Name: Zhuoyuan Xu

NetID: zx1137

Due Date: March 7, 2021

Problem 1

(a) Define $f(x) = \alpha||x||_2^2$, then

$$
\begin{aligned}
prox_f(y) &= arg\min_x f(x) + \frac{1}{2}||x-y||_2^2 \\
&= arg\min_x \alpha||x||_2^2 + \frac{1}{2}||x-y||_2^2
\end{aligned}
$$

Both $f(x)$ and $\frac{1}{2}||x-y||_2^2$ are convex, and then to find the proximal operator we can take the gradient of objective function in $prox_f(y)$ with respect to x and set the expression equal to 0.

$$
\begin{aligned}
\nabla_x \alpha||x||_2^2 + \frac{1}{2}||x-y||_2^2 &= 2\alpha x + \frac{1}{2}(2x-2y) \\
&= 2\alpha x + x - y
\end{aligned}
$$

Set to 0

$$
\begin{aligned}
2\alpha x + x - y &= 0 \\
x &= \frac{y}{2\alpha+1}
\end{aligned}
$$

(b) In this problem, the proximity operator is

$$prox_{\alpha||\cdot||_1}(y) = arg\min_x \alpha||x||_1 + \frac{1}{2}||x-y||_2^2$$

This expression can be divided into 2 parts:

$$F(x) = \frac{1}{2}||x-y||_2^2$$

which is convex and differentiable, and

$$G(x) = \alpha||x||_1$$

which is convex but not differentiable everywhere

If we take the subgradient of the operator, and by the optimality condition we need to have

$$0 \in \partial(G(x) + \frac{1}{2}||x-y||_2^2)$$

note ∂ is used to denote the subgradient in this case

Thus,

$$x - y \in \partial G(x)$$

The i-th entry of $\partial G(x)$ is

$$\partial |x_i| = \begin{cases} \alpha, & x_i > 0 \\ -\alpha, & x_i < 0 \\ [-\alpha, \alpha], & x_i = 0 \end{cases}$$

Therefore the $prox_{\alpha||\cdot||_1}(y)$ can be re-written entrywise by

$$(prox_{\alpha||\cdot||_1}(y))_i = \begin{cases} y_i - \alpha, & y_i \geq \alpha \\ 0, & |y_i| < \alpha \\ y_i + \alpha, & y_i \leq -\alpha \end{cases}$$

which can also be written as

$$(prox_{\alpha||\cdot||_1}(y))_i = \begin{cases} y_i - sign(y_i)\alpha, & |y_i| \geq \alpha \\ 0, & \text{otherwise} \end{cases}$$

This gives $prox_{\alpha||\cdot||_1}(y) = S_\alpha(y)$

(c) If $X \in R^{pxn}$ has orthonormal rows ($p \leq n$), then $XX^T = I$ and

$$||X(y - X^T\beta)||_2^2 = ||y - X^T\beta||_2^2$$

Thus,

$$\begin{aligned} arg \min_\beta \frac{1}{2}||y - X^T\beta||_2^2 + f(\beta) &= arg \min_\beta \frac{1}{2}||X(y - X^T\beta)||_2^2 + f(\beta) \\ &= arg \min_\beta \frac{1}{2}||Xy - XX^T\beta||_2^2 + f(\beta) \\ &= arg \min_\beta \frac{1}{2}||Xy - \beta||_2^2 + f(\beta) \end{aligned}$$

(d) Set $w = Xy$, then the previous representation can be written as

$$\begin{aligned} arg \min_\beta \frac{1}{2}||y - X^T\beta||_2^2 + f(\beta) &= arg \min_\beta \frac{1}{2}||w - \beta||_2^2 + f(\beta) \\ &= prox_f(w) \end{aligned}$$

For a lasso regression, from the previous problem, it has the form of $prox_f(w)$ with

$$f(\beta) = \alpha||\beta||_1$$

and the corresponding estimator is

$$x_i = (prox_{\alpha||\cdot||_1}(w))_i = \begin{cases} w_i - sign(w_i)\alpha, & |w_i| \geq \alpha \\ 0, & \text{otherwise} \end{cases}$$

For a ridge regression, from the previous problem, it has the form of $prox_f(w)$ with

$$f(\beta) = \alpha||\beta||_2$$

and the corresponding estimator is

$$x = \frac{y}{2\alpha + 1}$$

Note in this case we have orthonormal rows in X, thus $XX^T = I$, which shows that the OLS estimator for such a matrix X can be

$$\beta_{OLS} = Xy$$

Both ridge and lasso regression find the β which is close to the OLS estimator, while penalize the β that variates too much. Lasso uses l1 norm to add penalty, while ridge uses l2 norm. Lasso will give sparse result while ridge will not.

Problem 2

(a) Set the minimizer of f_x to be y^*, then

$$\begin{aligned} y^* &= arg\min_y (f(x) + \nabla f(x)^T(y-x) + \frac{1}{2\alpha}||y-x||_2^2) \\ &= arg\min_y (\frac{1}{2\alpha}||y-x+\alpha\nabla f(x)||_2^2) \end{aligned}$$

This is a convex function and it reaches its minimum 0 when

$$0 \in \partial(||y-x+\alpha\nabla f(x)||_2^2)$$

Thus,

$$y^* = x - \alpha\nabla f(x)$$

(b) For this composite function

$$h(x) = f_1(x) + f_2(x)$$

Then the minimizer of $h(x)$ can be written as

$$\begin{aligned} x^* &= arg\min_x h(x) \\ &= arg\min_x f_1(x) + f_2(x) \end{aligned}$$

If we have $f_2(x) = 0$, then x^* is the gradient descent update

$$x^* = x - \alpha\nabla f_1(x) = arg\min_x f_1(z) + \nabla f_1(z)^T(x-z) + \frac{1}{2\alpha}||x-z||_2^2$$

Now we have any $f_2(x)$ not differentiable, so

$$\begin{aligned} x^* &= arg\min_x f_1(z) + \nabla f_1(z)^T(x-z) + \frac{1}{2\alpha}||x-z||_2^2 + f_2(x) \\ &= arg\min_x \frac{1}{2\alpha}||x-(z-\alpha\nabla f_1(z))||_2^2 + f_2(x) \end{aligned}$$

If f_2 has a proximal operator that is easy to compute, then we can write the expression as

$$x^* = prox_{\alpha,f_2}(z - \alpha\nabla f_1(z))$$

In gradient descent update, to make the symbol clearer we can write as

$$x^{k+1} = prox_{\alpha,f_2}(x^k - \alpha\nabla f_1(x^k))$$

(c) If x^* is the optimal solution to the optimization problem, then

$$0 \in \nabla f_1(x^*) + \partial f_2(x^*)$$

note ∂ represents the subgradient here

Note x^* is a solution to

$$\min_x \frac{1}{2\alpha}||x-(x^*-\alpha\nabla f_1(x^*))||_2^2+f_2(x)$$

and

$$x^*=prox_{\alpha,f_2}(x^*-\alpha\nabla f(x^*))$$

Then for x^* to be a solution of this problem, we should have $0\in\alpha\nabla f_1(x^*)+\alpha\partial f_2(x^*)$

So the 2 conditions are equivalent if $\alpha>0$.

Problem 3

(a) The proximal gradient problem is equivalent to solve problem

$$arg \min_{\beta} f(\beta) = arg \min_{\beta} f_1(\beta) + f_2(\beta)$$

with

$$f_1(\beta) = \frac{1}{2}||y - X\beta||_2^2$$

which is convex and differentiable and

$$f_2(\beta) = \lambda|\beta|_1$$

which is convex but not differentiable everywhere

Thus the problem has gradient update formula

$$\begin{aligned}\beta_{k+1} &= prox_{\alpha, f_2(.)}(\beta_k - \alpha\nabla f_1(\beta_k)) \\ &= S_{\alpha,\lambda}(\beta_k - \alpha\nabla f_1(\beta_k))\end{aligned}$$

S, similar to the problem statement in 1(b), is the soft-thresholding operator and the gradient

$$\nabla f_1(\beta_k) = X^T(X\beta_k - y)$$

(b) To check whether this expression reaches an optimum, we can check if this is a convex function. Then if it is, it reaches the optimum at a vector x if and only if the 0 vector is a subgradient at x.

$$0 \in \nabla f_1(x) + g_{f_2}$$

One way to account for numerical inaccuracy is that instead of let strictly 0 vector in the subgradient, we can add tolerance to the calculation. The tolerance limits the acceptable precision of the result or gives a range around 0 for the acceptable result. In this problem, we can calculate the subgradient, and check if the absolute result of the subgradient falls within the tolerance.

$$X(X^T\beta - y) + \lambda g_{f_2} = 0$$

$$g_{f_2} = \frac{1}{\lambda}X(X^T\beta - y)$$

These 2 expressions may be used for this problem, with g_{f_2} the sign of β.

