1 Preliminary setup

[1]:
```
import numpy as np
import pandas as pd
from ISLP import load_data
from matplotlib.pyplot import subplots, show
import matplotlib.pyplot as plt


# Load and preprocess data
Hitters = load_data('Hitters').dropna()
```

2 Task

1. Use the final model (tuning parameter) obtained from 10-fold CV and fit the model again using the full dataset and display the corresponding coefficients.

[45]:
```
from sklearn.linear_model import RidgeCV, Lasso, LassoCV, Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle

# Construct feature matrix and outcome variable
X = pd.get_dummies(Hitters.drop(columns='Salary'), drop_first=True)
y = Hitters['Salary'].values

# Task 1: Ridge regression with CV to find best lambda (alpha)
alphas = np.exp(np.linspace(0, 8, 50))
ridge_cv = RidgeCV(alphas=alphas, store_cv_results=True)
ridge_cv.fit(X, y)
best_alpha_ridge = ridge_cv.alpha_
ridge_model = Ridge(alpha=best_alpha_ridge)
ridge_model.fit(X, y)
coef_task1 = ridge_model.coef_

print(coef_task1)
```

```
[ -2.02860796   7.58424738   4.03379342  -2.32240494  -0.88451987
   6.19542148  -3.22976069  -0.17077862   0.10671085  -0.20456668
   1.49604977   0.81815031  -0.80943149   0.28390582   0.37371839
  -3.20449556  36.02559023 -98.31856376  -1.03497201]
```

2. Multiply the feature Errors by 1/1000 and again fit the model from Task 1. Display the coefficients and interpret.

[52]:
```
# Task 2: Scale one variable manually and fit ridge with standardization
X_scaled = X.copy()

# Rescale 'Errors' by 1/1000
X_scaled['Errors'] = X_scaled['Errors'].astype(float) / 1000

# Standardize (if required by the task)
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_scaled), columns=X.columns)

# Fit ridge model
ridge_model2 = Ridge(alpha=best_alpha_ridge)
ridge_model2.fit(X_scaled, y)
coef_task2 = ridge_model2.coef_
print("Coefficients after rescaling + standardization:\n", coef_task2)
```

```
Coefficients after rescaling + standardization:
 [-120.75620416  141.98271425  -10.7355903    23.13998844   16.78667849
   79.16269699  -46.75097671  -13.02193553   93.80996664   57.58064435
  110.86292201   89.70452309  -86.55162346   74.72391629   27.29818898
  -24.87620768   27.59197633  -61.81228093  -10.31249136]
```

3. Redo Task 2 BUT without the normalizing (standardize) the data. Refit the same model again and display the coefficients. Interpret.

[53]:
```
# Task 3: Same variable but no standardization
X_no_std = X.copy()

# Rescale 'Errors' only
X_no_std['Errors'] = X_no_std['Errors'].astype(float) / 1000

# Do not standardize
ridge_model3 = Ridge(alpha=best_alpha_ridge)
ridge_model3.fit(X_no_std, y)
coef_task3 = ridge_model3.coef_
print("Coefficients after rescaling WITHOUT standardization:\n", coef_task3)
```

```
Coefficients after rescaling WITHOUT standardization:
 [-2.08316424e+00  7.83574168e+00  4.03538434e+00 -2.46707401e+00
 -1.01184803e+00  6.30012470e+00 -2.72891307e+00 -1.63751994e-01
  4.96433825e-02 -2.72096047e-01  1.55164973e+00  8.48452938e-01
 -8.20932977e-01  2.79014031e-01  2.72444155e-01 -1.43983852e+00
```

```
3.27987035e+01 -9.84477207e+01  1.27421956e-01]
```

4. Split the dataset into a training set using 80% of the observations and validation set using all other observations.

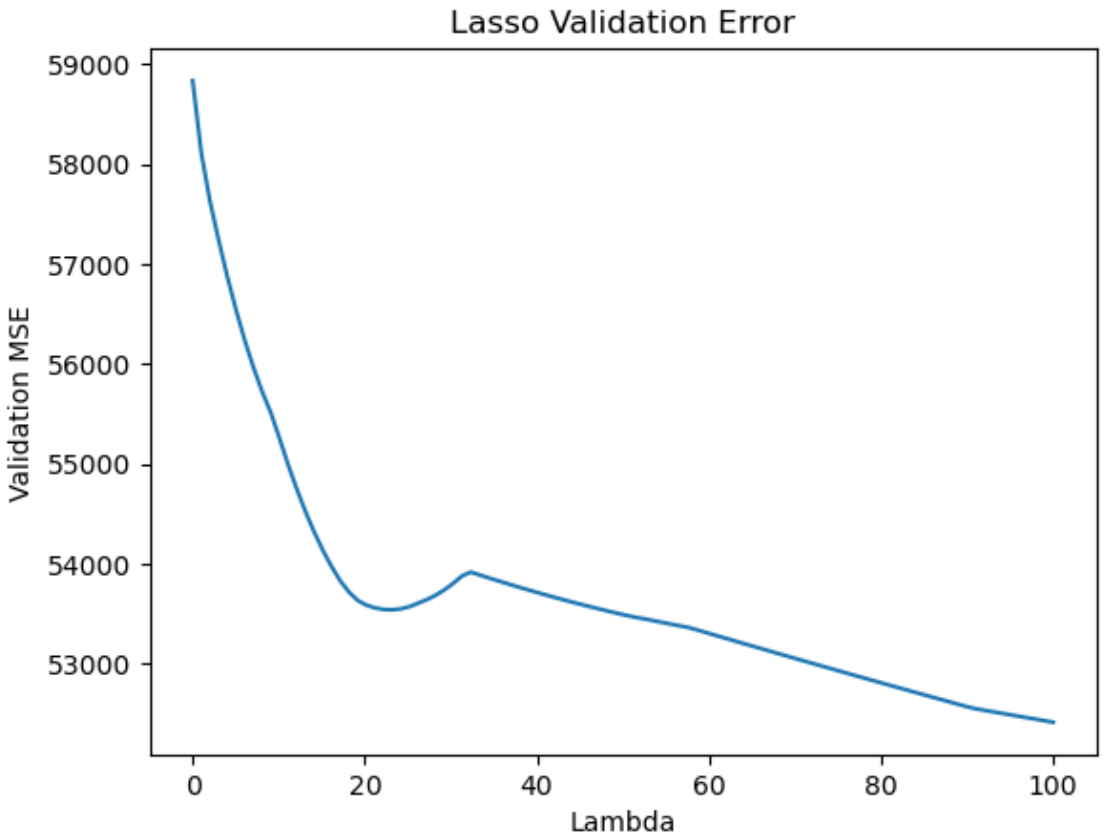
[54]:
```
# Task 4: Shuffle data and split
Hitters_shuffled = shuffle(Hitters, random_state=2)
n = len(Hitters_shuffled)
nTr = int(n * 0.8)
train_data = Hitters_shuffled[:nTr]
val_data = Hitters_shuffled[nTr:]

X_train = pd.get_dummies(train_data.drop(columns='Salary'), drop_first=True)
y_train = train_data['Salary'].values
X_val = pd.get_dummies(val_data.drop(columns='Salary'), drop_first=True)
y_val = val_data['Salary'].values
```

5. Set up a grid for the tuning parameter λ and fit Lasso regressions for all tuning parameters using the training data. Make sure that you choose the mininmum and maximum values of λ so that it allows you to determine the optimal λ parameter in the next task (you might need to play with the grid size a bit).

[ ]:
```
# Task 5: Lasso grid
grid = np.linspace(0.001, 100, 100)
val_errors = []

for alpha in grid:
    lasso = Lasso(alpha=alpha, max_iter=10000)
    lasso.fit(X_train, y_train)
    y_pred = lasso.predict(X_val)
    val_errors.append(mean_squared_error(y_val, y_pred))
```

6. For each model (tuning parameter), compute the mean squared prediction error in the validation dataset. Plot the validation error as a function of λ and find the best model which minimizes the validation error. Display the estimated coefficients for the best model and check whether some features are not selected in the final regression.

[26]:
```
# Task 6: Lasso grid plot
plt.plot(grid, val_errors)
plt.xlabel('Lambda')
plt.ylabel('Validation MSE')
plt.title('Lasso Validation Error')
plt.show()
```



[43]:
```
print(min(val_errors))
```

```
52415.86383077253
```

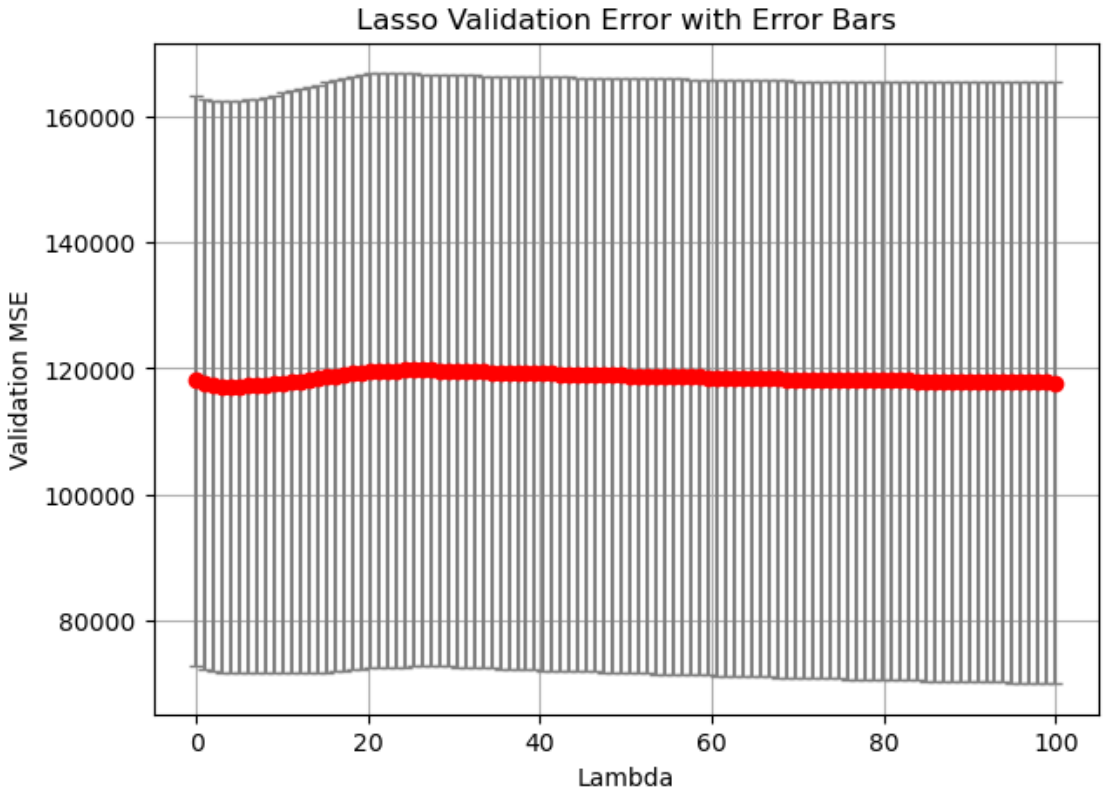
[27]:
```
# Find best model
min_index = np.argmin(val_errors)
best_lambda_lasso_cv = grid[min_index]
print("Best lambda: ",best_lambda_lasso_cv,'\n')

# Refit model with best lambda
best_lasso_model = Lasso(alpha=best_lambda_lasso_cv, max_iter=10000).
 ↪fit(X_train, y_train)
best_lasso_coefs = pd.Series(best_lasso_model.coef_, index=X_train.columns)
print("Lasso results: \n",best_lasso_coefs,'\n')

# Display non-zero coefficients
non_zero_coefs = best_lasso_coefs[best_lasso_coefs != 0]
non_zero_coefs.sort_values(ascending=False)
print("Non-zero Lasso coefficients: \n",non_zero_coefs.
 ↪sort_values(ascending=False))
```

```
Best lambda:  100.0

Lasso results:
 AtBat          -2.015511
Hits            6.056399
HmRun          -0.000000
Runs           -0.000000
RBI             0.000000
Walks           5.565911
Years          -0.000000
CAtBat         -0.303844
CHits           0.521263
CHmRun         -0.000000
CRuns           1.402914
CRBI            0.900971
CWalks         -0.753718
PutOuts         0.292351
Assists         0.362418
Errors         -0.000000
League_N        0.000000
Division_W     -0.000000
NewLeague_N     0.000000
dtype: float64

Non-zero Lasso coefficients:
 Hits        6.056399
Walks       5.565911
CRuns       1.402914
CRBI        0.900971
CHits       0.521263
Assists     0.362418
PutOuts     0.292351
CAtBat     -0.303844
CWalks     -0.753718
AtBat      -2.015511
dtype: float64
```

2.1 Extra code (not same! 10 folds!)

[41]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
```

```
# Create lambda grid
grid = np.linspace(0.001, 100, 100)
val_errors_mean = []
val_errors_std = []

# Repeat train/validation split multiple times
n_runs = 10

for alpha in grid:
    run_errors = []
    for seed in range(n_runs):
        # Shuffle and split each time
        Hitters_shuffled = shuffle(Hitters, random_state=seed)
        n = len(Hitters_shuffled)
        nTr = int(n * 0.8)
        train_data = Hitters_shuffled[:nTr]
        val_data = Hitters_shuffled[nTr:]

        X_train = pd.get_dummies(train_data.drop(columns='Salary'),␣
 ↪drop_first=True)
        y_train = train_data['Salary'].values
        X_val = pd.get_dummies(val_data.drop(columns='Salary'), drop_first=True)
        y_val = val_data['Salary'].values

        # Align columns (in case some category levels are missing)
        X_val = X_val.reindex(columns=X_train.columns, fill_value=0)

        model = Lasso(alpha=alpha, max_iter=10000)
        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)
        run_errors.append(mean_squared_error(y_val, y_pred))

    val_errors_mean.append(np.mean(run_errors))
    val_errors_std.append(np.std(run_errors))

# Plot with error bars
plt.figure(figsize=(7,5))
plt.errorbar(grid, val_errors_mean, yerr=val_errors_std, fmt='o',␣
 ↪ecolor='gray', capsize=3, color='red')
plt.xlabel('Lambda')
plt.ylabel('Validation MSE')
plt.title('Lasso Validation Error with Error Bars')
plt.grid(True)
plt.show()
```



[42]:
```
print(min(val_errors_mean))
```

```
117142.78701588386
```

7. Finally compare the best Lasso model obtained from the validation set approach from Task 6 to the best Lasso model obtained by 5-fold cross-validation.

[33]:
```
# Task 7 & 8: LassoCV and RidgeCV
lasso_cv = LassoCV(cv=5, random_state=1, max_iter=10000, alphas=alphas)
lasso_cv.fit(X, y)
y_pred_lasso = lasso_cv.predict(X)
mse_lasso_cv = mean_squared_error(y, y_pred_lasso)
best_lambda_lasso = lasso_cv.alpha_
lasso_cv_coefs = lasso_cv.coef_

print(lasso_cv_coefs)

print('\nLasso-MSE: ',mse_lasso_cv, 'with alpha', best_lambda_lasso)
```

```
[-1.66927222  5.71591888  0.         -0.          0.          4.64628557
 -0.         -0.23063656  0.35464977  0.          1.26847803  0.74243336
 -0.57932076  0.28981039  0.26073614 -0.          0.         -0.
  0.        ]
```

```
Lasso-MSE:  96459.01482992568 with alpha 113.82860789196029
```

8. Compare the best model from Task 7 to the best ridge regression obtained from 5-fold cross validation. How do the coefficients of the two models differ?

[34]:
```
ridge_cv_5fold = RidgeCV(alphas=alphas, cv=5)
ridge_cv_5fold.fit(X, y)
y_pred_ridge = ridge_cv.predict(X)
mse_ridge_cv = mean_squared_error(y, y_pred_ridge)
best_lambda_ridge = ridge_cv_5fold.alpha_
ridge_cv_coefs = ridge_cv_5fold.coef_

print(ridge_cv_coefs)

print('\nRidge-MSE: ',mse_ridge_cv, 'with alpha', best_lambda_ridge)
```

```
[-1.92090495  6.39400133  0.35386008 -0.66701689  0.47113815  5.2897077
 -0.31582864 -0.2195458   0.30632354  0.07691325  1.38722192  0.68697756
 -0.67956448  0.29499384  0.35765861 -2.07935974  0.90188371 -2.38262927
  0.65171112]

Ridge-MSE:  92141.38240788862 with alpha 2980.9579870417283
```