Q2) $(w^*, b^*, \xi^*) \rightarrow$ solution to the soft margin hyperplane quadratic program.

$$\left.\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i \\ \text{st} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}\right\} \quad (\#)$$

2a) Given that,

x_i is missclassified.

If x_i is missclassified then we know that,

$$y_i(w^{*T} x_i + b^*) \leq 0$$

Reason: y_i and $w^{*T} x_i + b^*$ will have opposite sign's if x_i is missclassified

From the problem structure (#), we also know that

$$\overbrace{1 - \xi_i \leq y_i(w^{*T} x_i + b^*)} \leq 0$$

$$\Rightarrow 1 - \xi_i \leq 0$$

$$\boxed{\Rightarrow \xi_i \geq 1}$$

Training error = $\dfrac{\text{No. of misclassifications}}{n}$

$$= \frac{\sum_i \mathbb{1}\{y_i \neq \text{sign}(w^{*T} x_i + b^*)\}}{n}$$

We have, for misclassified x_i's, $\xi^* \geq 1$

$$\Rightarrow \quad 1 \times \mathbb{1}_{\{y_i \neq \text{sign}(w^{*T}x_i + b^*)\}} \leq \xi^* \times \mathbb{1}_{\{y_i \neq \text{sign}(w^{*T}x_i + b^*)\}}$$

$$\frac{\sum_i \mathbb{1}_{\{y_i \neq \text{sign}(w^{*T}x_i + b^*)\}}}{n} \leq \frac{\sum \xi_i^* \mathbb{1}_{\{y_i \neq \text{sign}(w^{*T}x_i + b^*)\}}}{n}$$

$$\Rightarrow \quad \frac{\text{No. of misclassifications}}{n} \leq \frac{\sum_i \xi_i^*}{n}$$

$$\Rightarrow \quad \boxed{\text{training error} \leq \frac{\sum_i \xi_i^*}{n}}$$

$\frac{\sum_i \xi_i}{n}$ upper bounds the training error

Objective

$$\min_{w, b, \xi} \quad \underbrace{\frac{1}{2}\|w\|^2} + C \times \left[\frac{\sum \xi_i}{n}\right]$$

$$\min_{w, b, \xi} \quad \frac{1}{2}\|w\|^2 \Rightarrow \max \left\{\frac{1}{\|w\|}\right\} \xrightarrow{\alpha} \text{margin maximization}$$

$$\min_{w, b, \xi} \quad C\left[\frac{\sum \xi_i}{n}\right] \Rightarrow C\left[\text{upper bound on error}\right]$$

Here the SVM is balacing out margin maximization with minimizing a bound on the training error

b) $\xi_i^* > 0$

Distance of a point z to the plane $w^T z + b = 0$ is

$$d_z = \frac{|w^T z + b|}{\|w\|}$$

the max-margin hyperplane associated with class y_i,

$$\Rightarrow w^{*T} x + b^* = y_i$$

distance of point x_i from this max-margin hyperplane,

$$d_{x_i} = \frac{|w^{*T} x_i + b^* - y_i|}{\|w^*\|} \quad - (1)$$

We know that, from the soln of OSM

$$\xi^* = \min \{ 0, 1 - y_i (w^{*T} x_i + b^*) \}$$

Since we are given $\xi^* > 0$

$$\Rightarrow \xi^* = 1 - y_i (w^{*T} x_i + b^*)$$

$$w^{*T} x_i + b^* = \left[\frac{1 - \xi^*}{y_i} \right] \quad - (2)$$

Substitute (2) in (1)

$$d_{x_i} = \frac{\left| \frac{1 - \xi_i^*}{y_i} - y_i \right|}{\|w^*\|} = \frac{\left| \frac{1 - \xi_i^* - y_i^2}{y_i} \right|}{\|w^*\|}$$

We can use the fact that $y_i = \{-1, 1\}$

So, $|y_i| = 1$ and $\underline{\underline{y_i^2 = 1}}$

$$d_{xi} = \frac{\left| \frac{\cancel{1} - \xi_i^* \cancel{-1}}{1} \right|}{\|w^*\|}$$

$$d_{xi} = \frac{\xi_i^*}{\|w^*\|}$$

$$\Rightarrow \xi_i^* = d_{xi} \|w^*\|$$

$\therefore \xi_i^* \propto d_{xi}$ & constant of proportionality is $\|w^*\|$

Q1)

(b) $w_0 = 181.678874$

First 5 optimal weights of ridge regression. $[w_1\ w_2\ w_3\ w_4\ w_5]$

$[3.02355147,\ 4.58605397,\ 2.38090826,\ 0.122324962,\ -0.14434401]$

(d) The train data MSE increases as λ is increased, where as the test data MSE drops till a certain optimal λ^*_{test} then increases again. Both the curves closely resemble a quadratic function

$$\left.\begin{aligned} \lambda^*_{train} &= 1 \\ \lambda^*_{test} &= 87.5628 \end{aligned}\right\} \text{ } \lambda\text{'s corresponding to minimal MSE}$$

Q3) Given,

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}\left(L(y_i, w^T x_i + b) + \frac{\lambda}{2}\|w\|^2\right)$$

where $L(y,t) = \max\{0, 1-yt\}$ - Hinge Loss.

(a) Determine $J_i(w,b)$

$$J(w,b) = \sum_{i=1}^{n} J_i(w,b)$$

Let, $\theta = \begin{bmatrix} b \\ w \end{bmatrix}$ and $\tilde{x}_i = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$, then. $J(w,b)$ can be written as,

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\max\{0, 1-y_i(\theta^T\tilde{x}_i)\} + \frac{\lambda}{2}\theta^T A\theta\right) \quad \text{—} (1)$$

where $A = \begin{bmatrix} 0 & 0_{1\times d} \\ 0_{d\times 1} & I_{d\times d} \end{bmatrix}$

We can now write this $J(\theta)$ as,

$$J(\theta) = \sum_{i=1}^{n} J_i(\theta)$$

where $J_i(\theta) = \frac{1}{n}\max\{0, 1-y_i\theta^T\tilde{x}_i\} + \frac{\lambda}{2n}\theta^T A\theta$

Just to write it more explicitly,

$$J_i(\theta) = \begin{cases} \frac{\lambda}{2n}\theta^T A\theta & \text{when } 1-y_i\theta^T\tilde{x}_i < 0 \\ \frac{1}{n}(1-y_i\theta^T\tilde{x}_i) + \frac{\lambda}{2n}\theta^T A\theta & \text{when } 1-y_i\theta^T\tilde{x}_i \geq 0 \end{cases}$$

We can now find the sub-gradient of $\nabla J(\theta) = \sum_i \nabla J_i(\theta)$

$$\nabla J(\theta) = \sum_i \nabla J_i(\theta) = \sum_i u_i$$

$$\nabla J_i(\theta) = \begin{cases} \frac{\lambda}{\cancel{2}n} \cancel{2} A\theta & 1 - y_i\theta^T\tilde{x}_i < 0 \\ -\frac{y_i\tilde{x}_i}{n} + \frac{\lambda}{\cancel{2}n} \times \cancel{2} A\theta & \underbrace{1 - y_i\theta^T\tilde{x}_i \geq 0} \end{cadses}$$

derived in earlier HW's

Explanation

$$\nabla\left(\frac{\lambda}{2n}\theta^TA\theta\right) = \frac{\lambda}{\cancel{2}n}(\cancel{2}A\theta) = \frac{\lambda}{n}A\theta$$

$$\nabla\left(\frac{1 - y_i\theta^T\tilde{x}_i}{n}\right) = \frac{-y_i\tilde{x}_i}{n}$$

for, $1 - y_i\theta^T\tilde{x}_i = 0$

the sub-gradient is not unique but I have considered this as per Prof Hero Lecture side 08 page 32.

$$\therefore \quad \nabla J_i(\theta) = \underline{u_i} = \begin{cases} \frac{\lambda}{n} A\theta & 1 - y_i\theta^T\tilde{x}_i < 0 \\ -\frac{y_i\tilde{x}_i}{n} + \frac{\lambda}{n} A\theta & 1 - y_i\theta^T\tilde{x}_i \geq 0 \end{cases}$$

$\therefore \underbrace{\nabla J(\theta)}_{\text{(subgradient)}} = \sum_i \nabla J_i(\theta) = \sum_i u_i$, where u_i is defined as above

Q3c) The ~~plot~~ plot of rate of convergence / decay of objective is provided in the next page.

