hw06

November 10, 2023

1 CSC380 - Homework 6

INDIVIDUAL HOMEWORK The homework is not collaborative anymore. Please respect the academic integrity. **Remember: if you get caught on cheating, you get F.**

Each subproblem is worth 10 pts. All cells are marked with instructions to insert your code. Please complete all cells as directed.

What to turn in: - Please print the notebook containing the answers and results into a pdf file (you can use `File - Print`). Submit this pdf file to the main homework entry in gradescope. Be sure to locate your answers for each problem when you submit, as ususal. In the worst case where you cannot print it into a pdf file somehow, you can create a Microsoft word document and then copy-paste screenshots showing your code and output parts by parts. - You also need to submit this jupyter notebook file filled with your answers in the code entry in gradescope.

Description:

This homework will familiarize you with linear regression. You will be using the *Prostate Cancer Dataset* from a study by Stamey et al. (1989). The study aims to predict prostate-specific antigen levels from clinical measures in men about to receive a radical prostatectomy.

The data contain 8 features: * log cancer volume (lcavol) * log prostate weight (lweight) * age (age) * log amount of benign prostatic hyperplasia (lbph) * seminal vesicle invasion (svi) * log of capsular penetration (lcp) * Gleason score (gleason) * percent of Gleason scores 4 or 5 (pgg45)

The data use a fixed Train / Test split, which we will load below.

[109]:
```
#All finalised needed imports
import pandas as pd
import itertools
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn import linear_model
import matplotlib.pyplot as plt
import numpy as np
import warnings
# Suppress warnings
warnings.filterwarnings("ignore")
```

[110]:
```
df_train = pd.read_csv('prostate_train.csv')
df_train.head()
```

```
[110]:      lcavol   lweight  age      lbph  svi       lcp  gleason  pgg45      lpsa
        0 -0.579818  2.769459   50 -1.386294    0 -1.386294        6      0 -0.430783
        1 -0.994252  3.319626   58 -1.386294    0 -1.386294        6      0 -0.162519
        2 -0.510826  2.691243   74 -1.386294    0 -1.386294        7     20 -0.162519
        3 -1.203973  3.282789   58 -1.386294    0 -1.386294        6      0 -0.162519
        4  0.751416  3.432373   62 -1.386294    0 -1.386294        6      0  0.371564
```

1.1 Problem 1: Your First Regression

We will begin by fitting our first ordinary least squares regression model. But first we need to do a little data management. You will notice that the data exist in a single data frame (one for Train and one for Test). The last column of the data frame ('lpsa') is the quantity that we wish to predict (the Y-value).

1.1.1 (a)

Do the following in the cell below, * Create X_train and Y_train by separating out the last column ('lpsa') and store it in Y_train * Do the same for X_test and Y_test * Display the DataFrame X_train

```
[111]: features = ['lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp', 'gleason',␣
        ↪'pgg45']
       output = ['lpsa']
       X_train = df_train[features]
       Y_train = df_train[output]

       df_test = pd.read_csv('prostate_test.csv')
       X_test = df_test[features]
       Y_test = df_test[output]
       # Display training inputs
       # Insert code here
       X_train
```

```
[111]:       lcavol   lweight  age      lbph  svi       lcp  gleason  pgg45
        0   -0.579818  2.769459   50 -1.386294    0 -1.386294        6      0
        1   -0.994252  3.319626   58 -1.386294    0 -1.386294        6      0
        2   -0.510826  2.691243   74 -1.386294    0 -1.386294        7     20
        3   -1.203973  3.282789   58 -1.386294    0 -1.386294        6      0
        4    0.751416  3.432373   62 -1.386294    0 -1.386294        6      0
        ..        …        …  …        …  …        …      …    …
        62   3.246491  4.101817   68 -1.386294    0 -1.386294        6      0
        63   2.532903  3.677566   61  1.348073    1 -1.386294        7     15
        64   2.830268  3.876396   68 -1.386294    1  1.321756        7     60
        65   3.821004  3.896909   44 -1.386294    1  2.169054        7     40
        66   2.882564  3.773910   68  1.558145    1  1.558145        7     80

        [67 rows x 8 columns]
```

1.1.2 (b)

Now we will fit our first model using a single feature (‘lcavol’). Do the following in the cell below, * Train a linear regression model on the ‘lcavol’ feature * Compute the R-squared score of the model on the training data * Scatterplot the training data for the ‘lcavol’ feature * Plot the regression line over the scatterplot * Label the plot axis / title and report the R-squared score