(c)

pgd_lasso-question

March 6, 2021

```
[14]: %matplotlib inline
      import numpy as np
      from matplotlib import pyplot as plt
```

0.0.1 Utility functions

```
[15]: def obj(w):
          ## calculates the obj functions
          r = X*w-y;
          return np.sum(np.multiply(r,r))/2 +  lamda * np.sum(np.abs(w))
```

0.1 Data

```
[16]: np.random.seed(50)

      N = 100
      dim = 30
      lamda = 1/np.sqrt(N);

      w = np.zeros(dim)
      n_nonzero = 15
      w[np.random.choice(range(dim), n_nonzero, False)] = np.random.randn(n_nonzero)
      w = np.matrix(w.reshape(-1, 1))

      X = np.matrix(np.random.multivariate_normal([0.0]*dim, np.eye(dim), size = N))
      y = X*w
```

Our objective function of interest is:

$$\frac{1}{2}\|Xw - y\|^2 + \lambda|w|_1$$

In the cell above, the variables X, y, w and lamda corresponds to X, y, w and λ in the equation above.

```
[17]: opt = obj(w)
      print('Optimal Objective Function Value: ', opt)
```

```
Optimal Objective Function Value:  1.3043384900597284
```

0.2 Optimal Value using SKLearn

[18]:
```
from sklearn import linear_model
clf = linear_model.Lasso(alpha=lamda / N, fit_intercept = False)
clf.fit(X, y)
```

[18]:
```
Lasso(alpha=0.001, fit_intercept=False)
```

[19]:
```
print('SKLearn obj val: ', obj(clf.coef_.reshape(-1, 1)) )
```

```
SKLearn obj val:  1.303641803846212
```

0.3 Proximal Gradient

[20]:
```
max_iter = 100 # max number of iterations of proximal gradient method
```

[21]:
```
alpha_array = np.logspace(-5, -2, num = 10, base = 10.0) #range over which you␣
 ↪search hyperparam
```

[22]:
```
def g_prox(x, step_size):
    """
    L1 regularization
    """
    return np.fmax(x - step_size * alpha, 0) - np.fmax(- x - step_size * alpha,␣
 ↪0)
```

[23]:
```
def f_prime(x, b, y):
    return x.T @ (x @ b - y)
```

[24]:
```
## Proximal Gadient

obj_pg = {} #stores obj function value as a function of iteration for each alpha
w_pg = {} #stores the final weight vector learned for each alpha

tol = 0.01

for alpha in alpha_array:
    print('Alpha: ', alpha)

    w_pg[alpha] = np.matrix([0.0]*dim).T
    obj_pg[alpha] = []


    for t in range(0, max_iter):
        obj_val = obj(w_pg[alpha])
        obj_pg[alpha].append(obj_val.item())

        ## fill in your code
        ## be sure to include your stopping condition
        grad_fk = f_prime(X, w_pg[alpha], y)
        wk_grad = w_pg[alpha] - alpha * grad_fk
        prx = g_prox(wk_grad, alpha)
        w_pg[alpha] = prx

        sign = np.zeros(shape=(len(prx), 1))
        for elem in range(len(prx)):
            if prx[elem] > 0:
                sign[elem] = 1
            elif prx[elem] < 0:
                sign[elem] = -1
        if np.add(grad_fk, lamda * sign).all() < tol:
            print("Achieved relative tolerance at iteration %s" % t)
            break

        if (t%10==0):
            print('iter= {},\tobjective= {:3f}'.format(t, obj_val.item()))
```

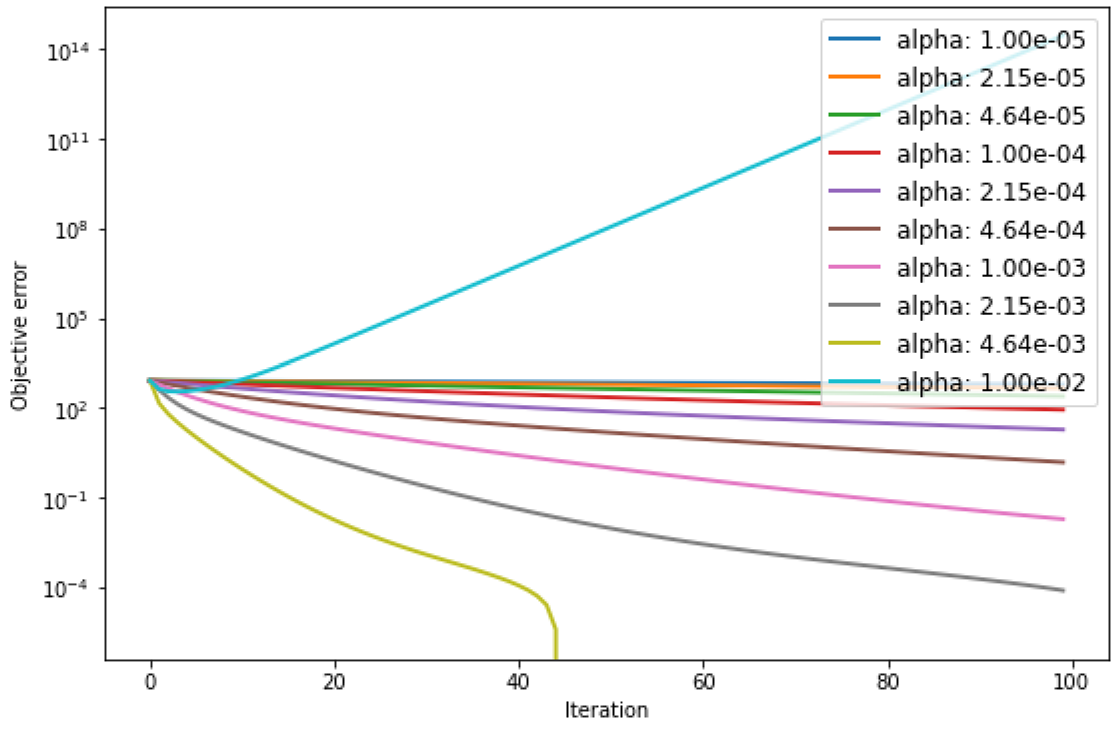
```
Alpha:  1e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 807.827723
iter= 20,       objective= 784.858060
iter= 30,       objective= 762.639041
iter= 40,       objective= 741.144378
iter= 50,       objective= 720.348736
iter= 60,       objective= 700.227700
iter= 70,       objective= 680.757744
iter= 80,       objective= 661.916192
iter= 90,       objective= 643.681189
Alpha:  2.1544346900318823e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 781.331858
iter= 20,       objective= 734.561707
iter= 30,       objective= 691.008037
iter= 40,       objective= 650.433717
iter= 50,       objective= 612.619760
iter= 60,       objective= 577.363910
iter= 70,       objective= 544.479326
iter= 80,       objective= 513.793383
iter= 90,       objective= 485.146564
Alpha:  4.641588833612782e-05
iter= 0,        objective= 831.575313
iter= 10,       objective= 727.433455
iter= 20,       objective= 638.133069
iter= 30,       objective= 561.416201
iter= 40,       objective= 495.380805
iter= 50,       objective= 438.423409
```

```
iter= 60,       objective= 389.191302
iter= 70,       objective= 346.542337
iter= 80,       objective= 309.510795
iter= 90,       objective= 277.279950
Alpha:  0.0001
iter= 0,        objective= 831.575313
iter= 10,       objective= 624.748521
iter= 20,       objective= 475.808645
iter= 30,       objective= 367.531386
iter= 40,       objective= 287.993498
iter= 50,       objective= 228.906114
iter= 60,       objective= 184.481708
iter= 70,       objective= 150.658630
iter= 80,       objective= 124.570515
iter= 90,       objective= 104.182049
Alpha:  0.00021544346900318823
iter= 0,        objective= 831.575313
iter= 10,       objective= 454.378222
iter= 20,       objective= 265.577994
iter= 30,       objective= 165.989038
iter= 40,       objective= 110.232907
iter= 50,       objective= 77.004301
iter= 60,       objective= 55.974355
iter= 70,       objective= 41.937197
iter= 80,       objective= 32.146807
iter= 90,       objective= 25.079156
Alpha:  0.00046415888336127773
iter= 0,        objective= 831.575313
iter= 10,       objective= 241.094828
iter= 20,       objective= 95.958464
iter= 30,       objective= 47.808779
iter= 40,       objective= 27.010517
iter= 50,       objective= 16.390989
iter= 60,       objective= 10.457647
iter= 70,       objective= 6.977412
iter= 80,       objective= 4.876632
iter= 90,       objective= 3.583209
Alpha:  0.001
iter= 0,        objective= 831.575313
iter= 10,       objective= 81.019724
iter= 20,       objective= 22.099363
iter= 30,       objective= 8.286735
iter= 40,       objective= 3.865040
iter= 50,       objective= 2.299016
iter= 60,       objective= 1.709656
iter= 70,       objective= 1.476937
iter= 80,       objective= 1.381248
iter= 90,       objective= 1.340408
```

```
Alpha:  0.002154434690031882
iter= 0,        objective= 831.575313
iter= 10,       objective= 17.158520
iter= 20,       objective= 2.982339
iter= 30,       objective= 1.538040
iter= 40,       objective= 1.345306
iter= 50,       objective= 1.313666
iter= 60,       objective= 1.307106
iter= 70,       objective= 1.305362
iter= 80,       objective= 1.304776
iter= 90,       objective= 1.304527
Alpha:  0.004641588833612777
iter= 0,        objective= 831.575313
iter= 10,       objective= 2.203964
iter= 20,       objective= 1.322278
iter= 30,       objective= 1.305567
iter= 40,       objective= 1.304452
iter= 50,       objective= 1.304277
iter= 60,       objective= 1.304275
iter= 70,       objective= 1.304275
iter= 80,       objective= 1.304275
iter= 90,       objective= 1.304275
Alpha:  0.01
iter= 0,        objective= 831.575313
iter= 10,       objective= 904.886576
iter= 20,       objective= 13936.982306
iter= 30,       objective= 278153.963288
iter= 40,       objective= 5630707.199543
iter= 50,       objective= 114069346.474656
iter= 60,       objective= 2310992861.402610
iter= 70,       objective= 46819976227.000832
iter= 80,       objective= 948558929396.402710
iter= 90,       objective= 19217529660282.031250
```

[25]:
```
## Plot objective error vs. iteration (log scale)

fig, ax = plt.subplots(figsize = (9, 6))

for alpha in alpha_array:
    plt.semilogy(np.array(obj_pg[alpha])-opt,  linewidth = 2, label = 'alpha:␣
 ↪'+'{:.2e}'.format(alpha) )
plt.legend(prop={'size':12})
plt.xlabel('Iteration')
plt.ylabel('Objective error')
```

[25]:
```
Text(0, 0.5, 'Objective error')
```



From this plot, we can see as alpha remains at a very small value, the algorithm converges slowly and the objective error can hardly decrease as the iteration number increases. When alpha/step size becomes larger, the error decreases faster or the algorithm converges faster as iteration number increases. However, if we have too large a step size, i.e. in this case when alpha=0.01, the algorithm diverges.

