Homework 2: Lasso Regression

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. It's preferred that you write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython), though scanning handwritten work is fine as well. You may find the minted package convenient for including source code in your LaTeX document. If you are using LyX, then the listings package tends to work better.

1 Introduction

In this homework you will investigate regression with ℓ_1 regularization, both implementation techniques and theoretical properties. On the methods side, you'll work on coordinate descent (the "shooting algorithm"), homotopy methods, and [optionally] projected SGD. On the theory side you'll derive the largest ℓ_1 regularization parameter you'll ever need to try, and optionally you'll derive the explicit solution to the coordinate minimizers used in coordinate descent, you'll investigate what happens with ridge and lasso regression when you have two copies of the same feature, and you'll work out the details of the classic picture that "explains" why ℓ_1 regularization leads to sparsity.

1.1 Data Set and Programming Problem Overview

For the experiments, we are generating some artifical data using code in the file `setup_problem.py`. We are considering the regression setting with the 1-dimensional input space $\mathbf{R}$. An image of the training data, along with the target function (i.e. the Bayes prediction function for the square loss function) is shown in Figure 1 below.

You can examine how the target function and the data were generated by looking at `setup_problem.py`. The figure can be reproduced by running the `LOAD_PROBLEM` branch of the main function.

As you can see, the target function is a highly nonlinear function of the input. To handle this sort of problem with linear hypothesis spaces, we will need to create a set of features that perform nonlinear transforms of the input. A detailed description of the technique we will use can be found in the Jupyter notebook `basis-fns.ipynb`, included in the zip file.

In this assignment, we are providing you with a function that takes care of the featurization. This is the "featurize" function, returned by the `generate_problem` function in `setup_problem.py`. The `generate_problem` function also gives the true target function, which has been constructed to be a sparse linear combination of our features. The coefficients of this linear combination are

Figure 1: Training data and target function we will be considering in this assignment.

also provided by generate_problem, so you can compare the coefficients of the linear functions you find to the target function coefficients. The generate_problem function also gives you the train and validation sets that you should use.

To get familiar with using the data, and perhaps to learn some techniques, it's recommended that you work through the main() function of the include file ridge_regression.py. You'll go through the following steps (on your own - no need to submit):

1. Load the problem from disk into memory with load_problem.

2. Use the featurize function to map from a one-dimensional input space to a d-dimensional feature space.

3. Visualize the design matrix of the featurized data. (All entries are binary, so we will not do any data normalization or standardization in this problem, though you may experiment with that on your own.)

4. Take a look at the class RidgeRegression. Here we've implemented our own RidgeRegression using the general purpose optimizer provided by scipy.optimize. This is primarily to introduce you to the sklearn framework, if you are not already familiar with it. It can help with hyperparameter tuning, as we will see shortly.

5. Take a look at compare_our_ridge_with_sklearn. In this function, we want to get some evidence that our implementation is correct, so we compare to sklearn's ridge regression. Comparing the outputs of two implementations is not always trivial – often the objective functions are slightly different, so you may need to think a bit about how to compare the results. In this case, sklearn has total square loss rather than average square loss, so we needed to account for that. In this case, we get an almost exact match with sklearn. This is because ridge regression is a rather easy objective function to optimize. You may not get as exact a match for other objective functions, even if both methods are "correct."

6. Next take a look at do_grid_search, in which we demonstrate how to take advantage of the fact that we've wrapped our ridge regression in an sklearn "Estimator" to do hyperparameter tuning. It's a little tricky to get GridSearchCV to use the train/test split that you want, but an approach is demonstrated in this function. In the line assigning the param_grid variable, you can see my attempts at doing hyperparameter search on a different problem. Below you will be modifying this (or using some other method, if you prefer) to find the optimal L2 regularization parameter for the data provided.

7. Next is some code to plot the results of the hyperparameter search.

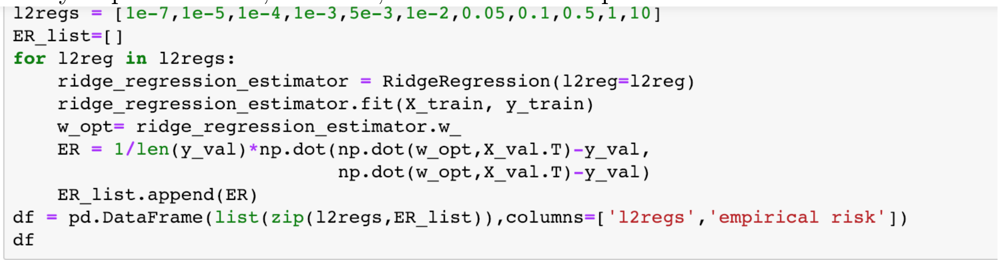
8. Next we want to visualize some prediction functions. We plotted the target function, along with several prediction functions corresponding to different regularization parameters, as functions of the original input space $\mathbf{R}$, along with the training data. Next we visualize the coefficients of each feature with bar charts. Take note of the scale of the y-axis, as they may vary substantially, buy default.

2 Ridge Regression

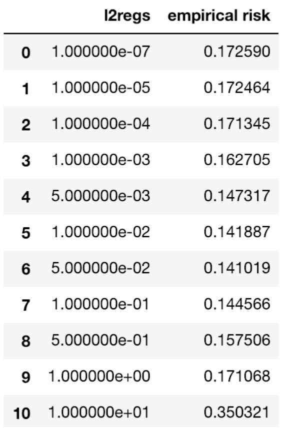
In the problems below, you do not need to implement ridge regression. You may use any of the code provided in the assignment, or you may use other packages. However, your results must correspond to the ridge regression objective function that we use, namely

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

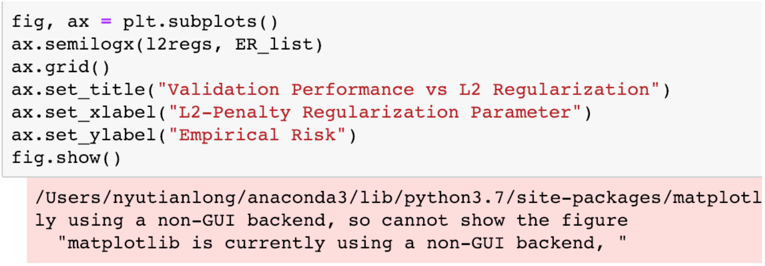
1. Run ridge regression on the provided training dataset. Choose the λ that minimizes the empirical risk (i.e. the average square loss) on the validation set. Include a table of the parameter values you tried and the validation performance for each. Also include a plot of the results.
   Im my implementation, $\lambda = 0.05$, minimized the empirical risk on the validation set.



```
l2regs = [1e-7,1e-5,1e-4,1e-3,5e-3,1e-2,0.05,0.1,0.5,1,10]
ER_list=[]
for l2reg in l2regs:
    ridge_regression_estimator = RidgeRegression(l2reg=l2reg)
    ridge_regression_estimator.fit(X_train, y_train)
    w_opt= ridge_regression_estimator.w_
    ER = 1/len(y_val)*np.dot(np.dot(w_opt,X_val.T)-y_val,
                             np.dot(w_opt,X_val.T)-y_val)
    ER_list.append(ER)
df = pd.DataFrame(list(zip(l2regs,ER_list)),columns=['l2regs','empirical risk'])
df
```

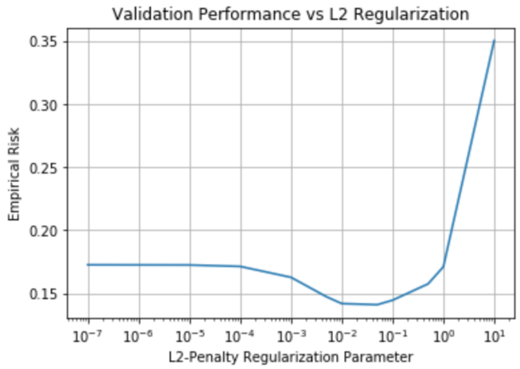


	l2regs	empirical risk
0	1.000000e-07	0.172590
1	1.000000e-05	0.172464
2	1.000000e-04	0.171345
3	1.000000e-03	0.162705
4	5.000000e-03	0.147317
5	1.000000e-02	0.141887
6	5.000000e-02	0.141019
7	1.000000e-01	0.144566
8	5.000000e-01	0.157506
9	1.000000e+00	0.171068
10	1.000000e+01	0.350321



