DS-GA 1003 HW2

Yiyan Chen

February 2018

1 Introduction

2 Ridge Regression

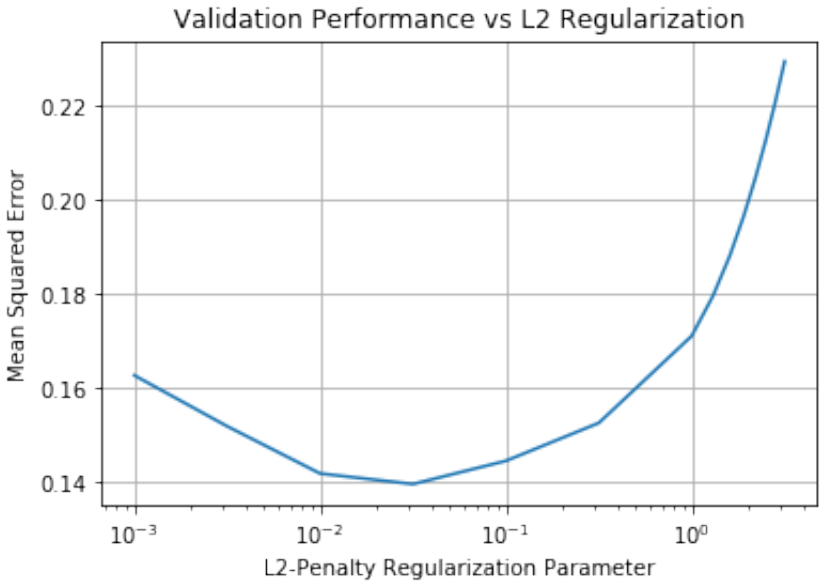
2.1

```
from ridge_regression import RidgeRegression, plot_prediction_functions,
    compare_parameter_vectors
import setup_problem
import numpy as np
import pandas as pd
from sklearn.model_selection import GridSearchCV, PredefinedSplit
from sklearn.metrics import mean_squared_error, make_scorer
import matplotlib.pyplot as plt
%matplotlib inline
lasso_data_fname = "lasso_data.pickle"
x_train, y_train, x_val, y_val, target_fn, coefs_true, featurize =
    setup_problem.load_problem(lasso_data_fname)
X_train = featurize(x_train)
X_val = featurize(x_val)
X_train_val = np.vstack((X_train, X_val))
y_train_val = np.concatenate((y_train, y_val))
val_fold = [-1]*len(X_train) + [0]*len(X_val)

param_grid = [{'l2reg':np.concatenate((10.**np.arange(-3, 1, 0.5), np.arange(1,3,0.3))) }]

ridge_regression_estimator = RidgeRegression()
grid = GridSearchCV(ridge_regression_estimator,
                        param_grid,
                        return_train_score=False,
                        cv = PredefinedSplit(test_fold=val_fold),
                        refit = True,
                        scoring = make_scorer(mean_squared_error,
```

```
                                                    greater_is_better = False))
grid.fit(X_train_val, y_train_val)
df = pd.DataFrame(grid.cv_results_)
df['mean_test_score'] = -df['mean_test_score']
cols_to_keep = ["param_l2reg", "mean_test_score"]
df_toshow = df[cols_to_keep].fillna('-')
df_toshow = df_toshow.sort_values(by=["param_l2reg"])
df_toshow
```