0.4 Visualize Coefficients

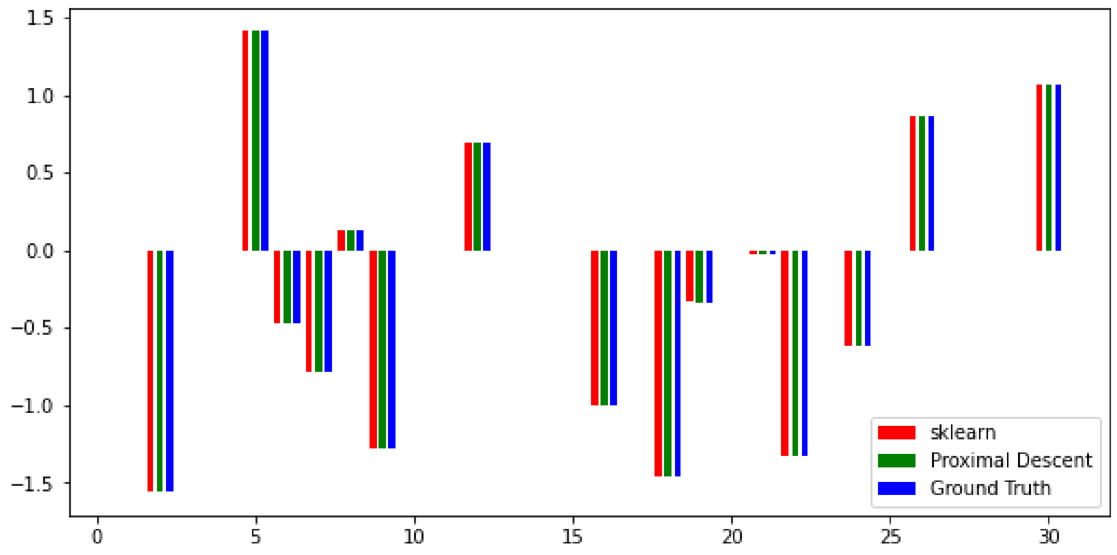
pick the coefficient corresponding to alpha value with the minimum objective function value

[34]:
```
min_obj= np.inf
min_alpha = None
```

[35]:
```
for alpha in alpha_array:
    if obj_pg[alpha][-1] < min_obj:
        min_alpha = alpha
        min_obj = obj_pg[alpha][-1]
```

[36]:
```
plt.figure(figsize = (10, 5))

ax = plt.subplot(111)

x = np.arange(1, dim+1)
```

```
ax.bar(x-0.3, clf.coef_, width=0.2, color='r', align='center', label =␣
↪'sklearn')
ax.bar(x, np.ravel(np.array(w_pg[min_alpha])), width=0.2, color='g',␣
↪align='center', label = 'Proximal Descent')
ax.bar(x+0.3, np.ravel(np.array(w)), width=0.2, color='b', align='center',␣
↪label = 'Ground Truth')

plt.legend()

plt.show()
```



This plot shows the comparison of coefficients from 3 different methods. Proximal descent result of the smallest obj value and sklearn both give close result to the ground truth. Notably, because we stop the proximal descent algorithm when the subgradient is close to 0 within an 1% error, the minimal objective value here corresponds to the best estimated weight.

Problem 4

1st Plot: Lasso: n=340

In this plot, the n in the title is the number of training instances. The x-axis represents the value of hyperparameter λ. The line of symbol 'o' represents the validation error against λ while the line of symbol 'x' the training error. Other lines on the plot are the coefficients from lasso regression along the algorithm path. We can see initially as λ increases, the training error increases and validation error decreases because we add a little bit bias to trade for decreasing variance. However, as λ becomes too large, the model becomes very bias, and thus both errors increases. The errors show horizontal segment when the lasso term dominates and the algorithm does not implement. More explanation is in the following part about the coefficients.

At the same time, initially as λ increases, the variation in coefficients decreases. Some coefficients are forced to be 0 as λ increases, with others remain relatively much larger. This is why the regularization is sparse and can help dimension reduction and feature selection. Then, when lambda is very large, which happens at the same time when both training and validation reaches the maximum, all coefficients are forced to be 0. Since all coefficients are forced to be 0 after λ reaches the threshold, the error also stays the same.

2nd Plot

This plot isolates the training error and validation error out from the previous plot to have a clearer graph.

3rd Plot

This plot is the coefficients on the ridge regression path against different λ. Different from lasso, even though ridge regression also shrinks coefficients as λ increases, the coefficients are not forced to be 0.

4th Plot

This graph plots the error against different numbers of features chosen by FSR algorithm. The purple line represents training error and the red represents validation error. Initially both training and validation error decreases as number of features increases. This is reasonable because more features are added for modeling while overfitting has not become a problem yet due to lack of generality. However, as too many features are added into the mode, even though the training error decreases, the validation error increases because of overfitting. Correlated features or features that do not contain much useful information can also cause problems. The decreasing rate of training error and the increasing rate of validation error both decrease because the features that added to the model and the end are the least informative ones.

5th Plot

This plot is the performance of the lasso and ridge estimator on the given temperature data. Both test errors on the data and the 2016 data and the training errors are plotted. For all training error trends, the errors first increase as the number of training data increases, and then keeps on the level. The test errors first decrease and then keep on the same level. Initially when data size is small, more information are added to the training process which give better fit to the test data and more variation to the training. Generally, lasso prediction has better prediction for all data sizes than ridge until the point where the estimators approach the OLS solution.

6th Plot

This plot is the regularization parameter against number of training data. As the number of training data increases, the size of the regularization parameter decreases. When we have smaller data size, the regularization effect is larger because of larger risk of overfitting, and thus larger regularization parameter is more useful.

7th Plot

This plot is the number of features gainst number of training data. As the number of training data increases, the number of features also increases. Since we have more training data, we can fit more features which contain more information or noises.

8th and 9th Plot

The 8th and the 9th plot are the coefficients of the ridge regression and those of the lasso estimators from the given data for different sizes of training set. The 8th plot is the ridge regression and the 9th is the lasso. All coefficients are depicted in light blue except the 3 that have the largest magnitudes. The 9th plot has a much sparser linear model except for large training set. When the size of the training set is very large, both estimators get close to the least squares solution.

10th and 11th Plot

The 10th and the 11th plot are the coefficients against regularization parameter for ridge regression and lasso estimators. The 10th plot is for ridge regression while the 11th plot is for the lasso. All coefficients are depicted in light blue except the 3 that have the largest magnitudes. In the 10th plot, ridge regression shrinks the coefficients as the regularization parameter increases, but the coefficients are not forced to 0, and the decreasing trend is relatively smooth. In the 11th plot, lasso estimator also shrinks coefficients as the regularization parameter increases, but the coefficients are all finally forced to be 0. The result of the lasso is also sparse, with the features with largest coefficients in magnitude are kept relatively large and others are shrunk to 0.

FSR

March 6, 2021

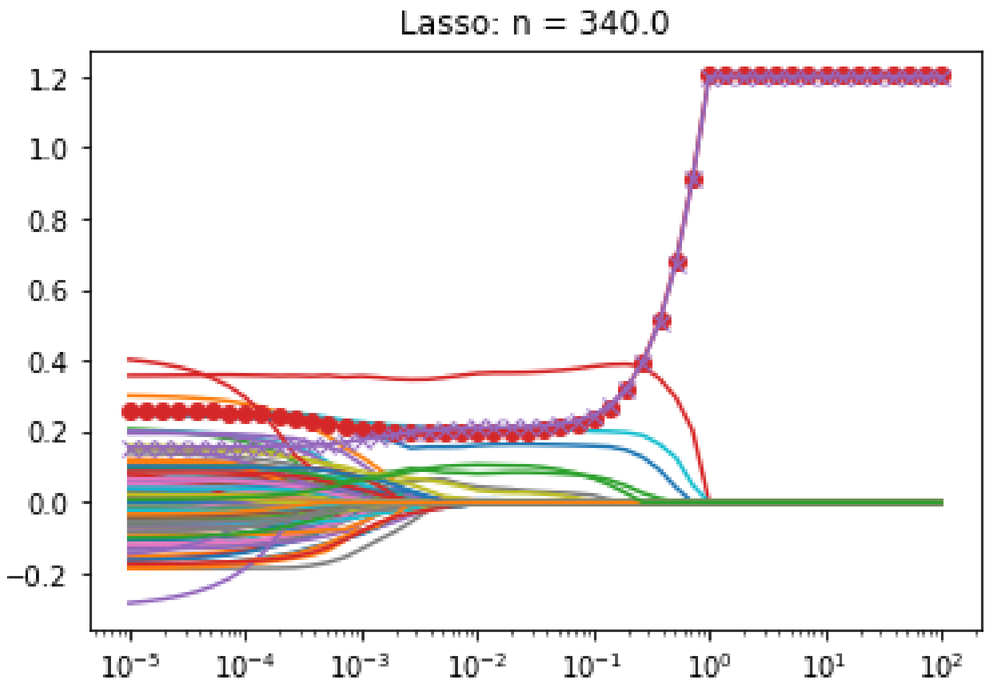
[9]:
```python
import matplotlib.pyplot as plt
import numpy as np
from os import listdir
from sklearn import linear_model
from sklearn.linear_model import lasso_path
from warnings import filterwarnings
filterwarnings('ignore')

np.random.seed(2019)

def extract_temp(file_name,col_ind):
    data_aux = np.loadtxt(file_name, usecols=range(10))
    data = data_aux[:,col_ind]
    err_count = 0
    ind_errs = []
    for ind in range(data.shape[0]):
        if data[ind] > 100 or data[ind] < -100:
            err_count = err_count + 1
            ind_errs.append(ind)
            data[ind] = data[ind-1]
    print("File name: " + file_name)
    print("Errors: " + str(err_count) + " Indices: " + str(ind_errs))
    return data

def create_data_matrix(str_path):
    file_name_list = listdir(str_path)
    file_name_list.sort()
    col_ind = 8 # 8 = last 5 minutes, 9 = average over the whole hour
    data_matrix = []
    ind = 0
    for file_name in file_name_list:
        if file_name[0] == '.':
            continue
        else:
            print("Station " + str(ind))
            ind = ind + 1
            data_aux = extract_temp(str_path + file_name,col_ind)
            if len(data_matrix) == 0:
                data_matrix = data_aux
            else:
                data_matrix = np.vstack((data_matrix,data_aux))
    return data_matrix.T

def FSR(X,y,num_features=20):
    '''
    X: feature matrix
    y: response
    return
        coeff: coefficient betta
    '''
    S = []
    r = y
    coeff = np.zeros(shape=X.shape[1])
    iteration = 1
    position = np.linspace(0, X.shape[1]-1, num = X.shape[1])
    while iteration <= num_features:
        i_val = np.empty(shape=len(position), dtype=float)
        for i in range(len(position)):
            i_val[i] = np.linalg.norm(X[:, i].T @ r)
        i_star = np.argmax(i_val)
        S.append(int(position[i_star]))
        position = np.delete(position, i_star)
        beta = np.linalg.inv(X[:, S].T @ X[:, S]) @ X[:, S].T @ y
        r = y - X[:, S] @ beta
        for elem in range(len(S)):
            coeff[S[elem]] = beta[elem]
        iteration += 1
    return coeff
```

