0.0023632664700486711,
0.002414701218637317,
0.0024093903001773938,
0.0023945368441851311,
0.0023469429199296488,
0.0023197414038306999,
0.0023099825268006117,
0.0023612642655074074,
0.0023853686449230894,
0.0023548382332838919,
0.0024911651249762239,
0.0024305591664395051,
0.0024214797381246021,
0.002395713561295967,
0.0024564018842137495,
0.0024246834406469802,
0.0024210968008406271,
0.0024100879941253827,
0.0024071296310441068,
0.0024087848478114527,
0.0023332879315593473,
0.0023354766707409098,
0.0023464969137068454,
0.00230838046534534,
0.0022971656957452608,
0.0022984664130818572,
0.0023521352808820381,
0.0024085269711463793,

0.0024267631834151322,
0.0023894329147598104,
0.0024115304570601305,
0.0023427088635181283,
0.0023318733419312849,
0.0026003383206942461,
0.0024631349054485378,
0.0023795766422484853,
0.0023835218063842571,
0.0022334318550404964,
0.0022347189199291889,
0.0022385744506947486,
0.0022478913917112694,
0.0022390331262950509,
0.0023083206476738893,
0.0022767462336435481,
0.0023106355774016115,
0.0023154135072337434,
0.0023431617426145574,
0.002348633814544609,
0.0023615717487404852,
0.002370767499949859,
0.0023281103317080109,
0.0023252306497773133,
0.0022868524717952585,
0.0022871157831501421,
0.002287117521520411,
0.002273239784497155,
0.0023185487411911724,
0.0023051221951768579,
0.0022439522335216881,
0.0022356578577803123,
0.0022384556688425849,
0.0022546609602960245,
0.0022624546201294323,
0.0022911485379095101,
0.0023336653143817866,
0.0023255598035435234,
0.0025461916776854015,
0.0025741098814425784,
0.0023703989314196997,
0.0023395743291364621,
0.0023306233759912225,
0.0023265351117092224,
0.0023273266420306502,
0.002298769022748478,
0.0022942864138877578,
0.0022962018546124978,

0.002392773367920402,
0.0022586787760786537,
0.002421119943635273,
0.0024093264975045673,
0.0022343518566098654,
0.0022403763897433579,
0.0022662722909552848,
0.0023394299718176208,
0.0023831312811855095,
0.0023589167780325728,
0.0023584394423526787,
0.0024658903087566345,
0.0024740761922216353,
0.0024873557384959967,
0.0024908847873398106,
0.002674873183838228,
0.0024811291189551919,
0.0024785193026395252,
0.0024747128196360244,
0.0023493240708563397,
0.0023361473488420705,
0.0022572511615261723,
0.0023613532215813323,
0.002335740189488874,
0.0023545277786078088,
0.0022279835290251715,
0.0022645343469137992,
0.0026401686097132053,
0.0027177364945401303,
0.0026298966668502032,
0.0025429181037381395,
0.0025535324937321427,
0.0025669176435015625,
0.0026508091378192812,
0.0026649395036228091,
0.0023166700588071435,
0.0022509948872326759,
0.0023132916234332979,
0.0022899880541969236,
0.0022436106153287884,
0.0022255464274584915,
0.0022553045606444051,
0.0022520931110858208,
0.0022616679500459826,
0.0022668230627109162,
0.002266886585082264,
0.002257094926625717,
0.0023046584443969262,

0.0022488967581439389,
0.0022315547708246985,
0.0023150462363265801,
0.0023926558315433503,
0.0022661606539336821,
0.0022824302769210935,
0.0022834101400482877,
0.0022878961978800025,
0.0023059097913588865,
0.0023153822079540519,
0.00231496059431376 4,
0.0023060283009342522,
0.0023661763053570966,
0.0022787210399328817,
0.0022364900576353685,
0.0022280882469777464,
0.00223051198570532 3,
0.0022343785191318057,
0.0022269670266455471,
0.0022259717113184579,
0.0022355627082758824,
0.0022358095163072965,
0.00225191268737693 4,
0.0022922870502986789,
0.0022769303411322764,
0.00226665225023621 31,
0.00225556072854175 4,
0.0022748373474287924,
0.0022570554473426252,
0.0022465439266957164,
0.00225175437106791 85,
0.0022521626346643326,
0.0022475051710392408,
0.0022437790644555841,
0.0022430587954582627,
0.0024729513433033256,
0.00247198936840802 93,
0.0024258133432131378,
0.00227893461683013 5,
0.0022695202468831031,
0.0023260138299478371,
0.0023543737172938275,
0.0022659568701803032,
0.0022659281606085858,
0.0022534016934515825,
0.0022294938513995003,
0.0022434801634681144,
0.0022838769494385211,

0.0023003919500844778,
0.0023407436990652763,
0.0022792672277468207,
0.0022750272276003767,
0.0022526277345247987,
0.0022530171541297016,
0.0022489422470603772,
0.0022506624118458067,
0.0022734100382847031,
0.0022799499685370686,
0.0023061439466313966,
0.0022689385698878993,
0.0022310165318044309,
0.0022414022389516618,
0.0022324247147467178,
0.002313393950963642,
0.0027025930386355636,
0.0023849074166418611,
0.0026362020364471394,
0.0024120523415274066,
0.002225449366653586,
0.0022360687395864901,
0.0022302106005295135,
0.0022299142259991885,
0.0022406360639220749,
0.0023076805559334372,
0.0023041203676432802,
0.0022667409432710994,
0.0022463761576624816,
0.0022453120990141738,
0.0022270501956655401,
0.0022251512316004874,
0.0022268945567540284,
0.0022281116315824275,
0.0022253364620505129,
0.0022257376289979134,
0.0022270082352957407,
0.0022407088059579533,
0.0023322907725588991,
0.0023314784825566753,
0.0022895391341965512,
0.0023487687569320012,
0.0023481898121131941,
0.0023517498044438919,
0.0024082381648725792,
0.0024014750098308229,
0.0022604747743930585,
0.0022414805985846455,

```
0.002266837231752548,
0.0023494800389692533,
0.0023003208807436792,
0.0023020544195461903,
0.0023145870640619709,
0.0023466319170228917,
0.0022975284450734239,
0.002235531635486363,
0.0022369226236278717,
0.0022428720584885529,
0.0022931959457300655,
0.0022991658381573848,
0.0023944323425905832,
0.0023301922796599736,
0.0026307626764031232,
0.0025799517711757956,
0.0025374313679904711,
0.0027695425485866681,
0.0027000181567374618,
0.0025552235413580328,
0.0025584580600093235,
0.0025155644644200906,
0.0025232937257422022,
0.0023584005565081599,
0.0023906625170275943,
0.0023683601445221312,
0.0023303671270149783,
0.0023468056089085251,
0.0023274468304432484,
0.0023745458448838647,
0.0024390352340112101,
0.0023963140727039506,
0.0024063040807818527,
0.0025139682821345992,
0.0024008370204946904,
0.0022747115324511951,
0.0022680223320627759,
0.0022660692012354561,
0.0023390876672833723,
0.002319486933327993,
0.0022862364771843516,
0.002370592010665246,
0.0023348958302521366,
0.0023464973993713991,
0.0023334979210114697,
0.0023413298971230361,
0.0022567427698233929,
0.0022791049455316236,
```

```
0.0022782011360529238,
0.0022851159267016888,
0.0022527922504416145,
0.0022411102614047361,
0.0022413911455312946,
0.0023075585607658998,
0.0023128276859132007,
0.0022830353493137286,
0.0022710655185711804,
0.0022350335190000122,
0.0022265084584273251,
0.0022295954675563886,
0.0022268688680055523,
0.0022300701947293275,
0.0022439745477255673,
0.0022444925339946131,
0.0022466141660986371,
0.0022624014716390111,
0.0022706212775233 52,
0.0022632882977157774,
0.0023224611869502908,
0.0023144711677318035,
0.0022347130352374242,
0.0022689589231825531,
0.0022274929183701431,
0.0022709135215277726,
0.0022305217655939 42,
0.0022339903738119149,
0.0022669995250323787,
0.0022794540869221155,
0.0022886702247692688,
0.0024003103948800489,
0.0023623389711174879,
0.0024587795859918825,
0.0023521945652910997,
0.0023379069080810026,
0.0024605918869229858,
0.0023644900418734157,
0.0023819820690116657,
0.0023748529486709294,
0.0024298333630481208,
0.0025643821664873396,
0.0025554971950450919,
0.0024169615378472819,
0.0024676684153900961,
0.0024656938751984382,
0.0024514205391025258,
0.0024721882038107367,
```

```
0.0023286192579737384,
0.0023263991049930865,
0.0023150290434956365,
0.0023102914016413585,
0.0022453735616558869,
0.0022273570623852477,
0.0022277746497815624,
0.002231022809876151,
0.0023292983218992733,
0.0023398107719784459,
0.0025808902861207987,
0.0028423703231295564,
0.002726702274351526,
0.00279771540844764,
0.0024243701356632644,
0.0024729373688322444,
0.0023759787582904885,
0.0023886830220319613,
0.0023654052591065553,
0.0023696225092371924,
0.0024290235164638884,
0.0023991013496069621,
0.0024463172851730591,
0.0023091588056180501,
0.0022912568197302361,
0.0025321216918959568,
0.0025370673181892896,
0.002315441026873831,
0.0023213319608653706,
0.0023401625244658382,
0.0024522491217333878,
0.0024481156209102658,
0.0024505817736852967,
0.0023134935669975568,
0.002264527362002818,
0.0022592002921117873,
0.0022598257768076105,
0.0022574003338009792,
0.0023090562013107587,
0.0023059544348134286,
0.0023027746332140125,
0.0022922137351372458,
0.0022928354698955548,
0.002361986042410676,
0.0023701827587315908,
0.0023692968033868994,
0.0023687533643194235,
0.0022947673344382693,
```