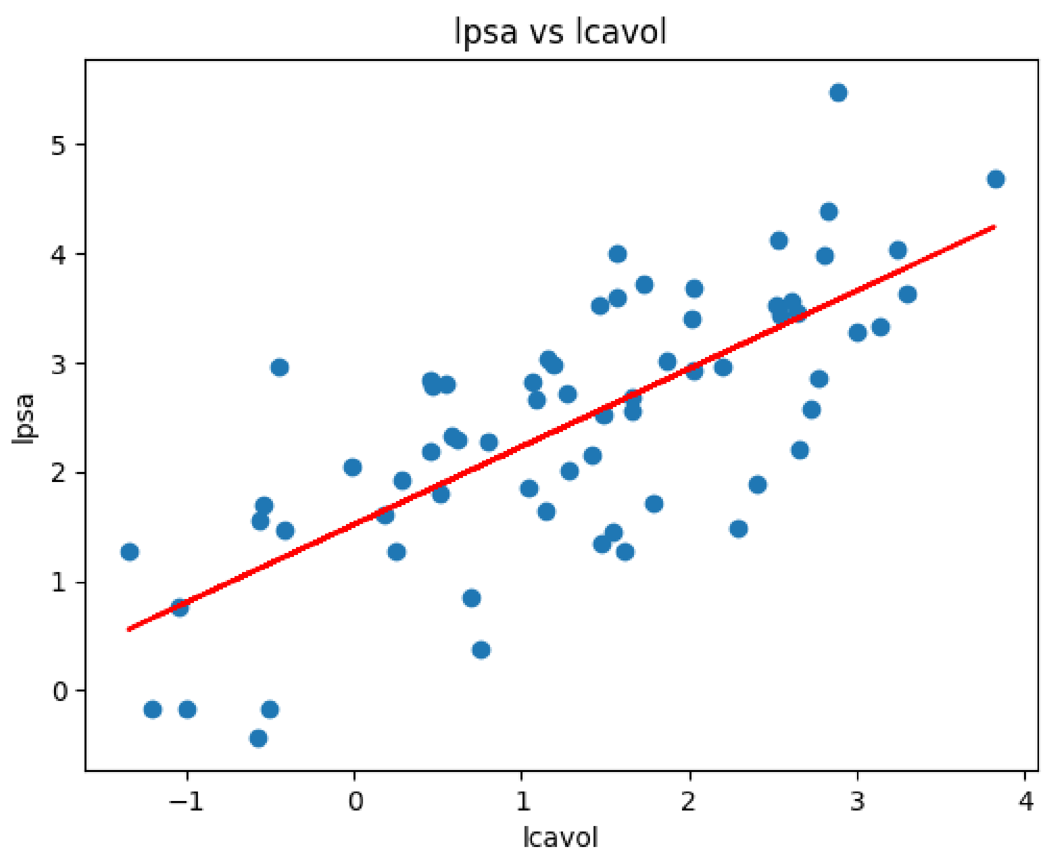
A couple of notes: * Scikit-learn gets cranky when you pass in single features. In some versions you will need to use, X_train[‘lcavol’].values.reshape(-1, 1) * To plot the regression line you can create a dense grid of points using numpy.arange, between the min() and max() of the feature values.

Documentation - Scikit-Learn - LinearRegression

[112]:
```
# Fit one feature
# Insert code here
lcavol = df_train[['lcavol']]
reg = linear_model.LinearRegression().fit(lcavol, Y_train)
print('R-squared score:', reg.score(lcavol, Y_train))

# plot
# Insert code here
plt.scatter(lcavol, Y_train)
plt.plot(lcavol, reg.predict(lcavol), color='red')
plt.xlabel('lcavol')
plt.ylabel('lpsa')
plt.title('lpsa vs lcavol')
```

```
R-squared score: 0.5375164690552882
```

[112]:
```
Text(0.5, 1.0, 'lpsa vs lcavol')
```



1.2 Problem 2: Best Subset Feature Selection

Now we will look at finding the best subset of features out of all possible subsets. To do this, you will implement the Best Feature Subset Selection as presented in lecture (see lecture slides). We will break this into subproblems to walk through it. To help you with this we have provided a function findsubsets(S,k). When passed a set S this function will return a set of all subsets of size k, which you can iterate through to train models.