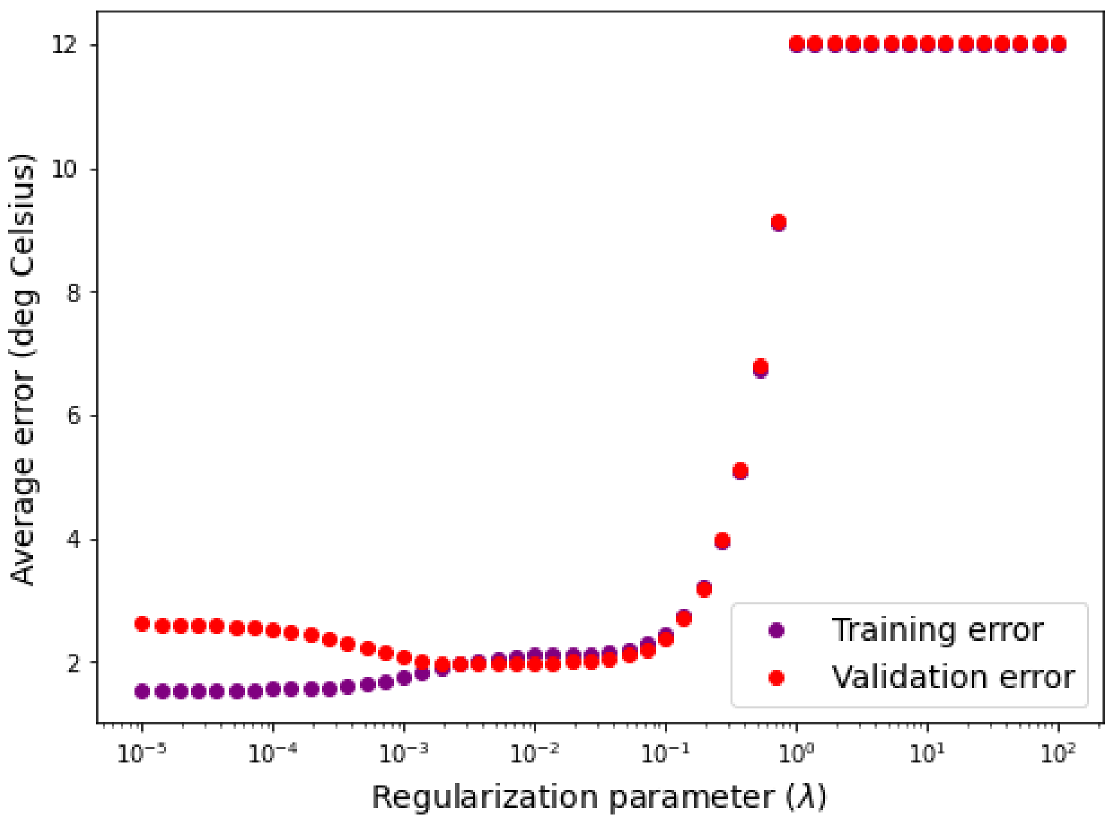
[10]:
```
str_path = "./weather/"

load_files = False
if load_files:
    str_path_2015 = str_path + "hourly/2015/"
    data_matrix = create_data_matrix(str_path_2015)
    str_path_2016 = str_path + "hourly/2016/"
    data_matrix_2016 = create_data_matrix(str_path_2016)
else:
    data_matrix = np.load(str_path +"hourly_temperature_2015.npy")
    data_matrix_2016 = np.load(str_path +"hourly_temperature_2016.npy")

file_name_list = listdir(str_path + "hourly/2015/")
file_name_list.sort()
```

```
ind_response = 78 # 53 = Manhattan, 18 = Troy has 2 correlated features
# 23 = Williams dense linear model 30 = Death Valley 16 = AL_Talladega_10_NNE␣
 ↪good for elastic net
# 78 = ND Jamestown also good for enet
print("Response is " + str(file_name_list[ind_response]))
y_raw = data_matrix[:,ind_response]
y_2016 = data_matrix_2016[:,ind_response]
ind_X = np.hstack((np.arange(0,ind_response),np.
 ↪arange(ind_response+1,data_matrix.shape[1])))
X_raw = data_matrix[:,ind_X]
X_2016 = data_matrix_2016[:,ind_X]
n_features = X_raw.shape[1]

n_train_values = 40
n_test = int(1e3)
n_val = int(1e3)
n_train_max = data_matrix.shape[0] - n_test # 5e2
n_train_min = 133
n_train_list = np.around(np.logspace(np.log10(n_train_min),np.
 ↪log10(n_train_max),n_train_values))
train_error_lasso_vec = []
val_error_lasso_vec = []
test_error_lasso_vec = []
test_2016_lasso_vec = []
train_error_ridge_vec = []
val_error_ridge_vec = []
test_error_ridge_vec = []
test_2016_ridge_vec = []

train_error_fsr_vec = []
val_error_fsr_vec = []
test_error_fsr_vec = []
test_2016_fsr_vec = []

coeffs_lasso_matrix = np.zeros((n_features,len(n_train_list)))
coeffs_ridge_matrix = np.zeros((n_features,len(n_train_list)))
coeffs_fsr_matrix = np.zeros((n_features,len(n_train_list)))
lambda_lasso_vec = []
lambda_ridge_vec = []
n_features_fsr_vec = []

n_lambda = 50
lambdas_ridge_aux = np.logspace(-5, 2, n_lambda)
lambdas_lasso = np.logspace(-5, 2, n_lambda)
eps = 1e-5  # the smaller it is the longer is the path
fixed_n = n_train_list[9]
```