0.0022653571808847498,
0.0022421093470200554,
0.0022419921500541132,
0.0022545213090864298,
0.0022489819484603482,
0.0022524408772435531,
0.0022568963926156496,
0.0024156835069918054,
0.0023486787934680451,
0.0023173590579795072,
0.002325614548659411,
0.0023394097607940438,
0.0023558581000772951,
0.0023356066090877291,
0.0022819646546684549,
0.0022903413051829336,
0.0022685979822029364,
0.0022685832899438873,
0.0022834074349118985,
0.0023012421035043815,
0.0022658806192639947,
0.0022748751731482761,
0.0022383504904598278,
0.0022401203548891658,
0.0022593268430431833,
0.0022693285265299704,
0.0023688921510265303,
0.0023236081947665467,
0.0022468007364344045,
0.0022553850179078613,
0.0023166508979794035,
0.0023625740020796686,
0.0024045871333991048,
0.0023748689342357413,
0.0023699280677820489,
0.0023803075756518243,
0.0023515167529850728,
0.0023030113609623379,
0.00231499145757132,
0.0023165856616147039,
0.0023194761716052305,
0.0023236451473501976,
0.002342765521115365,
0.0023244535930256579,
0.002370574324543134,
0.0023709885303038617,
0.0023455224332522393,
0.002329421734534889,

0.0023602311914191986,
0.0024129081108568069,
0.002424759337131916,
0.0023011276718414393,
0.0023402807999799557,
0.0023414407599028362,
0.0023376087362394797,
0.0023629570446057004,
0.0023854015447088346,
0.0023821845329554084,
0.0024549587985851855,
0.002365322033385795,
0.0023462951066947958,
0.0024223846259468149,
0.0024403215791406387,
0.0024230182124879021,
0.0024919177379840044,
0.002315135906267602,
0.0023181749423236129,
0.0022956847599451555,
0.0023332078309506506,
0.0023265216576046424,
0.0023279725264699598,
0.0023186757077574278,
0.0023687997026018712,
0.0024014799579674859,
0.0023484317192510869,
0.002339042408435358,
0.0023651490927701726,
0.0023624261066304696,
0.002410226244524864,
0.0022574525765045278,
0.0022569215112546344,
0.0022573782662578487,
0.0022584468725717652,
0.0023022277444411407,
0.0023137266989500835,
0.0023142361042969633,
0.0023005546972550014,
0.0022903475498701063,
0.0023341941210441321,
0.0022620491994662191,
0.002277364166880944,
0.0022798419489606881,
0.0022779789030884876,
0.002270155643766552,
0.0022994594826510994,
0.0022612443415475168,

0.0022560880631428176,
0.0022535158468401676,
0.002225106388534904,
0.00222625626423069877,
0.0022387053055281975,
0.0022274224178010045,
0.0022396395655529711,
0.0022528823690337115,
0.0022376007515063187,
0.0022322291453672123,
0.0022317275931755653,
0.0022280494011335827,
0.0022276284684592533,
0.0022317310824052461,
0.0022563357180613607,
0.0022600222811811777,
0.0025474460607681371,
0.0024952999521304672,
0.0024079199505387941,
0.0024380522185964456,
0.0024040953378531432,
0.0024012840332156091,
0.0024545477989785535,
0.0024950521672736032,
0.0026059769365164294,
0.0024040184996513398,
0.00236055154063983,
0.0023475176812561812,
0.0023147834514376936,
0.002315155575993288,
0.0023267299931038206,
0.0024176470738949134,
0.0024069756226050075,
0.0023069477538422999,
0.0023073699950116222,
0.0023018467501872354,
0.0024373512369513461,
0.0024243575556127219,
0.0025779414819649504,
0.0025529436637995256,
0.00251488183497903,
0.0023265258231944547,
0.0022762868067748398,
0.0022757262037107159,
0.0022740676581903254,
0.0022341640975579733,
0.002234136633930454,
0.0022634482597956741,

0.0023315372030155083,
0.0022594595641931376,
0.0022669999979820885,
0.0022639704388789264,
0.00225039186633644,
0.0022322892043077406,
0.0023184337236648576,
0.00226776597056479,
0.0022735760922282969,
0.0022706933899232112,
0.0022337795466105506,
0.0022251895844590257,
0.0022597766333178147,
0.0022314717412924984,
0.0022261631103917129,
0.0022282516852633813,
0.0022771834402409348,
0.0022571236306562319,
0.0022436189764871361,
0.0022300883207171875,
0.0022315530407557133,
0.0022283173968239856,
0.0022581363203215588,
0.0022356091368766467,
0.00227093677245042,
0.0022706115760144224,
0.0022574852232184017,
0.0023234394038352583,
0.0022400893554106399,
0.0022507442818406587,
0.0022352705946699922,
0.0022444384320239198,
0.00224416499962269 4,
0.0022652615804931086,
0.00226930281225492,
0.0022730807747925 19,
0.0022767693922254332,
0.0022320071491451266,
0.0022544367677866423,
0.0023805763531628095,
0.0023621671743165491,
0.0023431533719788832,
0.0024280154611265913,
0.00226931082638441 5,
0.0022719130267019 64,
0.0023088420884740243,
0.0023558747150142743,
0.0023114442313657553,

```
0.0023162496514657861,
0.0022951102107685879,
0.0023122669882754106,
0.0022276700944027922,
0.0022304147114188613,
0.0022273564596521698,
0.0022452675930264183,
0.0022422549861728729,
0.0022984289840681169,
0.0022915111741060598,
0.002311184542028596,
0.0022678586117136569,
0.0022279875019849222,
0.0022280368598032877,
0.0022286269117072007,
0.002244922222263271,
0.0022348374137276432,
0.0022298618123552903,
0.0022407155835897814,
0.0023541948412319383,
0.0023094526620125681,
0.0023091325462176781,
0.0022855781225487367,
0.0024071904886559018,
0.0024123724802999256,
0.002346266406393762,
0.0023820664045179746,
0.0023329055036905811,
0.0022258285992929165,
0.0022262140224932618,
0.0022495819796132053,
0.0022719065958067717,
0.0022523046447641237,
0.002267525253886838,
0.0022302209374594623,
0.0022315817086904707,
0.0024160379403433724,
0.0024272504752802407,
0.0024561727570589628,
0.0022902576383062481,
0.0023859418623734382,
0.0023795442522829158,
0.0023803859047502288,
0.002420252490023281,
0.0023041376601545985,
0.0023057668401531622,
0.0023125898353808134,
0.0022264181539497913,
```

```
0.0022269973606287609,
0.0022319722104656224,
0.0022380730141556269,
0.0022431725982136356,
0.0022356614826111142,
0.0022300301271762362,
0.0022255797993536134,
0.0022506251437030577,
0.0022584183578516514,
0.0022543726306130567,
0.0022429407255948838,
0.0022426460269999006,
0.0022407542312038878,
0.0022325029282162187,
0.0022484376999264573,
0.0022701366735838149,
0.0023001772846452822,
0.0023076959473403253,
0.0022504754196898151,
0.0022466737248387453,
0.0022449785609397086,
0.0022445247655718602,
0.002243492071392171,
0.0022440946646260711,
0.0022466385641530242,
0.0022463459840924706,
0.0022448104048719258,
0.00225816867240576,
0.0022269312585990204,
0.0022297903529475012,
0.0022480678190066666,
0.0022259231763234006,
0.0022258397056495252,
0.0022264752545641891,
0.0022262868368202976,
0.0022939092148946505,
0.0023187989220169324,
0.0023391163104138093,
0.0022370965042960091,
0.0023064560411211426,
0.0025374189139445395,
0.0025295875121880873,
0.0023543381682227383,
0.0023535574395260519,
0.0022800876990690597,
0.0022324407045951639,
0.002246113881347607,
0.0022354782269078167,
```