The stochastic sub gradient descent has steeper curve than the subgradient descent. The plot for subgradient descent and stochastic sub gradient descent is plotted with J (cost) on y-axis and log (iterations).

subgradient descent → had a nearly linear decay of $J(\theta)$ with log (iterations)

Stochastic subgradient descent → had a exponential kind of decay of $J(\theta)$ with log (iterations)

Q 3b) gradient-descent results

hyperplane parameters $[b\ w^T]^T \rightarrow$ [12.068, -17.816, -9.1171]

total margin : 0.1657

minimum value of objective function:- 0.44988

3c) stochastic gradient-descent results:

hyperplane parameters: $[b\ w^T]^T \rightarrow$ [4.0051, -5.8246, - 4.4142]

total margin: 0.4993

minimum value of objective function achieved: 0.125827824

Q4)

$$F_{ridge}(w) = \sum_{j=1}^{n} (y_j - \hat{y}_j(w))^2 + \lambda \sum_{i=1}^{p} |w_i|^2,$$

$$\lambda > 0 \quad \hat{y}_j = w_0 + \sum_{i=1}^{p} w_i x_{ij} = w_0 + x^T w_1$$

a)

(a) $$\frac{\partial F_{ridge}}{\partial w_0} = \frac{\partial}{\partial w_0}\left[\sum_{j=1}^{n} (y_j - (w_0 + x_j^T w_1))^2 + \lambda \sum_{i=1}^{p} |w_i|^2\right]$$

$$= \sum_{j=1}^{n} \frac{\partial}{\partial w_0}(y_i - (w_0 + x_j^T w_1))^2 + \cancel{\lambda \sum_{i=1}^{p} \frac{\partial}{\partial w_0} |w_i|^2} \to 0$$

Sum is a

linear operator

$$= \sum_{j=1}^{n} 2\,(y_i - (w_0 + x_j^T w_1))\,(1)$$

$$= 2\left[\sum_{j=1}^{n} y_i - \sum_{j=1}^{n} w_0 - \sum_{j=1}^{n} x_j^T w_1\right]$$

$$= 2\left[\sum_{j=1}^{n} y_i - \sum_{j=1}^{n} w_0 - \cancel{\left[\sum_{j=1}^{n} x_j^T\right]} w_1\right] \to 0 \quad \text{as } \sum_{j=1}^{n} x_j^T = 0$$

* [data is sphered off]

$$\frac{\partial F_{ridge}}{\partial w_0} = 2\left[\sum_{j=1}^{n} y_i - \sum_{j=1}^{n} w_0\right]$$

$$\frac{\partial F_{ridge}}{\partial w_0} = 0 \Rightarrow \sum_{j=1}^{n} w_0 = \sum_{j=1}^{n} y_i$$

$$n w_0 = \sum_{j=1}^{n} y_i$$

$$\boxed{w_0 = \frac{1}{n} \sum_{j=1}^{n} y_i}$$

(b) w_i $\quad i=1,2\ldots d$

data dependent term in F_{ridge} $= \underbrace{\frac{\partial}{\partial w_i} \sum (y_i - \hat{y}_j(w))^2}$

$$\Rightarrow \frac{\partial}{\partial w_i} \sum_{j=1}^{n} (y_i - \hat{y}_j(w))^2 \qquad \underline{\hat{y}_j(w) = x_j^T w_1 + w_0}$$

$$\Rightarrow \frac{\partial}{\partial w_i} \sum_{j=1}^{n} \left[y_i - (w_0 + x_j^T w_1)\right]^2$$

$$\Rightarrow \sum_{j=1}^{n} \frac{\partial}{\partial w_i} \left[y_i - w_0 - x_j^T w_i\right]^2$$

✓

as sum is a linear operator we can pull it out.

$$\Rightarrow \sum_{j=1}^{n} 2\left[y_i - w_0 - x_j^T w_1\right]\left(\underbrace{\frac{\partial}{\partial w_i}\left[y_i - w_0 - x_j^T w_1\right]}\right)$$

$$\frac{\partial y_i}{\partial w_i} = 0 \qquad \frac{\partial w_0}{\partial w_1} = 0 \qquad \frac{\partial x_j^T w_1}{\partial w_i} \quad \frac{\partial x_{ij} w_i + \ldots}{\partial w_i} \qquad \boxed{\frac{\partial x_j^T w_1}{\partial w_i} = x_{ij}}$$

$$\Rightarrow -\sum_{j=1}^{n} 2\left[y_i - w_0 - x_j^T w_1\right] x_{ij}$$

$$\Rightarrow -\sum_{j=1}^{n} 2\left[y_i - w_0 - (x_{1j} w_1 + x_{2j} w_2 \cdots x_{ij} w_i \cdots x_{nj} w_n)\right] x_{ij}$$

just pull out this term

$$\Rightarrow -\sum_{j=1}^{n} 2\left[y_i - \hat{y}_j(w_{-i}) - x_{ij} w_i\right] x_{ij} \quad x_{ij} w_i$$

$\hat{y}_j(w_{-i}) \rightarrow$ is $\underline{\hat{y}_j(w_i) - x_{ij} w_i}$

$$= -\sum_{j=1}^{n} 2\left[y_i - \hat{y}_j(w_{-i})\right] x_{ij} + \sum_{j=1}^{n} 2 x_{ij}^2 w_i$$

$$\Rightarrow 2\sum_{j=1}^{n} x_{ij}^2 w_i - 2\sum\left[y_i - \hat{y}_j(w_{-i})\right] x_{ij} \Rightarrow a_i w_i - c_i$$

where $a_i = 2\sum_{j=1}^{n} x_{ij}^2 \qquad c_i = 2\sum_{j=1}^{n} x_{ij}(y_i - \underline{\hat{y}_j(w_{-i})})$

(c) $F_{ridge}(w)$, set $\frac{\partial F_{ridge}}{\partial w_i} = 0$

$$F_{ridge} = \sum_{j=1}^{n} (y_j - \hat{y}_j(w))^2 + \lambda \sum_{j=1}^{p} |w_j|^2$$

From 4(b)

$$\frac{\partial}{\partial w_i}\left(\sum_{j=1}^{n} (y_j - \hat{y}_j(w))^2\right) \rightarrow a_i w_i - c_i$$

where, $a_i = 2\sum_{j=1}^{n} x_{ij}^2$ $\quad c_i = 2\sum_{j=1}^{n} x_{ij}(y_j - \hat{y}_j(w_{-i}))$

$$\frac{\partial}{\partial w_i} \lambda \sum_{j=1}^{p} |w_i|^2$$

$$\Rightarrow \underline{\underline{2\lambda w_i}}$$

So, when we set $\frac{\partial F_{ridge}}{\partial w_i} = 0$

$$\Rightarrow a_i w_i - c_i + 2\lambda w_i = 0$$

$$(a_i + 2\lambda) w_i - c_i = 0$$

$$\Rightarrow w_i^* = \frac{c_i}{a_i + 2\lambda}$$

$\frac{\partial^2 F_{ridge}}{\partial w_i^2}$ is $(a_i + 2\lambda) > 0$

as $\lambda > 0$ & $x_{ij}^2 \geq 0$

Hence w_i^* is minimizer

$$\underline{\arg\min_{w_i} F_{ridge}(w) = w_i^* = \frac{c_i}{a_i + 2\lambda}} \quad i \geq 1.$$

Q.5)

$$F_{lasso}(W) = \sum_{j=1}^{n} (y_j - \hat{y}_j(W))^2 + \lambda \sum_{i=1}^{p} |w_i|$$

$$\hat{y}_j(W) = w_0 + \sum_{i=1}^{p} w_i x_{ij} = w_0 + X^T W_1$$

a) Given

$$\partial_{w_i} F_{lasso}(W) = a_i w_i - c_i + \lambda \partial_{w_i} |w_i| \quad , i \geq 1$$

where

$$a_i = 2 \sum_{j=1}^{n} x_{ij}^2 \qquad c_i = 2 \sum_{j=1}^{n} x_{ij} (y_j - \hat{y}_j(w_{-i})) \quad , \underline{i \geq 1}$$

For optimality $\partial_{w_i} F_{lasso}(w) = 0$

$$\Rightarrow a_i w_i - c_i + \lambda \partial_{w_i} |w_i| = 0$$

We know that

$$\partial_w |w| = \begin{cases} 1 & w > 0 \\ [-1, 1] & w = 0 \\ -1 & w < 0 \end{cases}$$

Case-1

(i) assume, $w_i > 0$

$\partial_{w_i} F_{lasso} = a_i w_i - c_i + \lambda = 0$

$w_i = \dfrac{c_i - \lambda}{a_i}$ $\qquad w_i > 0$ is our assumption

So $c_i - \lambda > 0$ as $\underline{a_i > 0 \text{ always}}$

$\Rightarrow \underline{c_i > \lambda}$

Case-2

(ii) assume, $w_i < 0$

$\partial_{w_i} F_{lasso} = a_i w_i - c_i - \lambda = 0$

$w_i = \dfrac{c_i + \lambda}{a}$ $\qquad w_i < 0$ was our assumption

So, $c_i + \lambda < 0$ and $\underline{a_i > 0 \text{ always}}$

$\Rightarrow \underline{c_i < -\lambda}$

(iii) Case 3

assume $w_i = 0$

$\Rightarrow \partial_{w_i} F_{lasso} = a_i w_i - c_i + \lambda \partial_{w_i} |w_i| \ni 0$, with $a_i w_i \to 0$ and $\partial_{w_i}|w_i| \to [-1, 1]$

$\Rightarrow$ ~~$-c_i \pm \lambda = 0$~~ $\Rightarrow -c_i + [-\lambda, \lambda] \ni 0$

~~$c_i = \pm\lambda$~~

$\underbrace{c_i \in [-\lambda, \lambda]}$

From Case-1, Case-2, and Case-3,

$$w_i = \text{soft}\left(\frac{c_i}{a_i}, \frac{\lambda}{a_i}\right) = \begin{cases} \frac{c_i - \lambda}{a_i} & c_i > \lambda \\ 0 & c_i \in [-\lambda, \lambda] \\ \frac{c_i + \lambda}{a_i} & c_i < -\lambda \end{cases}$$

Hence verified that if w_i is specified, it satisfies the subdifferential optimality condition.

$0 \in \partial_{w_i} F_{lasso}(w)$

Also,

$\rightarrow \partial^2_{w_i} F_{lasso} = a_i$

and $a_i > 0$

hence w_i is minimizer of F_{lasso}

[obtained by setting $\partial_{w_i} F_{lasso} = 0$]

Q) 5 (b)   Final test MSE = 981.092

CD lasso solution.