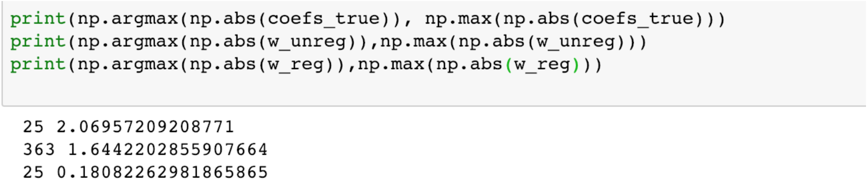
```
fig, ax = plt.subplots()
ax.semilogx(l2regs, ER_list)
ax.grid()
ax.set_title("Validation Performance vs L2 Regularization")
ax.set_xlabel("L2-Penalty Regularization Parameter")
ax.set_ylabel("Empirical Risk")
fig.show()
```

```
/Users/nyutianlong/anaconda3/lib/python3.7/site-packages/matplotl
ly using a non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



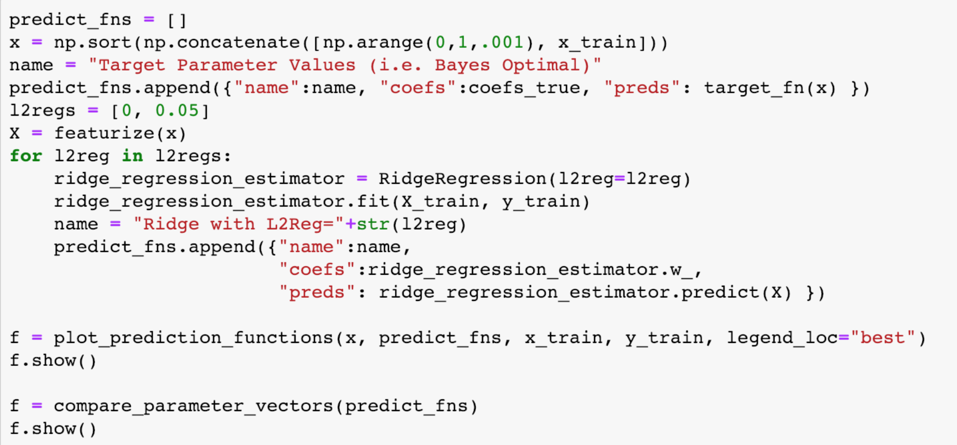
2. Now we want to visualize the prediction functions. On the same axes, plot the following: the training data, the target function, an unregularized least squares fit (still using the featurized data), and the prediction function chosen in the previous problem. Next, along the lines of the bar charts produced by the code in compare_parameter_vectors, visualize the coefficients for each of the prediction functions plotted, including the target function. Describe the patterns, including the scale of the coefficients, as well as which coefficients have the most weight.

   The coefficients of target function are very sparse, followed by the unregularized square fit.The coefficients for ridge regression with $\lambda = 0.05$ are very dense. This is expected as any parameter in ridge regression is unlikely to be exact zero. As for the scale of coefficients, both the target function and unregularized least square fit have parameters with much larger scale compared with regularized ridge regression. For the target function and regularized regression, the 25th coefficient have most weights, which are 2.0696 and 0.1808. For unregularized fit, the 363rd coefficient has most weight, which is 1.6442.

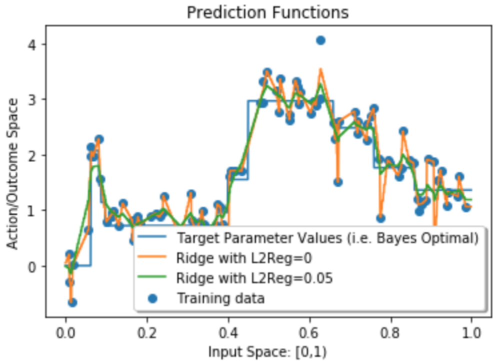


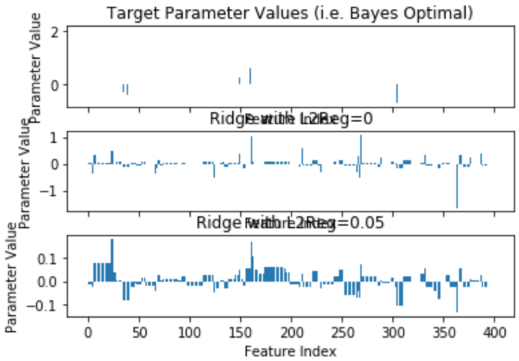
```
print(np.argmax(np.abs(coefs_true)), np.max(np.abs(coefs_true)))
print(np.argmax(np.abs(w_unreg)),np.max(np.abs(w_unreg)))
print(np.argmax(np.abs(w_reg)),np.max(np.abs(w_reg)))
```

```
25 2.06957209208771
363 1.6442202855907664
25 0.18082262981865865
```

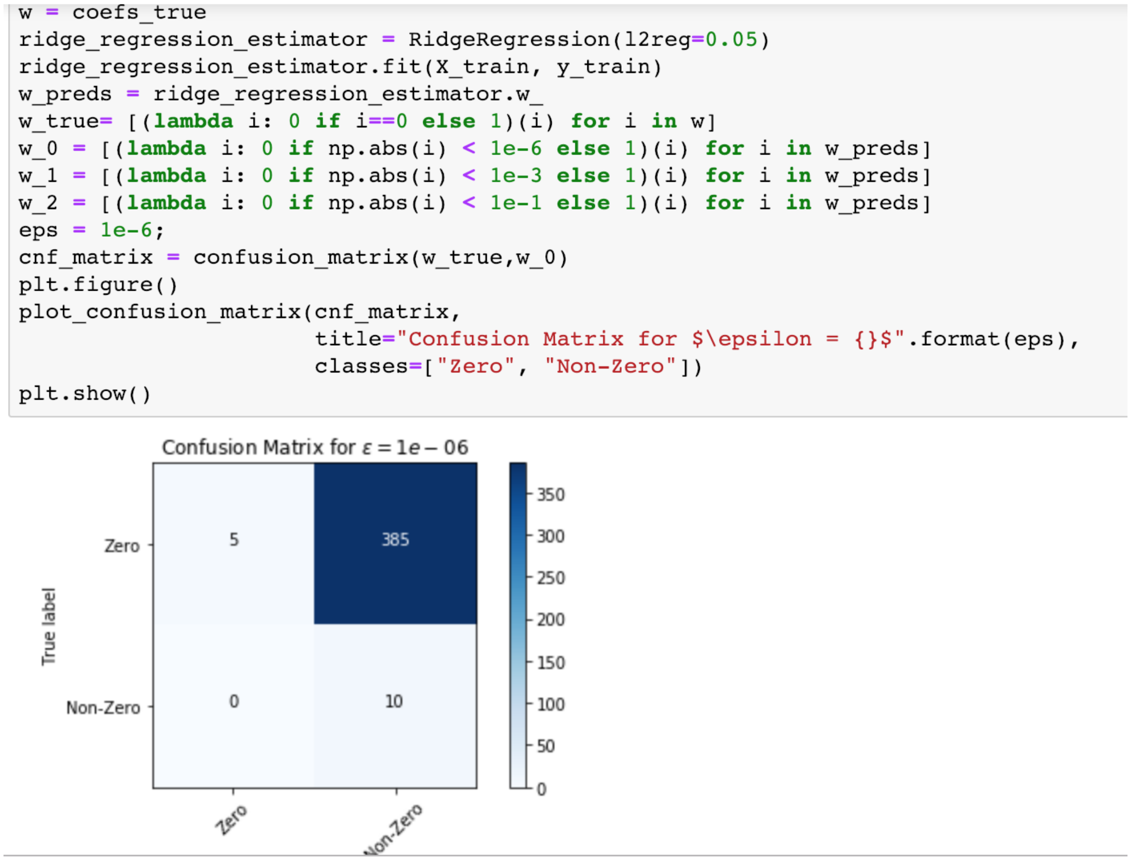