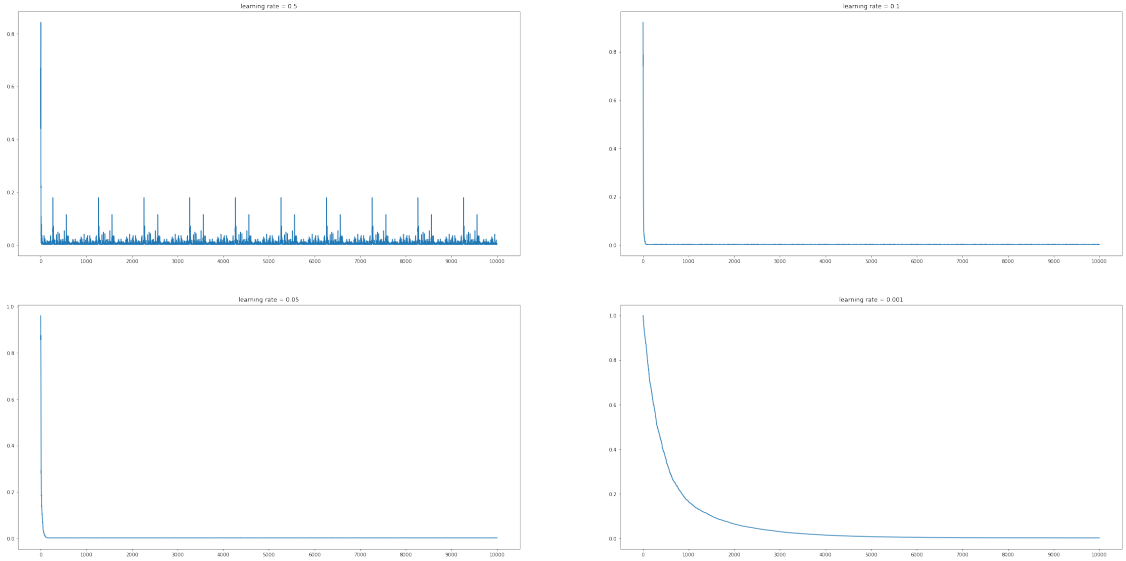
```
0.0022358371721238417,
0.0022356408929608109,
0.0022963541956157944,
0.0023140679834420491,
0.0023924152975513986,
0.0023946223655932036,
0.0023614340955196125,
0.0023627978351014579,
0.0023814424592127096,
0.0023812496706221422,
0.0023398756203956705,
0.00230577097010228 1,
0.0022817860698806014,
0.00228109565794915 3,
0.0023803581229633435,
0.0023605385453511554,
0.0023665826025004062,
0.00255768043408657 5,
0.0025455102893134378,
0.0023423630608517673,
0.0022981845010264382,
0.00229835438912063 7,
0.0022613975928123427,
0.0022331332993324789,
0.00223191757133963 7,
0.0022301818890814937,
0.0022309456150640177,
0.0022302940479097008,
0.0024324280456032481,
0.00241530065331731 3,
0.00241839315602830 4,
0.0024209260239402789,
0.0024238396909865541,
0.0024792636540813362,
0.0024802116438679455,
0.0024161251647360233,
0.0024514297762305592,
0.0024279542097674495,
0.0024268331067092944,
0.0024241973309723167,
0.0023404337392580491,
0.0024189816860191431,
0.0022290563234555832,
0.0022765183109048118,
0.0024849651765738931,
0.0022825672461685739,
0.0023657775275183041,
0.0023677387572746357,
```

0.0023774654068560038,
0.0022399998775168837,
0.0022510573899103018,
0.0022278509433302034,
0.0022261266880636253,
0.0022357064377363143,
0.0022569623103719529,
0.0022376896579546389,
0.0022441925862453048,
0.00224949018517718668,
0.0023067419217834128,
0.0023303323998382429,
0.0023343386150646113,
0.0023111447122605763,
0.0023052391124069897,
0.0023042334500208757,
0.0023110677975542149,
0.0023171659963827462,
0.0024921114515946748,
0.0026524669099644766,
0.0027136255936197722,
0.0024225706058237958,
0.0024172256977076281,
0.0024172957085294234,
0.0023426894191887104,
0.0023192672332156754,
0.0023258093638766448,
0.0023282534919920384,
0.0023148177120030874,
0.0023248936024482338,
0.0023254569297837166,
0.0022698868073636122,
0.0022811574284673174,
0.0023117582963058086,
0.0023522773792678179,
0.0022982311103954098,
0.0023016683129744235,
0.0023190742138694872,
0.0022656200508375129,
0.0022887439975387888,
0.0022825999350010741,
0.0023046082068548844,
0.0022576491519904502,
0.00224587540532667 5,
0.0022643726533688379,
0.0022960909977515608,
0.0022920787444854554,
0.0022304470697562047,

