Yen-Chun Chen
chenyenc@oregonstate.edu

Q1

For $y = \beta_0 + \beta_1 x \rightarrow \hat{\beta}_1 = \dfrac{S_{xy} - n\bar{x}\bar{y}}{S_{xx} - n\bar{x}^2}$, $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

(a) If $\bar{x} = 0$,

$$\hat{\beta}_1 = \frac{S_{xy} - 0}{S_{xx} - 0} = \frac{S_{xy}}{S_{xx}} \quad \#$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \cdot 0 = \bar{y} \quad \#$$

(b) $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = 0 \rightarrow \sum_{i=1}^{n} x_i = 0$

$$\rightarrow \overline{x^{(-j)}} = \frac{1}{n-1}\left[\left(\sum_{i=1}^{n} x_i\right) - x_j\right] = \frac{-x_j}{n-1} \quad \#$$

($\sum_{i=1}^{n} x_i = 0$)

$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \rightarrow \sum_{i=1}^{n} y_i = n\bar{y}$

$$\overline{y^{(-j)}} = \frac{1}{n-1}\left[\left(\sum_{i=1}^{n} y_i\right) - y_j\right] = \frac{n\cdot\bar{y} - y_j}{n-1} \quad \#$$

(c) $\hat{\beta}_1 = \dfrac{S_{xy} - n\bar{x}\bar{y}}{S_{xx} - n\bar{x}^2}$, $\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

$$\hat{\beta}_1^{(-j)} = \frac{\sum_{i\neq j}^{n} x_i y_i - (n-1)\,\overline{x^{(-j)}}\;\overline{y^{(-j)}}}{\sum_{i\neq j}^{n} x_i^2 - (n-1)\,\overline{x^{(-j)}}^2} \quad \#$$

$$\hat{\beta}_0^{(-j)} = \overline{y^{(-j)}} - \hat{\beta}_1^{(-j)} \cdot \overline{x^{(-j)}}$$

$$= \frac{n\bar{y} - y_j}{n-1} + \hat{\beta}_1^{(-j)} \frac{x_j}{n-1} \quad \#$$

(d)

$$\hat{\beta}_1^{(-j)} = \frac{\sum_{i\neq j}^{n} x_i y_i - (n-1)\,\overline{x^{(-j)}}\;\overline{y^{(-j)}}}{\sum_{i\neq j}^{n} x_i^2 - (n-1)\,\overline{x^{(-j)}}^2}$$

$$S_{xy} = \sum_{i=1}^{n} x_i y_i = \sum_{i\neq j}^{n} x_i y_i + x_j y_j$$

$$\rightarrow \begin{cases} \sum_{i\neq j}^{n} x_i y_i = S_{xy} - x_j y_j \\ \sum_{i\neq j}^{n} x_i^2 = S_{xx} - x_j^2 \end{cases}$$

$$= \frac{S_{xy} - x_j y_j + (n-1)\cdot \frac{(+x_j)\cdot(n\bar{y} - y_j)}{(n-1)(n-1)}}{S_{xx} - x_j^2 - (n-1)\left(\frac{-x_j}{n-1}\right)^2}$$

$$= \frac{S_{xy} + \left(\frac{1}{n-1}\right)(-n x_j y_j + x_j y_j + n x_j \bar{y} - x_j y_j)}{S_{xx} + \left(\frac{1}{n-1}\right)(-n^2 x_j^2 + x_j^2 - x_j^2)}$$

$$= \frac{S_{xy} + \left(\frac{n}{n-1}\right)(x_j \bar{y} - x_j y_j)}{S_{xx} + \left(\frac{n}{n-1}\right)(-x_j^2)} = \frac{S_{xy} - \left(\frac{n}{n-1}\right) x_j (y_j - \bar{y})}{S_{xx} - \left(\frac{n}{n-1}\right) x_j^2} \quad \#$$

$$y_j - \hat{\beta}_0^{(-j)} = \frac{n y_j - y_j}{n-1} - \frac{n\bar{y} - y_j}{n-1} - \hat{\beta}_1^{(-j)} \frac{x_j}{n-1}$$

$$= \frac{n}{n-1}(y_j - \bar{y}) - \frac{1}{n-1}\hat{\beta}_1^{(-j)} \cdot x_j \quad \#$$