```
predict_fns = []
x = np.sort(np.concatenate([np.arange(0,1,.001), x_train]))
name = "Target Parameter Values (i.e. Bayes Optimal)"
predict_fns.append({"name":name, "coefs":coefs_true, "preds": target_fn(x) })
l2regs = [0, 0.05]
X = featurize(x)
for l2reg in l2regs:
    ridge_regression_estimator = RidgeRegression(l2reg=l2reg)
    ridge_regression_estimator.fit(X_train, y_train)
    name = "Ridge with L2Reg="+str(l2reg)
    predict_fns.append({"name":name,
                        "coefs":ridge_regression_estimator.w_,
                        "preds": ridge_regression_estimator.predict(X) })

f = plot_prediction_functions(x, predict_fns, x_train, y_train, legend_loc="best")
f.show()

f = compare_parameter_vectors(predict_fns)
f.show()
```



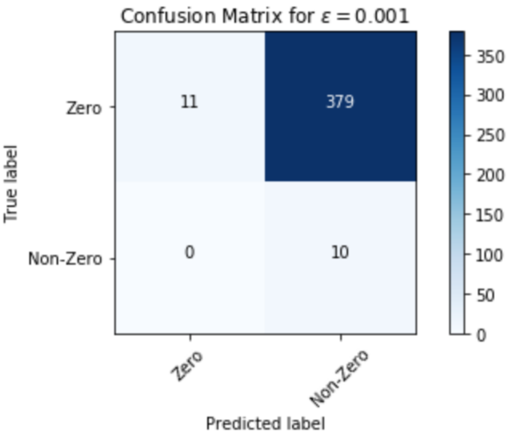


3. For the chosen λ, examine the model coefficients. For ridge regression, we don't expect any parameters to be exactly 0. However, let's investigate whether we can predict the sparsity pattern of the true parameters (i.e. which parameters are 0 and which are nonzero) by thresholding the parameter estimates we get from ridge regression. We'll predict that $w_i = 0$ if $|\hat{w}_i| < \varepsilon$ and $w_i \neq 0$ otherwise. Give the confusion matrix for $\varepsilon = 10^{-6}, 10^{-3}, 10^{-1}$, and any other thresholds you would like to try.

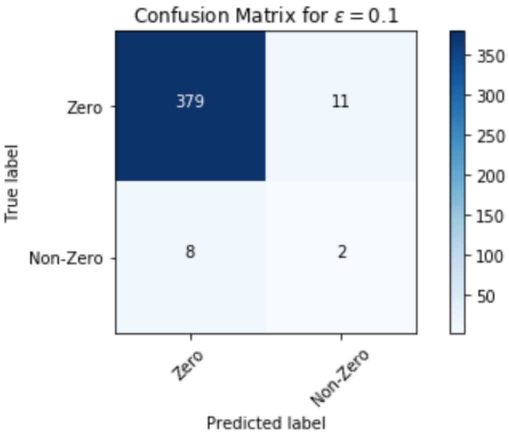


```
w = coefs_true
ridge_regression_estimator = RidgeRegression(l2reg=0.05)
ridge_regression_estimator.fit(X_train, y_train)
w_preds = ridge_regression_estimator.w_
w_true= [(lambda i: 0 if i==0 else 1)(i) for i in w]
w_0 = [(lambda i: 0 if np.abs(i) < 1e-6 else 1)(i) for i in w_preds]
w_1 = [(lambda i: 0 if np.abs(i) < 1e-3 else 1)(i) for i in w_preds]
w_2 = [(lambda i: 0 if np.abs(i) < 1e-1 else 1)(i) for i in w_preds]
eps = 1e-6;
cnf_matrix = confusion_matrix(w_true,w_0)
plt.figure()
plot_confusion_matrix(cnf_matrix,
                      title="Confusion Matrix for $\epsilon = {}$".format(eps),
                      classes=["Zero", "Non-Zero"])
plt.show()
```

```
eps = 1e-3
cnf_matrix = confusion_matrix(w_true,w_1)
plt.figure()
plot_confusion_matrix(cnf_matrix, title="Confusion Matrix for $\
plt.show()
```