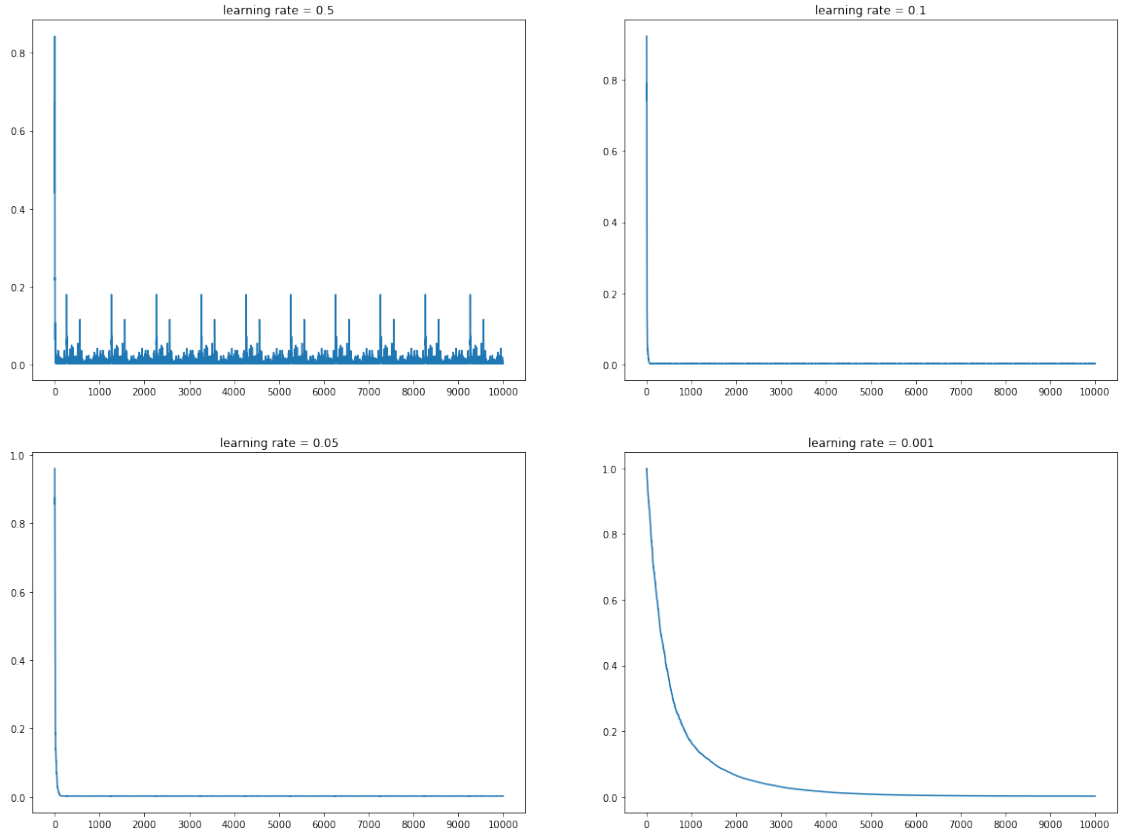
0.0022469902055504316,
0.0022472180548389611,
0.0022283417153940144,
0.0022249058196780122,
0.0022251216480642533,
0.0022293437255335256,
0.0022333376732659449,
0.0022927075422854138,
0.0022810477718597441,
0.0022332608791273325,
0.0023031884598570069,
0.0023734808909378138,
0.002374943121705561,
0.0022270080984734763,
0.0022379642583055456,
0.00223531162146497,
0.00224678399241658,
0.0023209507582753401,
0.0023838238880661538,
0.0024437864872237673,
0.0024011513650206966,
0.0022921895472427485,
0.0022305655796772692,
0.0022306044432808233,
0.0022303593578488464,
0.0022278163889350696,
0.00223693397678682 8,
0.0024189635029862517,
0.0023696308397245448,
0.0025382072474775617,
0.0024783218298286814,
0.0025615007367861759,
0.0025954569842317576,
0.0026840946171110655,
0.0027073407721768632,
0.0028321369560567202,
0.0028153812269820092,
0.0027181113761424493,
0.0027512937304988443,
0.0027448452604055102,
0.0027437824533048131,
0.0028818183346580074,
0.0027134718187972781,
0.0026297794390434154,
0.0024925412282134182,
0.0023811321654567488,
0.0023262528516990819,
0.0023518759849050076,

```
 0.0022847283011561315,
 0.0023292408109638731,
 0.0023328990380746487,
 0.0023280086933775614,
 0.0023339891332641266,
 0.0024978024294446078,
 0.0023222865588871651,
 0.0022486843477984319,
 0.0022450115909427486,
 0.0022407115452043333,
 0.0022489230165630004,
 0.0022413100948448336,
 0.0022424343821681354,
 0.0022459774062358671,
 0.0022559206201622551,
 0.0022732519186846024,
 0.0025371270328560006,
 0.0023993938010077294,
 0.0023673809857128834,
 0.0023827436241175789,
 ...]
```

In [179]:
```
learning_rate = [0.5,0.1,0.05,0.001]
bsgd_group = [BSGD(learning_rate= learning_rate[i]) for i in range(len(learning_rate))]
plt.figure(figsize=(40,20))
for i in range(len(learning_rate)):
    plt.subplot(2,2,i+1)
    bsgd_group[i].train(train_x,train_y,1)
    plt.plot(np.arange(1,10001),bsgd_group[i].MSE)
    plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
    plt.title('learning rate = '+str(learning_rate[i]))
plt.show()
```



```
In [171]: plt.figure(figsize=(20,15))
          for i in range(len(learning_rate)):
              plt.subplot(2,2,i+1)
              plt.plot(np.arange(1,10001),bsgd_group[i].MSE)
              plt.title('learning rate = '+str(learning_rate[i]))
              plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
          plt.show()
```




```
In [18]: sgd[0].w[-1]

Out[18]: array([[ 0.54397541],
                [ 0.29172211]])
```

1.3 (c)

```
In [136]: class FLS(object):
              def __init__(self,
                           total_step_count = 10,
```

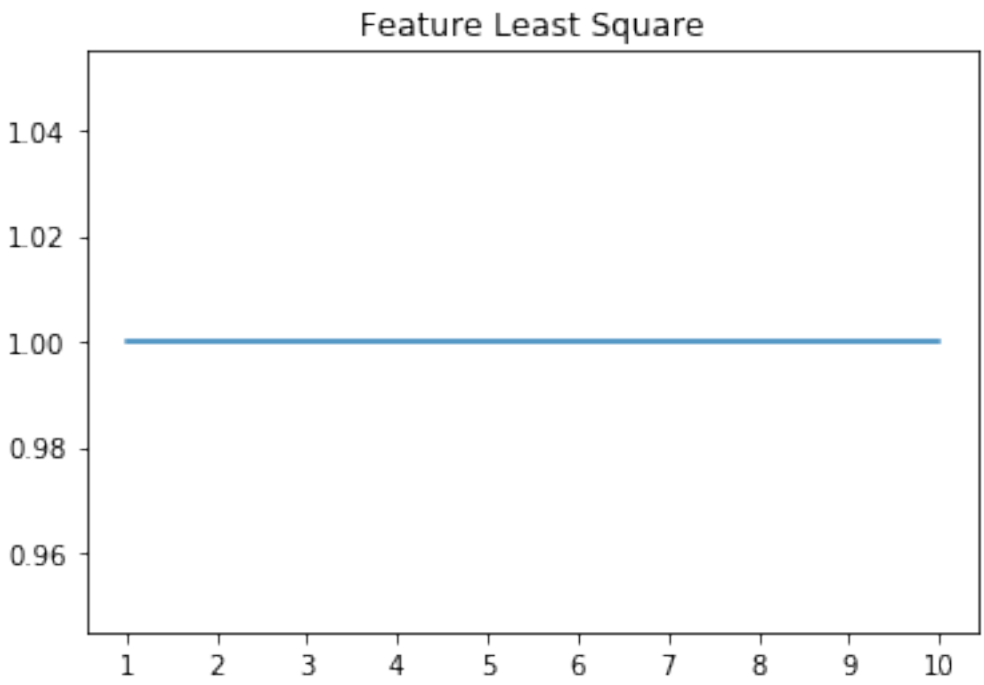
```
                    err = 0):
        self.total_step_count = total_step_count
        self.err = err
        self.MSE = []

    def LS(self,x,y):
        return np.sum(x*y)/np.sum(x*x)

    def train(self, x, y):
        _,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        for i in range(self.total_step_count):
            index = np.random.randint(0,m)
            self.w[index,:] = self.LS(x[:,index],y)
            self.MSE.append(np.mean(np.linalg.norm(np.dot(x,self.w)-y,axis=1)**2))
```

In [156]:
```
np.random.seed(0)
fls = FLS()
fls.train(train_x,train_y)
```

In [157]:
```
fls.MSE
```

Out[157]:
```
[1.0000000000000009,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002,
 1.0000000000000002]
```

In [158]:
```
plt.figure()
plt.plot(np.arange(1,11),fls.MSE)
plt.title('Feature Least Square')
plt.xticks(np.arange(1,11),np.arange(1,11))
plt.show()
```