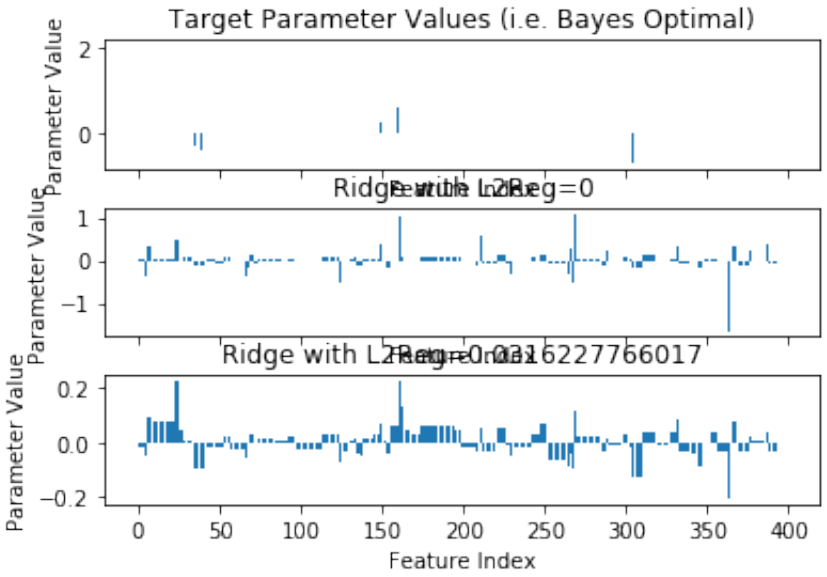
	param_l2reg	mean_test_score
0	0.001000	0.162705
1	0.003162	0.151900
2	0.010000	0.141887
3	0.031623	0.139648
4	0.100000	0.144566
5	0.316228	0.152607
6	1.000000	0.171068
8	1.000000	0.171068
9	1.300000	0.179521
10	1.600000	0.187993
11	1.900000	0.196361
12	2.200000	0.204553
13	2.500000	0.212530
14	2.800000	0.220271
7	3.162278	0.229295

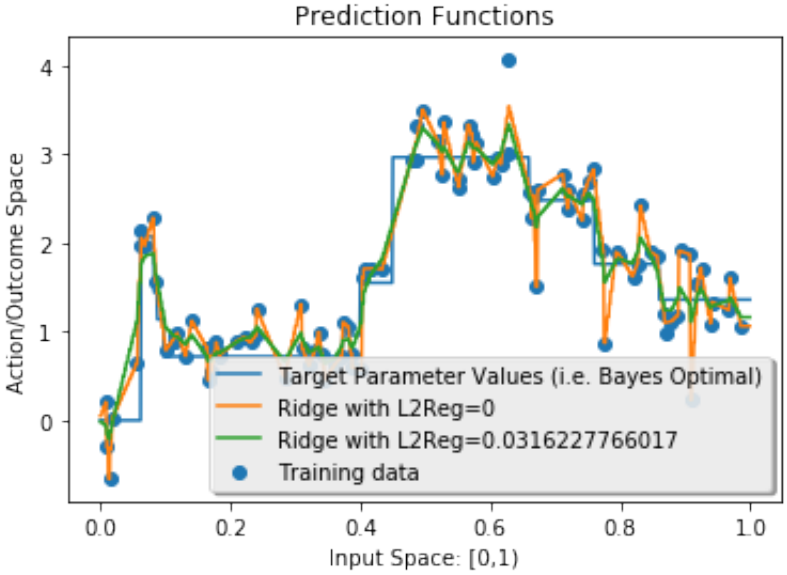
```
fig, ax = plt.subplots()
ax.semilogx(df_toshow["param_l2reg"], df_toshow["mean_test_score"])
ax.grid()
ax.set_title("Validation Performance vs L2 Regularization")
ax.set_xlabel("L2-Penalty Regularization Parameter")
ax.set_ylabel("Mean Squared Error")
fig.show()
```



2.2

```
para_final = df_toshow['param_l2reg'][np.argmin(df_toshow['mean_test_score'])]
pred_fns = []
x = np.sort(np.concatenate([np.arange(0,1,.001), x_train]))
name = "Target Parameter Values (i.e. Bayes Optimal)"
pred_fns.append({"name":name, "coefs":coefs_true, "preds": target_fn(x) })

l2regs = [0, para_final]
X = featurize(x)
for l2reg in l2regs:
    ridge_regression_estimator = RidgeRegression(l2reg=l2reg)
    ridge_regression_estimator.fit(X_train, y_train)
    name = "Ridge with L2Reg="+str(l2reg)
    pred_fns.append({"name":name,
                     "coefs":ridge_regression_estimator.w_,
                     "preds": ridge_regression_estimator.predict(X) })

plot_prediction_functions(x, pred_fns, x_train, y_train, legend_loc="best")
compare_parameter_vectors(pred_fns)
```






```
max_ls = []
min_ls = []
max_idx_ls = []
for i in range(len(pred_fns)):
    max_ls.append(max(pred_fns[i]['coefs']))
    min_ls.append(min(pred_fns[i]['coefs']))
    max_idx_ls.append(np.argmax(pred_fns[i]['coefs']))
```

```

df_pred = pd.DataFrame(np.concatenate((max_ls, min_ls, max_idx_ls)).reshape(3,3),
                       index = ['max', 'min', 'max index'],
                      columns = ['target', 'no regularization', 'best reguulatizaion'])
```