```
s = 1e-1
f_matrix = confusion_matrix(w_true,w_2)
t.figure()
ot_confusion_matrix(cnf_matrix, title="Confusion Matrix for $\epsilon = {}$"
t.show()
```



3 Coordinate Descent for Lasso (a.k.a. The Shooting algorithm)

The Lasso optimization problem can be formulated as[1]

$$\hat{w} \in \arg\min_{w \in \mathbf{R}^d} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + \lambda \|w\|_1,$$

where $h_w(x) = w^T x$, and $\|w\|_1 = \sum_{i=1}^{d} |w_i|$. Note that to align with Murpy's formulation below, and for historical reasons, we are using the total square loss, rather than the average square loss, in the objective function.

Since the ℓ_1-regularization term in the objective function is non-differentiable, it's not immediately clear how gradient descent or SGD could be used to solve this optimization problem directly. (In fact, as we'll see in the next homework on SVMs, we can use "subgradient" methods when the objective function is not differentiable, in addition to the two methods discussed in this homework assignment.)

Another approach to solving optimization problems is coordinate descent, in which at each step we optimize over one component of the unknown parameter vector, fixing all other components. The descent path so obtained is a sequence of steps, each of which is parallel to a coordinate axis in $\mathbf{R}^d$, hence the name. It turns out that for the Lasso optimization problem, we can find a closed form solution for optimization over a single component fixing all other components. This gives us the following algorithm, known as the **shooting algorithm**:

(Source: Murphy, Kevin P. Machine learning: a probabilistic perspective. MIT press, 2012.)

The "soft thresholding" function is defined as

$$\text{soft}\,(a, \delta) = \text{sign}(a)\left(|a| - \delta\right)_+,$$

for any $a, \delta \in \mathbf{R}$.

NOTE: Algorithm 13.1 does not account for the case that $a_j = c_j = 0$, which occurs when the jth column of X is identically 0. One can either eliminate the column (as it cannot possibly help the solution), or you can set $w_j = 0$ in that case since it is, as you can easily verify, the coordinate minimizer. Note also that Murphy is suggesting to initialize the optimization with the ridge regession solution. Although theoretically this is not necessary (with exact computations and enough time, coordinate descent will converge for lasso from any starting point), in practice it's helpful to start as close to the solution as we're able.

There are a few tricks that can make selecting the hyperparameter λ easier and faster. First, as we'll see in a later problem, you can show that for any $\lambda \geq 2\|X^T(y - \bar{y})\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\bar{y}$ is the mean of values in the vector y, and $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum over the absolute values of the components of a vector. Thus we need to search for an optimal λ in $[0, \lambda_{\max}]$, where $\lambda_{\max} = 2\|X^T(y - \bar{y})\|_\infty$. (Note: This expression for $\lambda_{\max}$ assumes we have an unregularized bias term in our model. That is, our decision functions are of the form $h_{w,b}(x) = w^T x + b$. In our the experiments, we do not have an unregularized bias term, so we should use $\lambda_{\max} = 2\|X^T y\|_\infty$.)

[1]

The second trick is to use the fact that when λ and λ' are close, the corresponding solutions $\hat{w}(\lambda)$ and $\hat{w}(\lambda')$ are also close. Start with $\lambda = \lambda_{\max}$, for which we know $\hat{w}(\lambda_{\max}) = 0$. You can run the optimization anyway, and initialize the optimization at $w = 0$. Next, λ is reduced (e.g. by a constant factor close to 1), and the optimization problem is solved using the previous optimal point as the starting point. This is called **warm starting** the optimization. The technique of computing a set of solutions for a chain of nearby λ's is called a **continuation** or **homotopy method**. The resulting set of parameter values $\hat{w}(\lambda)$ as λ ranges over $[0, \lambda_{\max}]$ is known as a **regularization path**.

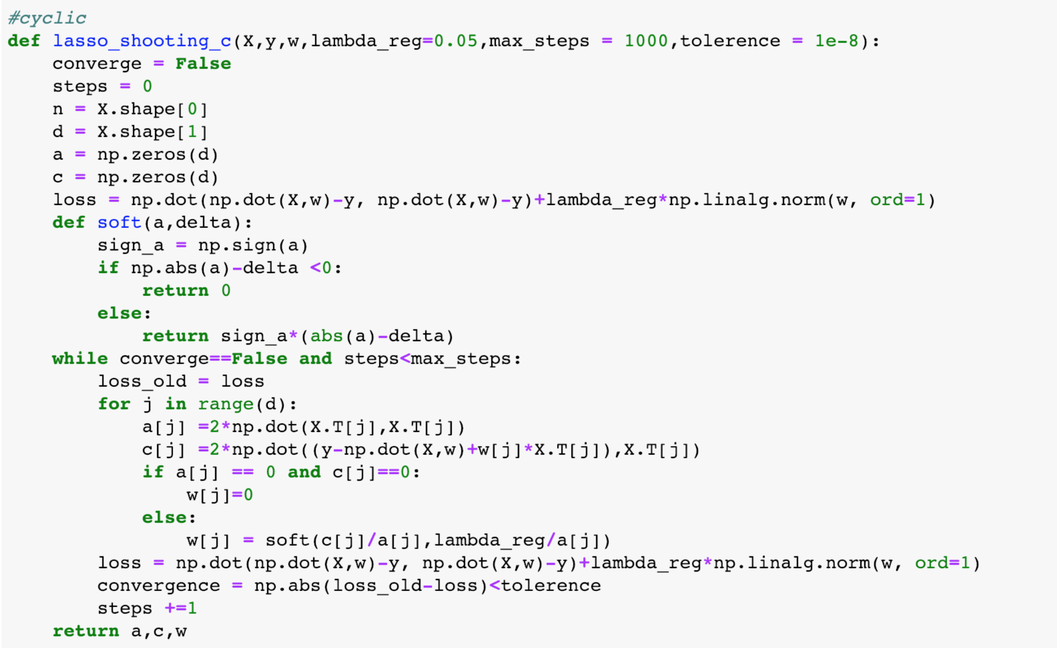
3.1 Experiments with the Shooting Algorithm

1. The algorithm as described above is not ready for a large dataset (at least if it has being implemented in Python) because of the implied loop in the summation signs for the expressions for a_j and c_j. Give an expression for computing a_j and c_j using matrix and vector operations, without explicit loops. This is called "vectorization" and can lead to dramatic speedup when implemented in languages such as Python, Matlab, and R. Write your expressions using X, w, $y = (y_1, \ldots, y_n)^T$ (the column vector of responses), $X_{\cdot j}$ (the jth column of X, represented as a column matrix), and w_j (the jth coordinate of w – a scalar).

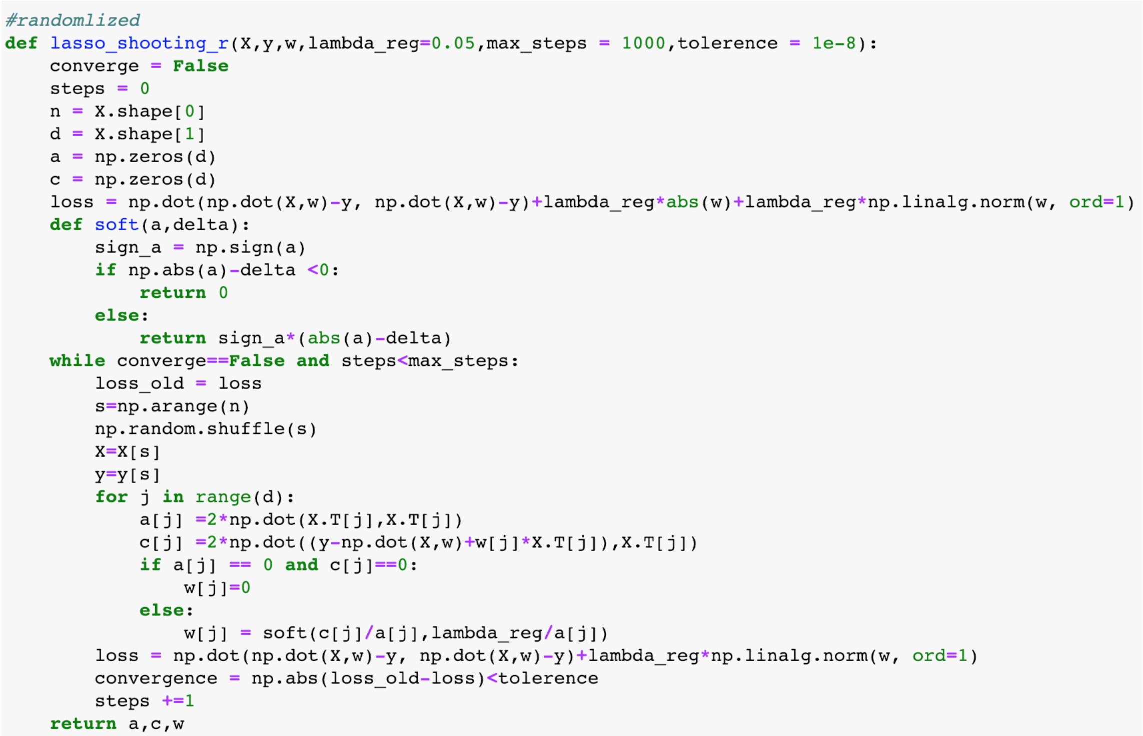
$$\begin{aligned} a_j &= 2X_j^T X_j \\ c_j &= 2X_j^T(y - Xw + w_j X_j) \end{aligned}$$

2. Write a function that computes the Lasso solution for a given λ using the shooting algorithm described above. For convergence criteria, continue coordinate descent until a pass through the coordinates reduces the objective function by less than 10^{-8}, or you have taken 1000 passes through the coordinates. Compare performance of cyclic coordinate descent to randomized coordinate descent, where in each round we pass through the coordinates in a different random order (for your choices of λ). Compare also the solutions attained (following the convergence criteria above) for starting at 0 versus starting at the ridge regression solution suggested by Murphy (again, for your choices of λ). If you like, you may adjust the convergence criteria to try to attain better results (or the same results faster).

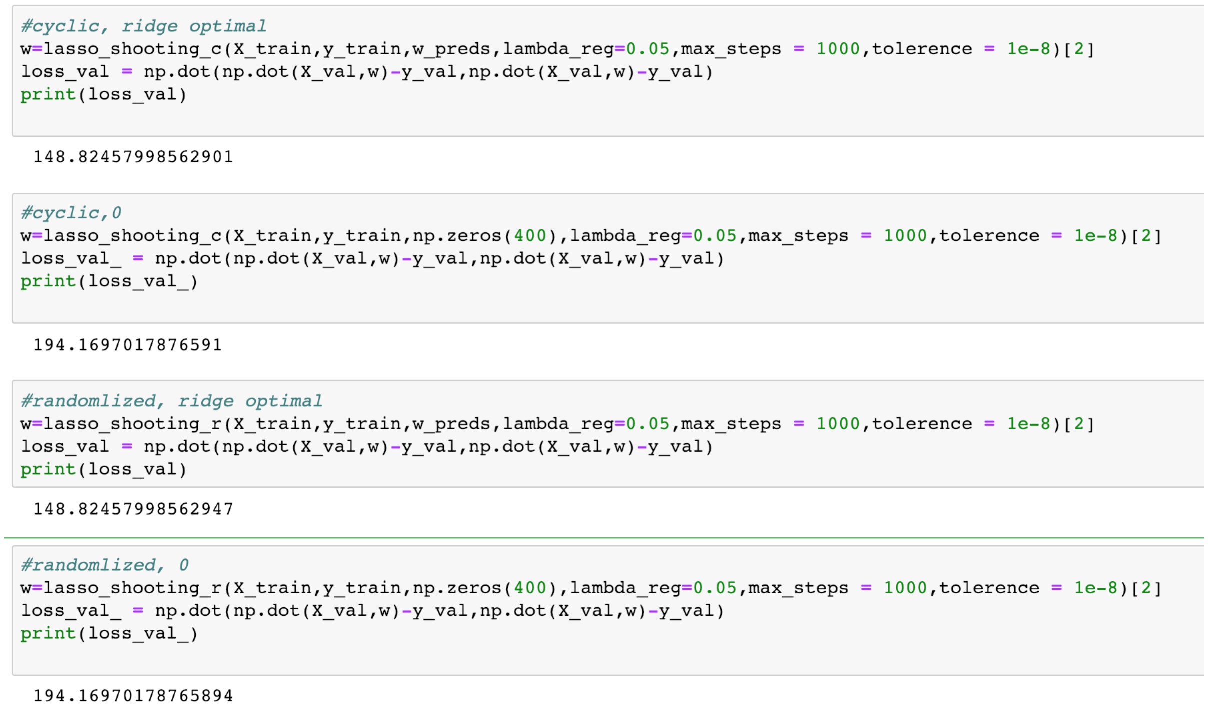
   In my implementation, cyclic coordinate descent and randomized coordinate descent gave very similar validation loss, though the validation loss from cyclic coordinate descent is slightly lower. And starting at the ridge regression solution suggested by Murphy ended up with less validation loss than starting from zero.