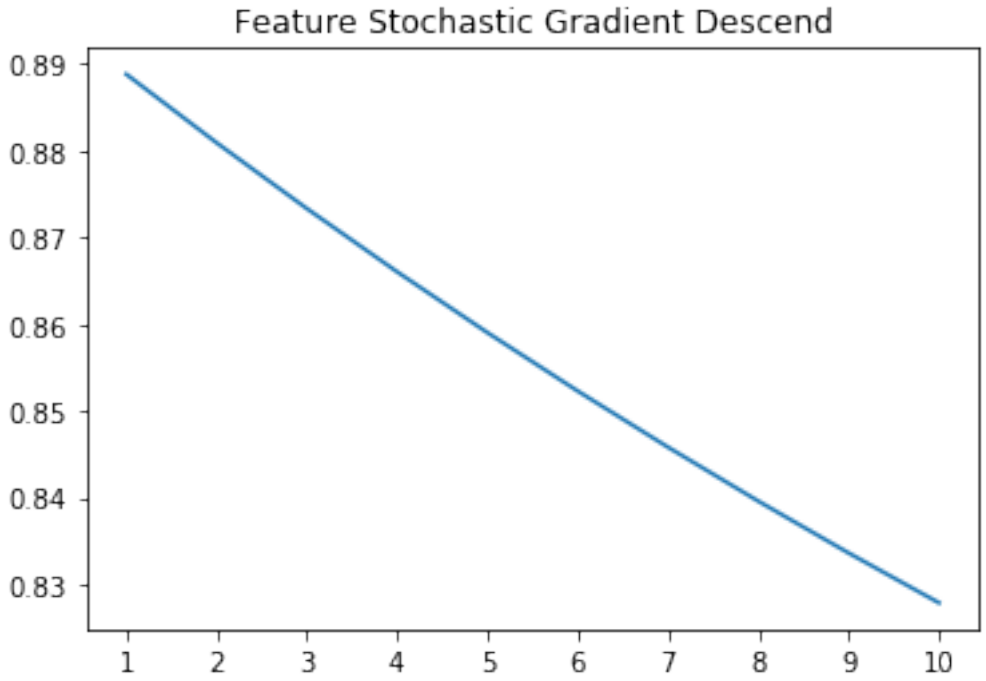


In [89]:
```
class FSGD(object):
    def __init__(self,
                 total_step_count = 10,
                 learning_rate = 0.05):
        self.total_step_count = total_step_count
        self.learning_rate = learning_rate
        self.MSE = np.zeros((total_step_count,1))

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)


    def gradient_descend(self, x, y, w):
        x = x[:,np.newaxis]
        w = w[np.newaxis,:]
        W = [np.array(w)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step(x, y, W[-1], k)
            self.MSE[k] = np.mean(np.linalg.norm(np.dot(x,new)-y,axis=1)**2)
            W.append(new)
        return W[-1]
```

```
            def train(self,x,y):
                _,m = np.shape(x)
                _,n = np.shape(y)
                self.w = np.zeros((m,n))
                for i in range(self.total_step_count):
                    index = np.random.randint(0,m)
                    self.w[index,:] = self.gradient_descend(x[:,index],y,self.w[index,:])
```

```
In [90]: np.random.seed(0)
         fsgd = FSGD()
         fsgd.train(train_x,train_y)
```

```
In [109]: plt.figure()
          plt.plot(np.arange(1,11),fsgd.MSE)
          plt.title('Feature Stochastic Gradient Descend')
          plt.xticks(np.arange(1,11),np.arange(1,11))
          plt.show()
```

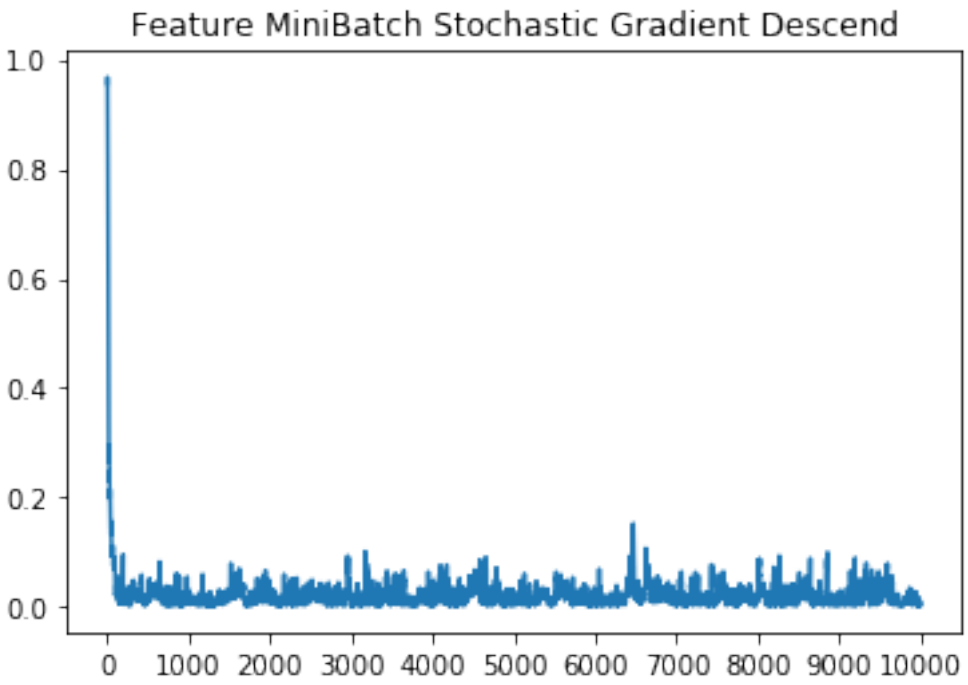



```
In [103]: class FBSGD(object):
              def __init__(self,
                           total_step_count = 10,
                           learning_rate = 0.05):
                  self.total_step_count = total_step_count
                  self.learning_rate = learning_rate
```

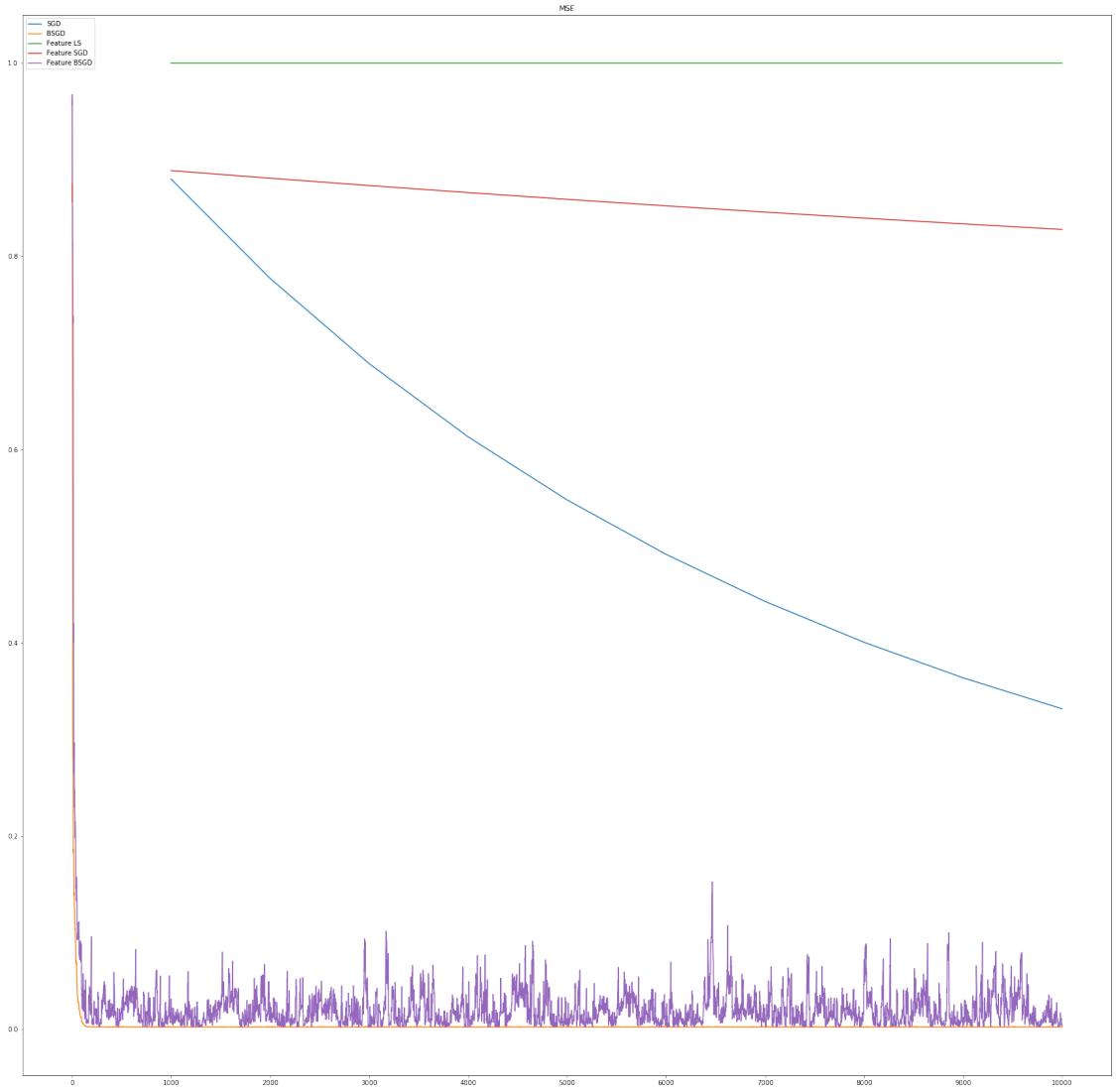
```
        self.MSE = []

    def gradient(self, x, y, w):
        return np.dot(x.T,np.dot(x,w)-y)/len(y)


    def gradient_descend_step(self,x, y, w, step_count):
        return w - self.learning_rate * self.gradient(x, y, w)

    def train(self, x, y, batchSize):
        _,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        for k in range(self.total_step_count):
            for i in range(0,len(y),batchSize):
                index = np.random.randint(0,m)
                end = min(i+batchSize,len(y))
                batchX = x[i:end,index]
                batchY = y[i:end,:]
                batchX = batchX[:,np.newaxis]
                w = self.w[index,:]
                w = w[np.newaxis,:]
                self.w[index,:] = self.gradient_descend_step(batchX, batchY, w, k)
                self.MSE.append(np.mean(np.linalg.norm(np.dot(x,self.w)-y,axis=1)**2))
```