Comparision of MSE

CD ridge regression	CD lasso regression
→ MSE: 751.1934	→ MSE: 981.092
→ W_{max} ~ 15 (max weight) after 2900 iterations	→ W_{max} ~ 23 (max weight) after 2900 iteration

It is observed that MSE is less for ridge regression, and the solution of the CD lasso regression is a lot sparser. and only some important features are given priority.

CD lasso regression → sparse

No. of zero weights - 34

weights after 2900 iterations [4, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 28, 29, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 44, 45, 46, 50, 51, 52, 53, 54, 56]

The weight vector starts from (0).

Problem 1 (a)

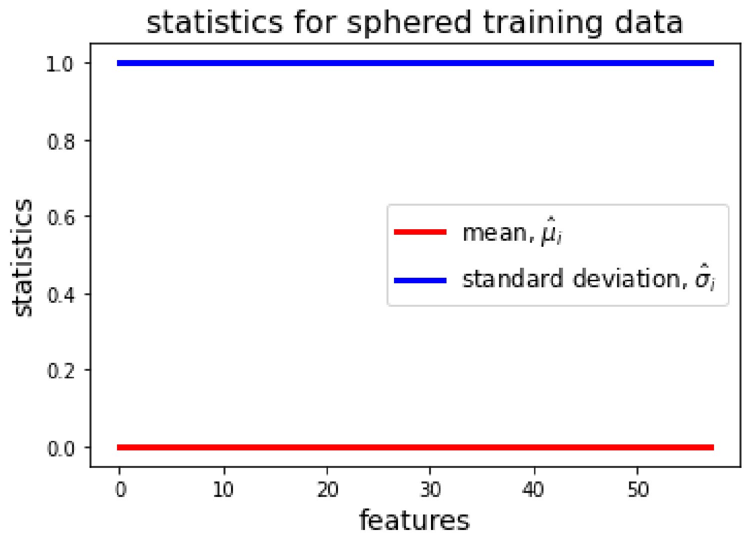


Figure 1: statistics of sphered training data

Problem 1 (c)

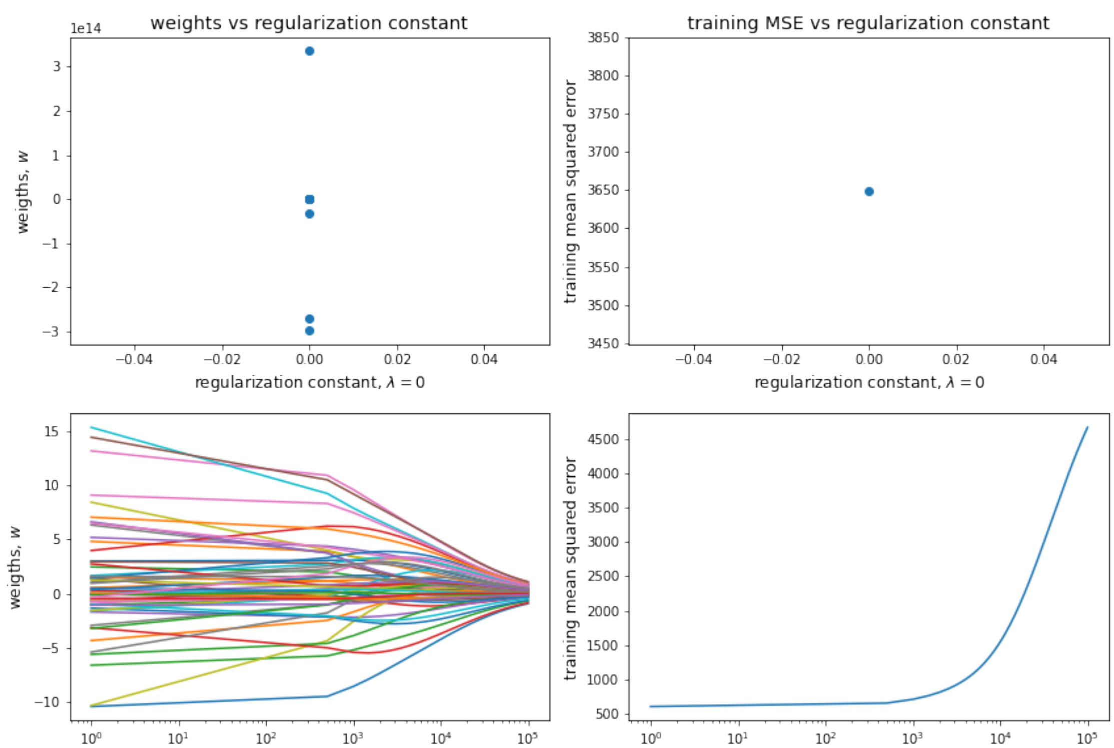


Figure 2: weights vs regularization constant

Problem 1 (d)

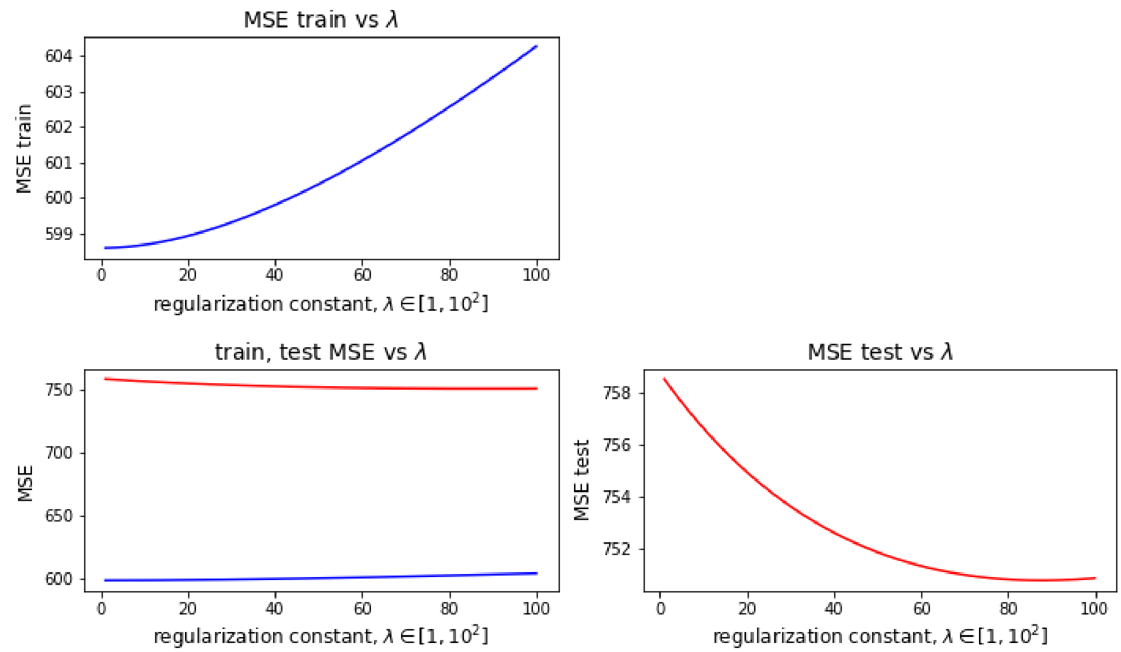


Figure 3: MSE train and test vs λ

Problem 3 (b)

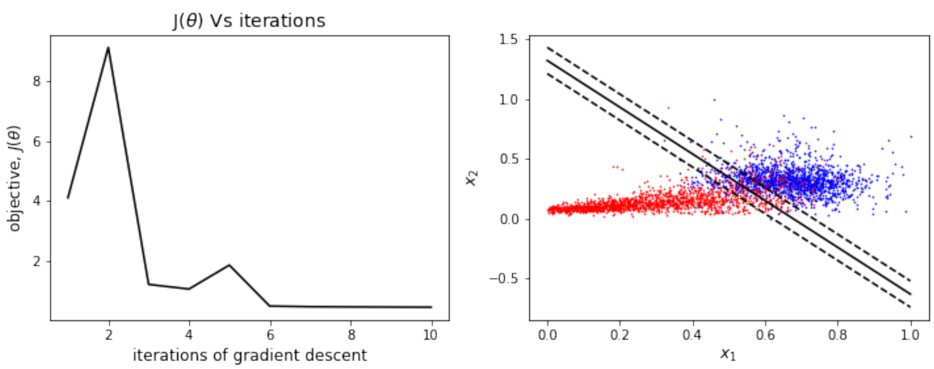


Figure 4: gradient-descent results for OSM after 10 iterations

Problem 3 (c)