```python
#cyclic
def lasso_shooting_c(X,y,w,lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8):
    converge = False
    steps = 0
    n = X.shape[0]
    d = X.shape[1]
    a = np.zeros(d)
    c = np.zeros(d)
    loss = np.dot(np.dot(X,w)-y, np.dot(X,w)-y)+lambda_reg*np.linalg.norm(w, ord=1)
    def soft(a,delta):
        sign_a = np.sign(a)
        if np.abs(a)-delta <0:
            return 0
        else:
            return sign_a*(abs(a)-delta)
    while converge==False and steps<max_steps:
        loss_old = loss
        for j in range(d):
            a[j] =2*np.dot(X.T[j],X.T[j])
            c[j] =2*np.dot((y-np.dot(X,w)+w[j]*X.T[j]),X.T[j])
            if a[j] == 0 and c[j]==0:
                w[j]=0
            else:
                w[j] = soft(c[j]/a[j],lambda_reg/a[j])
        loss = np.dot(np.dot(X,w)-y, np.dot(X,w)-y)+lambda_reg*np.linalg.norm(w, ord=1)
        convergence = np.abs(loss_old-loss)<tolerence
        steps +=1
    return a,c,w
```



```python
#randomlized
def lasso_shooting_r(X,y,w,lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8):
    converge = False
    steps = 0
    n = X.shape[0]
    d = X.shape[1]
    a = np.zeros(d)
    c = np.zeros(d)
    loss = np.dot(np.dot(X,w)-y, np.dot(X,w)-y)+lambda_reg*abs(w)+lambda_reg*np.linalg.norm(w, ord=1)
    def soft(a,delta):
        sign_a = np.sign(a)
        if np.abs(a)-delta <0:
            return 0
        else:
            return sign_a*(abs(a)-delta)
    while converge==False and steps<max_steps:
        loss_old = loss
        s=np.arange(n)
        np.random.shuffle(s)
        X=X[s]
        y=y[s]
        for j in range(d):
            a[j] =2*np.dot(X.T[j],X.T[j])
            c[j] =2*np.dot((y-np.dot(X,w)+w[j]*X.T[j]),X.T[j])
            if a[j] == 0 and c[j]==0:
                w[j]=0
            else:
                w[j] = soft(c[j]/a[j],lambda_reg/a[j])
        loss = np.dot(np.dot(X,w)-y, np.dot(X,w)-y)+lambda_reg*np.linalg.norm(w, ord=1)
        convergence = np.abs(loss_old-loss)<tolerence
        steps +=1
    return a,c,w
```



```
#cyclic, ridge optimal
w=lasso_shooting_c(X_train,y_train,w_preds,lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8)[2]
loss_val = np.dot(np.dot(X_val,w)-y_val,np.dot(X_val,w)-y_val)
print(loss_val)
```

```
148.82457998562901
```

```
#cyclic,0
w=lasso_shooting_c(X_train,y_train,np.zeros(400),lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8)[2]
loss_val_ = np.dot(np.dot(X_val,w)-y_val,np.dot(X_val,w)-y_val)
print(loss_val_)
```

```
194.1697017876591
```

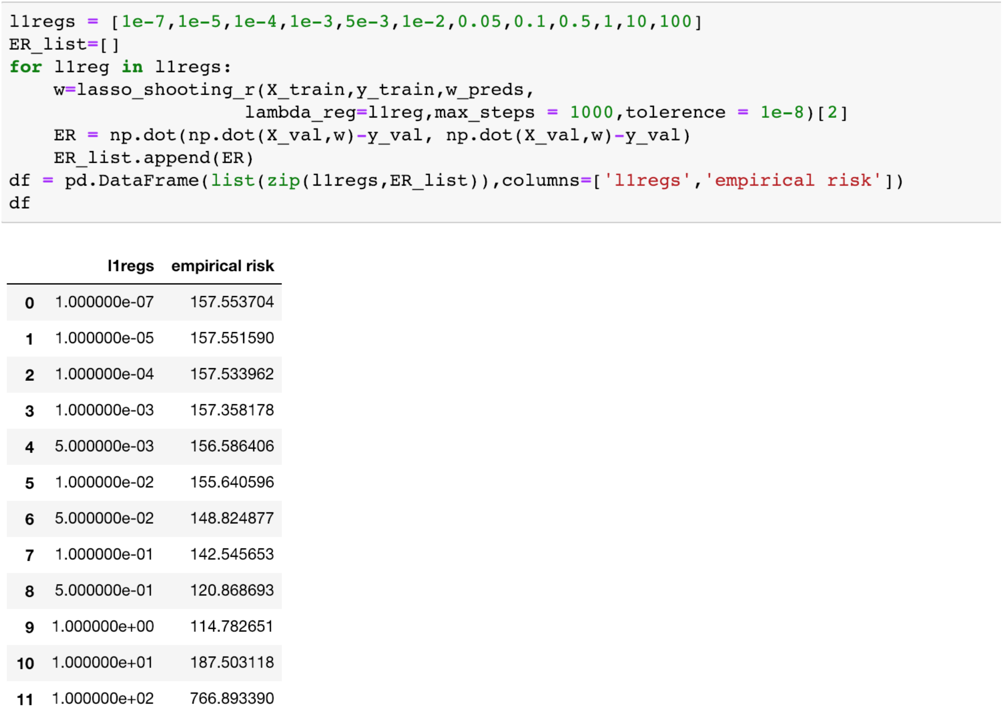
```
#randomlized, ridge optimal
w=lasso_shooting_r(X_train,y_train,w_preds,lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8)[2]
loss_val = np.dot(np.dot(X_val,w)-y_val,np.dot(X_val,w)-y_val)
print(loss_val)
```