	target	no regularization	best regularization
max	2.069572	1.093516	0.222279
min	-0.717818	-1.644220	-0.204951
max index	25.000000	269.000000	24.000000

The Bayes optimal's pattern is the most sparse one. The range of its coefficients' values is the largest, [-0.717818, 2.069572]. The 25th coefficient has the largest weight. While the regularized least squared fit pattern's sparsity is between target and the regularized one. The range of its coefficients' values is [-1.644220, 1.093516]. The 269th coefficient has the largest weight. Compared to the other models, the regularized model have the most non-zero coefficients, with the range of which is [-0.204951, 0.222279]. The largest coefficient weight happens at the 24th.

2.3

```
any(pred_fns[2]['coefs']==0)
```

The chosen λ is 0.0316, and the coefficients are mapped into the plot showed in last question. None of the coefficients is zero.

```
from sklearn.metrics import confusion_matrix
threshold = [10**-6, 10**-3, 10**-2, 10**-1]
pred_fns[0]['coefs'][abs(pred_fns[0]['coefs']) > 0] = 1
target_coef = pred_fns[0]['coefs']
reg_coef = pred_fns[2]['coefs']
for i in threshold:
    copy_val = np.copy(reg_coef)
    copy_val[abs(reg_coef) >= i] = 1
    copy_val[abs(reg_coef) < i] = 0
    print(confusion_matrix(target_coef, copy_val))
```

The confusion matrix when threshold, $\epsilon = 10^{-6}$:

$$\begin{bmatrix} 5 & 385 \\ 0 & 10 \end{bmatrix}$$

The confusion matrix when $\epsilon = 10^{-3}$:

$$\begin{bmatrix} 9 & 381 \\ 0 & 10 \end{bmatrix}$$

The confusion matrix when $\epsilon = 10^{-2}$:

$$\begin{bmatrix} 61 & 329 \\ 0 & 10 \end{bmatrix}$$

The confusion matrix when $\epsilon = 10^{-1}$:

$$\begin{bmatrix} 376 & 14 \\ 7 & 3 \end{bmatrix}$$

3 Coordinate Descent for Lasso

3.1 Experiments with the Shooting Algorithm

3.1.1

$$a_j = 2X_{.j}^T X_{.j}$$

$$c_j = 2(X_{.j}^T y - X_{.j}^T X w + w_j X_{.j}^T X_{.j})$$

3.1.2

```python
def soft(a, delta):
    if abs(a)- delta > 0:
        return np.sign(a)*(abs(a)-delta)
    else:
        return 0

def objective_function(X, y, w, lambd):
    y = y.reshape(-1)
    return np.sum((y-np.dot(X, w))**2) + lambd*sum(abs(i) for i in w)

def shooting_alg(X, y, w, lambd, iteration = 1000, tolerance = 10**-6):
    D = X.shape[1]
    criteria = 1
    count = 1
    while criteria > tolerance or count < iteration:
        obj_prev = objective_function(X, y, w, lambd)
        for j in range(D):
            aj = 2*np.dot(X[:,j], X[:,j])
            cj = 2*(np.dot(X[:,j],y) - np.dot(np.dot(X[:,j], X), w)
                    + w[j]*np.dot(X[:,j],X[:,j]))
            wj = soft(cj/aj, lambd/aj)
            w[j] = wj
        criteria = obj_prev - objective_function(X,y,w, lambd)
        count += 1
    return w
w = np.zeros(X.shape[1])
w_cyc = shooting_alg(X_train, y_train, w, 1)
```

```python
from random import sample
def shooting_alg_random(X, y, w, lambd, iteration = 1000, tolerance = 10**-6):
    D = X.shape[1]
    criteria = 1
    count = 1
```

```
    while criteria > tolerance or count < iteration:
        obj_prev = objective_function(X, y, w, lambd)
        shuffled_list = sample(list(range(D)), D)
        for j in shuffled_list:
            aj = 2*np.dot(X[:,j], X[:,j])
            cj = 2*(np.dot(X[:,j],y) - np.dot(np.dot(X[:,j], X), w)
                    + w[j]*np.dot(X[:,j],X[:,j]))
            wj = soft(cj/aj, lambd/aj)
            w[j] = wj
        criteria = obj_prev - objective_function(X,y,w, lambd)
        count += 1
    return w

w_random = shooting_alg_random(X_train, y_train, w, 1)

def RMSE(X, y, w):
    return np.sum((y-np.dot(X, w))**2)/X.shape[0]

print(RMSE(X_val, y_val, w_cyc))
print(RMSE(X_val, y_val, w_random))
```