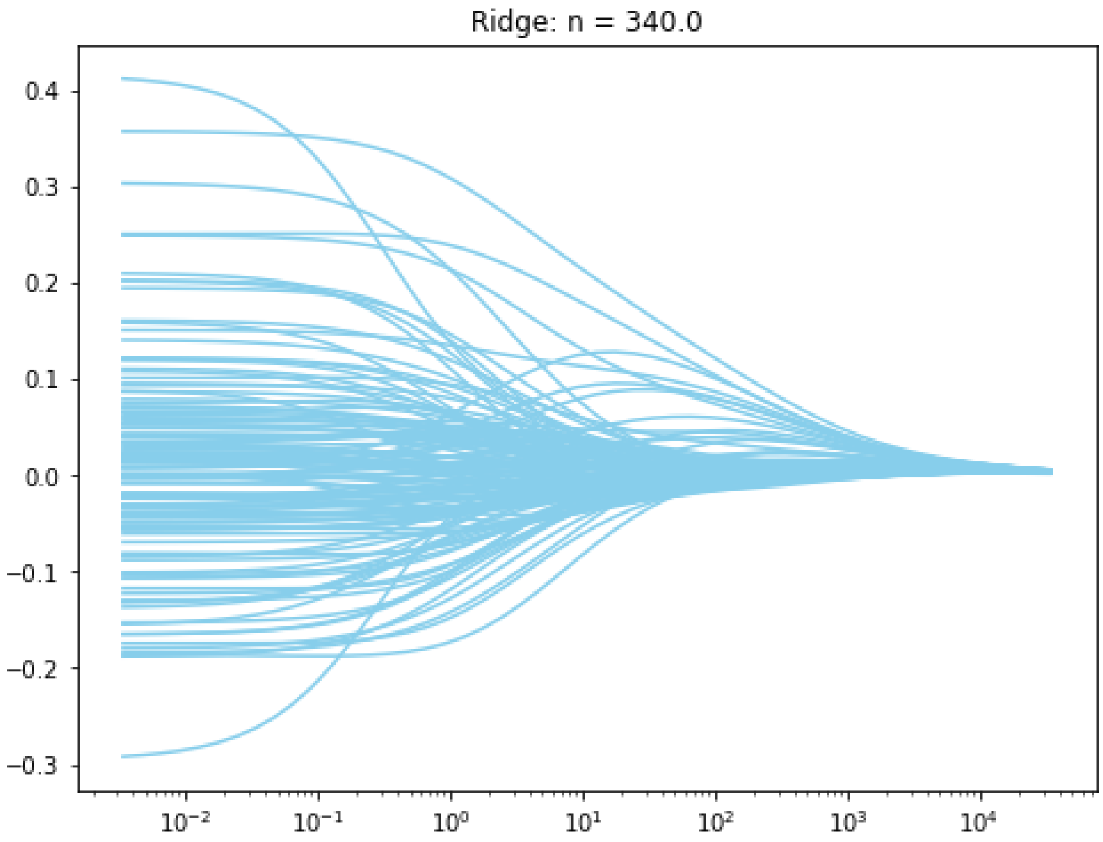
```
i_m = 0
for n_train in n_train_list:
    lambdas_ridge = lambdas_ridge_aux * n_train
    aux_ind = np.random.permutation(range(data_matrix.shape[0]))
    ind_test = aux_ind[:n_test]
    ind_val = aux_ind[n_test:(n_test+n_val)]
    X_test = X_raw[ind_test,:]
    y_test = y_raw[ind_test]
    X_val = X_raw[ind_val,:]
    y_val = y_raw[ind_val]
    ind_train = aux_ind[(n_test+n_val):int(n_test+n_val+n_train)]
    X_train = X_raw[ind_train,:]
    y_train = y_raw[ind_train]

    center_vec = X_train.mean(axis=0)
    X_train_centered = X_train - center_vec
    col_norms = np.linalg.norm(X_train_centered, axis=0) / np.sqrt(n_train)
    X_train_norm = np.true_divide(X_train_centered, col_norms)
    X_test_centered = X_test - center_vec
    X_test_norm = np.true_divide(X_test_centered, col_norms)
    X_val_centered = X_val - center_vec
    X_val_norm = np.true_divide(X_val_centered, col_norms)
    X_2016_centered = X_2016 - center_vec
    X_2016_norm = np.true_divide(X_2016_centered, col_norms)
    y_train_center = y_train.mean()
    y_train_centered = y_train - y_train_center
    norm_y_train = np.linalg.norm(y_train_centered) / np.sqrt(n_train)
    y_train_norm = y_train_centered / norm_y_train

    print("Computing regularization path using the lasso...")
    lambdas_lasso, coeffs_lasso, _ = lasso_path(X_train_norm, y_train_norm,␣
↪eps,  normalize=False,
                                          alphas = lambdas_lasso,␣
↪fit_intercept=False)

    min_error_val = 1e4
    i_lambda = 0
    val_error_lasso_lambdas = np.zeros(n_lambda)
    train_error_lasso_lambdas = np.zeros(n_lambda)
    for i, coeffs_est in enumerate(coeffs_lasso.T):
        y_val_lasso = norm_y_train * np.dot(X_val_norm, coeffs_est) +␣
↪y_train_center
        error_val = np.linalg.norm(y_val - y_val_lasso) / np.sqrt(len(y_val))
        # error_val_vec.append(error_val)
        if error_val < min_error_val:
            min_error_val = error_val
            lambda_best_lasso = lambdas_lasso[i]
            coeffs_lasso_best = coeffs_est
        y_train_lasso = norm_y_train * np.dot(X_train_norm, coeffs_est) +␣
↪y_train_center
        train_error_lasso_lambdas[i] =  np.linalg.norm(y_train - y_train_lasso␣
↪) / np.sqrt(len(y_train))
        val_error_lasso_lambdas[i] = error_val
    lambda_lasso_vec.append(lambda_best_lasso)
    y_train_lasso = norm_y_train * np.dot(X_train_norm, coeffs_lasso_best) +␣
↪y_train_center
    y_test_lasso = norm_y_train * np.dot(X_test_norm, coeffs_lasso_best) +␣
↪y_train_center
    y_2016_lasso = norm_y_train * np.dot(X_2016_norm, coeffs_lasso_best) +␣
↪y_train_center
    train_error_lasso = np.linalg.norm(y_train - y_train_lasso) / np.
↪sqrt(len(y_train))
    test_error_lasso = np.linalg.norm(y_test - y_test_lasso) / np.
↪sqrt(len(y_test))
    test_2016_lasso = np.linalg.norm(y_2016 - y_2016_lasso) / np.
↪sqrt(len(y_2016))
    train_error_lasso_vec.append(train_error_lasso)
    val_error_lasso_vec.append(min_error_val)
    test_error_lasso_vec.append(test_error_lasso)
    test_2016_lasso_vec.append(test_2016_lasso)
    coeffs_lasso_matrix[:,i_m] = coeffs_lasso_best


    if n_train == fixed_n:
        coeffs_lasso_fixed = np.copy(coeffs_lasso)
        plt.figure()
        plt.plot(lambdas_lasso,coeffs_lasso.T)
        plt.plot(lambdas_lasso,val_error_lasso_lambdas/10.,marker='o')
        plt.plot(lambdas_lasso,train_error_lasso_lambdas/10.,marker='x')
        plt.xscale('log')
        plt.title('Lasso: n = ' + str(n_train))

        plt.figure(figsize=(8,6))
        plt.
↪plot(lambdas_lasso,train_error_lasso_lambdas,marker='o',linestyle='none',color='purple',lab
↪error')
        plt.
↪plot(lambdas_lasso,val_error_lasso_lambdas,marker='o',linestyle='none',color='red',label='V
↪error')
        plt.xlabel(r"Regularization parameter ($\lambda$)",fontsize=14)
        plt.ylabel('Average error (deg Celsius)',fontsize=14)
        plt.gcf().subplots_adjust(bottom=0.15)
        plt.xscale('log')
        plt.legend(loc = 'lower right',fontsize=14)


    # ridge regression
    clf = linear_model.Ridge(fit_intercept=False, normalize=False,)
    min_error_val = 1e3
    lambda_best = 0
    coeffs_ridge = np.zeros((n_features,n_lambda))
    val_error_ridge_lambdas = np.zeros(n_lambda)
    for ind_a,a in enumerate(lambdas_ridge):
        # print "lambda: " + str(a)
        clf.set_params(alpha=a)
        clf.fit(X_train_norm, y_train_norm)
        coeffs_ridge[:,ind_a] = clf.coef_
        y_val_ridge = norm_y_train * np.dot(X_val_norm, clf.coef_) +␣
↪y_train_center
        error_val = np.linalg.norm(y_val - y_val_ridge) / np.sqrt(len(y_val))
        val_error_ridge_lambdas[ind_a] = error_val
        # error_val_vec.append(error_val)
        if error_val < min_error_val:
            min_error_val = error_val
            lambda_best = a
            coeffs_ridge_best = clf.coef_
    lambda_ridge_vec.append(lambda_best)
    y_train_ridge = norm_y_train * np.dot(X_train_norm, coeffs_ridge_best) +␣
↪y_train_center
    y_test_ridge = norm_y_train * np.dot(X_test_norm, coeffs_ridge_best) +␣
↪y_train_center
    y_2016_ridge = norm_y_train * np.dot(X_2016_norm, coeffs_ridge_best) +␣
↪y_train_center
    train_error_ridge = np.linalg.norm(y_train - y_train_ridge) / np.
↪sqrt(len(y_train))
    test_error_ridge = np.linalg.norm(y_test - y_test_ridge) / np.
↪sqrt(len(y_test))
    test_2016_ridge = np.linalg.norm(y_2016 - y_2016_ridge) / np.
↪sqrt(len(y_2016))
    train_error_ridge_vec.append(train_error_ridge)
    val_error_ridge_vec.append(min_error_val)
    test_error_ridge_vec.append(test_error_ridge)
    test_2016_ridge_vec.append(test_2016_ridge)
    coeffs_ridge_matrix[:,i_m] = coeffs_ridge_best

    if n_train == fixed_n:
        coeffs_ridge_fixed = np.copy(coeffs_ridge)
        plt.figure(figsize=(8,6))
        # plt.plot(lambdas_ridge,val_error_lasso_lambdas/10.,marker='o')
        for ind in range(n_features):
            plt.plot(lambdas_ridge,coeffs_ridge[ind,:],color='skyblue')
            plt.xscale('log')
            plt.title('Ridge: n = ' + str(n_train))


    # FSR
    min_error_val = 1e3
    n_features_best_fsr = 0
    n_feature = X_train_norm.shape[1]
    coeffs_fsr = np.zeros((X_train_norm.shape[1],X_train_norm.shape[1]))
    val_error_fsr_lambdas = np.zeros(n_feature)
    train_error_fsr_lambdas = np.zeros(n_feature)
    for i,a in enumerate(range(1,n_feature)):
        coeffs_fsr = FSR(X_train_norm,y_train_norm,num_features=a)
        y_train_fsr = norm_y_train * np.dot(X_train_norm, coeffs_fsr) +␣
↪y_train_center
        y_val_fsr = norm_y_train * np.dot(X_val_norm, coeffs_fsr) +␣
↪y_train_center
        error_val = np.linalg.norm(y_val - y_val_fsr) / np.sqrt(len(y_val))
    #         error_val_vec.append(error_val)
        if error_val < min_error_val:
            min_error_val = error_val
            n_features_best_fsr = a
            coeffs_fsr_best = coeffs_est

        train_error_fsr_lambdas[i] =  np.linalg.norm(y_train - y_train_fsr ) /␣
↪np.sqrt(len(y_train))
        val_error_fsr_lambdas[i] = error_val
    n_features_fsr_vec.append(n_features_best_fsr)

    y_train_fsr = norm_y_train * np.dot(X_train_norm, coeffs_fsr_best) +␣
↪y_train_center

    y_train_fsr = norm_y_train * np.dot(X_train_norm, coeffs_fsr_best) +␣
↪y_train_center
    y_test_fsr = norm_y_train * np.dot(X_test_norm, coeffs_fsr_best) +␣
↪y_train_center
    y_2016_fsr = norm_y_train * np.dot(X_2016_norm, coeffs_fsr_best) +␣
↪y_train_center
    train_error_fsr = np.linalg.norm(y_train - y_train_fsr) / np.
↪sqrt(len(y_train))
    test_error_fsr = np.linalg.norm(y_test - y_test_fsr) / np.sqrt(len(y_test))
    test_2016_fsr = np.linalg.norm(y_2016 - y_2016_fsr) / np.sqrt(len(y_2016))
    train_error_fsr_vec.append(train_error_fsr)
    val_error_fsr_vec.append(min_error_val)
    test_error_fsr_vec.append(test_error_fsr)
    test_2016_fsr_vec.append(test_2016_fsr)

    coeffs_fsr_matrix[:,i_m] = coeffs_fsr_best

    if n_train == fixed_n:
        plt.figure(figsize=(8,6))
        plt.
↪plot(range(len(train_error_fsr_lambdas)),train_error_fsr_lambdas,marker='o',linestyle='none
↪error')
        plt.
↪plot(range(len(val_error_fsr_lambdas)),val_error_fsr_lambdas,marker='o',linestyle='none',co
↪error')
        plt.xlabel(r"Number of features",fontsize=14)
        plt.ylabel('Average error (deg Celsius)',fontsize=14)
        plt.gcf().subplots_adjust(bottom=0.15)
        plt.legend(loc = 'lower right',fontsize=14)

    i_m = i_m + 1
```