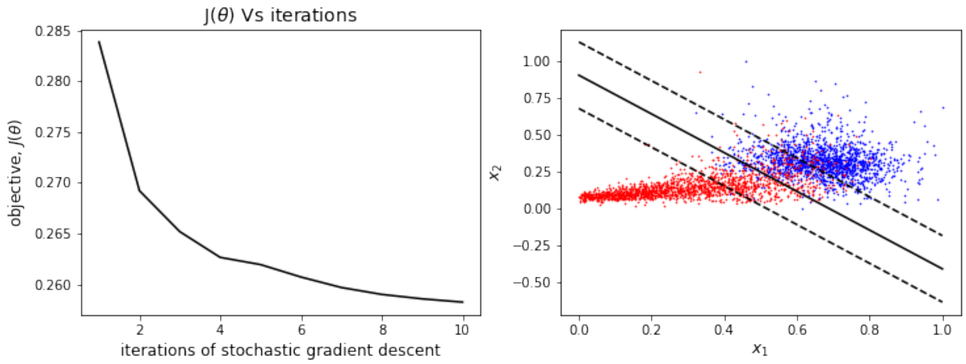


Figure 5: stochastic-gradient-descent results for OSM after 10 iterations

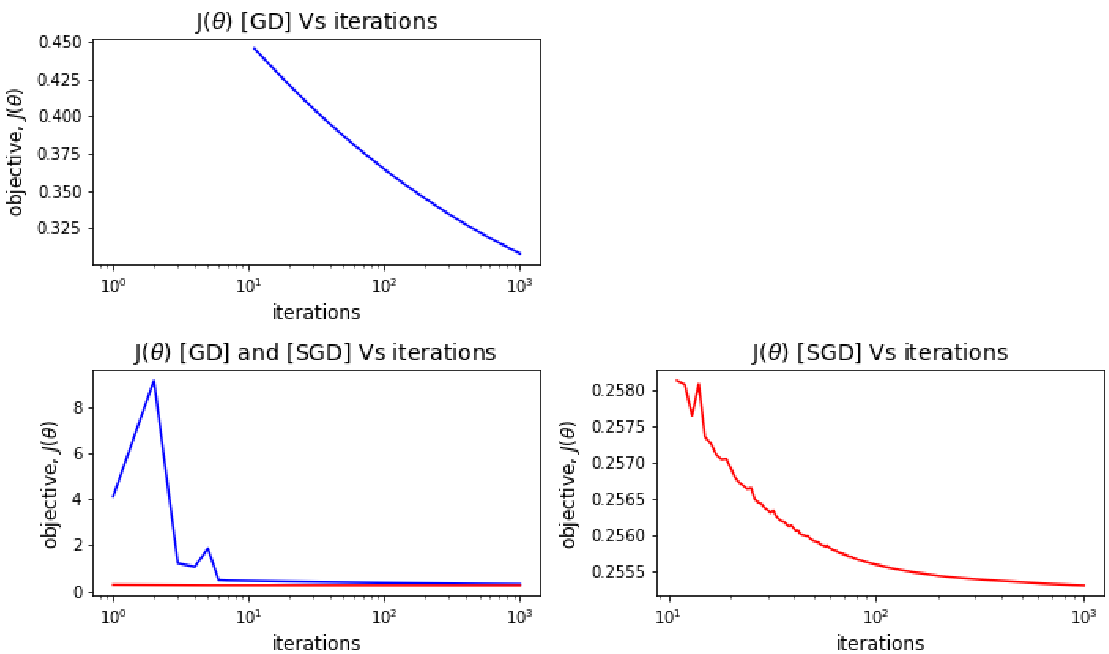


Figure 6: Comparison of Gradient Descent and Stochastic gradient descent over 100 iterations

Problem 4 (d)

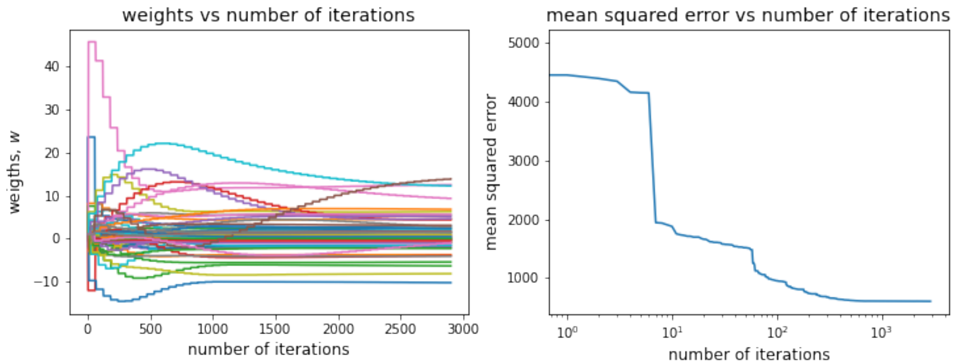


Figure 7: weights, MSE vs iterations for CD regression

Problem 5 (b)

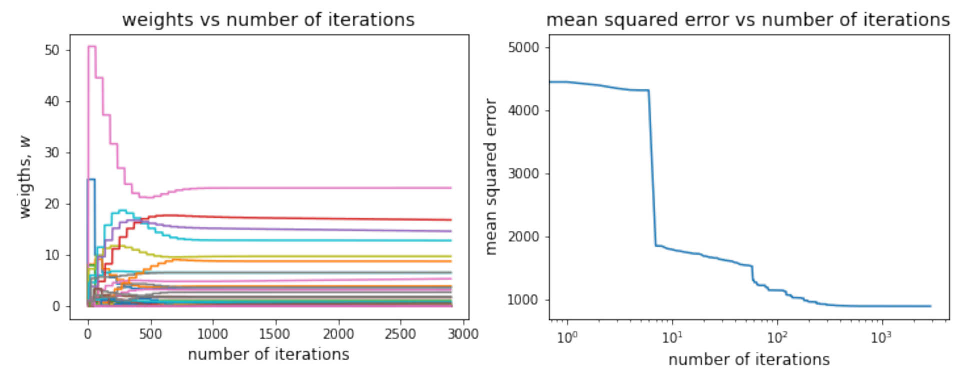


Figure 8: weights, MSE vs iterations for CD lasso regression

Problem 1

In [60]:
```
%matplotlib inline

# load relevant packages
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
```

In [61]:
```
# load test and train data
Xtrain = np.load("hw3_housing/housing_train_features.npy")
Xtest = np.load("hw3_housing/housing_test_features.npy")
ytrain = np.load("hw3_housing/housing_train_labels.npy")
ytest = np.load("hw3_housing/housing_test_labels.npy")
```

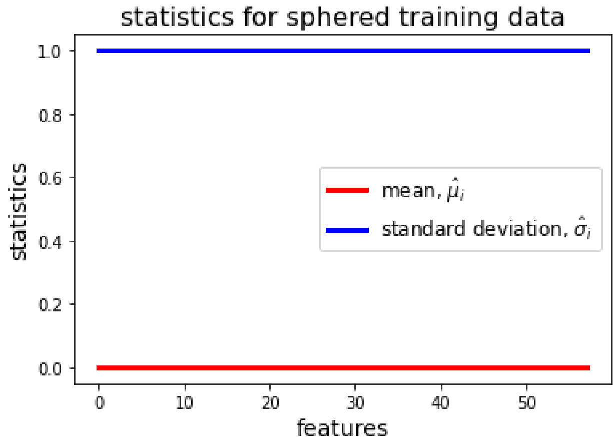
In [62]:
```
#
feature_names = np.load("hw3_housing/housing_feature_names.npy", allow_pickle=True)
print("First feature name: ", feature_names[0])
print("Lot frontage for first train sample:", Xtrain[0,0])
```

```
First feature name:  Lot.Frontage
Lot frontage for first train sample: 141.0
```

In [63]:
```
# define features and dimensions
nfeatures = Xtrain.shape[0]
ntrain = Xtrain.shape[1]
ntest = Xtest.shape[1]

# print
print('nfeatures: {} \nntrain: {} \nntest: {}\n'.format(nfeatures,ntrain,ntest))
```

```
nfeatures: 58
ntrain: 2000
ntest: 925
```

In [64]:
```
## Part (a)
# computation of statistics of the data
Xtrain_mean = np.mean(Xtrain,axis = 1)
Xtrain_std = np.std(Xtrain, axis = 1)

# sphere the data
Xtrain_sphere = ((Xtrain.T - Xtrain_mean)/Xtrain_std).T

# verify by plotting
features = np.arange(nfeatures)

plt.figure()
plt.plot(features,np.mean(Xtrain_sphere, axis = 1),
         color = 'red', linewidth = 3, label = 'mean, $\hat{\mu}_{i}$')
plt.plot(features,np.std(Xtrain_sphere, axis = 1),
         color = 'blue', linewidth = 3, label = 'standard deviation, $\hat{\sigma}_{i}$')
plt.xlabel('features', fontsize=14)
plt.ylabel('statistics', fontsize=14)
plt.title('statistics for sphered training data', fontsize=16)
plt.legend(fontsize=12)
plt.show()
plt.savefig('prob1a.png')
```




```
<Figure size 432x288 with 0 Axes>
```

In [65]:

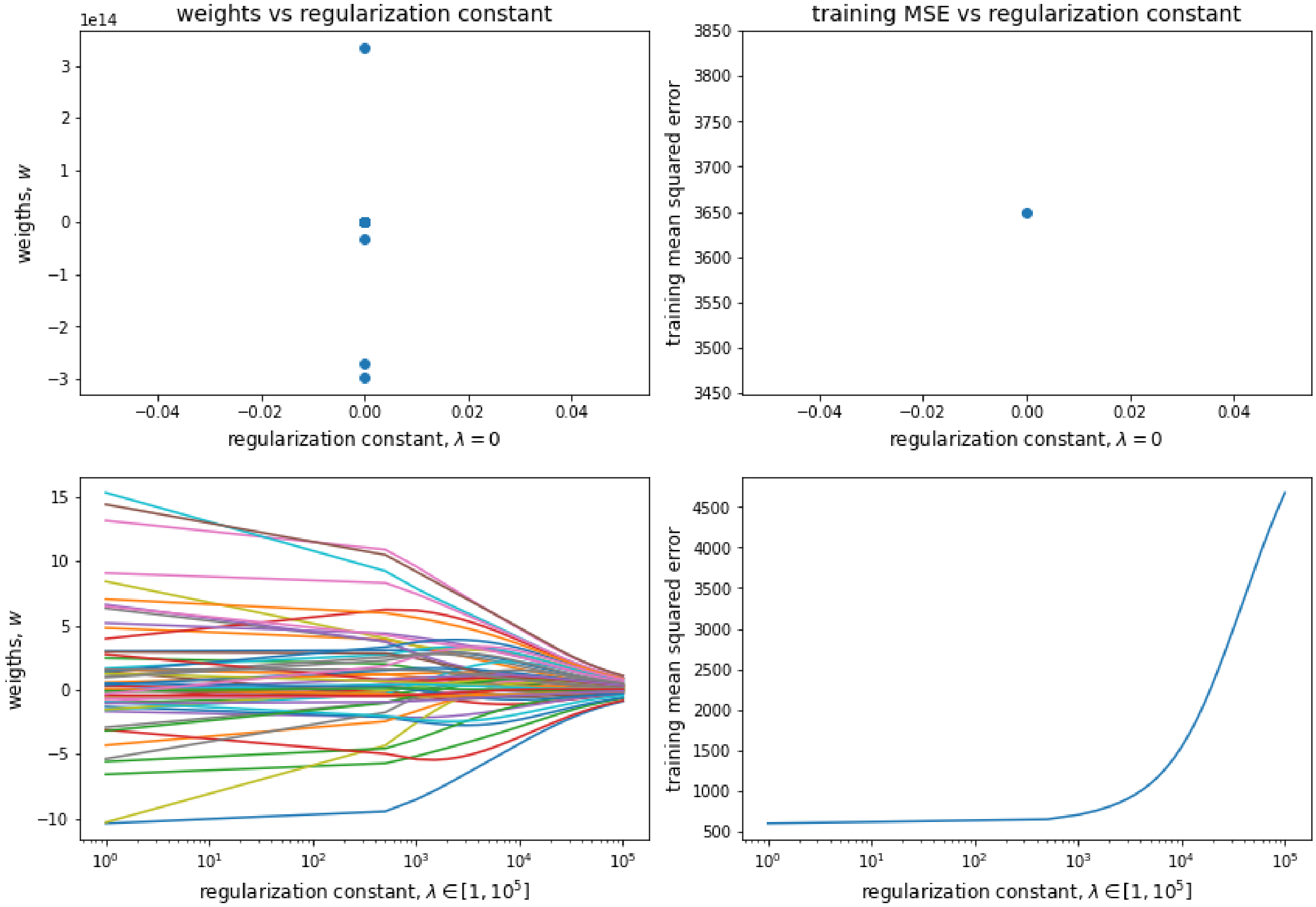
```python
## Part (b)
# computing Xbar
# [this zero but still I am adding it here for completeness]
Xbar = np.mean(Xtrain_sphere, axis = 1)
ybar = np.mean(ytrain) # computing ybar

Xtilde = (Xtrain_sphere.T - Xbar).T   # computing Xtilde
ytilde = ytrain-ybar                  # computing ytilde
reg_const = 100                       # class - 001 convention

# compute w
w = np.linalg.inv(Xtilde.dot(Xtilde.T) + reg_const*np.eye(nfeatures)).dot(Xtilde).dot(ytilde)
print('First 5 optimal weights of ridge regression:{}'.format(w[:5]))

# compute w0
w0 = ybar - w.dot(Xbar)
print('w0 of ridge regression: {}\n'.format(w0))
```

```
First 5 optimal weights of ridge regression:[ 3.02355147  4.58605397  2.38090826  0.22324962 -0.4434401 ]
w0 of ridge regression: 181.678874
```

In [66]:

```python
## Part (c)
# loop to find w for a range of regression constant


# regularization constant [0] - special case
reg_const = 0

w_reg_0 = np.linalg.inv(Xtilde.dot(Xtilde.T)
            + reg_const*np.eye(nfeatures)).dot(Xtilde).dot(ytilde)
w0_reg_0 = ybar - w_reg_0.dot(Xbar)   # constant put here for completeness
MSE_reg_0 = 1/ntrain*np.sum((w_reg_0.dot(Xtrain_sphere)
            + w0_reg_0*np.ones(ntrain) - ytrain)**2)

# plots
fig, axs = plt.subplots(2, 2, figsize = (12,8))

axs[0, 0].scatter(reg_const*np.ones(nfeatures),w_reg_0.T)
axs[0, 0].set_xlabel('regularization constant, $\lambda = 0$', fontsize=12)
axs[0, 0].set_ylabel('weigths, $w$', fontsize=12)
axs[0, 0].set_title('weights vs regularization constant',fontsize=14)

axs[0, 1].scatter(reg_const,MSE_reg_0)
axs[0, 1].set_xlabel('regularization constant, $\lambda = 0$', fontsize=12)
axs[0, 1].set_ylabel('training mean squared error', fontsize=12)
axs[0, 1].set_title('training MSE vs regularization constant',fontsize=14)
fig.tight_layout()

# regularization constant [1,10^5]
reg_const = np.linspace(1,10**5,200)

w_loop = np.zeros([nfeatures, len(reg_const)]) # weights for 200 cases
MSE_loop = np.zeros([len(reg_const), 1])
for i in range(len(reg_const)):
    w_loop[:,i] = np.linalg.inv(Xtilde.dot(Xtilde.T)
                + reg_const[i]*np.eye(nfeatures)).dot(Xtilde).dot(ytilde)
    w0_loop = ybar - w_loop[:,i].dot(Xbar)   # constant put here for completeness
    MSE_loop[i] = 1/ntrain*np.sum((w_loop[:,i].dot(Xtrain_sphere)
                + w0_loop*np.ones(ntrain) - ytrain)**2)
```

```
# plots
axs[1, 0].semilogx(reg_const,w_loop.T)
axs[1, 0].set_xlabel('regularization constant, $\lambda \in [1, 10^5]$', fontsize=12)
axs[1, 0].set_ylabel('weigths, $w$', fontsize=12)

axs[1, 1].semilogx(reg_const,MSE_loop)
axs[1, 1].set_xlabel('regularization constant, $\lambda \in [1, 10^5]$', fontsize=12)
axs[1, 1].set_ylabel('training mean squared error', fontsize=12)
plt.savefig('prob1c.png')
```