The validation error for cyclic coordinate descent is 0.167818772197, while that of randomized coordinate descent is 0.167818825676. Those two are very much similar.

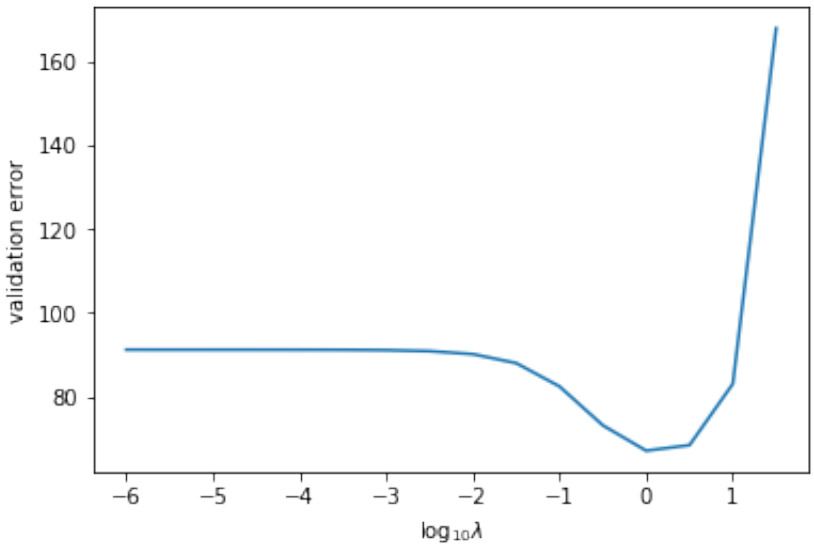
```
lambda_val = 1
w_ridge = np.dot(np.dot(np.linalg.inv(np.dot(X_train.transpose(), X_train)
                                      + lambda_val*np.identity(X_train.shape[1])),
                        X_train.transpose()), y_train)
w_ridge_cyc = shooting_alg(X_train, y_train, w_ridge, 1)
RMSE(X_val, y_val, w_ridge_cyc)
```

The validation error for cyclic coordinate descent when using ridge regression solution suggested by Murphy is 0.12643800274653294, which are the smallest among all.