In [104]:
```
np.random.seed(0)
fbsgd = FBSGD()
fbsgd.train(train_x,train_y,1)
```

In [108]:
```
plt.figure()
plt.plot(np.arange(1,10001),fbsgd.MSE)
plt.title('Feature MiniBatch Stochastic Gradient Descend')
plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
plt.show()
```



In [118]:
```
plt.figure(figsize=(30,30))
plt.plot(np.arange(1000,10001,1000),sgd.MSE,label='SGD')
plt.plot(np.arange(1,10001),bsgd.MSE,label='BSGD')
plt.plot(np.arange(1000,10001,1000),fls.MSE,label='Feature LS')
plt.plot(np.arange(1000,10001,1000),fsgd.MSE,label='Feature SGD')
plt.plot(np.arange(1,10001),fbsgd.MSE,label='Feature BSGD')
plt.title('MSE')
plt.legend()
plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
plt.show()
```



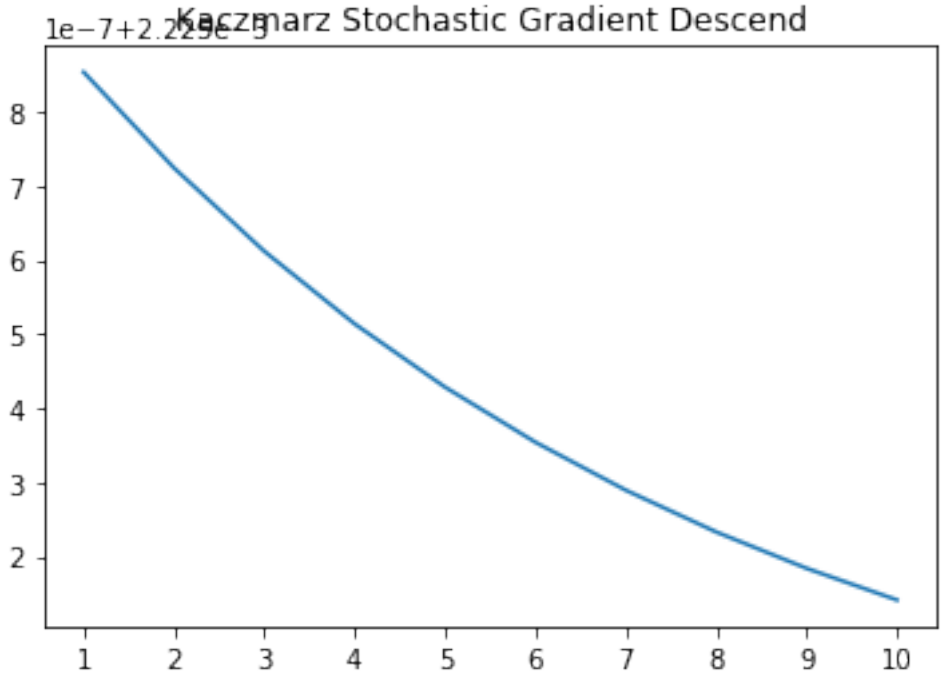
1.4 (d)

```
In [121]: class KSGD(object):
              def __init__(self,
                           total_step_count = 10):
                  self.total_step_count = total_step_count
                  self.MSE = np.zeros((total_step_count,1))

              def gradient(self, x, y, w):
                  return np.dot(x.T,np.dot(x,w)-y)/len(y)


              def gradient_descend_step(self,x, y, w, step_count):
                  return w - self.learning_rate * self.gradient(x, y, w)
```

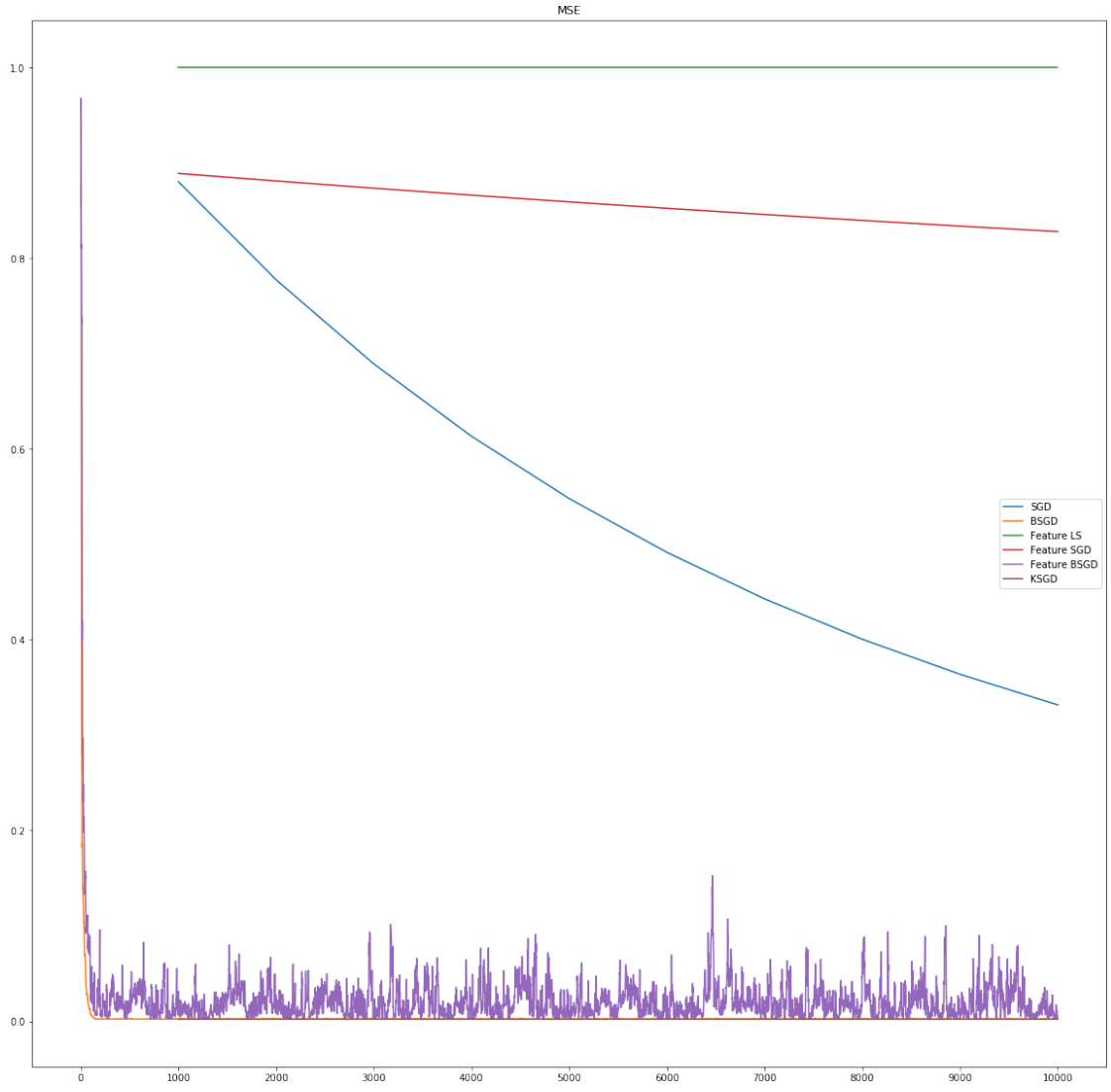
```
    def gradient_descend(self, x, y, w):
        W = [np.array(w)]
        for k in range(self.total_step_count):
            new = self.gradient_descend_step(x, y, W[-1], k)
            self.MSE[k] = np.mean(np.linalg.norm(np.dot(x,new)-y,axis=1)**2)
            W.append(new)
        return W[-1]

    def train(self,x,y):
        r,m = np.shape(x)
        _,n = np.shape(y)
        self.w = np.zeros((m,n))
        for i in range(self.total_step_count):
            J = [np.linalg.norm(x[i,:])**2 for i in range(r)]
            j = np.argmax(J/np.linalg.norm(x)**2)
            self.learning_rate = 1/J[j]
            self.w = self.gradient_descend(x,y,self.w)
```