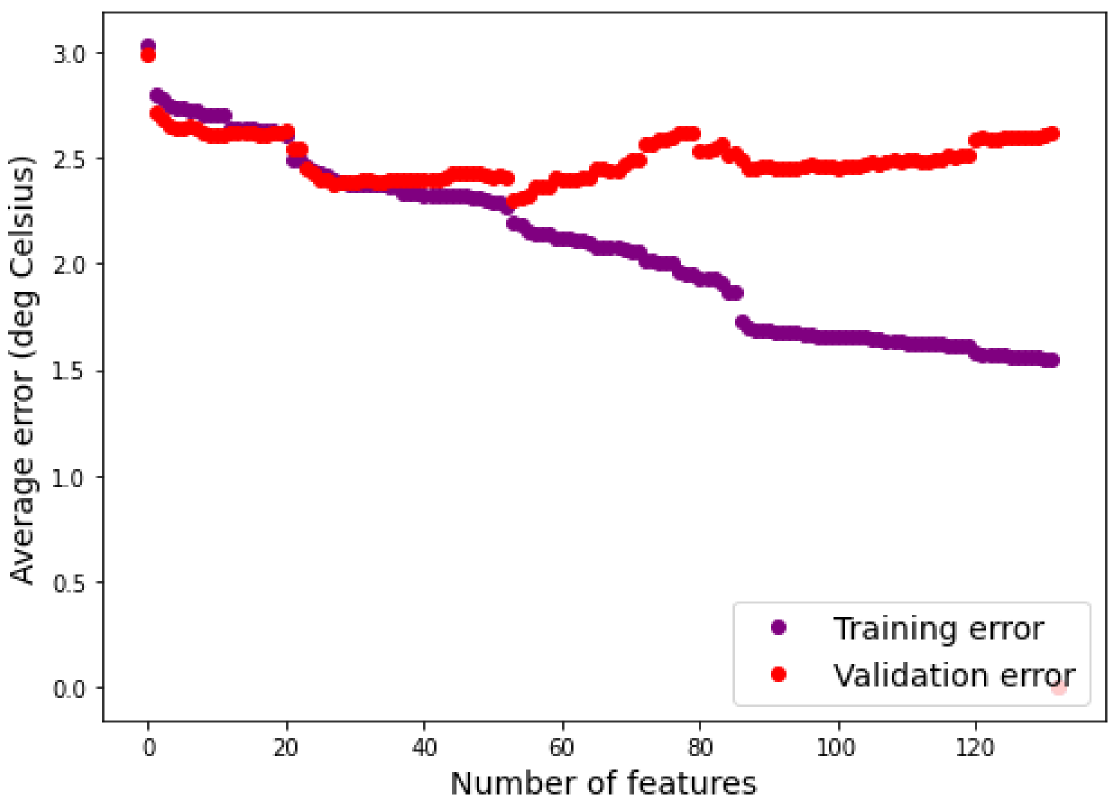
```
Response is CRNH0203-2015-ND_Medora_7_E.txt
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
Computing regularization path using the lasso…
```





12
10
8
6
4
2
Average error (deg Celsius)
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Regularization parameter (λ)
Training error
Validation error

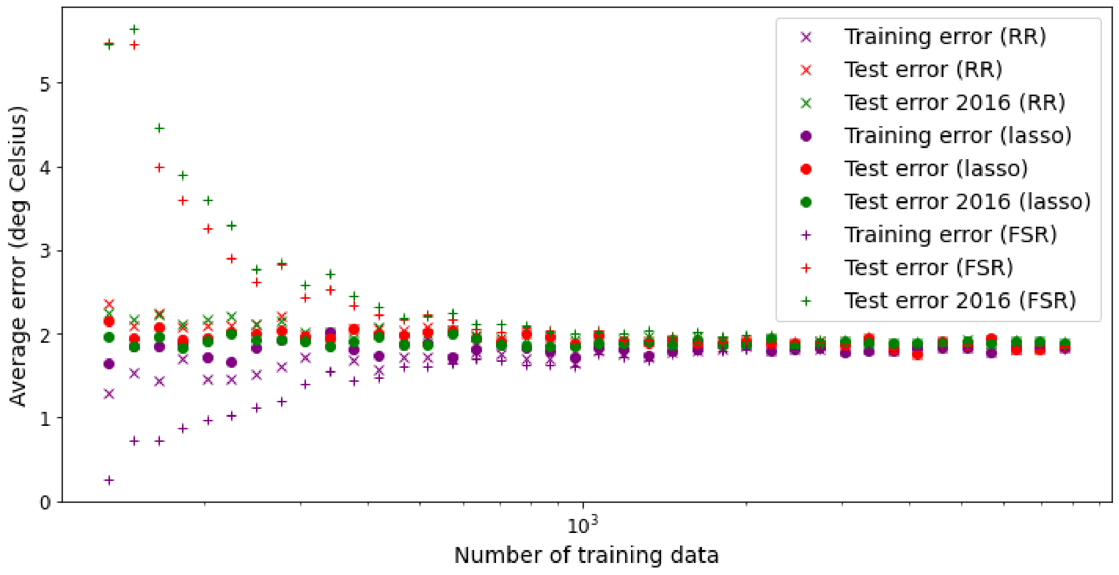

Ridge: n = 340.0
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}

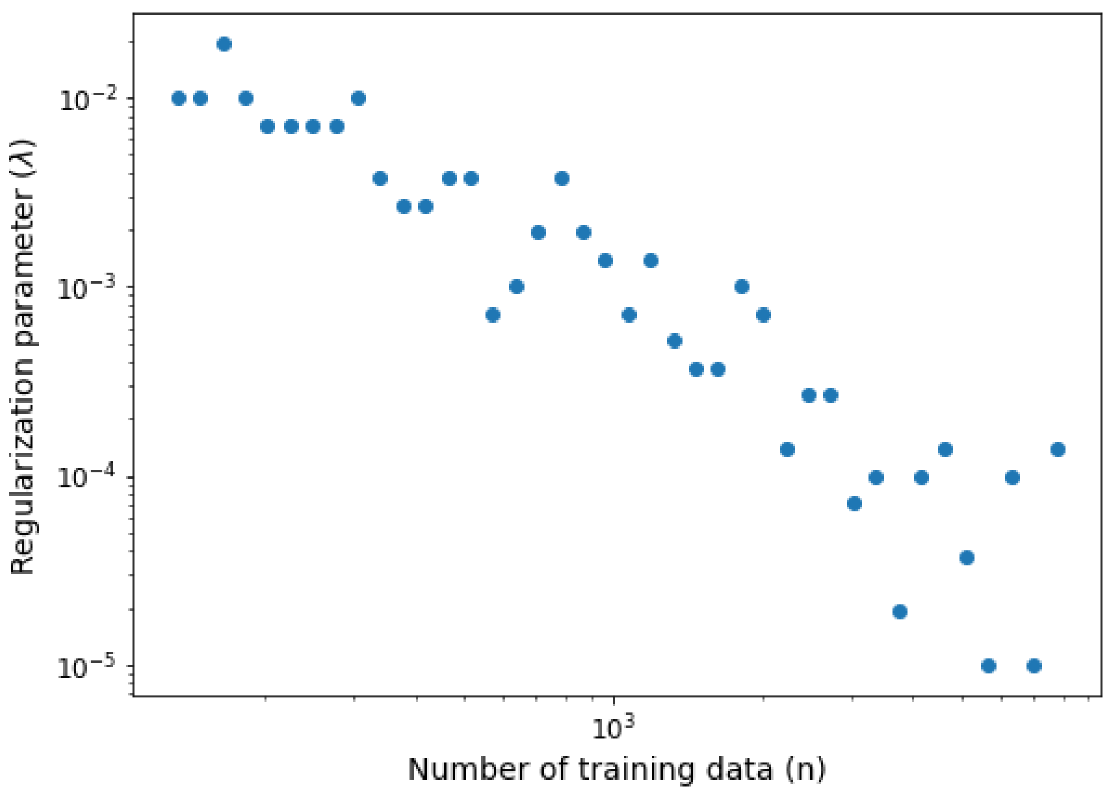


```
[11]: # xticks=(np.array([200,500,1000,2000,5000]))
      # xtick_labels = ('200','500','1000','2000','5000')
      plt.figure(figsize=(12,6))
      linstyle = 'None'
      plt.
       ↪plot(n_train_list,train_error_ridge_vec,linestyle=linstyle,marker='x',color='purple',label=
       ↪error (RR)")
      plt.
       ↪plot(n_train_list,test_error_ridge_vec,linestyle=linstyle,marker='x',color='red',label="Tes
       ↪error (RR)")
      plt.
       ↪plot(n_train_list,test_2016_ridge_vec,linestyle=linstyle,marker='x',color='green',label="Te
       ↪error 2016 (RR)")
      plt.
       ↪plot(n_train_list,train_error_lasso_vec,linestyle=linstyle,marker='o',color='purple',label=
       ↪error (lasso)")
      plt.
       ↪plot(n_train_list,test_error_lasso_vec,linestyle=linstyle,marker='o',color='red',label="Tes
       ↪error (lasso)")
```