(e)

$$y_j - \hat{y}_j = y_j - \hat{\beta}_0^{(-j)} - \hat{\beta}_1^{(-j)} \cdot x_j$$

$$= \frac{n(y_j - \bar{y})}{n-1} - \frac{1}{n-1}\hat{\beta}_1^{(-j)} x_j - \hat{\beta}_1^{(-j)} \cdot x_j$$

$$= \frac{n(y_j - \bar{y})}{n-1} - \frac{n}{n-1}\hat{\beta}_1^{(-j)} x_j = \frac{n}{n-1}\left(y_j - \bar{y} - \hat{\beta}_1^{(-j)} x_j\right)$$

$$= \frac{n}{n-1}\left(y_j - \bar{y} - \frac{(n-1)S_{xy} - n x_j (y_j - \bar{y})}{(n-1)S_{xx} - n x_j^2} x_j\right)$$

$$= \frac{(y_j - \bar{y})\left[n \cdot (n-1) S_{xx} - n^2 x_j^2\right] - n \cdot \left[(n-1) S_{xy} - n x_j (y_j - \bar{y})\right] x_j}{(n-1)\left[(n-1)S_{xx} - n x_j^2\right]}$$

Given that $1 - h_j = \dfrac{(n-1)S_{xx} - n x_j^2}{n \cdot S_{xx}} \Rightarrow (n-1)S_{xx} - n x_j^2 = (1-h_j)\, n \cdot S_{xx}$

$$\Rightarrow \frac{(y_j - \bar{y})\left[n \cdot (n-1) S_{xx} - n^2 x_j^2\right] - n \cdot \left[(n-1) S_{xy} - n x_j (y_j - \bar{y})\right] x_j}{(n-1) \cdot n \cdot S_{xx} (1-h_j)}$$

$$\Rightarrow \frac{(y_j - \bar{y})(n-1)S_{xx}}{(n-1)S_{xx}(1-h_j)} - \frac{(y_j - \bar{y}) n \cdot x_j^2}{(n-1)S_{xx}(1-h_j)} - \frac{(n-1)S_{xy} x_j}{(n-1)S_{xx}(1-h_j)} + \frac{(y_j - \bar{y}) x_j^2}{(n-1)S_{xx}(1-h_j)}$$

$$\frac{-\beta_1 x_j}{1-h_j} = \frac{\beta_1 x_j \cdot n \cdot S_{xx}}{(n-1)S_{xx} - n x_j^2}$$

$$= \underbrace{\frac{n \cdot S_{xx} x_j}{(n-1)S_{xx} - n x_j^2}} \cdot \underbrace{\frac{(n-1)S_{xy} - n \cdot x_j \cdot (y_j - \bar{y})}{(n-1)S_{xx} - n \cdot x_j^2}}$$

$$= \frac{n(n-1) S_{xx} S_{xy} \cdot x_j - n^2 S_{xx} \cdot x_j^2 (y_j - \bar{y})}{\left[(n-1)S_{xx} - n x_j^2\right]^2}$$

Q1 (f)

May 15, 2024

(f) In the homework folder you have access to the data file SimpleReg.csv. The data contains a feature column x and a response column y. Read the data, then center the x data, and fit a linear model in the form of $y = \beta 0 + \beta 1 x$. Now use an R or Python program to calculate the LOOCV CVn, as we did in the class (if you use R, pick the first element of delta). Also write a code that calculates the CVn using equation (3). You should see that the two methods produce identical results. You may also be surprised with how faster your customized code is, compared to the R cv.glm function!