In [122]:
```
ksgd = KSGD()
ksgd.train(train_x,train_y)
```

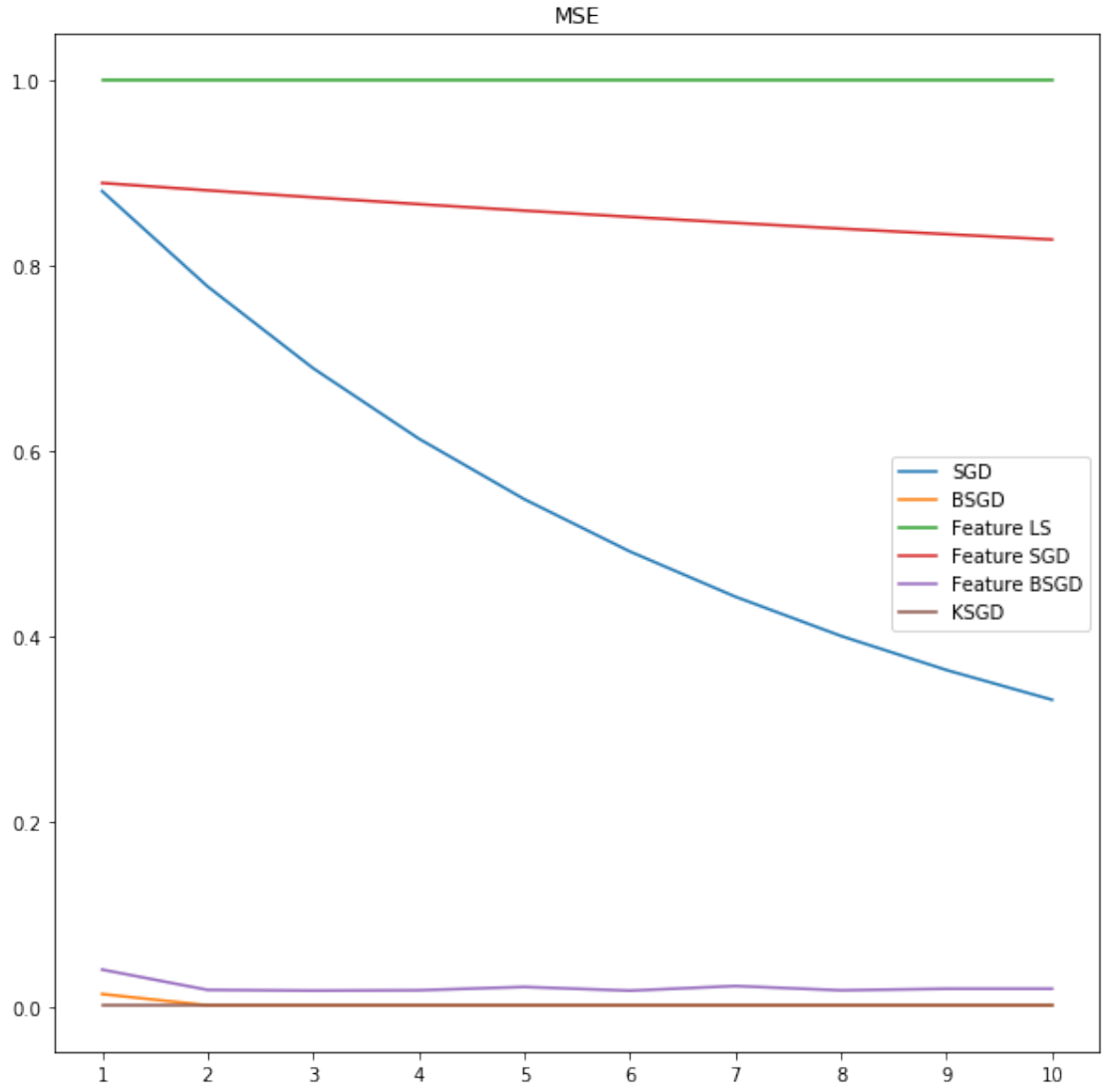
In [125]:
```
plt.figure()
plt.plot(np.arange(1,11),ksgd.MSE)
plt.title('Kaczmarz Stochastic Gradient Descend')
plt.xticks(np.arange(1,11),np.arange(1,11))
plt.show()
```



In [126]:
```
plt.figure(figsize=(20,20))
plt.plot(np.arange(1000,10001,1000),sgd.MSE,label='SGD')
plt.plot(np.arange(1,10001),bsgd.MSE,label='BSGD')
plt.plot(np.arange(1000,10001,1000),fls.MSE,label='Feature LS')
plt.plot(np.arange(1000,10001,1000),fsgd.MSE,label='Feature SGD')
plt.plot(np.arange(1,10001),fbsgd.MSE,label='Feature BSGD')
plt.plot(np.arange(1000,10001,1000),ksgd.MSE,label='KSGD')
plt.title('MSE')
plt.legend()
plt.xticks(np.arange(0,10001,1000),np.arange(0,10001,1000))
plt.show()
```