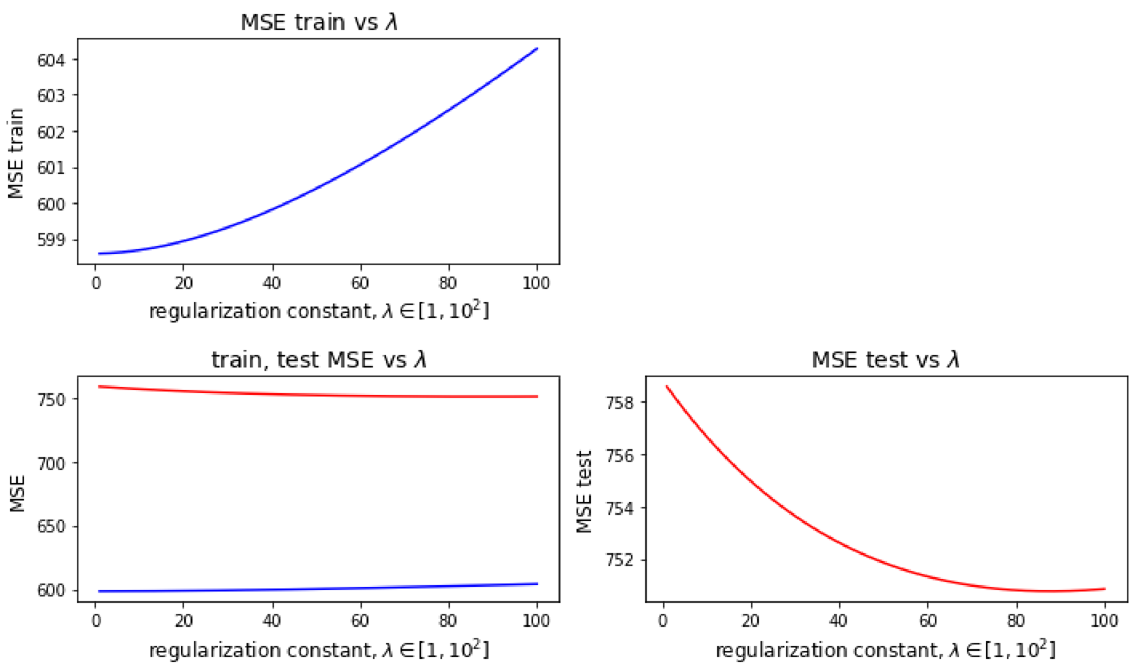


In [67]:

```
## Part (d)

# regularization constant
reg_const = np.linspace(1,100,200)

# loop
# w_loop = np.zeros([nfeatures, len(reg_const)]) # weights for 200 cases
MSE_train = np.zeros([len(reg_const), 1])      # MSE for 200 cases
MSE_test = np.zeros([len(reg_const), 1])       # MSE for 200 cases
for i in range(len(reg_const)):
    w_loop = np.linalg.inv(Xtilde.dot(Xtilde.T)
            + reg_const[i]*np.eye(nfeatures)).dot(Xtilde).dot(ytilde)
    w0_loop = ybar - w_loop.dot(Xbar)   # constant put here for completeness
    MSE_train[i] = 1/ntrain*np.sum((w_loop.dot(Xtrain_sphere)
                    + w0_loop*np.ones(ntrain) - ytrain)**2)
    Xtest_new = ((Xtest.T - Xtrain_mean)/Xtrain_std).T
    MSE_test[i] = 1/ntest*np.sum((w_loop.dot(Xtest_new)
                + w0_loop*np.ones(ntest) - ytest)**2)


# reg_const that mimimizes test error
reg_const_min_error_train = reg_const[(np.argmin(MSE_train))]
print('regularization cosntant corresponding to minimum train MSE: {:.4f}'.format(reg_const_min_error_train))

# reg_const that mimimizes test error
reg_const_min_error_test = reg_const[(np.argmin(MSE_test))]
print('regularization cosntant corresponding to minimum test MSE: {:.4f}'.format(reg_const_min_error_test))
print('minimum test MSE: {:.4f}'.format(np.min(MSE_test)))

# plots
fig, axs = plt.subplots(2, 2, figsize = (10,6))

axs[0, 0].plot(reg_const,MSE_train,color = 'blue')
```

```
axs[0, 0].set_title('MSE train vs $\lambda$',fontsize = 14)
axs[0, 0].set_ylabel('MSE train',fontsize = 12)
axs[0, 0].set_xlabel('regularization constant, $\lambda \in [1,10^2]$',fontsize = 12)

axs[1, 0].plot(reg_const,MSE_train, color = 'blue')
axs[1, 0].plot(reg_const,MSE_test, color = 'red')
axs[1, 0].set_title('train, test MSE vs $\lambda$',fontsize = 14)
axs[1, 0].set_xlabel('regularization constant, $\lambda \in [1,10^2]$',fontsize = 12)
axs[1, 0].set_ylabel('MSE',fontsize = 12)
axs[1, 0].sharex(axs[0, 0])

axs[0, 1].axis('off')

axs[1, 1].plot(reg_const,MSE_test, color = 'red')
axs[1, 1].set_title("MSE test vs $\lambda$", fontsize = 14)
axs[1, 1].set_xlabel('regularization constant, $\lambda \in [1,10^2]$',fontsize = 12)
axs[1, 1].set_ylabel('MSE test',fontsize = 12)
fig.tight_layout()
plt.savefig('prob1d.png')
```

```
regularization cosntant corresponding to minimum train MSE: 1.0000
regularization cosntant corresponding to minimum test MSE: 87.5628
minimum test MSE: 750.7789
```



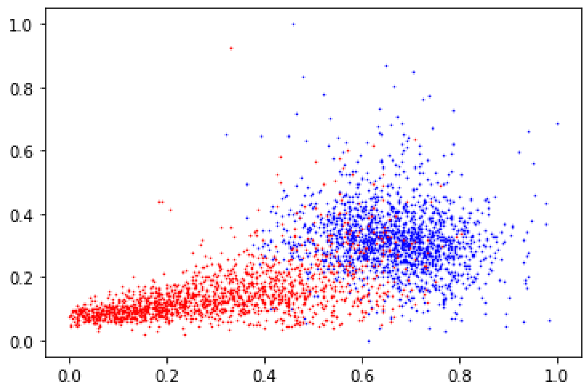
Problem 3

In [19]:
```
%matplotlib inline

# load relevant packages
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(0)
```

In [20]:
```
# load data
x = np.load("hw3_pulsars/pulsar_features.npy")
y = np.load("hw3_pulsars/pulsar_labels.npy")

nfeatures = x.shape[0]
nx = x.shape[1]

print(x.shape)
print(y.shape)
```

```
(2, 3278)
(1, 3278)
```

In [21]:
```
# plot data
negInd = y == -1
posInd = y == 1
plt.scatter(x[0, negInd[0, :]], x[1, negInd[0, :]], color='b', s=0.3)
plt.scatter(x[0, posInd[0, :]], x[1, posInd[0, :]], color='r', s=0.3)
plt.figure(1)
plt.show()
```



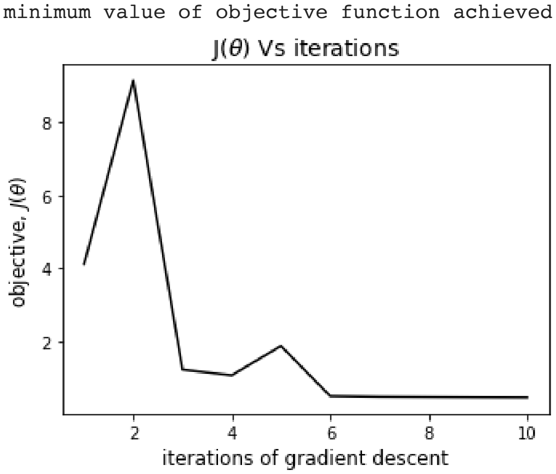
In [22]:
```
def objective_Ji(xi, yi, theta, A, reg_const):
    cond = 1-yi*theta.T.dot(xi)
    if cond < 0:
        return reg_const/(2*nx)*theta.T.dot(A).dot(theta)
    else:
        return cond/nx + reg_const/(2*nx)*theta.T.dot(A).dot(theta)


def subgradient_Ji(xi, yi, theta, A, reg_const):
    cond = 1-yi*theta.T.dot(xi)
    if cond < 0:
        return reg_const/nx*A.dot(theta)
    else:
        return -yi*xi/nx + reg_const/nx*A.dot(theta)


def gradient_descent(x, y, niter, reg_const):
    x_aug = np.vstack([np.ones(nx), x])  # augment x
    theta0 = np.zeros(nfeatures+1)        # construct theta0 = [b w^T]^T
    A = np.vstack([np.zeros(nfeatures+1), np.hstack([np.zeros([nfeatures,1]), np.eye(nfeatures)])])

    theta_new = theta0
    J = np.zeros(niter)

    for j in range(0,niter):
        grad = np.zeros(nfeatures+1)
        for i in range(nx):
            grad += subgradient_Ji(x_aug[:,i], y[0,i], theta_new, A, reg_const)
        theta_new -= 100/(j+1)*grad
```

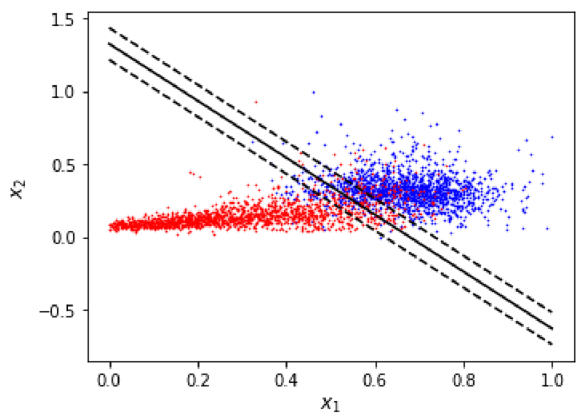
```python
            J[j] = np.sum([objective_Ji(x_aug[:,i], y[0,i], theta_new, A, reg_const) for i in range(nx)])
    return theta_new, J

def stochastic_gradient_descent(x, y, niter, reg_const):
    x_aug = np.vstack([np.ones(nx), x])  # augment x
    theta0 = np.zeros(nfeatures+1)       # construct theta0 = [b w^T]^T
    A = np.vstack([np.zeros(nfeatures+1), np.hstack([np.zeros([nfeatures,1]), np.eye(nfeatures)])])

    theta_new = theta0
    J = np.zeros(niter)
    for j in range(niter):
        randomized = np.random.permutation(nx)
        grad = np.zeros(nfeatures+1)
        for i in randomized:
            grad = subgradient_Ji(x_aug[:,i], y[0,i], theta_new, A, reg_const)
            theta_new -= 100/(j+1)*grad
        J[j] = np.sum([objective_Ji(x_aug[:,i], y[0,i], theta_new, A, reg_const) for i in range(nx)])
    return theta_new, J
```

In [23]:

```python
## part (b)
# problem setup

reg_const = 0.001  # regularization const
niter = 10         # number of iterations
theta, J = gradient_descent(x, y, niter, reg_const)  # gradient-descent
print('gradient-descent results')
print('hyperplane parametrs [b w^T]^T: {}'.format(theta))
print('toal margin: {}'.format(np.abs(2/theta[0])))
print('minimum value of objective function achieved: {}'.format(np.min(J)) )

# line equation to plot
line = lambda x, label: 1/theta[2]*(- theta[0]+label - theta[1]*x)

# plot
fig, axs = plt.subplots(1, 2, figsize = (12,4))
iternum = np.arange(1,niter+1)
axs[0].plot(iternum,J, color = 'k')
axs[0].set_xlabel('iterations of gradient descent', fontsize = 12)
axs[0].set_ylabel(r'objective, $J(\theta)$', fontsize = 12)
axs[0].set_title(r'J($\theta$) Vs iterations', fontsize = 14)

# plot data
negInd = y == -1
posInd = y == 1
points = np.linspace(np.min(x[0,:]), np.max(x[0,:]), 100)
axs[1].scatter(x[0, negInd[0, :]], x[1, negInd[0, :]], color='b', s=0.3)
axs[1].scatter(x[0, posInd[0, :]], x[1, posInd[0, :]], color='r', s=0.3)
axs[1].plot(points,line(points,0), color = 'k')
axs[1].plot(points,line(points,1), 'k--')
axs[1].plot(points,line(points,-1), 'k--')
axs[1].set_xlabel(r'$x_{1}$', fontsize = 12)
axs[1].set_ylabel(r'$x_{2}$', fontsize = 12)
plt.savefig('prob3b.png')
```

```
gradient-descent results
hyperplane parametrs [b w^T]^T: [ 12.0680196   -17.81627138  -9.11707611]
toal margin: 0.16572727473352247
minimum value of objective function achieved: 0.44988413706113406
```