[1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from ISLP.models import sklearn_sm
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
```

[2]:
```
Data = pd.read_csv('SimpleReg.csv')
y = Data['y']
X = Data[['x']]
# standardlize training data
X_cent = StandardScaler(with_mean=True,  with_std=False).fit_transform(X)
X = pd.DataFrame({'intercept': np.ones(X.shape[0]), 'x': X_cent.flatten()})
# X
```

1 ~ ~ ~ ~ ~ From Sample code ~ ~ ~ ~ ~

[3]:
```
M = sklearn_sm(sm.OLS)
M_CV = cross_validate(M, X, y, cv=Data.shape[0])
cv_error = np.mean(M_CV['test_score'])
print('CVn:', cv_error)
```

```
CVn: 0.3695297270024033
```

2 $\sim\sim\sim\sim\sim CV_n = \frac{1}{n}\sum_{j=1}^{n}(\frac{y_j-\hat{y}}{1-h_j})^2 \sim\sim\sim\sim\sim$

[4]:
```
n = len(Data)
MSE = np.zeros(n)

x = X['x']
Sxy = np.sum(Data.x * Data.y)
Sxx = np.sum(Data.x * Data.x)
muY = np.mean(y)

for j in range(n):

    beta_1_hat = (Sxy - (n/n-1) * x[j] * (y[j] - muY)) / (Sxx - (n/n-1) * x[j]␣
 ↪* x[j])
    beta_0_hat = y[j] - (n * (y[j] - muY) / (n-1)) + (beta_1_hat * x[j] / (n-1))

    # predict y_hat
    y_hat = beta_0_hat + (beta_1_hat * x[j])

    # calculate MSE
    h_j = 1/n + ((x[j] * x[j]) / Sxx)
    MSE[j] = ( (y[j] - y_hat) / (1 - h_j) ) ** 2

print('CVn:', np.sum(MSE)/len(Data))
```

```
CVn: 0.3703803767708858
```

The results are pretty close, all around 0.37, but not identical.

Q2

May 15, 2024

In HW3 you did an exercise on performing a classification on the HeartData.csv data file. We recently learned how to apply the LOOCV and K-Fold CV to regression problems. In this homework we would like to apply the LOOCV and K-Fold CV to the logistic regression, LDA and QDA models used in HW3. Using 10-fold CV and LOOCV fit the models and report the classification accuracy for the 3 models (logistic regression, LDA and QDA).For this question use num as the response variable and all the other variables as features.

[1]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
from ISLP.models import sklearn_sm
from ISLP.models import (ModelSpec as MS,summarize)
from sklearn.discriminant_analysis import \
     (LinearDiscriminantAnalysis as LDA,
      QuadraticDiscriminantAnalysis as QDA)
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
```

[2]:
```
Data = pd.read_csv('HeartData-1.csv')
y = Data['num']
X = Data.drop(['num'], axis = 1) # without intercept
X_LR = MS(X).fit_transform(Data) # with intercept
```

0.1 = = = = = Running 10-Fold CV= = = = =

[3]:
```
def split_into_10_Fold_chunks(X, y, train_index, test_index):
    X_train = X.iloc[train_index]
    y_train = y.iloc[train_index]
    X_test = X.iloc[test_index]
    y_test = y.iloc[test_index]
    return X_train, y_train, X_test, y_test
```

[4]:
```
def get_accuracy(probs, y_test):
    labels = (probs > 0.5)
    return np.mean(labels == y_test)
```

[5]:
```
# define LDA/QDA model
lda = LDA(store_covariance=True)
qda = QDA(store_covariance=True)
```

[6]:
```
K = 10
kf = KFold(n_splits=K, shuffle=True, random_state=0)
PROB_LR = np.zeros(K)
PROB_LDA = np.zeros(K)
PROB_QDA = np.zeros(K)

for i, (train_index, test_index) in enumerate(kf.split(Data)):
    X_train_LR, y_train, X_test_LR, y_test = split_into_10_Fold_chunks(X_LR, y,␣
 ↪train_index, test_index)
    X_train = X_train_LR.drop(['intercept'], axis = 1)
    X_test = X_test_LR.drop(['intercept'], axis = 1)

    # fit model
    lrm = sm.GLM(y_train, X_train_LR, family=sm.families.Binomial()).fit()
    lda.fit(X_train, y_train)
    qda.fit(X_train, y_train)

    # predict the values
    probs_LR = lrm.predict(exog=X_test_LR)
    probs_LDA = lda.predict(X_test)
    probs_QDA = qda.predict(X_test)

    # save accuracy
    labels = (probs_LR > 0.5)
    PROB_LR[i] = np.mean(labels == y_test)
    PROB_LDA[i] = np.mean(probs_LDA == y_test)
    PROB_QDA[i] = np.mean(probs_QDA == y_test)

print('~ ~ ~ ~ ~ Accuracy ~ ~ ~ ~ ~')
print("Logistic Regression:", np.mean(PROB_LR))
print("LDA:", np.mean(PROB_LDA))
print("QDA:", np.mean(PROB_QDA))
```

```
~ ~ ~ ~ ~ Accuracy ~ ~ ~ ~ ~
Logistic Regression: 0.8282758620689655
LDA: 0.8383908045977012
QDA: 0.8185057471264369
```