[113]:
```python
def findsubsets(S,k):
    return set(itertools.combinations(S, k))
```

1.2.1 (a)

We will start by getting familiar with the findsubsets() function. The variable 'features' was defined previously as a set of all feature names. In the cell do the following: * Use findsubsets to find all possible subsets of 3 features * Perform 5-fold cross validation to train a LinearRegression model on each set of 3 features * Find the model with the highest average R^2 score (scoring='r2') * Report the best performing set of features and the corresponding R^2 score

Documentation - Scikit-Learn - cross_val_score

[114]:
```
# Insert code here
from sklearn.metrics import r2_score
subsets = findsubsets(features, 3)
max_r2 = 0
best_subset = []
for subset in subsets:
    subset_features = []
    for feature in subset:
        subset_features.append(feature)
    X_train_subset = X_train[subset_features]
    reg_model = linear_model.LinearRegression().fit(X_train_subset, Y_train)
    cv_score = cross_val_score(reg_model, X_train_subset, Y_train, cv=5,␣
 ↪scoring='r2')
    r2 = r2_score(Y_train, reg_model.predict(X_train_subset))
    if (r2 > max_r2):
        max_r2 = r2
        best_subset = subset_features
print('r2 score:', max_r2)
print('best set of features', best_subset)
```

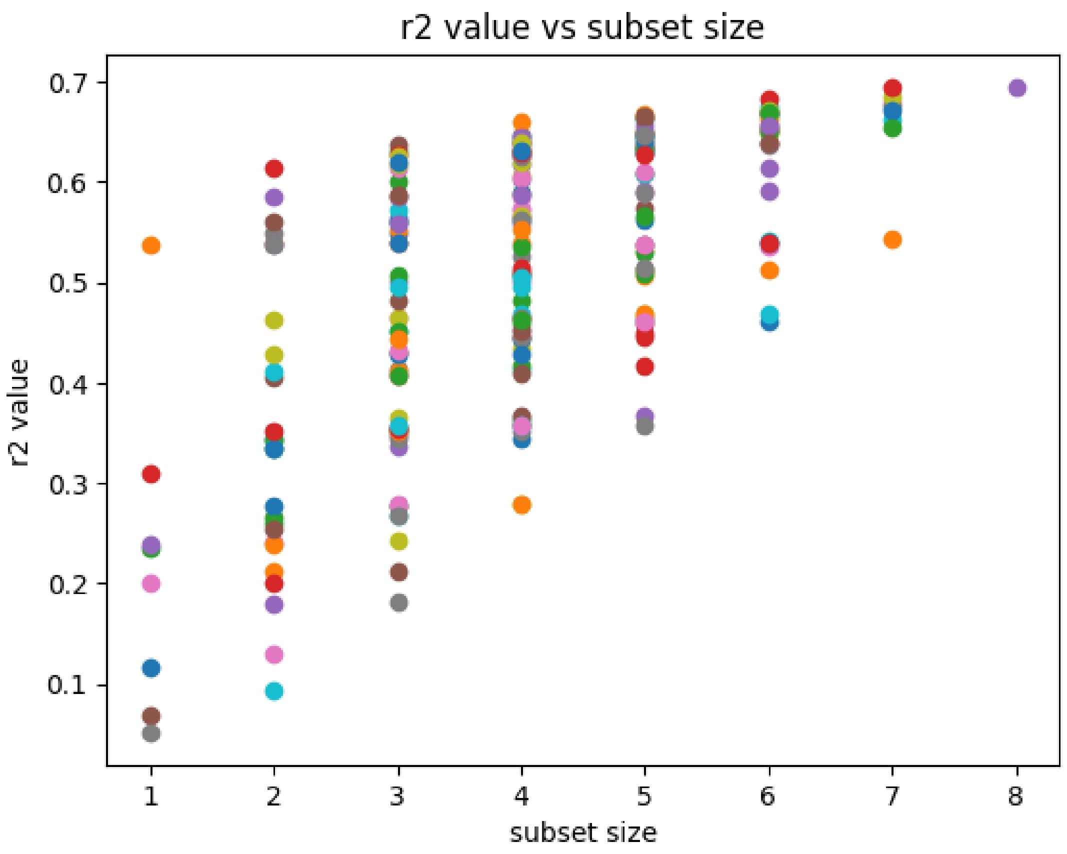
```
r2 score: 0.6374405385171893
best set of features ['lcavol', 'lweight', 'svi']
```

1.2.2 (b)

Now, repeat the above process for all subsets of all sizes. For each $k = 1, \ldots, 8$ find all possible subsets of k features and evaluate a model on each set of features using 5-fold cross validation. Report your findings as follows, * Produce a scatterplot of R^2 values for every run with subset size on the horizontal axis, and R^2 on the vertical axis (label your plot axes/title) * Find the best performing model overall and report the R^2 and features for that model