```
148.82457998562947
```

```
#randomlized, 0
w=lasso_shooting_r(X_train,y_train,np.zeros(400),lambda_reg=0.05,max_steps = 1000,tolerence = 1e-8)[2]
loss_val_ = np.dot(np.dot(X_val,w)-y_val,np.dot(X_val,w)-y_val)
print(loss_val_)
```

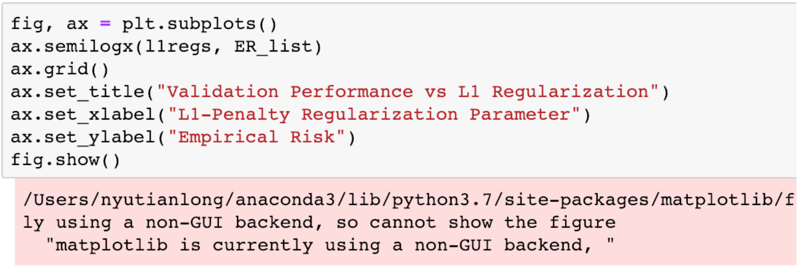
```
194.16970178765894
```

3. Run your best Lasso configuration on the training dataset provided, and select the λ that minimizes the square error on the validation set. Include a table of the parameter values you tried and the validation performance for each. Also include a plot of these results. Include also a plot of the prediction functions, just as in the ridge regression section, but this time add the best performing Lasso prediction function and remove the unregularized least squares fit. Similarly, add the lasso coefficients to the bar charts of coefficients generated in the ridge regression setting. Comment on the results, with particular attention to parameter sparsity and how the ridge and lasso solutions compare. What's the best model you found, and what's its validation performance?
In my implementation, $\lambda = 1$ gave the least square error on the validation set. The lasso solution was very sparse compared with the ridge solution. In terms of scale, lasso solution seems to have larger scale than the ridge solution. The best model I found was cyclic coordinate descent ($\lambda = 1$) with parameter attained from ridge regression as starting point.Its validation loss was 114.78.



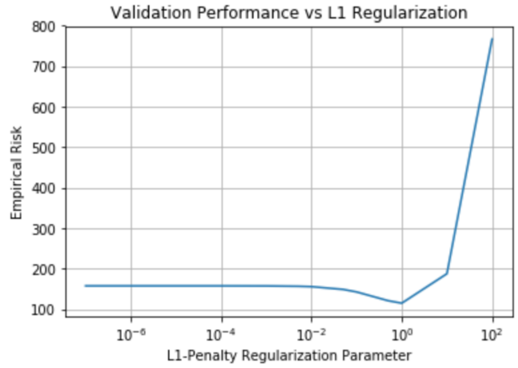
```
l1regs = [1e-7,1e-5,1e-4,1e-3,5e-3,1e-2,0.05,0.1,0.5,1,10,100]
ER_list=[]
for l1reg in l1regs:
    w=lasso_shooting_r(X_train,y_train,w_preds,
                       lambda_reg=l1reg,max_steps = 1000,tolerence = 1e-8)[2]
    ER = np.dot(np.dot(X_val,w)-y_val, np.dot(X_val,w)-y_val)
    ER_list.append(ER)
df = pd.DataFrame(list(zip(l1regs,ER_list)),columns=['l1regs','empirical risk'])
df
```

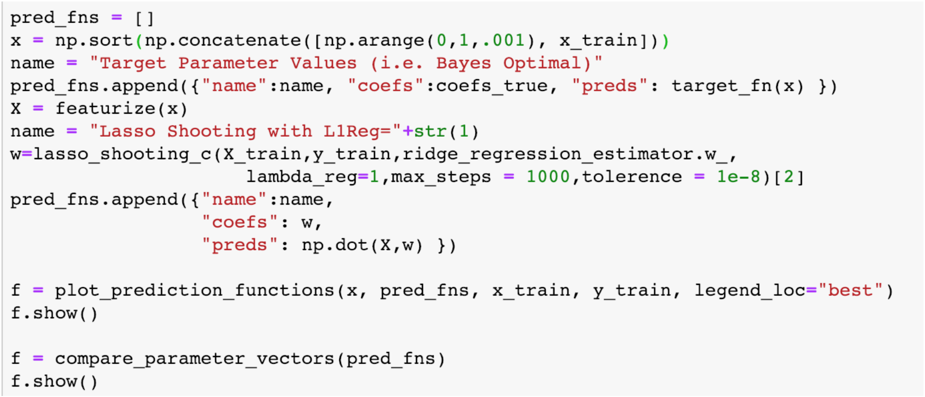
	l1regs	empirical risk
0	1.000000e-07	157.553704
1	1.000000e-05	157.551590
2	1.000000e-04	157.533962
3	1.000000e-03	157.358178
4	5.000000e-03	156.586406
5	1.000000e-02	155.640596
6	5.000000e-02	148.824877
7	1.000000e-01	142.545653
8	5.000000e-01	120.868693
9	1.000000e+00	114.782651
10	1.000000e+01	187.503118
11	1.000000e+02	766.893390



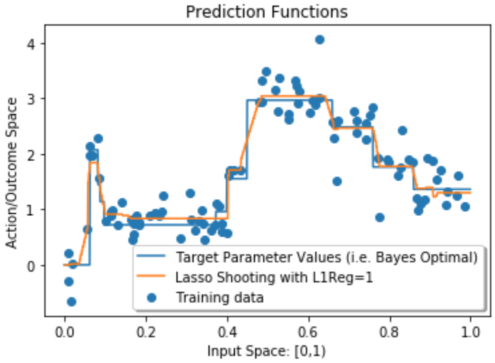
```
fig, ax = plt.subplots()
ax.semilogx(l1regs, ER_list)
ax.grid()
ax.set_title("Validation Performance vs L1 Regularization")
ax.set_xlabel("L1-Penalty Regularization Parameter")
ax.set_ylabel("Empirical Risk")
fig.show()
```

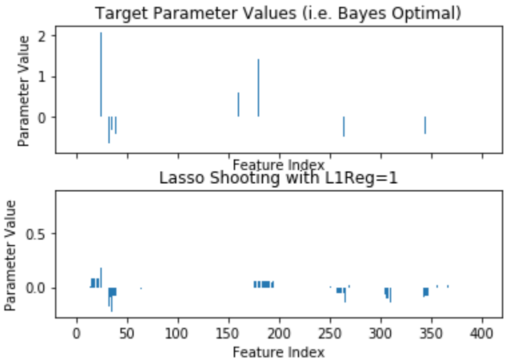
```
/Users/nyutianlong/anaconda3/lib/python3.7/site-packages/matplotlib/f
ly using a non-GUI backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



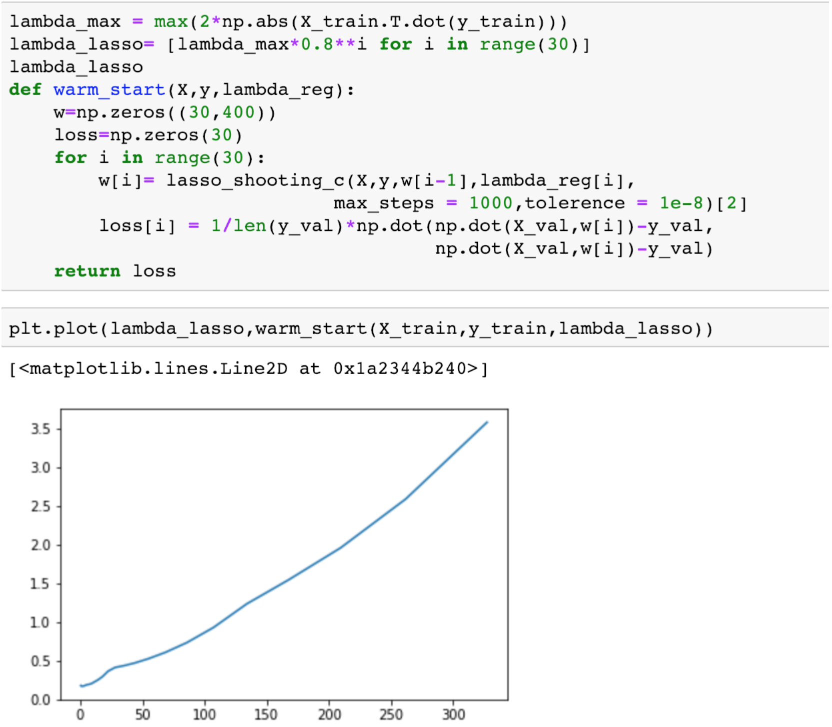


```
pred_fns = []
x = np.sort(np.concatenate([np.arange(0,1,.001), x_train]))
name = "Target Parameter Values (i.e. Bayes Optimal)"
pred_fns.append({"name":name, "coefs":coefs_true, "preds": target_fn(x) })
X = featurize(x)
name = "Lasso Shooting with L1Reg="+str(1)
w=lasso_shooting_c(X_train,y_train,ridge_regression_estimator.w_,
                    lambda_reg=1,max_steps = 1000,tolerence = 1e-8)[2]
pred_fns.append({"name":name,
                 "coefs": w,
                 "preds": np.dot(X,w) })

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

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





4. Implement the homotopy method described above. Compute the Lasso solution for (at least) the regularization parameters in the set $\{\lambda = \lambda_{\max}0.8^i \mid i = 0, \ldots, 29\}$. Plot the results (average validation loss vs λ).