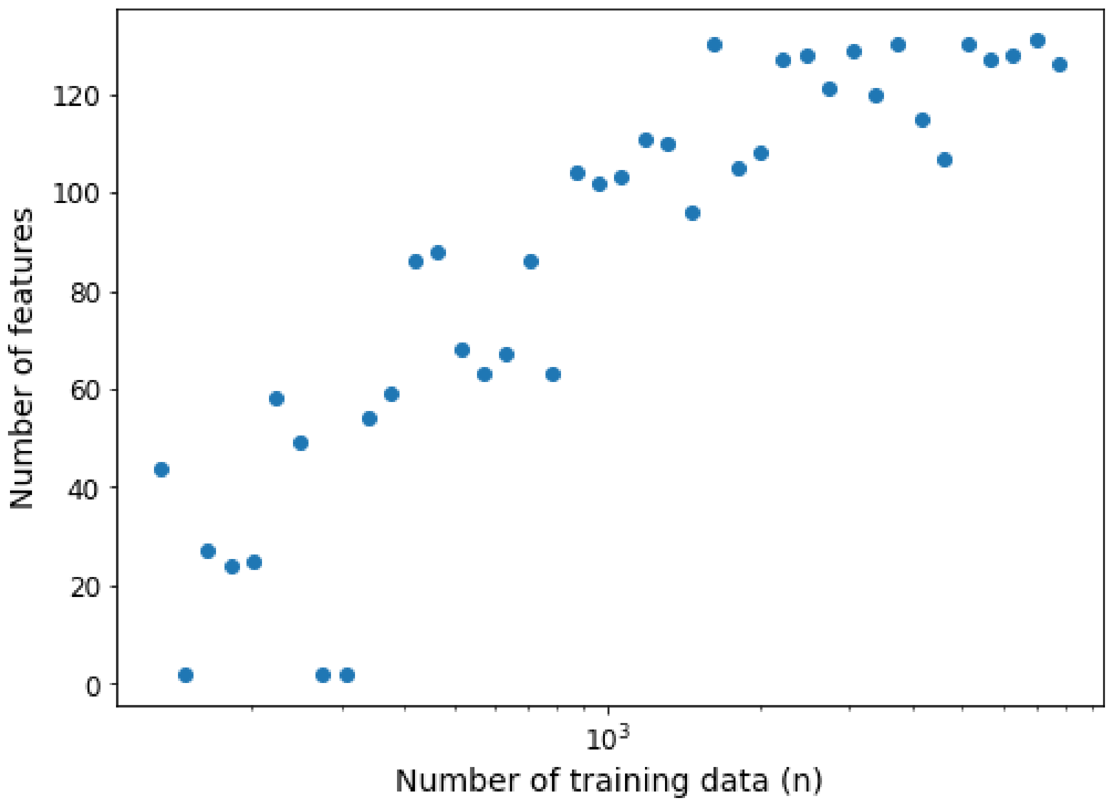
```
plt.
 ↪plot(n_train_list,test_2016_lasso_vec,linestyle=linstyle,marker='o',color='green',label="Te
 ↪error 2016 (lasso)")
plt.
 ↪plot(n_train_list,train_error_fsr_vec,linestyle=linstyle,marker='+',color='purple',label="T
 ↪error (FSR)")
plt.
 ↪plot(n_train_list,test_error_fsr_vec,linestyle=linstyle,marker='+',color='red',label="Test␣
 ↪error (FSR)")
plt.
 ↪plot(n_train_list,test_2016_fsr_vec,linestyle=linstyle,marker='+',color='green',label="Test
 ↪error 2016 (FSR)")
# plt.ylim((-0.5,7))
plt.xscale('log')
plt.xlabel('Number of training data',fontsize=14)
plt.ylabel('Average error (deg Celsius)',fontsize=14)
#plt.xticks(xticks,xtick_labels)
plt.tick_params(labelsize=12)
plt.legend(fontsize=14)
plt.gcf().subplots_adjust(bottom=0.15)

plt.figure(figsize=(8,6))
plt.plot(n_train_list,lambda_lasso_vec,marker='o',linestyle='none')
plt.xscale('log')
plt.yscale('log')
plt.ylabel(r"Regularization parameter ($\lambda$)",fontsize=14)
plt.xlabel('Number of training data (n)',fontsize=14)
# plt.xticks(xticks,xtick_labels)
plt.tick_params(labelsize=12)
# plt.gcf().subplots_adjust(left=0.2)
plt.gcf().subplots_adjust(bottom=0.15)


plt.figure(figsize=(8,6))
plt.plot(n_train_list,n_features_fsr_vec,marker='o',linestyle='none')
plt.xscale('log')
#plt.yscale('log')
plt.ylabel(r"Number of features",fontsize=14)
plt.xlabel('Number of training data (n)',fontsize=14)
# plt.xticks(xticks,xtick_labels)
plt.tick_params(labelsize=12)
# plt.gcf().subplots_adjust(left=0.2)
plt.gcf().subplots_adjust(bottom=0.15)


n_largest = 3
color_list = ['black','red','purple','green','orange']
```

```
sorted_coeffs = np.argsort(np.abs(coeffs_ridge_matrix[:,-1]))
largest_coeffs = sorted_coeffs[-1:(-n_largest-1):-1]
plt.figure(figsize=(8,6))
for ind in range(n_features):
    plt.plot(n_train_list,coeffs_ridge_matrix[ind,:],color='skyblue')
for ind in range(len(largest_coeffs)):
    ind_name = largest_coeffs[ind]
    if ind_name>=ind_response:
        ind_name = ind_name + 1
    aux_name = file_name_list[ind_name]
    aux_name = aux_name[14:]
    aux_name = aux_name[:-7]
    table = str.maketrans(dict.fromkeys('_0123456789'))
    aux_name = aux_name.translate(table)
    aux_name = aux_name[2:] + ", " + aux_name[:2]
    plt.plot(n_train_list,coeffs_ridge_matrix[largest_coeffs[ind],:
 ↪],color=color_list[ind],label=aux_name)
plt.ylim((-0.6,0.6))
plt.xscale('log')
plt.xlabel('Number of training data',fontsize=14)
plt.ylabel('Coefficients',fontsize=14)
# plt.xticks(xticks,xtick_labels)
plt.tick_params(labelsize=12)
# plt.gcf().subplots_adjust(left=0.2)
plt.gcf().subplots_adjust(bottom=0.15)
#plt.legend(loc = 'upper right',bbox_to_anchor=(1.3, 1.))
plt.legend(loc = 'lower right',fontsize=14)
aux_name = file_name_list[ind_response]
aux_name = aux_name[14:]
aux_name = aux_name[:-7]
table = str.maketrans(dict.fromkeys('_0123456789'))
aux_name = aux_name.translate(table)
aux_name = aux_name[2:] + ", " + aux_name[:2]
# plt.title(aux_name)

sorted_coeffs = np.argsort(np.abs(coeffs_lasso_matrix[:,-1]))
largest_coeffs = sorted_coeffs[-1:(-n_largest-1):-1]
plt.figure(figsize=(8,6))
for ind in range(n_features):
    plt.plot(n_train_list,coeffs_lasso_matrix[ind,:],color='skyblue')
for ind in range(len(largest_coeffs)):
    ind_name = largest_coeffs[ind]
    if ind_name>=ind_response:
        ind_name = ind_name + 1
    aux_name = file_name_list[ind_name]
    aux_name = aux_name[14:]
    aux_name = aux_name[:-7]
    table = str.maketrans(dict.fromkeys('_0123456789'))
    aux_name = aux_name.translate(table)
    aux_name = aux_name[2:] + ", " + aux_name[:2]
    plt.plot(n_train_list,coeffs_lasso_matrix[largest_coeffs[ind],:
 ↪],color=color_list[ind],label=aux_name)
plt.ylim((-0.6,0.6))
plt.xscale('log')
plt.xlabel('Number of training data',fontsize=14)
plt.ylabel('Coefficients',fontsize=14)
# plt.xticks(xticks,xtick_labels)
plt.tick_params(labelsize=12)
# plt.gcf().subplots_adjust(left=0.2)
plt.gcf().subplots_adjust(bottom=0.15)
#plt.legend(loc = 'upper right',bbox_to_anchor=(1.3, 1.))
plt.legend(loc = 'lower right',fontsize=14)
aux_name = file_name_list[ind_response]
aux_name = aux_name[14:]
aux_name = aux_name[:-7]
table = str.maketrans(dict.fromkeys('_0123456789'))
aux_name = aux_name.translate(table)
aux_name = aux_name[2:] + ", " + aux_name[:2]
# plt.title(aux_name)

plt.figure(figsize=(8,6))
for ind in range(n_features):
    plt.plot(lambdas_ridge,coeffs_ridge_fixed[ind,:],color='skyblue')
    plt.xscale('log')
for ind in range(len(largest_coeffs)):
    ind_name = largest_coeffs[ind]
    if ind_name>=ind_response:
        ind_name = ind_name + 1
    aux_name = file_name_list[ind_name]
    aux_name = aux_name[14:]
    aux_name = aux_name[:-7]
    table = str.maketrans(dict.fromkeys('_0123456789'))
    aux_name = aux_name.translate(table)
    aux_name = aux_name[2:] + ", " + aux_name[:2]
    plt.plot(lambdas_ridge,coeffs_ridge_fixed[largest_coeffs[ind],:
 ↪],color=color_list[ind],label=aux_name)
plt.xlabel(r"Regularization parameter ($\lambda/n$)",fontsize=14)
plt.ylabel('Coefficients',fontsize=14)
plt.gcf().subplots_adjust(bottom=0.15)
plt.legend(loc = 'upper right',fontsize=14)
plt.xscale('log')

plt.figure(figsize=(8,6))
for ind in range(n_features):
    plt.plot(lambdas_lasso,coeffs_lasso_fixed[ind,:],color='skyblue')
    plt.xscale('log')
for ind in range(len(largest_coeffs)):
    ind_name = largest_coeffs[ind]
    if ind_name>=ind_response:
        ind_name = ind_name + 1
    aux_name = file_name_list[ind_name]
    aux_name = aux_name[14:]
    aux_name = aux_name[:-7]
    table = str.maketrans(dict.fromkeys('_0123456789'))
    aux_name = aux_name.translate(table)
    aux_name = aux_name[2:] + ", " + aux_name[:2]
    plt.plot(lambdas_lasso,coeffs_lasso_fixed[largest_coeffs[ind],:
 ↪],color=color_list[ind],label=aux_name)
plt.xlabel(r"Regularization parameter ($\lambda/n$)",fontsize=14)
plt.ylabel('Coefficients',fontsize=14)
plt.gcf().subplots_adjust(bottom=0.15)
plt.legend(loc = 'upper right',fontsize=14)
plt.xscale('log')
```