[115]:
```
# Insert code here
max_r2 = 0
best_subset = []
for k in range(8):
    subsets = findsubsets(features, k + 1)
    for subset in subsets:
        subset_features = []
        for feature in subset:
            subset_features.append(feature)
        X_train_subset = X_train[subset_features]
        reg_model = linear_model.LinearRegression().fit(X_train_subset, Y_train)
        cv_score = cross_val_score(reg_model, X_train_subset, Y_train, cv=5,␣
 ↪scoring='r2')
        r2 = r2_score(Y_train, reg_model.predict(X_train_subset))
        if (r2 > max_r2):
            max_r2 = r2
            best_subset = subset_features
        plt.scatter(k + 1, r2)
plt.xlabel('subset size')
plt.ylabel('r2 value')
plt.title('r2 value vs subset size')
print('best r2 score:', max_r2)
print('best set of features', best_subset)
```

```
best r2 score: 0.6943711796768238
best set of features ['lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp',
'gleason', 'pgg45']
```



Excellent You have found the best set of features by brute-force search over all possible features. Good work.

1.3 Problem 3 : Ridge Regression

1.3.1 (a)

The problem with brute force search over features is that it doesn't scale well. We can do it for 8 features, but we can't do it for larger sets of features. Instead, we will look at a simpler model selection strategy by using L2 regularized linear regression (a.k.a. Ridge Regression). Do the following in the cell below, * Learn a Ridge regression model on training data with alpha=0.5 * Report the learned feature weights using the provided printFeatureWeights function

Documentation - Scikit-Learn - linear_model.Ridge

[116]:
```
def printFeatureWeights(f, w):
  for idx in range(len(f)):
    print('%s : %f' % (f[idx], w[idx]))

# Insert code here
ridge = linear_model.Ridge(alpha=0.5).fit(X_train, Y_train)
printFeatureWeights(ridge.feature_names_in_, ridge.coef_[0])
```

```
lcavol : 0.576706
lweight : 0.593447
age : -0.018544
lbph : 0.145617
svi : 0.683643
lcp : -0.193621
gleason : -0.034175
pgg45 : 0.009508
```

1.3.2 (b)

We chose the regularization coefficient alpha=0.5 somewhat arbitrarily. We now need to perform model selection in order to learn the best value of alpha. We will do that by using cross_val_score over a range of values for alpha. When searching for regularization parameters it is generally good practice to search in log-domain, rather than linear domain. For example, we will search in the range $[10^{-1}, 10^3]$. Using Numpy's "logspace" function this corresponds to the range $[-1, 3]$ in log-domain. In the cell below do the following, * Create a range of 50 alpha values spaced logarithmically in the range $[10^{-1}, 10^3]$ * Perform 5-fold cross-validation of Ridge regression model for each alpha and record R^2 score for each run (there will be 5x50 values) * Report the best R^2 score and the value of alpha that achieves that score * Use Matplotlib errorbar() function to plot the average R^2 with 1 standard deviation error bars for each of the 50 alpha values