```
lambda_max = max(2*np.abs(X_train.T.dot(y_train)))
lambda_lasso= [lambda_max*0.8**i for i in range(30)]
lambda_lasso
def warm_start(X,y,lambda_reg):
    w=np.zeros((30,400))
    loss=np.zeros(30)
    for i in range(30):
        w[i]= lasso_shooting_c(X,y,w[i-1],lambda_reg[i],
                               max_steps = 1000,tolerence = 1e-8)[2]
        loss[i] = 1/len(y_val)*np.dot(np.dot(X_val,w[i])-y_val,
                                      np.dot(X_val,w[i])-y_val)
    return loss
```

```
plt.plot(lambda_lasso,warm_start(X_train,y_train,lambda_lasso))
```

```
[<matplotlib.lines.Line2D at 0x1a2344b240>]
```

5. [Optional] Note that the data in Figure 1 is almost entirely nonnegative. Since we don't have an unregularized bias term, we have "pay for" this offset using our penalized parameters. Note also that λ_{max} would decrease significantly if the y values were 0 centered (using the training data, of course), or if we included an unregularized bias term. Experiment with one or both of these approaches, for both and lasso and ridge regression, and report your findings.

3.2 [Optional] Deriving the Coordinate Minimizer for Lasso

This problem is to derive the expressions for the coordinate minimizers used in the Shooting algorithm. This is often derived using subgradients (slide 15), but here we will take a bare hands approach (which is essentially equivalent).

In each step of the shooting algorithm, we would like to find the w_j minimizing

$$
\begin{aligned}
f(w_j) &= \sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 + \lambda \left|w\right|_1 \\
&= \sum_{i=1}^{n}\left[w_j x_{ij} + \sum_{k \neq j} w_k x_{ik} - y_i\right]^2 + \lambda\left|w_j\right| + \lambda \sum_{k \neq j}\left|w_k\right|,
\end{aligned}
$$

where we've written x_{ij} for the jth entry of the vector x_i. This function is convex in w_j. The only thing keeping f from being differentiable is the term with $|w_j|$. So f is differentiable everywhere except $w_j = 0$. We'll break this problem into 3 cases: $w_j > 0$, $w_j < 0$, and $w_j = 0$. In the first two cases, we can simply differentiate f w.r.t. w_j to get optimality conditions. For the last case, we'll use the fact that since $f : \mathbf{R} \to \mathbf{R}$ is convex, 0 is a minimizer of f iff

$$\lim_{\varepsilon \downarrow 0} \frac{f(\varepsilon) - f(0)}{\varepsilon} \geq 0 \quad \text{and} \quad \lim_{\varepsilon \downarrow 0} \frac{f(-\varepsilon) - f(0)}{\varepsilon} \geq 0.$$

This is a special case of the optimality conditions described in slide 6 here, where now the "direction" v is simply taken to be the scalars 1 and -1, respectively.

1. First let's get a trivial case out of the way. If $x_{ij} = 0$ for $i = 1, \ldots, n$, what is the coordinate minimizer w_j? In the remaining questions below, you may assume that $\sum_{i=1}^{n} x_{ij}^2 > 0$.

2. Give an expression for the derivative $f(w_j)$ for $w_j \neq 0$. It will be convenient to write your expression in terms of the following definitions:

$$\begin{aligned} \operatorname{sign}(w_j) &:= \begin{cases} 1 & w_j > 0 \\ 0 & w_j = 0 \\ -1 & w_j < 0 \end{cases} \\ a_j &:= 2\sum_{i=1}^{n} x_{ij}^2 \\ c_j &:= 2\sum_{i=1}^{n} x_{ij}\left(y_i - \sum_{k \neq j} w_k x_{ik}\right). \end{aligned}$$

3. If $w_j > 0$ and minimizes f, show that $w_j = \frac{1}{a_j}(c_j - \lambda)$. Similarly, if $w_j < 0$ and minimizes f, show that $w_j = \frac{1}{a_j}(c_j + \lambda)$. Give conditions on c_j that imply that a minimizer w_j is positive and conditions for which a minimizer w_j is negative.

4. Derive expressions for the two one-sided derivatives at $f(0)$, and show that $c_j \in [-\lambda, \lambda]$ implies that $w_j = 0$ is a minimizer.

5. Putting together the preceding results, we conclude the following:

$$w_j = \begin{cases} \frac{1}{a_j}(c_j - \lambda) & c_j > \lambda \\ 0 & c_j \in [-\lambda, \lambda] \\ \frac{1}{a_j}(c_j + \lambda) & c_j < -\lambda \end{cases}$$

   Show that this is equivalent to the expression given in 3.

4 Lasso Properties

4.1 Deriving $\lambda_{\max}$

In this problem we will derive an expression for $\lambda_{\max}$. For the first three parts, use the Lasso objective function excluding the bias term i.e, $J(w) = \|Xw - y\|_2^2 + \lambda \|w\|_1$. We will show that for any $\lambda \geq 2\|X^T y\|_\infty$, the estimated weight vector $\hat{w}$ is entirely zero, where $\|\cdot\|_\infty$ is the infinity norm (or supremum norm), which is the maximum absolute value of any component of the vector.

1. The one-sided directional derivative of $f(x)$ at x in the direction v is defined as:

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

Compute $J'(0;v)$. That is, compute the one-sided directional derivative of $J(w)$ at $w = 0$ in the direction v. [Hint: the result should be in terms of $X, y, \lambda,$ and v.]

$$\begin{aligned} J'(0;v) &= \lim_{h \downarrow 0} \frac{J(hv) - J(0)}{h} \\ &= \lim_{h \downarrow 0} \frac{\|Xhv - y\|_2^2 + \lambda \|hv\|_1 - \| - y\|_2^2}{h} \\ &= \lim_{h \downarrow 0} \frac{(Xhv)^T(Xhv) - 2y^T Xhv + y^2 + \lambda \|hv\|_1 - \| - y\|_2^2}{h} \\ &= -2y^T Xv + \lambda \|v\|_1 \end{aligned}$$

2. Since the Lasso objective is convex, w^* is a minimizer of $J(w)$ if and only if the directional derivative $J'(w^*;v) \geq 0$ for all $v \neq 0$. Show that for any $v \neq 0$, we have $J'(0;v) \geq 0$ if and only if $\lambda \geq C$, for some C that depends on $X, y,$ and v. You should have an explicit expression for C.

$$\begin{aligned} &\text{Let } -2y^T Xv + \lambda \|v\|_1 \geq 0 \Rightarrow \lambda \geq \frac{2y^T Xv}{\|v\|_1} \\ &C = \frac{2y^T Xv}{\|v\|_1} \end{aligned}$$

3. In the previous problem, we get a different lower bound on λ for each choice of v. Show that the maximum of these lower bounds on λ is $\lambda_{\max} = 2\|X^T y\|_\infty$. Conclude that $w = 0$ is a minimizer of $J(w)$ if and only if $\lambda \geq 2\|X^T y\|_\infty$.

$$\frac{2y^T Xv}{\|v\|_1} = \frac{\Sigma_{i=1}^n 2y_i x_i v_i}{\|v\|_1} = \frac{\Sigma_{i=1}^n 2(x_i y_i)(v_i)}{\Sigma_{i=1}^n v_i} \leq \frac{\Sigma_{i=1}^n 2(max(x_i y_i))(v_i)}{\Sigma_{i=1}^n v_i} = 2\|X^T y\|_\infty$$

The maximum of these lower bounds on λ is $2\|X^T y\|_\infty$ and we have $J'(0;v) \geq 0$ if and only if $\lambda \geq C$. Hence, C $= 2\|X^T y\|_\infty$ is the most strict condition on λ. We can conclude that $w = 0$ is a minimizer of $J(w)$ if and only if $\lambda \geq 2\|X^T y\|_\infty$.

4. [Optional] Let $J(w,b) = \|Xw + b\mathbf{1} - y\|_2^2 + \lambda \|w\|_1$, where $\mathbf{1} \in \mathbf{R}^n$ is a column vector of 1's. Let $\bar{y}$ be the mean of values in the vector y. Show that $(w^*, b^*) = (0, \bar{y})$ is a minimizer of $J(w,b)$ if and only if $\lambda \geq \lambda_{\max} = 2\|X^T(y - \bar{y})\|_\infty$.

4.2 Feature Correlation

In this problem, we will examine and compare the behavior of the Lasso and ridge regression in the case of an exactly repeated feature. That is, consider the design matrix $X \in \mathbf{R}^{m \times d}$, where

$X_{\cdot i} = X_{\cdot j}$ for some i and j, where $X_{\cdot i}$ is the i^{th} column of X. We will see that ridge regression divides the weight equally among identical features, while Lasso divides the weight arbitrarily. In an optional part to this problem, we will consider what changes when $X_{\cdot i}$ and $X_{\cdot j}$ are highly correlated (e.g. exactly the same except for some small random noise) rather than exactly the same.

1. Without loss of generality, assume the first two colums of X are our repeated features. Partition X and θ as follows:

$$X = \begin{pmatrix} x_1 & x_2 & X_r \end{pmatrix} \qquad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_r \end{pmatrix}$$

We can write the Lasso objective function as:

$$\begin{aligned} J(\theta) &= \|X\theta - y\|_2^2 + \lambda \|\theta\|_1 \\ &= \|x_1\theta_1 + x_2\theta_2 + X_r\theta_r - y\|_2^2 + \lambda|\theta_1| + \lambda|\theta_2| + \lambda \|\theta_r\|_1 \end{aligned}$$

With repeated features, there will be multiple minimizers of $J(\theta)$. Suppose that

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

is a minimizer of $J(\theta)$. Give conditions on c and d such that $\left(c, d, r^T\right)^T$ is also a minimizer of $J(\theta)$. [Hint: First show that a and b must have the same sign, or at least one of them is zero. Then, using this result, rewrite the optimization problem to derive a relation between a and b.]
WTS a and b have the same sign. Suppose a and b have different signs, At minimum, $J(\theta) = \|x_1 a + x_2 b + X_r r - y\|_2^2 + \lambda|a| + \lambda|b| + \lambda \|r\|_1$
$= \|x_1(a+b) + X_r r - y\|_2^2 + \lambda|a| + \lambda|b| + \lambda \|r\|_1$.
$\geq \|x_1(a+b) + X_r r - y\|_2^2 + \lambda|a+b| + \lambda \|r\|_1$.
Since a and b have opposite signs, we have $\|x_1(a+b) + X_r r - y\|_2^2 + \lambda|a| + \lambda|b| + \lambda \|r\|_1 > \|x_1(a+b) + X_r r - y\|_2^2 + \lambda|a+b| + \lambda \|r\|_1$, wich contradicts the fact that $(a, b, r^T)^T$ is a minimizer of $J(\theta)$. Hence, a and b must have the same sign or one of them is 0.

Next, WTS $c + d = a + b$. Since a and b have the same sign, we can rewrite $J(\theta) = \|x_1 S + X_r r - y\|_2^2 + \lambda|S| + \lambda \|r\|_1$where $S = a + b$. Since $(c, d, r^T)^T$ is also a minimizer of $J(\theta)$, $\|x_1 c + x_2 d + X_r r - y\|_2^2 + \lambda|c| + \lambda|d| + \lambda \|r\|_1 = \|x_1 S + X_r r - y\|_2^2 + \lambda|S| + \lambda \|r\|_1 \Rightarrow c + d = S = a + b$.

2. Using the same notation as the previous problem, suppose

$$\hat{\theta} = \begin{pmatrix} a \\ b \\ r \end{pmatrix}$$

minimizes the ridge regression objective function. What is the relationship between a and b, and why?

WTS $a = b$

$J(\theta) = \|x_1 a + x_2 b + X_r r - y\|^2 + \lambda(a^2 + b^2 + r^2)$

$= \|x_1(a + b) + X_r r - y\|^2 + \lambda(a^2 + b^2 + r^2)$

Denote $a + b$ as S, then $J(\theta) = \|x_1 S + X_r r - y\|^2 + \lambda(S^2 - 2(S - a) + r^2)$

$= x_1^2 S^2 + 2(X_r r - y)x_1 S + \|X_r r - y\|^2 + \lambda(S^2 - 2(S - a) + r^2)$

It is minimized when $a = \dfrac{S}{2}$, holding S constant. Hence, in $\hat{\theta}, a = b$.

3. [Optional] What do you think would happen with Lasso and ridge when $X_{\cdot i}$ and $X_{\cdot j}$ are highly correlated, but not exactly the same. You may investigate this experimentally or theoretically.

5 [Optional] The Ellipsoids in the ℓ_1/ℓ_2 regularization picture

Recall the famous picture purporting to explain why ℓ_1 regularization leads to sparsity, while ℓ_2 regularization does not. Here's the instance from Hastie et al's *The Elements of Statistical Learning:*

(While Hastie et al. use β for the parameters, we'll continue to use w.)

In this problem we'll show that the level sets of the empirical risk are indeed ellipsoids centered at the empirical risk minimizer $\hat{w}$.

Consider linear prediction functions of the form $x \mapsto w^T x$. Then the empirical risk for $f(x) = w^T x$ under the square loss is

$$\begin{aligned}\hat{R}_n(w) &= \frac{1}{n}\sum_{i=1}^{n}\left(w^T x_i - y_i\right)^2 \\ &= \frac{1}{n}\left(Xw - y\right)^T\left(Xw - y\right).\end{aligned}$$

1. [Optional] Let $\hat{w} = \left(X^T X\right)^{-1} X^T y$. Show that $\hat{w}$ has empirical risk given by

$$\hat{R}_n(\hat{w}) = \frac{1}{n}\left(-y^T X\hat{w} + y^T y\right)$$

2. [Optional] Show that for any w we have

$$\hat{R}_n(w) = \frac{1}{n}\left(w - \hat{w}\right)^T X^T X\left(w - \hat{w}\right) + \hat{R}_n(\hat{w}).$$

Note that the RHS (i.e. "right hand side") has one term that's quadratic in w and one term that's independent of w. In particular, the RHS does not have any term that's linear in w. On the LHS (i.e. "left hand side"), we have $\hat{R}_n(w) = \frac{1}{n}\left(Xw - y\right)^T\left(Xw - y\right)$. After expanding this out, you'll have terms that are quadratic, linear, and constant in w. Completing the square is the tool for rearranging an expression to get rid of the linear terms. The following "completing the square" identity is easy to verify just by multiplying out the expressions on the RHS:

$$x^T M x - 2b^T x \quad = \quad \left(x - M^{-1}b\right)^T M(x - M^{-1}b) - b^T M^{-1} b$$

3. [Optional] Using the expression derived for $\hat{R}_n(w)$ in 2, give a very short proof that $\hat{w} = \left(X^T X\right)^{-1} X^T y$ is the empirical risk minimizer. That is:

   $$\hat{w} = \arg\min_w \hat{R}_n(w).$$

   Hint: Note that $X^T X$ is positive semidefinite and, by definition, a symmetric matrix M is positive semidefinite iff for all $x \in \mathbf{R}^d$, $x^T M x \geq 0$.

4. [Optional] Give an expression for the set of w for which the empirical risk exceeds the minimum empirical risk $\hat{R}_n(\hat{w})$ by an amount $c > 0$. If X is full rank, then $X^T X$ is positive definite, and this set is an ellipse – what is its center?

6 [Optional] Projected SGD via Variable Splitting

In this question, we consider another general technique that can be used on the Lasso problem. We first use the variable splitting method to transform the Lasso problem to a differentiable problem with linear inequality constraints, and then we can apply a variant of SGD.

Representing the unknown vector θ as a difference of two non-negative vectors θ^+ and θ^-, the ℓ_1-norm of θ is given by $\sum_{i=1}^{d} \theta_i^+ + \sum_{i=1}^{d} \theta_i^-$. Thus, the optimization problem can be written as

$$(\hat{\theta}^+, \hat{\theta}^-) = \mathop{\arg\min}_{\theta^+, \theta^- \in \mathbf{R}^d} \sum_{i=1}^{m} (h_{\theta^+,\theta^-}(x_i) - y_i)^2 + \lambda \sum_{i=1}^{d} \theta_i^+ + \lambda \sum_{i=1}^{d} \theta_i^-$$
$$\text{such that } \theta^+ \geq 0 \text{ and } \theta^- \geq 0,$$

where $h_{\theta^+,\theta^-}(x) = (\theta^+ - \theta^-)^T x$. The original parameter θ can then be estimated as $\hat{\theta} = (\hat{\theta}^+ - \hat{\theta}^-)$.

This is a convex optimization problem with a differentiable objective and linear inequality constraints. We can approach this problem using projected stochastic gradient descent, as discussed in lecture. Here, after taking our stochastic gradient step, we project the result back into the feasible set by setting any negative components of θ^+ and θ^- to zero.

1. [Optional] Implement projected SGD to solve the above optimization problem for the same λ's as used with the shooting algorithm. Since the two optimization algorithms should find

essentially the same solutions, you can check the algorithms against each other. Report the differences in validation loss for each λ between the two optimization methods. (You can make a table or plot the differences.)

2. [Optional] Choose the λ that gives the best performance on the validation set. Describe the solution $\hat{w}$ in term of its sparsity. How does the sparsity compare to the solution from the shooting algorithm?