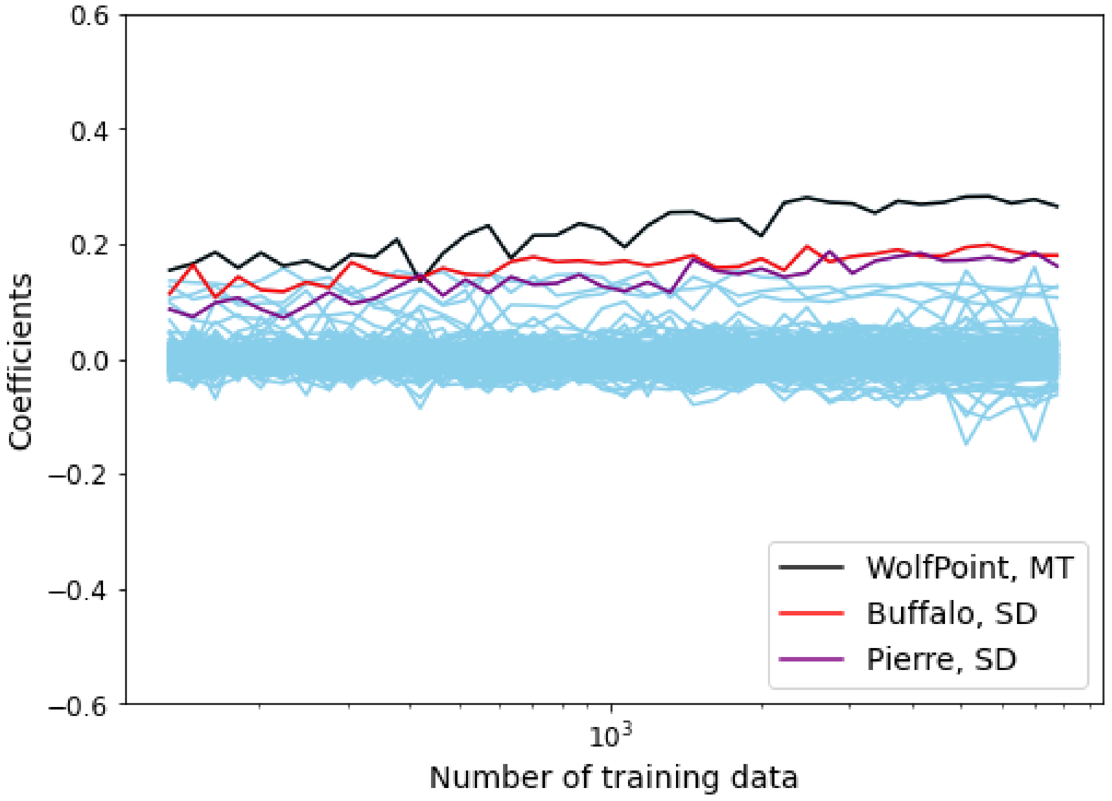


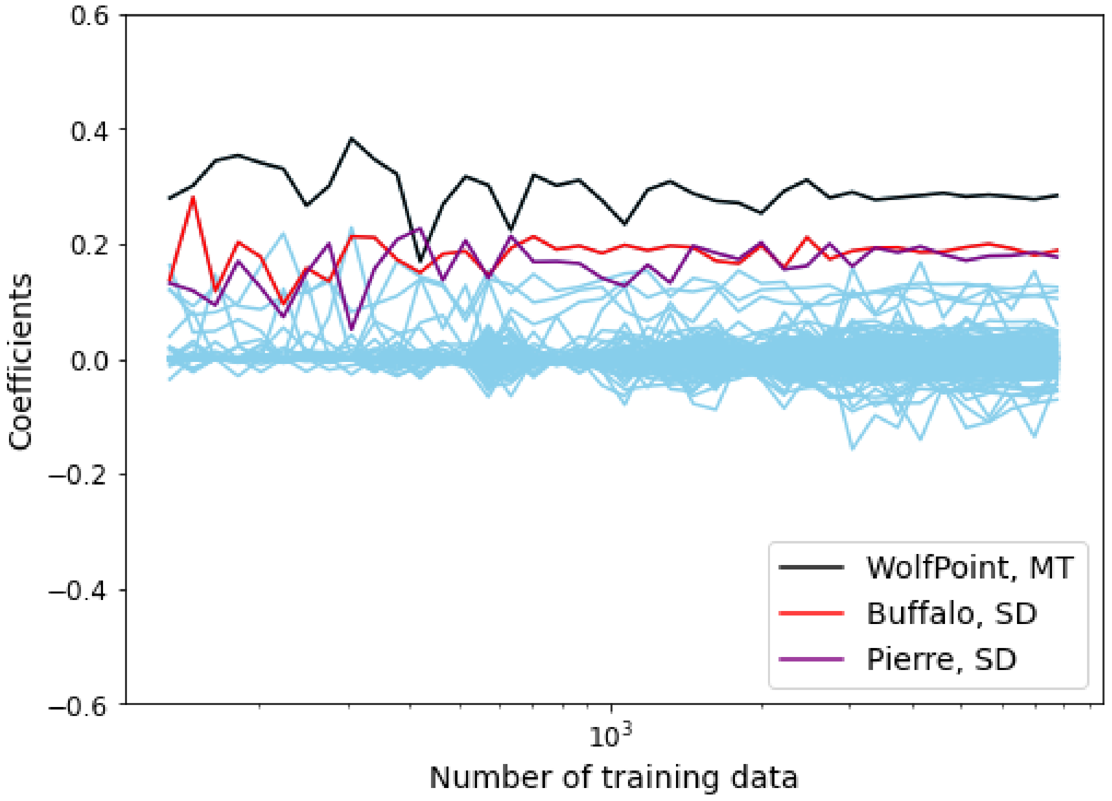


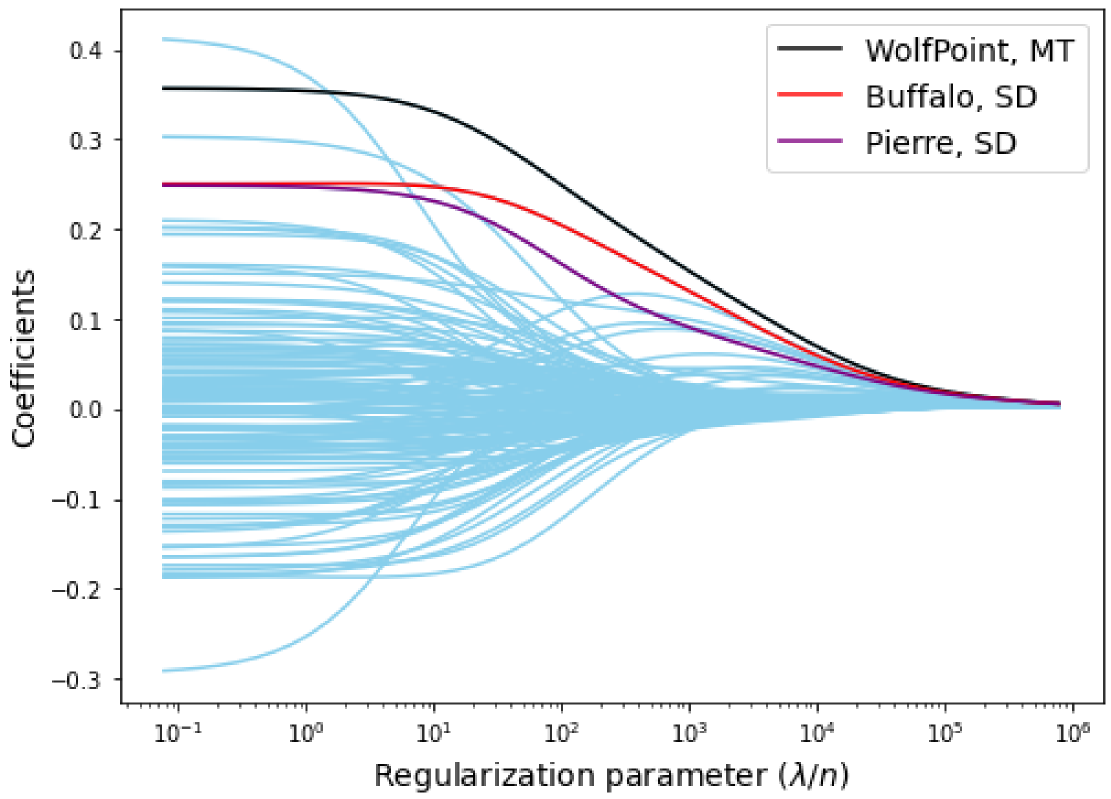
Regularization parameter (λ)
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{3}
Number of training data (n)

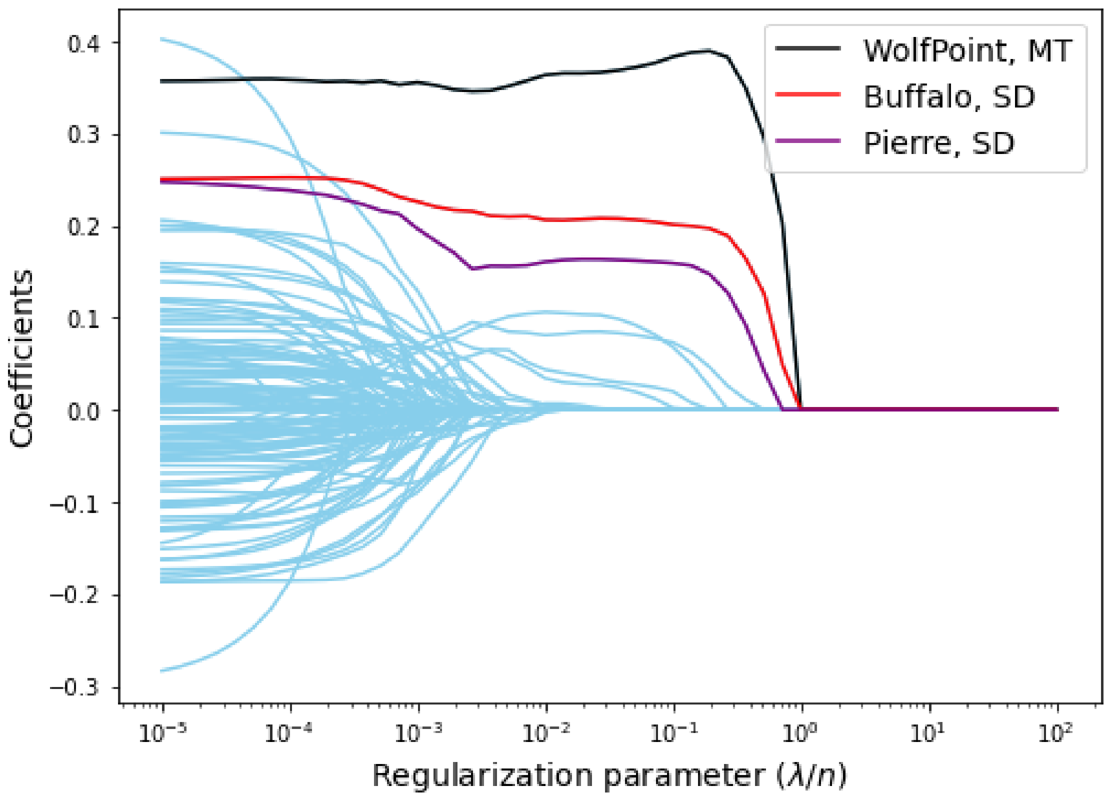


120
100
80
60
40
20
0
Number of features
10^3
Number of training data (n)


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
Coefficients
10^3
Number of training data
WolfPoint, MT
Buffalo, SD
Pierre, SD


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
Coefficients
10^3
Number of training data
WolfPoint, MT
Buffalo, SD
Pierre, SD


WolfPoint, MT
Buffalo, SD
Pierre, SD
Coefficients
Regularization parameter (λ/n)
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}



WolfPoint, MT
Buffalo, SD
Pierre, SD
Coefficients
Regularization parameter (λ/n)
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

```
[ ]:
```