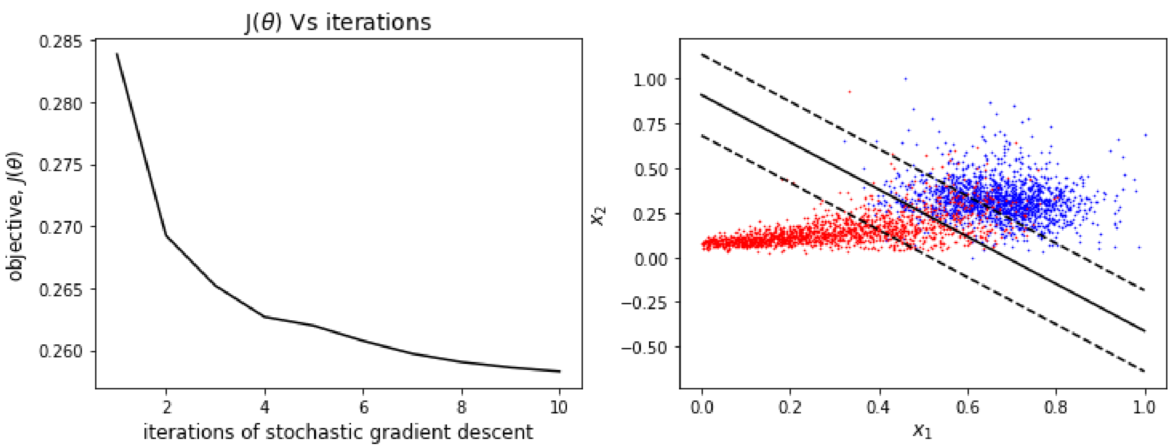


In [24]:

```python
# part (c)
# problem setup
```

```
reg_const = 0.001  # regularization const
niter = 10
theta, J = stochastic_gradient_descent(x, y, niter, reg_const)
print('stochastic-gradient-descent results')
print('hyperplane parametrs [b w^T]^T: {}'.format(theta))
print('toal margin: {}'.format(np.abs(2/theta[0])))
print('minimum value of objective function achieved: {}'.format(np.min(J)) )

# line equation to plot
line = lambda x, label: 1/theta[2]*(- theta[0]+label - theta[1]*x)

# plot
fig, axs = plt.subplots(1, 2, figsize = (12,4))
iternum = np.arange(1,niter+1)
axs[0].plot(iternum,J, color = 'k')
axs[0].set_xlabel('iterations of stochastic gradient descent', fontsize = 12)
axs[0].set_ylabel(r'objective, $J(\theta)$', fontsize = 12)
axs[0].set_title(r'J($\theta$) Vs iterations', fontsize = 14)

# plot data
# plot data
negInd = y == -1
posInd = y == 1
points = np.linspace(np.min(x[0,:]), np.max(x[0,:]), 100)
axs[1].scatter(x[0, negInd[0, :]], x[1, negInd[0, :]], color='b', s=0.3)
axs[1].scatter(x[0, posInd[0, :]], x[1, posInd[0, :]], color='r', s=0.3)
axs[1].plot(points,line(points,0), color = 'k')
axs[1].plot(points,line(points,1), 'k--')
axs[1].plot(points,line(points,-1), 'k--')
axs[1].set_xlabel(r'$x_{1}$', fontsize = 12)
axs[1].set_ylabel(r'$x_{2}$', fontsize = 12)
plt.savefig('prob3ci.png')
```

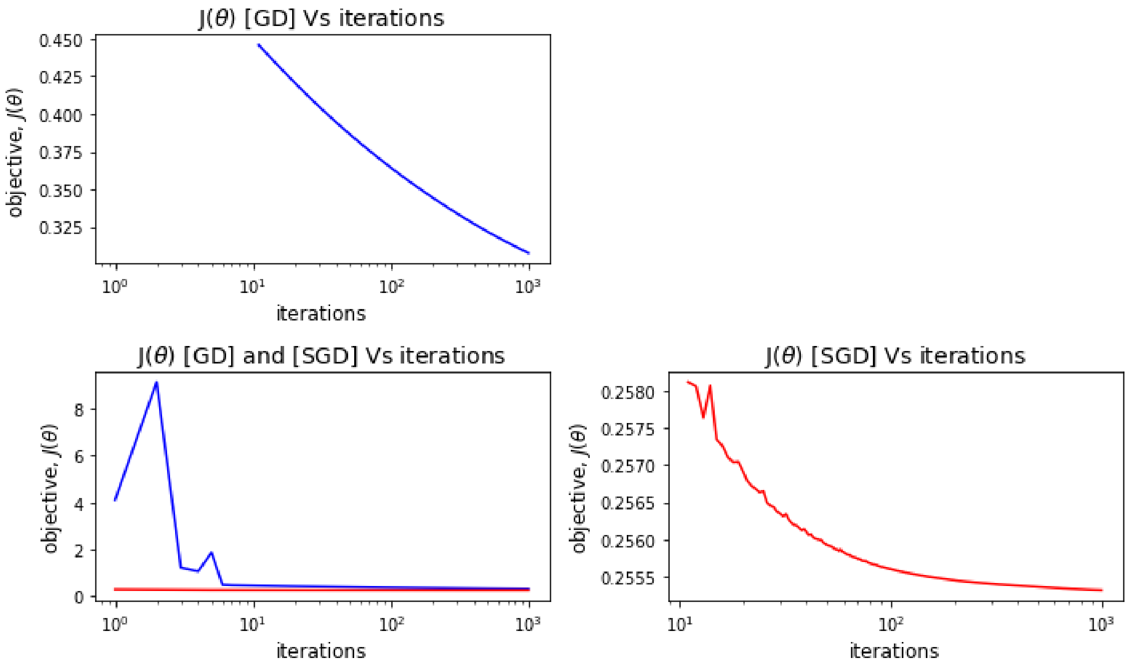
```
stochastic-gradient-descent results
hyperplane parametrs [b w^T]^T: [ 4.00515219 -5.82463117 -4.41417027]
toal margin: 0.4993568043331084
minimum value of objective function achieved: 0.2582782419707573
```



In [25]:

```
# examining the convergernce rate of the methods
reg_const = 0.001  # regularization const
niter = 1000        # running for 100 iterations

theta_gd, J_gd = gradient_descent(x, y, niter, reg_const)
theta_sgd, J_sgd = stochastic_gradient_descent(x, y, niter, reg_const)

iternum = np.arange(1,niter+1)

# plots
fig, axs = plt.subplots(2, 2, figsize = (10,6))

axs[0, 0].semilogx(iternum[10:], J_gd[10:], color = 'blue')
axs[0, 0].set_xlabel('iterations',fontsize = 12)
axs[0, 0].set_ylabel(r'objective, $J(\theta)$',fontsize = 12)
axs[0, 0].set_title(r'J($\theta$) [GD] Vs iterations',fontsize = 14)

axs[1, 0].semilogx(iternum, J_gd, color = 'blue')
axs[1, 0].semilogx(iternum, J_sgd, color = 'red')
axs[1, 0].set_title(r'J($\theta$) [GD] and [SGD] Vs iterations',fontsize = 14)
axs[1, 0].set_xlabel('iterations',fontsize = 12)
axs[1, 0].set_ylabel(r'objective, $J(\theta)$',fontsize = 12)
axs[1, 0].sharex(axs[0, 0])

axs[0, 1].axis('off')
```

```
axs[1, 1].semilogx(iternum[10:], J_sgd[10:], color = 'red')
axs[1, 1].set_title(r'J($\theta$) [SGD] Vs iterations',fontsize = 14)
axs[1, 1].set_xlabel('iterations',fontsize = 12)
axs[1, 1].set_ylabel(r'objective, $J(\theta)$',fontsize = 12)
fig.tight_layout()
plt.savefig('prob3cii.png')
```



Problem 4

In [1]:
```
%matplotlib inline

# load relevant packages
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
```

In [2]:
```
# load test and train data
Xtrain = np.load("hw3_housing/housing_train_features.npy")
Xtest = np.load("hw3_housing/housing_test_features.npy")
ytrain = np.load("hw3_housing/housing_train_labels.npy")
ytest = np.load("hw3_housing/housing_test_labels.npy")
```

In [3]:
```
# define features and dimensions
nfeatures = Xtrain.shape[0]
ntrain = Xtrain.shape[1]
ntest = Xtest.shape[1]

# print
print('nfeatures: {} \nntrain: {} \nntest: {}\n'.format(nfeatures,ntrain,ntest))

# computation of statistics of the data
Xtrain_mean = np.mean(Xtrain,axis = 1)
Xtrain_std = np.std(Xtrain, axis = 1)

# sphere the data
Xtrain_sphere = ((Xtrain.T - Xtrain_mean)/Xtrain_std).T
```

```
nfeatures: 58
ntrain: 2000
ntest: 925
```

In [4]:
```
# define functions

def sq_error(x, y, w0, w, reg_const, ntrain):
     return 1/ntrain*np.sum([(y[j] - (w0+x[:,j].dot(w.T)))**2 for j in range(ntrain)])


def coordinate_descent(x, y, reg_const, cycles, ntrain, nfeatures):
    w0 = np.sum(y)/ntrain      # bias term

    w_history = np.zeros([nfeatures,cycles*nfeatures+1])
    w_new = np.ones(nfeatures) # set initial weights to ones
    w_history[:,0] = w_new

    sq_error_history = np.zeros(cycles*nfeatures+1)
    sq_error_history[0] = sq_error(x, y, w0, w_new, reg_const, ntrain)

    for k in range(cycles):
        for i in range(nfeatures):
            # compute the constants ci and ai
            ci = 2*np.sum([x[i,j]*( y[j] - (w0+x[:,j].dot(w_new.T)-x[i,j]*w_new[i]) ) for j in range(ntrain)])
            ai = 2*np.sum(x[i,:]**2)

            # update w
            w_new[i] = ci/(ai+2*reg_const)

            # store the histories
            w_history[:,k*nfeatures+i+1] = w_new
            sq_error_history[k*nfeatures+i+1] = sq_error(x, y, w0, w_new, reg_const, ntrain)
    return w0, w_new, w_history, sq_error_history
```