3.1.3

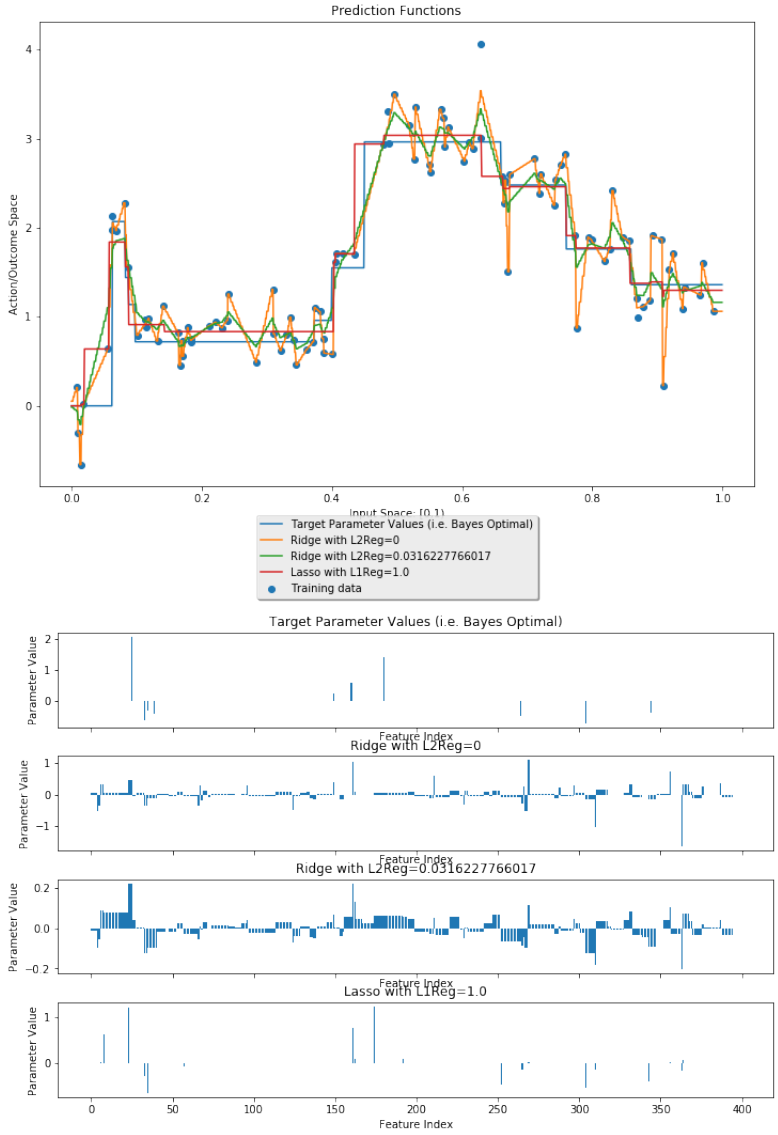
```
lambda_val_lst = np.arange(-6, 2, 0.5, dtype=float)
val_error = []
for val in lambda_val_lst:
    w_ridge_cyc_val = shooting_alg(X_train, y_train, w_ridge, 10**val)
    val_error.append(400*RMSE(X_val, y_val, w_ridge_cyc_val))
plt.plot(lambda_val_lst, val_error)
```

```
plt.xlabel('$\log_{10}\lambda$')
plt.ylabel("validation error")
plt.show()
```



	lambda value (10^)	validation error
0	-6.0	91.220809
1	-5.5	91.214366
2	-5.0	91.213619
3	-4.5	91.211285
4	-4.0	91.203904
5	-3.5	91.180578
6	-3.0	91.106946
7	-2.5	90.875421
8	-2.0	90.155156
9	-1.5	88.022245
10	-1.0	82.452074
11	-0.5	73.176563
12	0.0	67.127535
13	0.5	68.439045
14	1.0	83.069391
15	1.5	167.876487

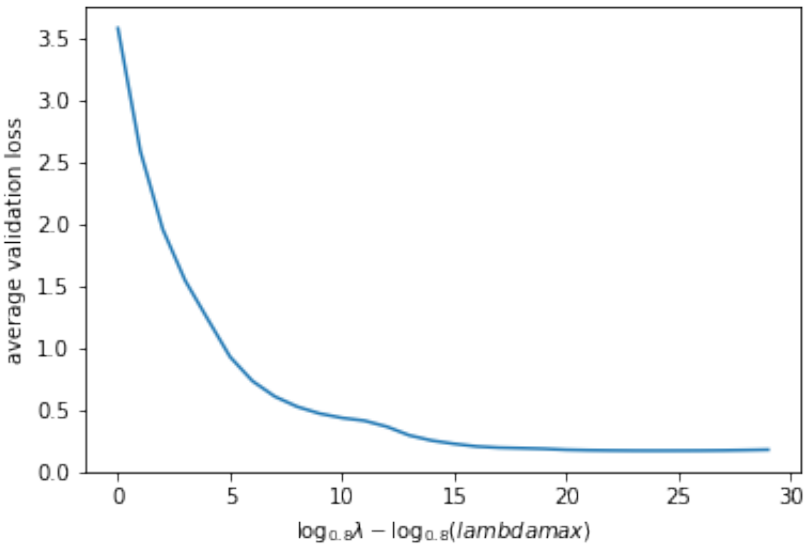
The λ that minimizes the square error on the validation set is 1 in my case. The plot of prediction functions and compare parameter vectors are showed as follow:



The lasso solutions' coefficients are much more sparse than the ridge ones. With less coefficients, lasso solutions are simpler and their performance are not bad compared to ridges as well. But the best model in terms of smaller validation error is the ridge regression with l2 regularization coefficient as 0.031623, and the smallest validation error is 0.139648.

3.1.4

```
lambda_max = 2*np.linalg.norm(np.dot(X_train.transpose(),y_train), ord = np.inf)
times_idx_list = np.arange(0,30, dtype=float)
validation_loss_homotopy = []
for idx in times_idx_list:
    lambda_homotopy = lambda_max*0.8**idx
    w = shooting_alg(X_train, y_train, w_ridge, lambda_homotopy)
    validation_loss_homotopy.append(RMSE(X_val, y_val, w))
```



4 Lasso Properties

4.1 Deriving λ_{Max}

4.1.1

$$\begin{aligned} J'(0;v) &= \lim_{h\to 0} \frac{J(0+hv)-J(0)}{h} \\ &= \lim_{h\to 0} \frac{J(hv)-J(0)}{h} \\ J(w) &= (Xw-y)^T(Xw-y)+\lambda||w||_1 \\ &= w^TX^TXw-2w^TX^Ty+y^Ty+\lambda||w||_1 \\ J(hv) &= (hv)^TX^TX(hv)-2(hv)^TX^Ty+y^Ty+\lambda||hv||_1 \\ &= h^2v^TX^TXv-2hv^TX^Ty+y^Ty+\lambda||hv||_1 \end{aligned}$$

$$\begin{aligned} J'(0;v) &= \lim_{h\to 0} \frac{h^2v^TX^TXv-2hv^TX^Ty+y^Ty+\lambda||hv||_1-y^Ty}{h} \\ &= \lim_{h\to 0} hv^TX^TXv-2v^TX^Ty+\lambda||v||_1 \\ &= -2v^TX^Ty+\lambda||v||_1 \end{aligned}$$

4.1.2

$$\begin{aligned} J'(0;v) &\geq 0 \\ -2v^TX^Ty+\lambda||v||_1 &\geq 0 \\ \lambda &\geq \frac{2v^TX^Ty}{||v||_1} \end{aligned}$$

The explicit expression for C is

$$\frac{2v^TX^Ty}{||v||_1}$$

4.1.3

$$\begin{aligned} \frac{2v^TX^Ty}{||v||_1} &= \frac{2\sum v_i(X^Ty)_i}{||v||_1} \leq \frac{2max(X^Ty)\sum v_i}{||v||_1} \\ &= \frac{2||X^Ty||_\infty \sum v_i}{||v||_1} \leq \frac{2||X^Ty||_\infty \sum |v_i|}{||v||_1} = 2||X^Ty||_\infty \end{aligned}$$

Therefore, the $max\frac{2v^TX^Ty}{||v||_1} = 2||X^Ty||_\infty$ If λ is larger than the greatest value, it means the $J'(0;v)$ is greater than zero, which indicates J increases after w = 0.

So the minimum happens when w = 0; If w = 0 is the minimizer, $J'(0; v) \geq 0$, then $\lambda \geq \frac{2v^T X^T y}{||v||_1}$ has to be satisfied whatever X, y, v values are given. $\lambda \geq max$. So $\lambda \geq 2||X^T y||_\infty$

4.1.4 OPTIONAL

To take the partial derivative on the function:

$$\begin{aligned} J_b(0,b) &= \lim_{h\to 0} \frac{((b+h)1-y)^T((b+h)1-y)-(b1-y)^T(b1-y)}{h} \\ &= \lim_{h\to 0} \frac{2bh1^T1+h^21^T1-2hy^T1}{h} \\ &= \lim_{h\to 0} 2b1^T1+h1^T1-2y^T1 \\ &= 2b1^T1-2y^T1 \end{aligned}$$