0.2 = = = = = Running LOOCV = = = = =

[7]:
```
def split_data_leave_one_out(X, y, i):
    # training data size 296
    X_train = X.drop(i)
```

```
    y_train = y.drop(i)
    # test data size 1
    X_test = X.iloc[i:i+1]
    y_test = y.iloc[i]
    return X_train, y_train, X_test, y_test
```

[8]:
```
PROB_LR = np.zeros(len(Data))
PROB_LDA = np.zeros(len(Data))
PROB_QDA = np.zeros(len(Data))

for i in range(len(Data)):
    X_train_LR, y_train, X_test_LR, y_test = split_data_leave_one_out(X_LR, y,␣
 ↪i)
    X_train = X_train_LR.drop(['intercept'], axis = 1)
    X_test = X_test_LR.drop(['intercept'], axis = 1)

    # fit model
    lrm = sm.GLM(y_train, X_train_LR, family=sm.families.Binomial()).fit()
    lda.fit(X_train, y_train)
    qda.fit(X_train, y_train)

    # predict the values
    probs_LR = lrm.predict(exog=X_test_LR)
    probs_LDA = lda.predict(X_test)
    probs_QDA = qda.predict(X_test)

    # save accuracy
    labels_loocv = (probs_LR > 0.5)
    PROB_LR[i] = np.mean(labels_loocv == y_test)
    PROB_LDA[i] = np.mean(probs_LDA == y_test)
    PROB_QDA[i] = np.mean(probs_QDA == y_test)

print('~ ~ ~ ~ ~ Accuracy ~ ~ ~ ~ ~')
print("Logistic Regression:", np.mean(PROB_LR))
print("LDA:", np.mean(PROB_LDA))
print("QDA:", np.mean(PROB_QDA))
```

```
~ ~ ~ ~ ~ Accuracy ~ ~ ~ ~ ~
Logistic Regression: 0.8249158249158249
LDA: 0.835016835016835
QDA: 0.8249158249158249
```

For 10-Fold CV, LDA has the best accuracy, 83.84%, among the three models.

For LOOCV, LDA still has the best accuracy, 83.5%, among the three models.

Q3.

(a) $RSS_{Ridge} = \sum_{i=1}^{n} (y_i - \beta - \beta x)^2 + \lambda \beta^2$

$$\frac{d\,RSS}{d\beta} = 2 \sum_{i=1}^{n} (y_i - \beta - \beta x_i)(-1 - x_i) + 2\lambda\beta = 0$$

$$\Rightarrow \lambda\beta = \sum (1+x_i) y_i - (1+x_i)\beta - (x_i + x_i^2)\beta$$

$$\Rightarrow \beta \left[\lambda + \sum_{i=1}^{n} (x_i + x_i^2 + 1 + x_i)\right] = \sum_{i=1}^{n} (1+x_i) y_i$$

$$\longrightarrow \hat{\beta}_R = \frac{\sum_{i=1}^{n} (1+x_i) y_i}{\lambda + \sum_{i=1}^{n} (x_i+1)^2} \quad \#$$

(b)

$E[\varepsilon] = 0, \quad Var(\varepsilon) = \sigma^2$

$y = \beta + \beta x + \varepsilon \longrightarrow Var(y) = \sigma^2$

$$Var(\hat{\beta}_R) = Var\left(\frac{\sum_{i=1}^{n} (1+x_i) y_i}{\lambda + \sum_{i=1}^{n} (x_i+1)^2}\right)$$

$$= \frac{\sum_{i=1}^{n} (1+x_i)^2 Var(y_i)}{\left[\lambda + \sum_{i=1}^{n} (1+x_i)^2\right]^2} = \frac{\sum_{i=1}^{n} (1+x_i)^2}{\left[\lambda + \sum_{i=1}^{n} (1+x_i)^2\right]^2} \sigma^2 \quad \#$$

(C) To show $Var(\hat{\beta}_R) \leq Var(\hat{\beta})$

$$\rightarrow \frac{\sum_{i=1}^{n}(1+x_i)^2}{\left[\lambda + \sum_{i=1}^{n}(1+x_i)^2\right]^2}\sigma^2 \leq \frac{\sigma^2}{\sum_{i=1}^{n}(1+x_i)^2}$$

$$\rightarrow \left[\sum_{i=1}^{n}(1+x_i)^2\right]^2 \leq \left[\lambda + \sum_{i=1}^{n}(1+x_i)^2\right]^2$$

since $\lambda > 0$, and $\sum_{i=1}^{n}(1+x_i)^2 \geq 0$

Thus, the equation $Var(\hat{\beta}_R) \leq Var(\hat{\beta})$ holds for all $\lambda > 0$

#

Q4

May 15, 2024

In this question we analyze the data file Fertility.csv which is available in the homework folder, and try to build a model for fertility. In this dataset Fertility, the first column, is the response variable, and the other variables are potential predictors. We will use several different statistical modeling techniques. The data set contains 47 rows (samples), split the data into training and test sets. Set the first 30 rows to training samples and the rows 31 through 47 as the test samples.

[1]:
```
import numpy as np
import pandas as pd
from ISLP.models import ModelSpec as MS
import sklearn.model_selection as skm
import sklearn.linear_model as skl
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge, Lasso, LinearRegression, RidgeCV,␣
 ↪LassoCV
from sklearn.metrics import mean_squared_error
from matplotlib.pyplot import subplots
```

[2]:
```
Data = pd.read_csv('Fertility.csv')
# print(Data)
train = (Data.index < 30)
X = MS(Data.columns.drop(['Fertility'])).fit_transform(Data)
Y = Data['Fertility']
y_train, X_train = Y.loc[train] , X.loc[train]
y_test, X_test = Y.loc[~train] , X.loc[~train]
```

(a) Fit a linear model on the training set, and report the test error (MSE) obtained.

[3]:
```
LinearModel = LinearRegression()
LinearModel.fit(X_train, y_train)
y_pred_linear = LinearModel.predict(X_test)
mse_linear = mean_squared_error(y_test, y_pred_linear)
# print(pd.DataFrame({'y':y_test, 'pred':pred_linear}))
print('MSE Linear:', mse_linear)
```

```
MSE Linear: 183.72179150160574
```

(b) Fit a Ridge regression model on the training set, with chosen by cross-validation on a dense grid similar to the example solved in the class. Report the test error obtained.

[4]:
```
if 'intercept' in X:
    X = X.drop(columns=['intercept'], axis = 1)
X_np = X.to_numpy()
Xs = X_np - X_np.mean(0)[None,:]
X_scale = X_np.std(0)
Xs = Xs / X_scale[None,:]
lambdas = 10**np.linspace(10, -10, 100) / Y.std()
```

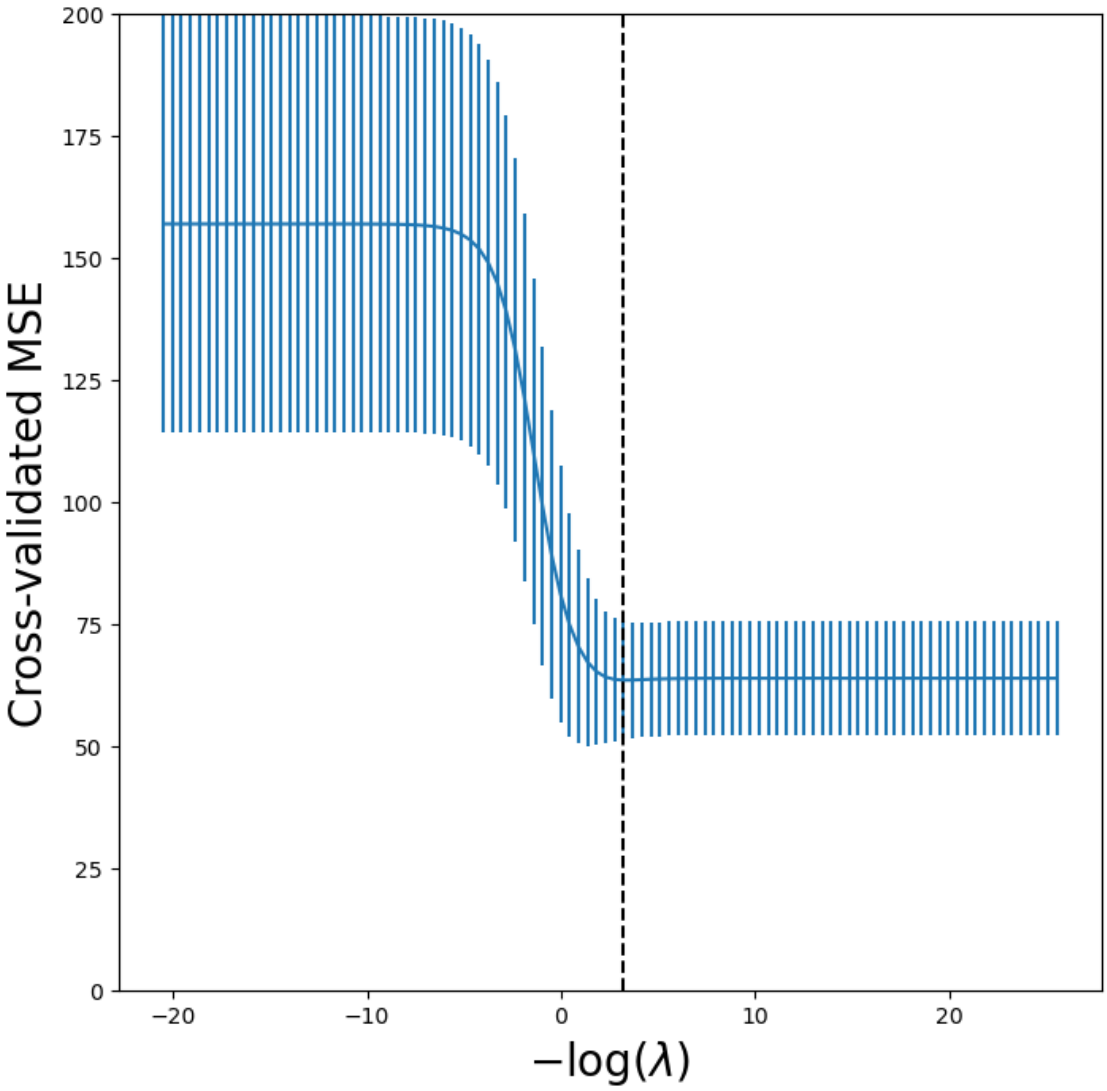
[5]:
```
param_grid = {'ridge__alpha': lambdas}
K = 5
kfold = skm.KFold(K, random_state=0, shuffle=True)
scaler = StandardScaler(with_mean=True,  with_std=True)

ridgeCV = skl.ElasticNetCV(alphas=lambdas,
                           l1_ratio=0,
                           cv=kfold)
pipeCV = Pipeline(steps=[('scaler', scaler),
                         ('ridge', ridgeCV)])
pipeCV.fit(X, Y)
```

```
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: UserWarning:
Coordinate descent without L1 regularization may lead to unexpected results and
is discouraged. Set l1_ratio > 0 to add L1 regularization.
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 3072.5159407502, tolerance: 0.6145031891891891
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: UserWarning:
Coordinate descent without L1 regularization may lead to unexpected results and
is discouraged. Set l1_ratio > 0 to add L1 regularization.
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 3072.5159376728498, tolerance: 0.6145031891891891
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: UserWarning:
Coordinate descent without L1 regularization may lead to unexpected results and
is discouraged. Set l1_ratio > 0 to add L1 regularization.
  model = cd_fast.enet_coordinate_descent_gram(
```

```
packages/sklearn/linear_model/_coordinate_descent.py:617: UserWarning:
Coordinate descent without L1 regularization may lead to unexpected results and
is discouraged. Set l1_ratio > 0 to add L1 regularization.
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:617: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 631.984809959277, tolerance: 0.4120903157894737
  model = cd_fast.enet_coordinate_descent_gram(
/usr/local/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_coordinate_descent.py:631: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 1.151e+03, tolerance: 7.178e-01 Linear regression models with null weight
for the l1 regularization term are more efficiently fitted using one of the
solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
  model = cd_fast.enet_coordinate_descent(
```

[5]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('ridge',
                 ElasticNetCV(alphas=array([8.00531736e+08, 5.02757261e+08,
3.15746212e+08, 1.98297823e+08,
       1.24536812e+08, 7.82127477e+07, 4.91198850e+07, 3.08487193e+07,
       1.93738948e+07, 1.21673706e+07, 7.64146333e+06, 4.79906167e+06,
       3.01395060e+06, 1.89284881e+06, 1.18876422e+06, 7.46578576e+05,
       4.68873104e+05, 2.94465974e+05,…
       5.51775473e-08, 3.46531078e-08, 2.17631616e-08, 1.36678998e-08,
       8.58383940e-09, 5.39090131e-09, 3.38564314e-09, 2.12628256e-09,
       1.33536742e-09, 8.38649657e-10, 5.26696426e-10, 3.30780706e-10,
       2.07739924e-10, 1.30466726e-10, 8.19369065e-11, 5.14587653e-11,
       3.23176043e-11, 2.02963974e-11, 1.27467291e-11, 8.00531736e-12]),
                              cv=KFold(n_splits=5, random_state=0,
shuffle=True),
                              l1_ratio=0))])
```

[6]:
```
tuned_ridge = pipeCV.named_steps['ridge']
ridgeCV_fig, ax = subplots(figsize=(8,8))
ax.errorbar(-np.log(lambdas),
            tuned_ridge.mse_path_.mean(1),
            yerr=tuned_ridge.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_ridge.alpha_), c='k', ls='--')
ax.set_ylim([0,200])
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20);
```



[7]:
```
print("The smallest test error for Ridge:", np.min(tuned_ridge.mse_path_.
  ↪mean(1)))
```

```
The smallest test error for Ridge: 63.45187653970055
```

(c) Fit a LASSO model on the training set, with chosen by cross-validation on a dense grid similar to the example solved in the class. Report the test error obtained, along with the number of non-zero coefficient estimates.

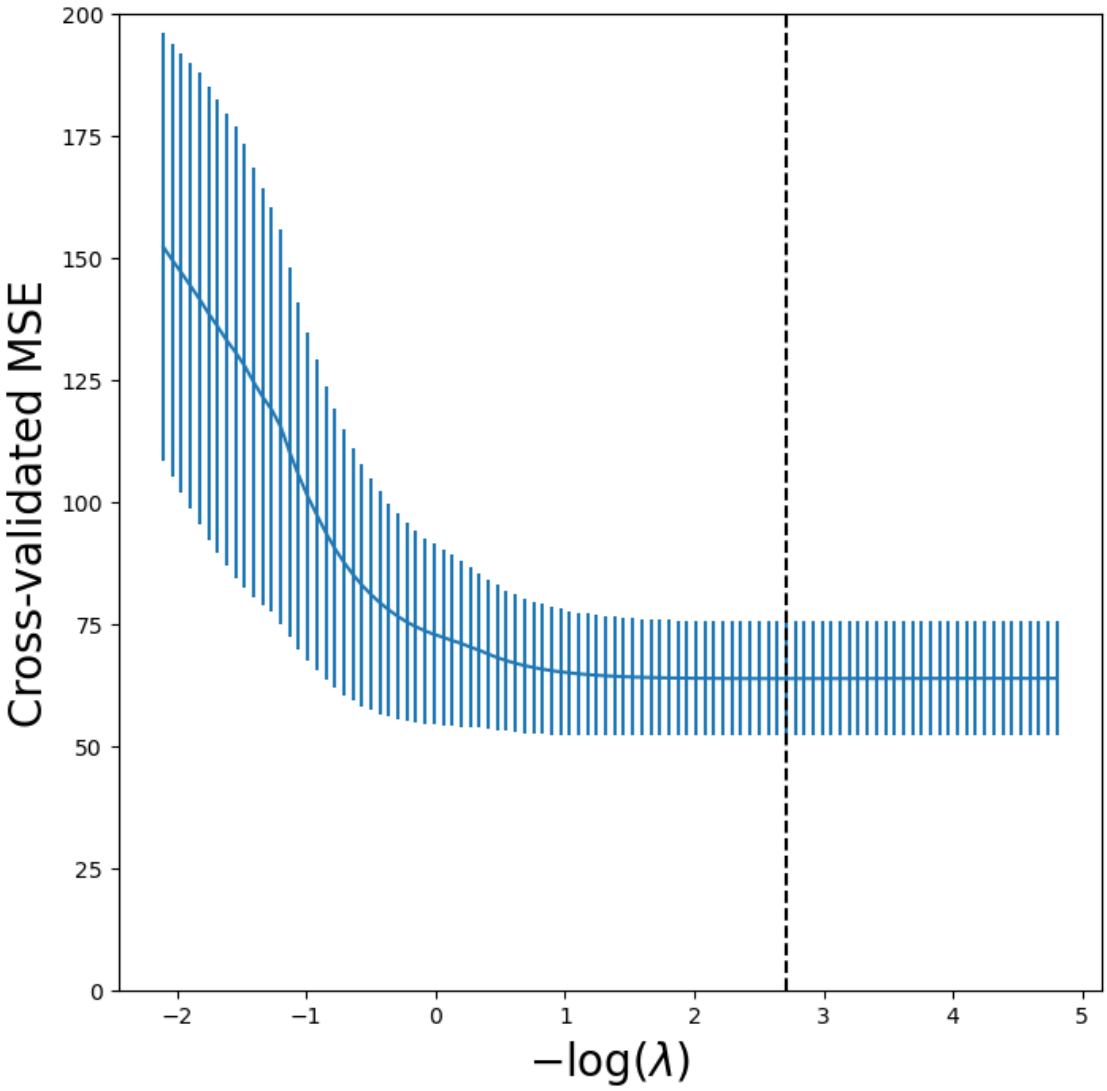
[8]:
```
lassoCV = skl.ElasticNetCV(n_alphas=100,
                           l1_ratio=1,
                           cv=kfold)
pipeCV = Pipeline(steps=[('scaler', scaler),
                         ('lasso', lassoCV)])
```

```
pipeCV.fit(X, Y)
tuned_lasso = pipeCV.named_steps['lasso']
tuned_lasso.alpha_
```

[8]:
```
0.06653839184028286
```

[9]:
```
lambdas, soln_array = skl.Lasso.path(Xs,
                                     Y,
                                     l1_ratio=1,
                                     n_alphas=100)[:2]
# soln_path = pd.DataFrame(soln_array.T,
#                          columns=X.columns,
#                          index=-np.log(lambdas))
```

[10]:
```
lassoCV_fig, ax = subplots(figsize=(8,8))
ax.errorbar(-np.log(tuned_lasso.alphas_),
            tuned_lasso.mse_path_.mean(1),
            yerr=tuned_lasso.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_lasso.alpha_), c='k', ls='--')
ax.set_ylim([0,200])
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20);
```



[11]:
```
# tuned_lasso.mse_path_
print("The smallest test error for LASSO:", np.min(tuned_lasso.mse_path_.
  ↪mean(1)))
```

```
The smallest test error for LASSO: 63.772950878902805
```

[12]:
```
print('The number of non-zero coefficient estimates:',np.sum(tuned_lasso.coef_ !
  ↪= 0))
```

```
The number of non-zero coefficient estimates: 5
```

(d) Compare the results of (a), (b), and (c). Which one seems to outperform the others for this specific setup?

In this case, the Linear Model got the highest test error(MSE). The results of Ridge and LASSO are pretty close and all around 63.5. The LASSO did not drop any features, which means that none of the features are useless.