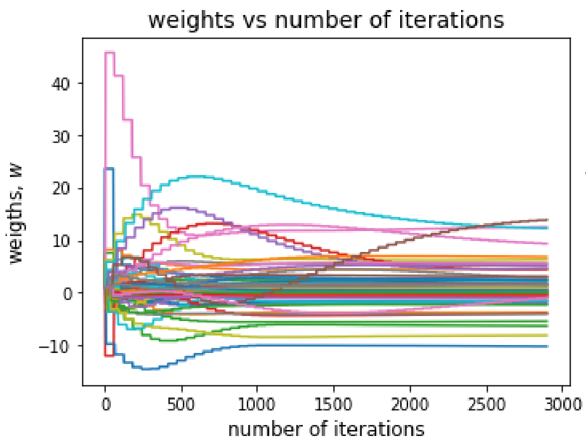
In [8]:
```
## part (d)
# problem setup

reg_const = 100
cycles = 50
w0, w, w_history, sq_error_history = coordinate_descent(Xtrain_sphere, ytrain, reg_const, cycles, ntrain, nfeatures)
```

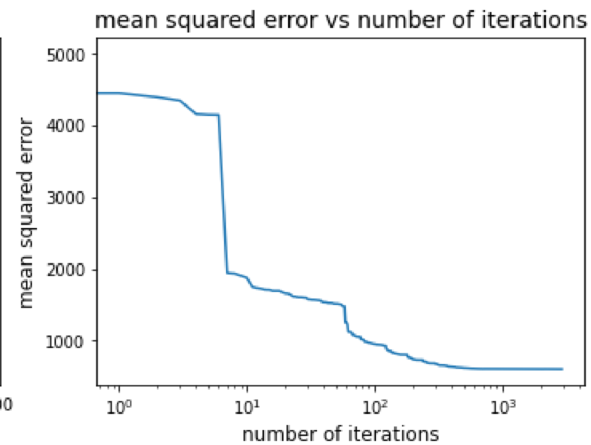
```python
# transform xtest
Xtest_new = ((Xtest.T - Xtrain_mean)/Xtrain_std).T
final_mse_test = 1/ntest*np.sum([(ytest[j] - (w0+Xtest_new[:,j].dot(w.T)))**2 for j in range(ntest)])
print('Final test MSE: {}'.format(final_mse_test))
```

```
Final test MSE: 751.1933845652543
```

In [7]:

```python
# plots
iterations = np.arange(nfeatures*cycles+1)

fig, axs = plt.subplots(1, 2, figsize = (12,4))
axs[0].plot(iterations,w_history.T)
axs[0].set_xlabel('number of iterations', fontsize=12)
axs[0].set_ylabel('weigths, $w$', fontsize=12)
axs[0].set_title('weights vs number of iterations',fontsize=14)

axs[1].semilogx(iterations,sq_error_history)
axs[1].set_xlabel('number of iterations', fontsize=12)
axs[1].set_ylabel('mean squared error', fontsize=12)
axs[1].set_title('mean squared error vs number of iterations',fontsize=14)

plt.savefig('prob4d.png')
```





Problem 5

In [1]:

```
%matplotlib inline

# load relevant packages
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
```

In [2]:

```
# load test and train data
Xtrain = np.load("hw3_housing/housing_train_features.npy")
Xtest = np.load("hw3_housing/housing_test_features.npy")
ytrain = np.load("hw3_housing/housing_train_labels.npy")
ytest = np.load("hw3_housing/housing_test_labels.npy")
```

In [3]:

```
# define features and dimensions
nfeatures = Xtrain.shape[0]
ntrain = Xtrain.shape[1]
ntest = Xtest.shape[1]

# print
print('nfeatures: {} \nntrain: {} \nntest: {}\n'.format(nfeatures,ntrain,ntest))

# computation of statistics of the data
Xtrain_mean = np.mean(Xtrain,axis = 1)
Xtrain_std = np.std(Xtrain, axis = 1)

# sphere the data
Xtrain_sphere = ((Xtrain.T - Xtrain_mean)/Xtrain_std).T
```

```
nfeatures: 58
ntrain: 2000
ntest: 925
```

In [4]:

```
# define functions

def sq_error(x, y, w0, w, reg_const, ntrain):
     return 1/ntrain*np.sum([(y[j] - (w0+x[:,j].dot(w.T)))**2 for j in range(ntrain)])


def coordinate_descent_lasso(x, y, reg_const, cycles, ntrain, nfeatures):
    w0 = np.sum(y)/ntrain      # bias term

    w_history = np.zeros([nfeatures,cycles*nfeatures+1])
    w_new = np.ones(nfeatures) # set initial weights to ones
    w_history[:,0] = w_new

    sq_error_history = np.zeros(cycles*nfeatures+1)
    sq_error_history[0] = sq_error(x, y, w0, w_new, reg_const, ntrain)

    for k in range(cycles):
        for i in range(nfeatures):
            # compute the constants ci and ai
            ci = 2*np.sum([x[i,j]*( y[j] - (w0+x[:,j].dot(w_new.T)-x[i,j]*w_new[i]) ) for j in range(ntrain)])
            ai = 2*np.sum(x[i,:]**2)

            # update w
            if ci > reg_const:
                w_new[i] = (ci-reg_const)/ai
            elif (ci <= reg_const or ci >= - reg_const):
                w_new[i] = 0
            else:
                w_new[i] = (ci+reg_const)/ai

            # store the histories
            w_history[:,k*nfeatures+i+1] = w_new
            sq_error_history[k*nfeatures+i+1] = sq_error(x, y, w0, w_new, reg_const, ntrain)
    return w0, w_new, w_history, sq_error_history
```

In [11]:

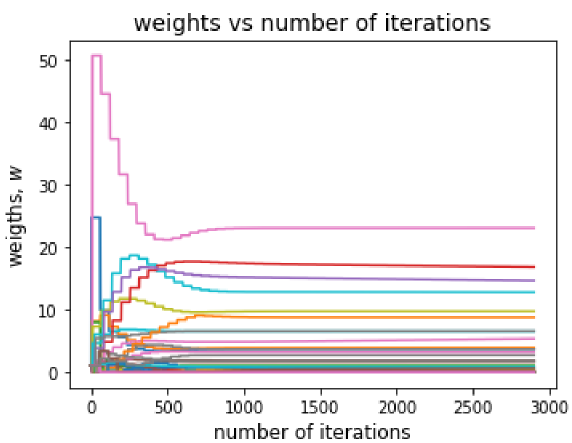
```
## part (b)
```

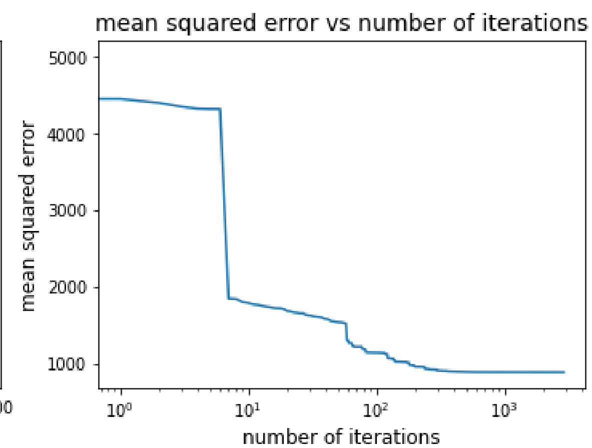
```python
# problem setup

reg_const = 100
cycles = 50
w0, w, w_history, sq_error_history = coordinate_descent_lasso(Xtrain_sphere, ytrain, reg_const, cycles, ntrain, nfea

# transform xtest
Xtest_new = ((Xtest.T - Xtrain_mean)/Xtrain_std).T
final_mse_test = 1/ntest*np.sum([(ytest[j] - (w0+Xtest_new[:,j].dot(w.T)))**2 for j in range(ntest)])
print('Final test MSE: {}'.format(final_mse_test))
```

```
Final test MSE: 981.0919212271457
```

In [6]:

```python
# plots
iterations = np.arange(nfeatures*cycles+1)


fig, axs = plt.subplots(1, 2, figsize = (12,4))
axs[0].plot(iterations,w_history.T)
axs[0].set_xlabel('number of iterations', fontsize=12)
axs[0].set_ylabel('weigths, $w$', fontsize=12)
axs[0].set_title('weights vs number of iterations',fontsize=14)

axs[1].semilogx(iterations,sq_error_history)
axs[1].set_xlabel('number of iterations', fontsize=12)
axs[1].set_ylabel('mean squared error', fontsize=12)
axs[1].set_title('mean squared error vs number of iterations',fontsize=14)

plt.savefig('prob5b.png')
```





In [8]:

```python
# find if some weights are set to 0
weights_0 = np.where(w == 0)[0]
print('number of 0 weights: {}\n'.format(len(weights_0)))
print(weights_0)
```

```
number of 0 weights: 34

[ 4 10 11 12 13 14 15 16 17 18 20 21 22 28 29 31 32 33 34 37 38 39 40 42
 43 44 45 46 50 51 52 53 54 56]
```