To minimize the partial derivative:

$$b = \frac{y^T1}{1^T1} = \frac{y_1+y_2+\dots y_n}{n} = \bar{y}$$

$$\begin{aligned} J(w) &= ||Xw+(\bar{y})-y||_2^2+\lambda||w||_1 \\ &= (Xw-(y-(\bar{y}))^T(Xw-(y-(\bar{y}))+\lambda||w|| \\ &= w^TX^Tw-2w^TX^T(y-\bar{y})+(y-\bar{y})^T(y-\bar{y})+\lambda||w|| \\ J(0;v) &= \lim_{h\to 0} \frac{J(hv)-J(0)}{h} \\ &= \lim_{h\to 0} \frac{h^2v^TX^TXv-2hv^TX^T(y-\bar{y})+(y-\bar{y})^T(y-\bar{y})+\lambda h||v||-(y-\bar{y})T^(y-\bar{y}}{h} \\ &= \lim_{h\to 0} \frac{h^2v^TX^TXv-2hv^TX^T(y-\bar{y})+\lambda h||v||}{h} \\ &= \lim_{h\to 0} hv^TX^TXv-2v^TX^T(y-\bar{y})+\lambda||v|| \\ &= -2v^TX^T(y-\bar{y})+\lambda||v|| \geq 0 \end{aligned}$$

$$\lambda \geq \frac{2v^TX^T(y-\bar{y})}{||v||}$$

According to previous problems:

$$\frac{2v^TX^T(y-\bar{y})}{||v||} \leq \frac{2||X^T(y-\bar{y})||_\infty \sum|v|}{\sum|v|} = 2||X^T(y-\bar{y})||_\infty$$

4.2 Feature Correlation

4.2.1

Suppose a and b have different signs: $a < 0, b > 0$ or $a > 0, b < 0$:

$$\begin{aligned} J(\hat{\theta}) &= ||x_1(a+b) + x_r r - y||_2^2 + \lambda|a| + \lambda|b| + \lambda||r||_1 \\ &= ||x_1(a+b) + x_r r - y||_2^2 + \lambda(|a| + |b|) + \lambda||r||_1 \end{aligned}$$

we set c+d = a+b:
Seen from this equation, we can use $c = \theta(a+b), d = (1-\theta)(a+b)$ to plug into the equation:

$$\begin{aligned} J(\theta^*) &= ||x_1(c+d) + x_r r - y||_2^2 + \lambda|c| + \lambda|d| + \lambda||r||_1 \\ &= ||x_1(a+b) + x_r r - y||_2^2 + \lambda\theta|a+b| + \lambda(1-\theta)|a+b| + \lambda||r||_1 \\ &= ||x_1(a+b) + x_r r - y||_2^2 + \lambda|a+b| + \lambda||r||_1 \end{aligned}$$

since $|a+b| \leq |a| + |b|$. $J(\hat{\theta})$ won't be the minimizer of $J(\theta)$, which is a contradiction. If $J(\hat{\theta})$ is a minimizer, we should ensure $|a+b| = |a| + |b|$. In order to make the equality hold, a and b have to be the same sign, or at least one of them be zero.

The $J(\theta^*)$ also needs to minimize, meaning $(c, d, r^T)^T$ is the minimizer: $J(\theta^*) = J(\hat{\theta})$.

Hence, $c + d = a + b, |c| + |d| = |a| + |b|$

4.2.2

$$\begin{aligned} J(\theta) &= ||X\theta - y||_2^2 + \lambda||\theta||_2^2 \\ &= ||x_1 a + x_2 b + x_r r - y||_2^2 + \lambda \sum \theta_i^2 \\ &= ||(a+b)x_1 + x_r r - y||_2^2 + \lambda a^2 + \lambda b^2 + \lambda||r||_2^2 \end{aligned}$$

to minimize $\lambda(a^2 + b^2)$:
since $a^2 + b^2 \geq 2ab$, the minimum of $a^2 + b^2$ happens when $a = b$
Hence, $a = b$ when minimizes the ridge regression objective function.