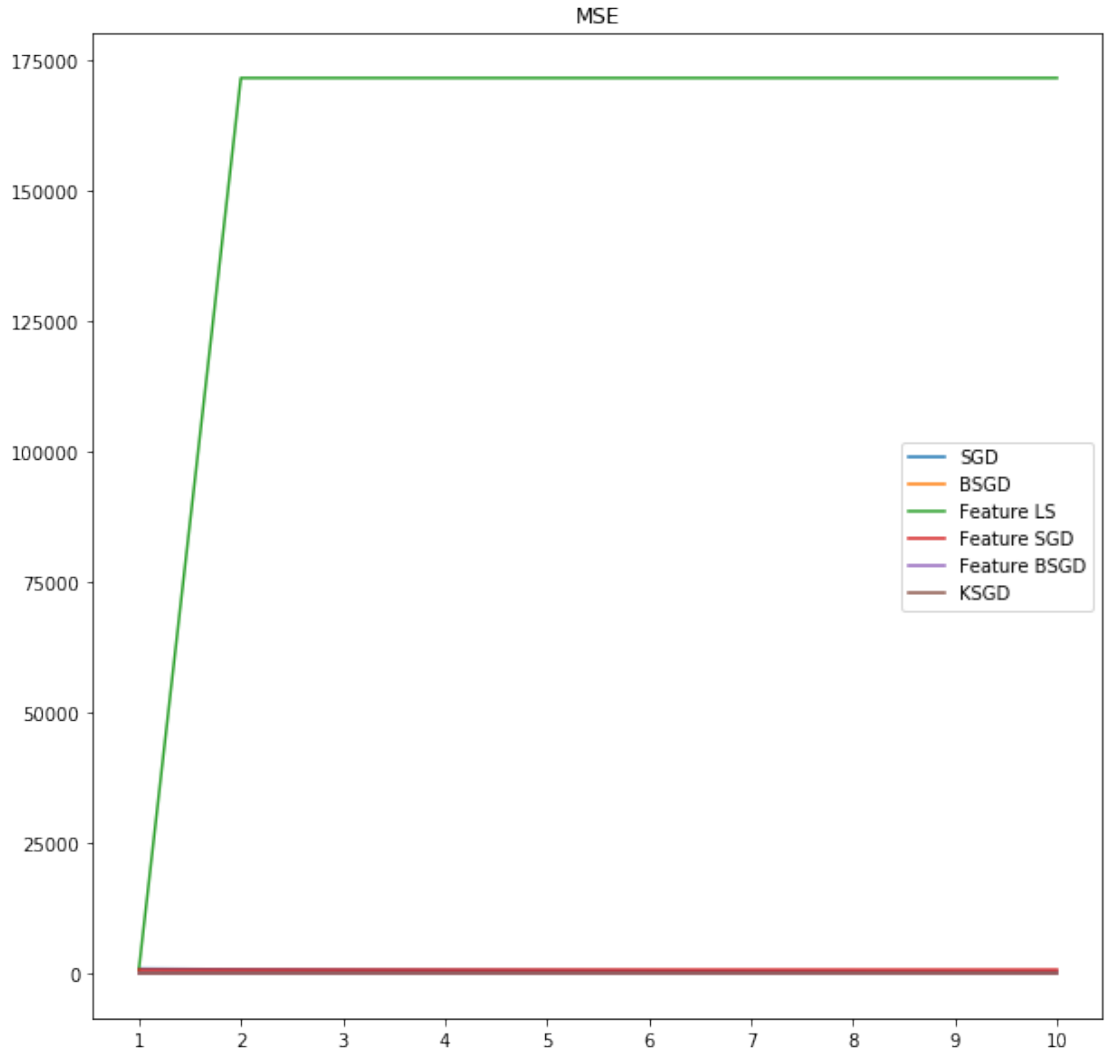


In [134]:
```
bsgd_MSE = np.mean(np.reshape(bsgd.MSE,(10,1000)),axis=1)
fbsgd_MSE = np.mean(np.reshape(fbsgd.MSE,(10,1000)),axis=1)
plt.figure(figsize=(10,10))
plt.plot(np.arange(1,11),sgd.MSE,label='SGD')
plt.plot(np.arange(1,11),bsgd_MSE,label='BSGD')
plt.plot(np.arange(1,11),fls.MSE,label='Feature LS')
plt.plot(np.arange(1,11),fsgd.MSE,label='Feature SGD')
plt.plot(np.arange(1,11),fbsgd_MSE,label='Feature BSGD')
plt.plot(np.arange(1,11),ksgd.MSE,label='KSGD')
plt.title('MSE')
plt.legend()
plt.xticks(np.arange(1,11),np.arange(1,11))
plt.show()
```

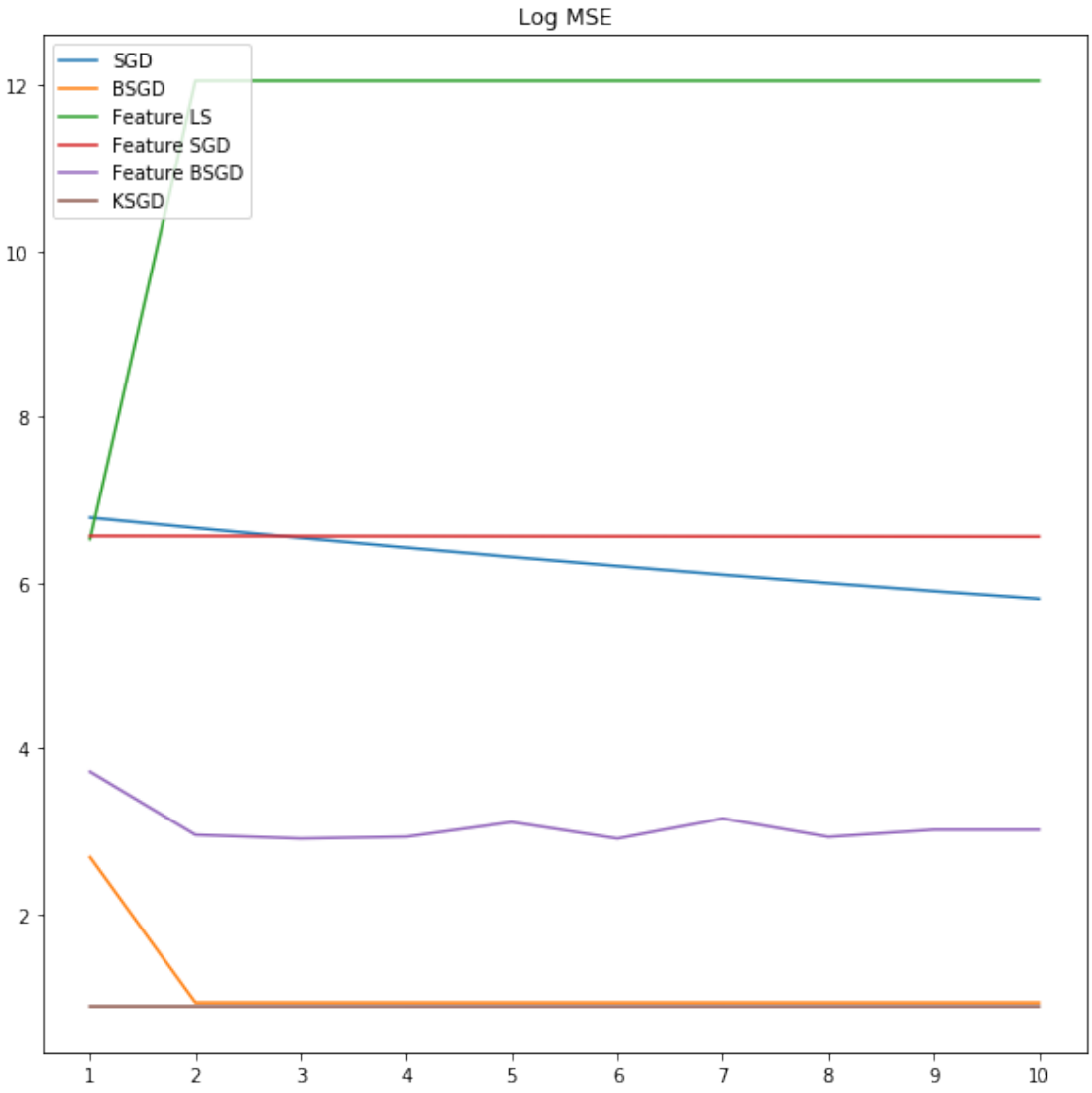


1.5 (e)

```
In [133]: y2 = y+0.1*x[:,1]
          train_y2 = (y2 - np.average(y2))/ np.std(y2)

In [137]: sgd2 = SGD()
          sgd2.train(train_x,train_y2)
          bsgd2 = BSGD()
          bsgd2.train(train_x,train_y2,1)
          np.random.seed(0)
          fls2 = FLS()
          fls2.train(train_x,train_y2)
          np.random.seed(0)
          fsgd2 = FSGD()
          fsgd2.train(train_x,train_y2)
          np.random.seed(0)
          fbsgd2 = FBSGD()
          fbsgd2.train(train_x,train_y2,1)
          ksgd2 = KSGD()
          ksgd2.train(train_x,train_y2)

In [138]: bsgd2_MSE = np.mean(np.reshape(bsgd2.MSE,(10,1000)),axis=1)
          fbsgd2_MSE = np.mean(np.reshape(fbsgd2.MSE,(10,1000)),axis=1)
          plt.figure(figsize=(10,10))
          plt.plot(np.arange(1,11),sgd2.MSE,label='SGD')
          plt.plot(np.arange(1,11),bsgd2_MSE,label='BSGD')
          plt.plot(np.arange(1,11),fls2.MSE,label='Feature LS')
          plt.plot(np.arange(1,11),fsgd2.MSE,label='Feature SGD')
          plt.plot(np.arange(1,11),fbsgd2_MSE,label='Feature BSGD')
          plt.plot(np.arange(1,11),ksgd2.MSE,label='KSGD')
          plt.title('MSE')
          plt.legend()
          plt.xticks(np.arange(1,11),np.arange(1,11))
          plt.show()
```




```
In [182]: plt.figure(figsize=(10,10))
          plt.plot(np.arange(1,11),np.log(sgd2.MSE),label='SGD')
          plt.plot(np.arange(1,11),np.log(bsgd2_MSE),label='BSGD')
          plt.plot(np.arange(1,11),np.log(fls2.MSE),label='Feature LS')
          plt.plot(np.arange(1,11),np.log(fsgd2.MSE),label='Feature SGD')
          plt.plot(np.arange(1,11),np.log(fbsgd2_MSE),label='Feature BSGD')
          plt.plot(np.arange(1,11),np.log(ksgd2.MSE),label='KSGD')
          plt.title('Log MSE')
          plt.legend()
          plt.xticks(np.arange(1,11),np.arange(1,11))
          plt.show()
```


Log MSE
SGD
BSGD
Feature LS
Feature SGD
Feature BSGD
KSGD
12
10
8
6
4
2
1
2
3
4
5
6
7
8
9
10

In [ ]: