Machine Learning HW02

Ben Zhang, bz957

February 14, 2018

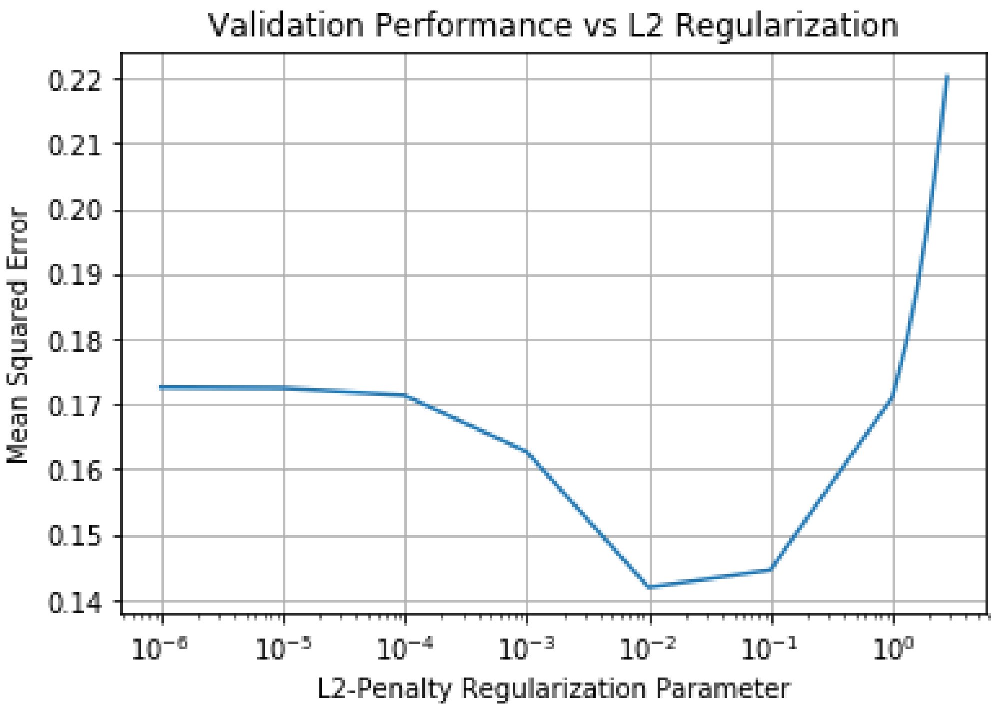
2 Ridge Regression

2.1

```
lasso_data_fname = "lasso_data.pickle"
x_train, y_train, x_val, y_val, target_fn,
                              coefs_true, featurize = load_problem(lasso_data_fname)

# Generate features
X_train = featurize(x_train)
X_val = featurize(x_val)
grid, results = do_grid_search_ridge(X_train, y_train, X_val, y_val)
print(results)

fig, ax = plt.subplots()
#     ax.loglog(results["param_l2reg"], results["mean_test_score"])
ax.semilogx(results["param_l2reg"], results["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")
plt.show()
```

$\Lambda = 10^{-2}$ **minimizes the empirical risk on the validation set**

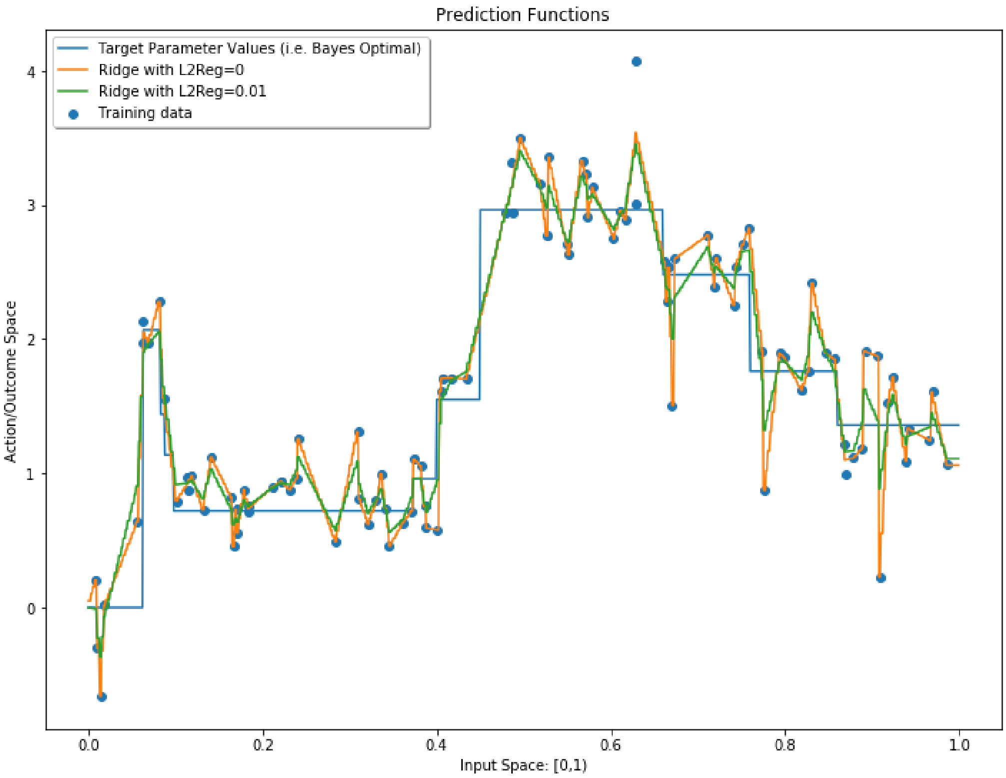
	param_l2reg	mean_test_score
0	0.000001	0.172579
1	0.000010	0.172464
2	0.000100	0.171345
3	0.001000	0.162705
4	0.010000	0.141887
5	0.100000	0.144566
6	1.000000	0.171068
7	1.300000	0.179521
8	1.600000	0.187993
9	1.900000	0.196361
10	2.200000	0.204553
11	2.500000	0.212530
12	2.800000	0.220271



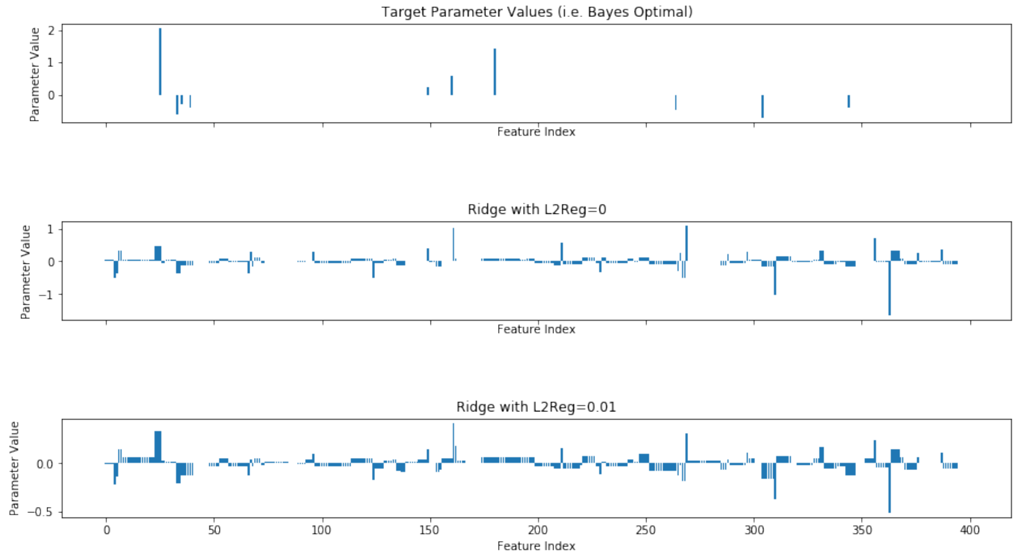
2.2

```
def plot_prediction_functions(x, pred_fns, x_train, y_train, legend_loc="best"):
    # Assumes pred_fns is a list of dicts, and each dict has a "name" key and a
    # "preds" key. The value corresponding to the "preds" key is an array of
    # predictions corresponding to the input vector x. x_train and y_train are
    # the input and output values for the training data
    fig, ax = plt.subplots(figsize = (12, 9))
    ax.set_xlabel('Input Space: [0,1)')
    ax.set_ylabel('Action/Outcome Space')
    ax.set_title("Prediction Functions")
    plt.scatter(x_train, y_train, label='Training data')
    for i in range(len(pred_fns)):
        ax.plot(x, pred_fns[i]["preds"], label=pred_fns[i]["name"])
    legend = ax.legend(loc=legend_loc, shadow=True)
    return fig

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, grid.best_params_['l2reg']]
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) })
```

```
f = plot_prediction_functions(x, pred_fns, x_train, y_train, legend_loc="best")
plt.show()
```




```
def compare_parameter_vectors(pred_fns):
    # Assumes pred_fns is a list of dicts, and each dict has a "name" key and a
    # "coefs" key
    fig, axs = plt.subplots(len(pred_fns),1, figsize = (15, 8),sharex=True)
    num_ftrs = len(pred_fns[0]["coefs"])
    for i in range(len(pred_fns)):
        title = pred_fns[i]["name"]
        coef_vals = pred_fns[i]["coefs"]
        axs[i].bar(range(num_ftrs), coef_vals)
        axs[i].set_xlabel('Feature Index')
        axs[i].set_ylabel('Parameter Value')
        axs[i].set_title(title)

    fig.subplots_adjust(hspace=1)
    return fig
f = compare_parameter_vectors(pred_fns)
plt.show()
```



The Target coefficients values are sparse and in the range $[-1, 2]$. Ringe Regression without regression makes coefficients less spare, shows ringe regression is not good for sparse dataset. With $Lambda = 0.01$, sparsity is almost the same as the ringe regression without regression, but the range of coefficients is reduced to $[-0.5, 0.5]$, coefficients are changed towards zero. As we can see on the plot, the coefficients which have the most weight on the ringe regression with and without regression are almost the same as the coefficients in the Target plot

2.3

```
pred_fns_best = []
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 = [grid.best_params_['l2reg']]
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_best.append({"name":name,
                          "coefs":ridge_regression_estimator.w_,
                          "preds": ridge_regression_estimator.predict(X) })

from sklearn.metrics import confusion_matrix
true_w = coefs_true.copy()
for i in range(true_w.shape[0]):
        if true_w[i] != 0:
            true_w[i] = 1
        else:
            true_w[i] = 0

elist = [1e-6,1e-3,1e-1]
for e in elist:
    pred_w = ridge_regression_estimator.w_.copy()
    for i in range(pred_w.shape[0]):
        if np.absolute(pred_w[i]) < e:
            pred_w[i] = 0
        else:
```

```python
            pred_w[i] = 1
print(confusion_matrix(true_w, pred_w))
```

Confusion matrix for $\varepsilon = 10^{-6}; 10^{-3}; 10^{-1}$

```
[[  5 385]
 [  0  10]]
[[  8 382]
 [  0  10]]
[[349  41]
 [  3   7]]
```

3.1 Experiments with the Shooting Algorithm

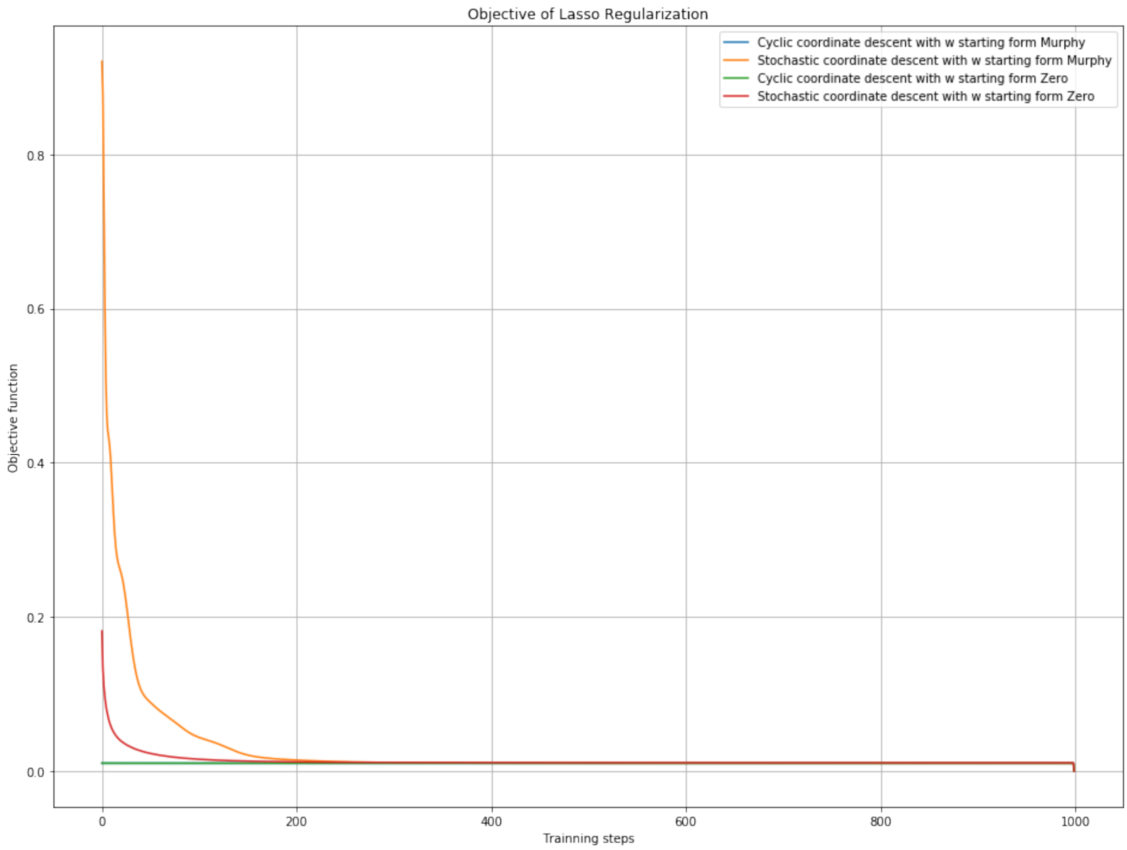
3.1.1

$a_j = 2 * X_{.j}^T * X_{.j}$
$c_j = 2 * X_{.j}^T * (y - W^T * X + W_j * X_{.j})$
$w_j = soft(c_j/a_j, \Lambda/a_j)$

3.1.2

```
def Lasso_obj(X, y, w, lambda_reg):
    residual = np.dot(X, w) - y
    empirical_risk = np.sum(residual**2)
    l1_norm = np.linalg.norm(w, ord=1)
    objective = empirical_risk + lambda_reg * l1_norm
    return objective

def soft(a,b):
    if a > 0:
        sign_a = 1
    else:
        sign_a = -1
    wnew = sign_a * max(np.abs(a)-b, 0)
    return wnew

def Lassodescent(X, y, w_choice, method, lambda_reg=0.01, num_iter=1000):
    (num_instances, num_features) = X.shape
    if w_choice == "Murphy":
        I = np.diag(np.ones(X.shape[1]))
        w = inv(np.dot(X.T,X)+ lambda_reg*I).dot(X.T).dot(y)
    else:
        w = np.zeros(num_features)

    w_next = np.zeros(num_features)
    w_hist = np.zeros((num_iter+1, num_features))  #Initialize w_hist
    w_hist[0] = w
    obj_hist = np.zeros(num_iter+1) #Initialize loss_hist
    objforplot_hist = np.zeros(num_iter+1) #Initialize loss_hist

    if method == "Cyclic":
        for i in range(num_iter):
            obj_hist[i] = Lasso_obj(X, y, w.T, lambda_reg)
            objforplot_hist[i] = obj_hist[i]/num_instances

            for j in range(num_features):

                a = (2*X[:,j].T).dot(X[:,j])
                c = (2*X[:,j]).dot(y - X.dot(w.T) + w[j]*X[:,j])
                if a==0:
                    w_next[j]=0
                else:
                    w_next[j] = soft(c/a,lambda_reg/a)
                w[j] = w_next[j]

            w_hist[i+1] = w

            if (obj_hist[i+1]-obj_hist[i])<1e-8:
                break
```

```
    else:
        l=list(range(num_features))
        for i in range(num_iter):
            obj_hist[i] = Lasso_obj(X, y, w.T, lambda_reg)
            objforplot_hist[i] = obj_hist[i]/num_instances

            np.random.shuffle(l)
            for j in l:   #stochastic
                a = (2*X[:,j].T).dot(X[:,j])
                c = (2*X[:,j]).dot(y - X.dot(w.T) + w[j]*X[:,j])
                if a==0:
                    w_next[j]=0
                else:
                    w_next[j] = soft(c/a,lambda_reg/a)
                w[j] = w_next[j]

            w_hist[i+1] = w


            if (obj_hist[i+1]-obj_hist[i])<1e-8:
                break

    return w_hist, objforplot_hist

Finding_Lasso(X_train,y_train)
```

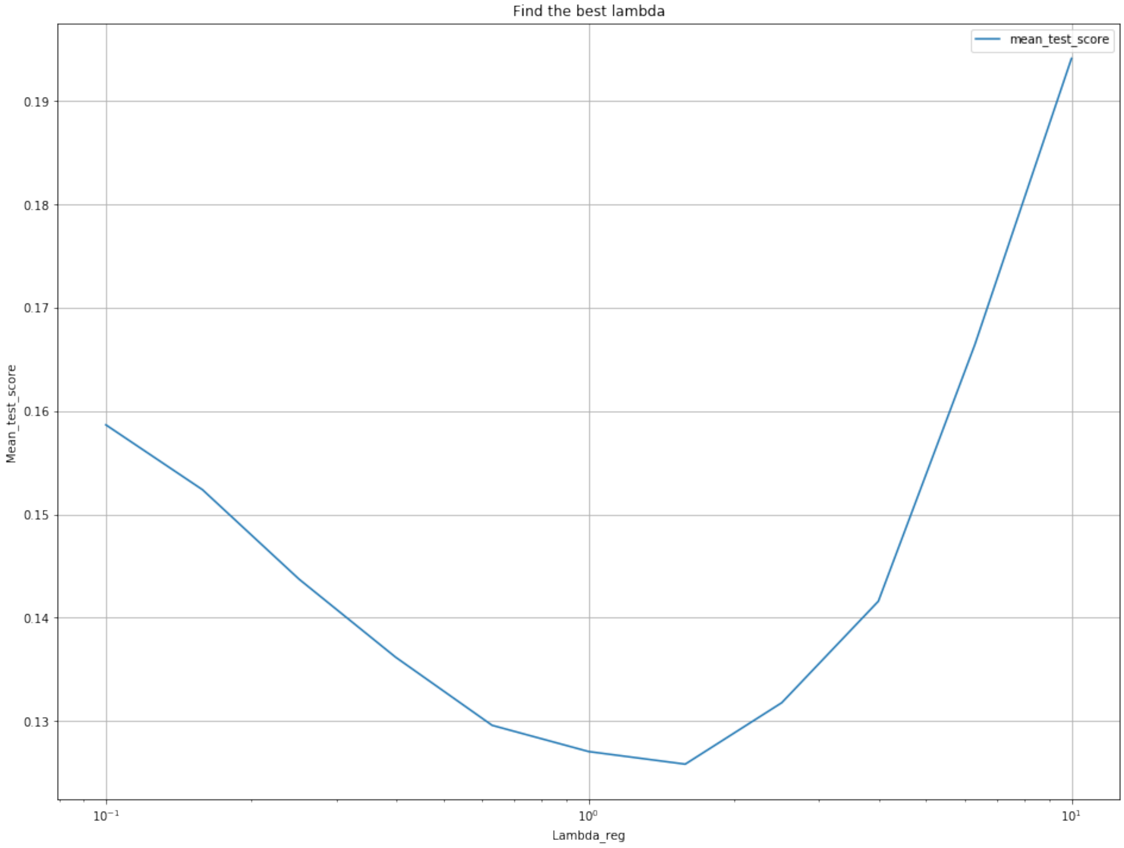


3.1.3

```
def Loss(X, y, w):
    residual = np.dot(X, w) - y
    empirical_risk = np.sum(residual**2)/X.shape[0]
    return empirical_risk

from numpy.linalg import inv
def SDL_Murphey(X, y, lambda_reg, num_iter=1000):
    (num_instances, num_features) = X.shape

    I = np.diag(np.ones(X.shape[1]))
    w = inv(np.dot(X.T,X)+ lambda_reg*I).dot(X.T).dot(y)
    w_next = np.zeros(num_features)

    l=list(range(num_features))
    for i in range(num_iter):

        np.random.shuffle(l)
        for j in l:  #stochastic
            a = (2*X[:,j].T).dot(X[:,j])
            c = (2*X[:,j]).dot(y - X.dot(w.T) + w[j]*X[:,j])
            if a==0:
                w_next[j]=0
            else:
                w_next[j] = soft(c/a,lambda_reg/a)
            w[j] = w_next[j]
#           if (obj_hist[i+1]-obj_hist[i])<1e-8:
#               break

    return w

def Finding_lambda(X_train,y_train,X_val,y_val,lambda_grid):

    Loss_hist = []
    for lambda_reg in lambda_grid:
        w = SDL_Murphey(X_train, y_train, lambda_reg, num_iter=1000)
        Loss_hist.append(Loss(X_val, y_val, w))

    return Loss_hist

lambda_grid = np.unique(10.**np.arange(-1,1.2,0.2))
Loss_hist = Finding_lambda(X_train,y_train, X_val,y_val, lambda_grid)
df = pd.DataFrame()
df['mean_test_score'] = Loss_hist
df['Lambda_list'] = lambda_grid
df

fig, ax = plt.subplots(figsize = (16, 12))
ax.semilogx(df["Lambda_list"], df["mean_test_score"])
ax.set_xlabel("Lambda_reg")
ax.set_ylabel("Mean_test_score")
ax.set_title("Find the best lambda")
legend = ax.legend(loc='best')
legend.FontSize = 8
plt.grid(True)
plt.show()
```

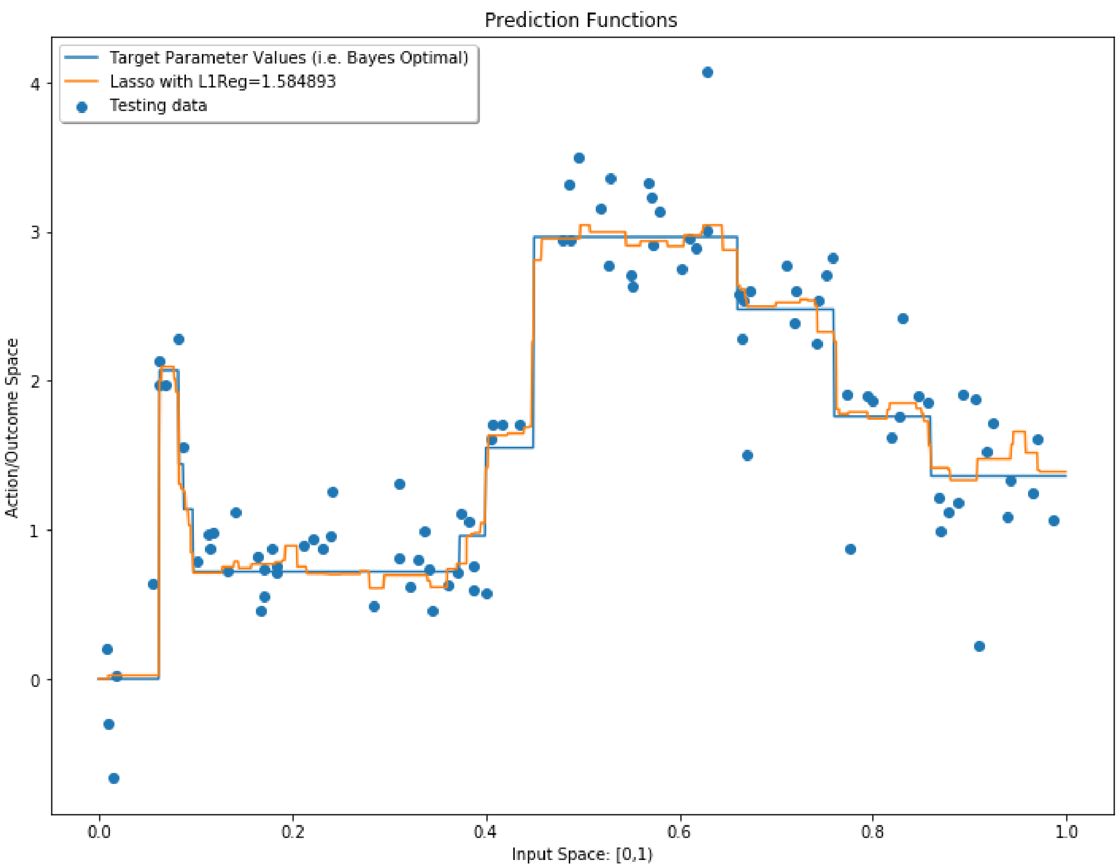
	mean_test_score	Lambda_list
0	0.158672	0.100000
1	0.152397	0.158489
2	0.143738	0.251189
3	0.136184	0.398107
4	0.129584	0.630957
5	0.127045	1.000000
6	0.125826	1.584893
7	0.131776	2.511886
8	0.141598	3.981072
9	0.166459	6.309573
10	0.194146	10.000000

$\lambda = 1.584893$ **has the best performance**



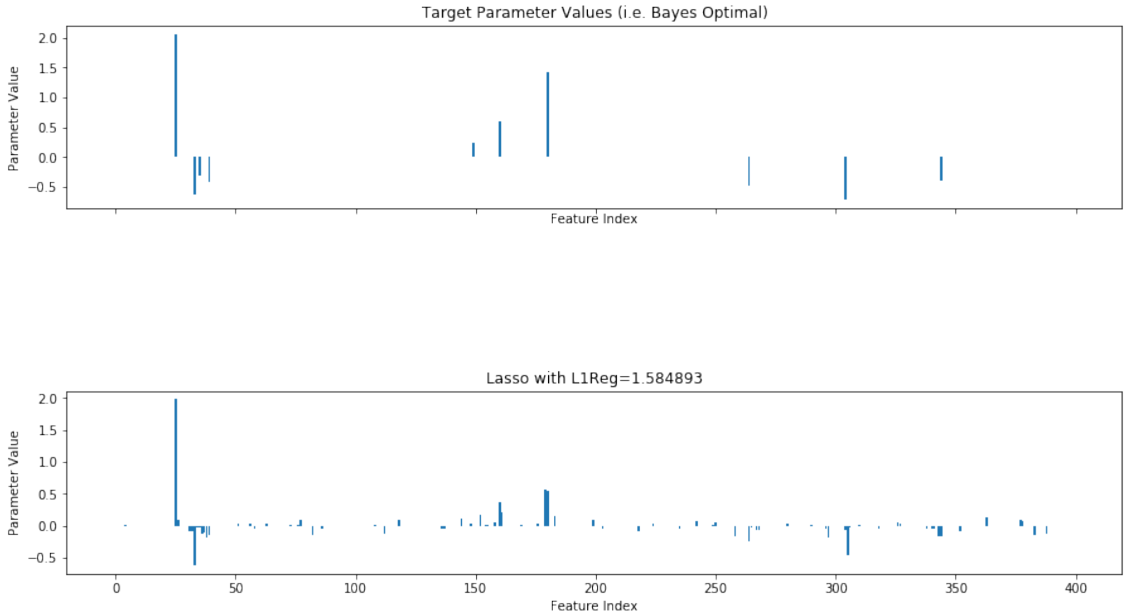

```
def plot_prediction_functions(x, pred_fns, X_val, y_val, legend_loc="best"):
    # Assumes pred_fns is a list of dicts, and each dict has a "name" key and a
    # "preds" key. The value corresponding to the "preds" key is an array of
    # predictions corresponding to the input vector x. x_train and y_train are
    # the input and output values for the training data
    fig, ax = plt.subplots(figsize = (12, 9))
    ax.set_xlabel('Input Space: [0,1)')
```

```
    ax.set_ylabel('Action/Outcome Space')
    ax.set_title("Prediction Functions")
    plt.scatter(X_val, y_val, label='Testing data')
    for i in range(len(pred_fns)):
        ax.plot(x, pred_fns[i]["preds"], label=pred_fns[i]["name"])
    legend = ax.legend(loc=legend_loc, shadow=True)
    return fig

pred_fns = []
x = np.sort(np.concatenate([np.arange(0,1,.001), x_val])) #
name = "Target Parameter Values (i.e. Bayes Optimal)"
pred_fns.append({"name":name, "coefs":coefs_true, "preds": target_fn(x) })

l1regs = [1.584893]
X = featurize(x)
for l1reg in l1regs:
    name = "Lasso with L1Reg="+str(l1reg)
    w = SDL_Murphey(X_val, y_val, l1reg)
    pred_fns.append({"name": name,
                     "coefs": w,
                     "preds": X.dot(w)})

f = plot_prediction_functions(x, pred_fns, x_train, y_train, legend_loc="best")
plt.show()
```




```
def compare_parameter_vectors(pred_fns):
    # Assumes pred_fns is a list of dicts, and each dict has a "name" key and a
    # "coefs" key
    fig, axs = plt.subplots(len(pred_fns),1, figsize = (15, 8),sharex=True)
    num_ftrs = len(pred_fns[0]["coefs"])
    for i in range(len(pred_fns)):
```

```
        title = pred_fns[i]["name"]
        coef_vals = pred_fns[i]["coefs"]
        axs[i].bar(range(num_ftrs), coef_vals)
        axs[i].set_xlabel('Feature Index')
        axs[i].set_ylabel('Parameter Value')
        axs[i].set_title(title)

    fig.subplots_adjust(hspace=1)
    return fig

f = compare_parameter_vectors(pred_fns)
plt.show()
```

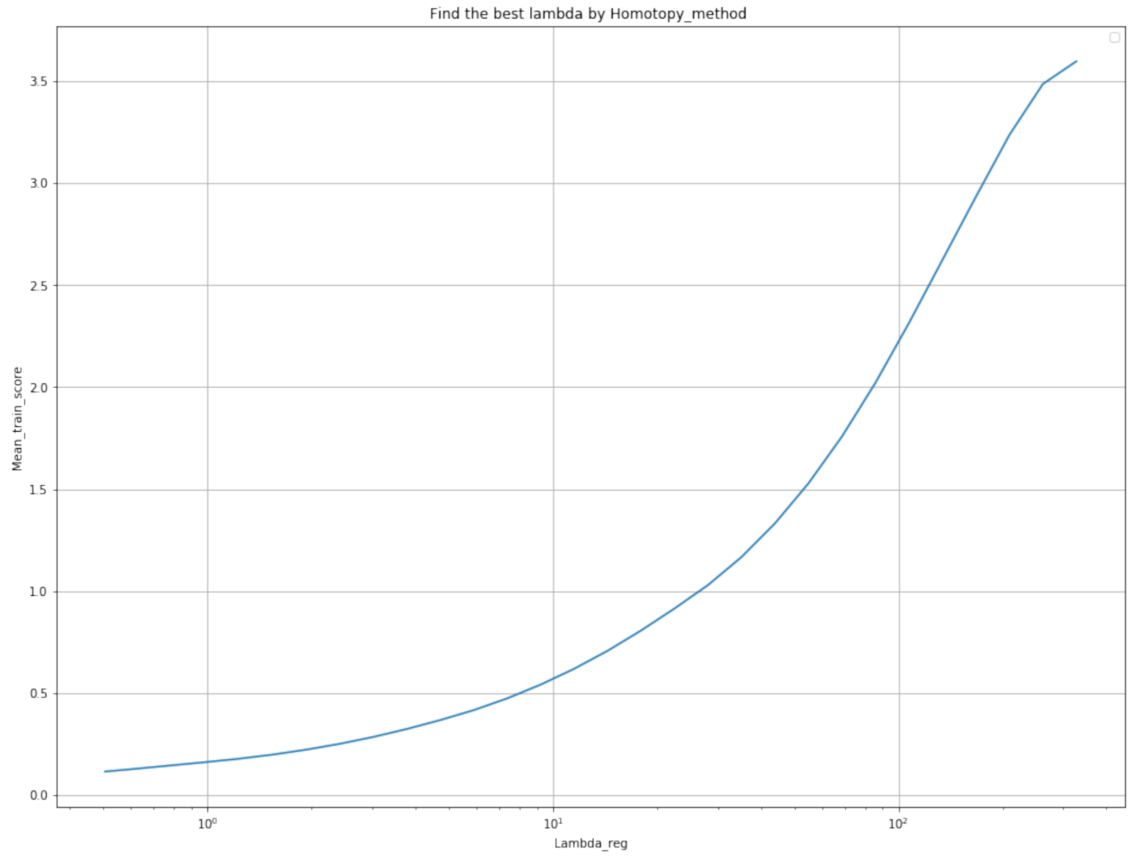


The Lasso with $L1Regression = 1.584893$, starting from the initial w calculated by Murphy method and stochastic descent is the best model.Its mean test score is 0.125826, which is very good. Its coefficients have almost the same sparsity and range as the true target function. Although is a little less spare than the true target function

3.1.4

```
def Homotopy_method(X, y, num_iter=1000):
    (num_instances, num_features) = X.shape

    from numpy.linalg import inv


    lambda_max = np.linalg.norm(2*X.T.dot(y), np.inf) #supernorm????
    I = np.diag(np.ones(X.shape[1]))
    w = inv(np.dot(X.T,X)+ lambda_max*I).dot(X.T).dot(y)
    w_next = np.zeros(num_features)

    objaver_hist = np.zeros(30)
    lambda_list = [lambda_max*(0.8**i) for i in range(30)]
    l=list(range(num_features))

    for l1 in range(30):
        lambda_reg = lambda_list[l1]
        for i in range(num_iter):
            np.random.shuffle(l)
```

```
            for j in l:  #stochastic
                a = (2*X[:,j].T).dot(X[:,j])
                c = (2*X[:,j]).dot(y - X.dot(w.T) + w[j]*X[:,j])
                if a==0:
                    w_next[j]=0
                else:
                    w_next[j] = soft(c/a,lambda_reg/a)
                w[j] = w_next[j]
#           if (obj_hist[i+1]-obj_hist[i])<1e-8:
#               break

        objaver_hist[l1] = Lasso_obj(X, y, w.T, lambda_reg)/num_instances

    return objaver_hist, lambda_list

objaver_hist, lambda_list = Homotopy_method(X_train, y_train)

fig, ax = plt.subplots(figsize = (16, 12))
ax.semilogx(lambda_list, objaver_hist)
ax.set_xlabel("Lambda_reg")
ax.set_ylabel("Mean_train_score")
ax.set_title("Find the best lambda by Homotopy_method ")
legend = ax.legend(loc='best')
legend.FontSize = 8
plt.grid(True)
plt.show()
```



3.2 [Optional] Deriving the Coordinate Minimizer for Lasso

3.2.1.

from the question we get

$$f(w_i) = \sum_{i=1}^{n} \Big[\sum_{k \neq j} (w_k x_{ik} - y_i) \Big]^2 + \lambda |w_i| + \lambda \sum_{k \neq j} |w_k|$$

$$\text{if } w_j > 0, \quad \frac{\partial f(w_j)}{\partial w_j} = 0 + \lambda + 0$$

$$\text{if } w_j < 0, \quad \frac{\partial f(w_j)}{\partial w_j} = 0 - \lambda + 0$$

3.2.2.

$$a_j = 2 \sum_{i=1}^{n} x_{ij}^2$$

$$c_j = 2 \sum_{i=1}^{n} x_{ij} y_i - \sum_{k \neq j} w_k x_{ik}$$

from 3.2.1

$$\Rightarrow \quad f(w_j) = \sum_{i=1}^{n} \Big(w_j x_{ij} + \sum_{k \neq j} (w_k x_{ik} - y_i) \Big)^2 + \lambda \Big(|w_j| + \sum_{k \neq j} |w_k| \Big)$$

$$f(w_j) = g + h$$

if $w_j \neq 0 \Rightarrow \begin{cases} \dfrac{\partial h}{\partial w_j} = \lambda \,\text{sign}(w_j) \\ \dfrac{\partial g}{\partial w_j} = w_j a_j - c_j \end{cases}$

$$\therefore \frac{\partial f}{\partial w_j} = w_j a_j - c_j + \lambda \,\text{sig}(w_j)$$

3.2.3.

$$w_j > 0 \Rightarrow w_j = \frac{1}{a_j}(c_j - \lambda)$$

$$w_j < 0 \Rightarrow w_j = \frac{1}{a_j}(c_j + \lambda)$$

$$a_j > 0 \Rightarrow \begin{array}{l} c_j < -\lambda, \quad w_j < 0 \\ c_j > \lambda, \quad w_j > 0 \end{array}$$

4.1 Deriving max

4.1.1

$$J'(0;v) = \lim_{h\downarrow 0} \frac{J(0+hv)-J(0)}{h}$$

$$= \lim_{h\downarrow 0} \frac{\|X\cdot hv - y\|_2^2 + \lambda\|hv\|_1 - \|-y\|_2^2 - \lambda\|0\|_1}{h}$$

$$= \lim_{h\downarrow 0} \frac{(X\cdot hv - y)^T(X\cdot hv - y) - y^T\cdot y + h\cdot\lambda\|v\|_1}{h}$$

$$= \lim_{h\downarrow 0} h\cdot v^T\cdot X^T\cdot X\cdot v - v^T\cdot X^T y - y^T\cdot X\cdot v + \lambda\|v\|_1$$

$$= \lambda\|v\|_1 - [(Xv)^T y + y^T(X\cdot v)]$$

$$= \lambda\|v\|_1 - 2(X\cdot v)^T y$$

4.1.2

$$J'(0;v) = \lambda\|v\|_1 - 2(X\cdot v)^T y \geq 0$$

$$\Rightarrow \lambda \geq \frac{2(X\cdot v)^T\cdot y}{\|v\|_1}$$

$$C = \frac{2(X\cdot v)^T\cdot y}{\|v\|_1}$$

Same logic, we have $v = \frac{\ }{2}$ $\Rightarrow$ $u = $

4.1.3 $\because \lambda \geq 2\|X^T y\|_\infty$

$\therefore$ for any $v \neq 0$,

$$\Rightarrow J'(0:v) = \lambda \|v\|_1 - 2(Xv)^T \cdot y$$

$$\geq 2\|X^T y\|_\infty \cdot \|v\|_1 - 2(Xv)^T \cdot y$$

$$= 2\|X^T y\|_\infty \cdot \|v\|_1 - 2(Xv)^T \cdot y \geq 0$$

$\therefore$ $w = 0$ is a minimizer of $J(w)$ if and only if $\lambda \geq 2\|X^T y\|_\infty$

4.1.4

$$J'(0, \bar{y}; v_1, v_2) = \lim_{h \to 0} \Big[\|X(0 + v_1 h) + (\bar{y} + v_2 h)\cdot 1 - y\|_2^2 + \lambda \|v_1 h\|_1 - \left(\|X \cdot 0 + \bar{y} 1 - y\|^2 + \lambda \|0\|_1\right)\Big] / h$$

$$= \lambda \|v_1\|_1 - 2(Xv_1)^T (y - \bar{y}) \geq 0$$

$$\Rightarrow \lambda \geq \lambda_{max} = 2\|X^T (y - \bar{y})\|_\infty$$

$$= 2 \cdot X^T (y - \bar{y})$$

4.2 Feature Correlation

iPad 9:44 PM 26%

classnotes Annotate Edit

Advance...thon | 03a.elastic-net | MI | hw2 | 04c.repr...theorem | ed

4.2.1. $J(\theta) = \| X_1\theta_1 + X_2\theta_2 + X_r\theta_r - y\|_2^2 + \lambda\|\theta_r\|_1 + (\lambda(|\theta_1| + |\theta_2|))$

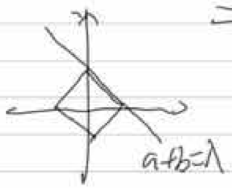
$X_1 = X_2 \rightarrow \quad = \|(\theta_1+\theta_2)X_1 + \theta_r X_r - y\|_2^2 + \lambda(|\theta_1| + |\theta_2|) + \lambda\|\theta_r\|_1$

a, b have to be the same sign,

$\because$ $a+b = a_1 + b_1$, a_1, b_1 same sign. if a_1, b are not same sign, $\Rightarrow |a| + |b| \geq |a_1| + |b_1|$

$\Rightarrow J(\theta)$ will not be optimal.

$\therefore$ a, b must be same sign



let $a+b = \lambda$

for any θ_1, θ_2, must meet $\theta_1 + \theta_2 = \lambda$

if $(c, d, r^*)^T$ is another minimizer,

$\Rightarrow c+d = \lambda = a+b$

4.2.2.

$J(\theta) = \|(\theta_1+\theta_2)X_1 + X_r\theta_r - y\|_2^2 + \lambda\|\theta_1\|_2^2 + \lambda(\|\theta_2\|_2^2 + \lambda\|\theta_r\|_2^2$

for $\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$, we have to minimize (a^2+b^2)

let $a^2+b^2 = w_1 \Rightarrow f(w_1) = a^2 + (w_1 - a)^2$

$\frac{\partial f(w_1)}{\partial a} = 2a - 2(w_1 - a) = 0 \Rightarrow a = \frac{w_1}{2}$

Same logic, we have $b = \frac{w_1}{2} \Rightarrow a = b$

4.2.3. the optimal solution will be unstable.

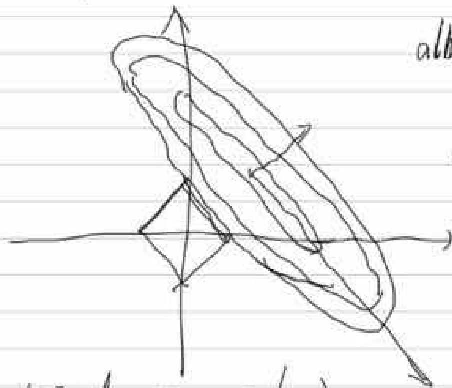
$\because$ radius $\propto \frac{1}{\text{singular value of } X^T X}$

$X = [X_{\cdot i}, X_{\cdot j}, X_{\cdot}]$

$\downarrow\ USV^T \Rightarrow \begin{bmatrix} \sigma_1 & & & \\ & \sigma_1 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}$

$\because X_{\cdot i}, X_{\cdot j}$ highly correlated, $\Rightarrow \sigma_n \to 0$

$\therefore$ ellipsoid will be like the following



almost parallel.

a little noise in the data set will change the optimal solution.

(2d. example)