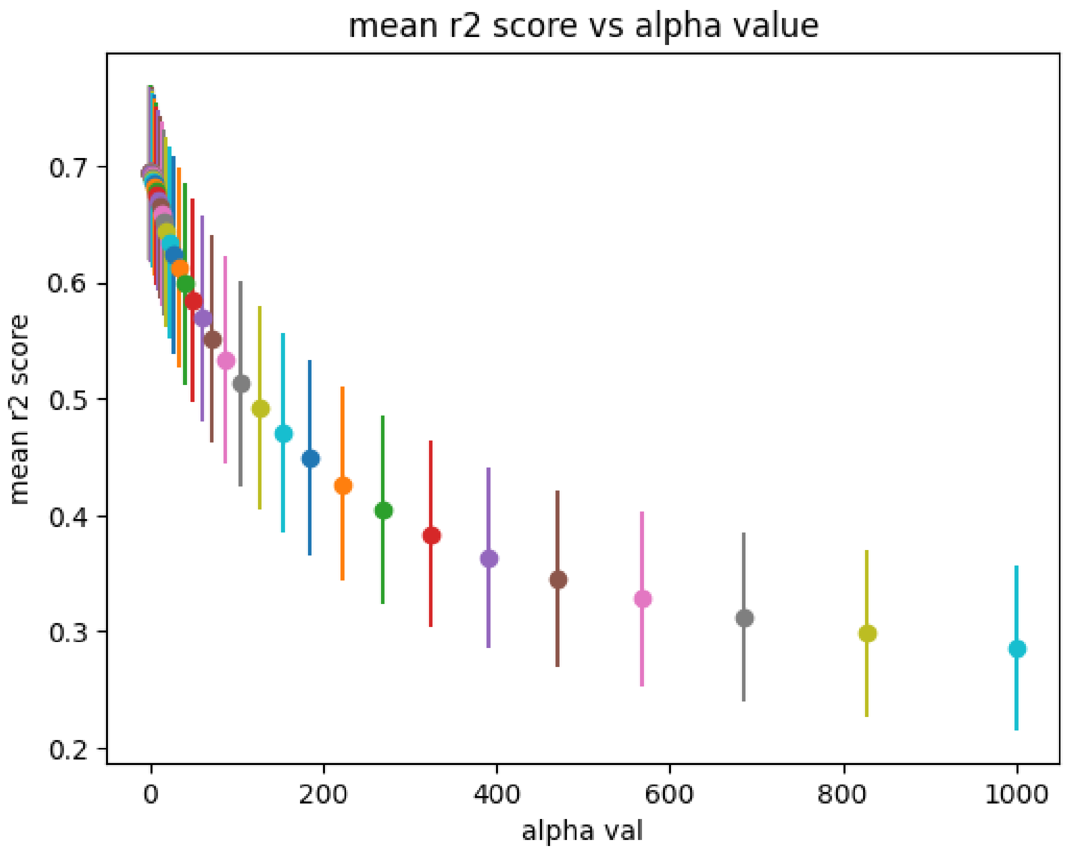
Documentation - Matplotlib - errorbar

Documentation - Numpy - logspace

[128]:
```
# Insert code here
alpha_vals = np.logspace(-1, 3, num=50)
max_r2 = 0
best_alpha = 0
for alpha_val in alpha_vals:
    ridge_model = linear_model.Ridge(alpha=alpha_val).fit(X_train, Y_train)
    cv_score = cross_val_score(ridge_model, X_train, Y_train, cv=5,␣
  ↪scoring='r2')
    cv_results = cross_validate(ridge_model, X_train, Y_train, cv = 5,␣
  ↪scoring=['r2'], return_train_score=True)
    # print('cv_results:', np.mean(cv_results['train_r2']))
    r2 = r2_score(Y_train, ridge_model.predict(X_train))
    # print(r2)
    if (np.mean(cv_results['train_r2']) > max_r2):
        max_r2 = np.mean(cv_results['train_r2'])
        best_alpha = alpha_val
    plt.errorbar(alpha_val, r2, np.std(cv_results['train_r2']), fmt='o-')
print('best r2 score:', max_r2)
print('best alpha score:', best_alpha)
plt.xlabel('alpha val')
plt.ylabel('mean r2 score')
plt.title('mean r2 score vs alpha value')
```

```
best r2 score: 0.6855502279605263
best alpha score: 0.1
```

[128]:
```
Text(0.5, 1.0, 'mean r2 score vs alpha value')
```



Now that we have a good model we will look at what it has learned. Train the Ridge regression model using the selected alpha from the previous cell. Report the learned feature weights using the printFeatureWeights() function previously provided.

[129]:
```
ridge = linear_model.Ridge(alpha=0.1).fit(X_train, Y_train)
printFeatureWeights(ridge.feature_names_in_, ridge.coef_[0])
```

```
lcavol : 0.576647
lweight : 0.609801
age : -0.018907
lbph : 0.144993
svi : 0.725754
lcp : -0.203669
gleason : -0.030539
pgg45 : 0.009476
```

1.4 Problem 4 : LASSO

Ridge regression performs shrinkage of the weights using the L2 norm. This will drive some weights *close* to zero, but not exactly zero. The LASSO method replaces the L2 penalty with an L1 penalty. Due to properties of L1 discussed in lecture, this has the effect of learning exactly zero weights on some features when it is supported by the data. In this problem we will repeat procedure of learning a Ridge regression model, but we will instead use LASSO. Let's start by fitting a LASSO model with a fixed alpha value.

1.4.1 (a)

In the cell below do the following, * Fit LASSO with alpha=0.1 * Use printFeatureWeights() to report the learned feature weights

Documentation - Scikit-Learn - linear_model.Lasso

[135]:
```
# Insert code here
lasso = linear_model.Lasso(alpha=0.1).fit(X_train, Y_train)
printFeatureWeights(lasso.feature_names_in_, lasso.coef_)
```

```
lcavol : 0.538986
lweight : 0.184891
age : -0.006352
lbph : 0.128433
svi : 0.000000
lcp : -0.000000
gleason : -0.000000
pgg45 : 0.007727
```

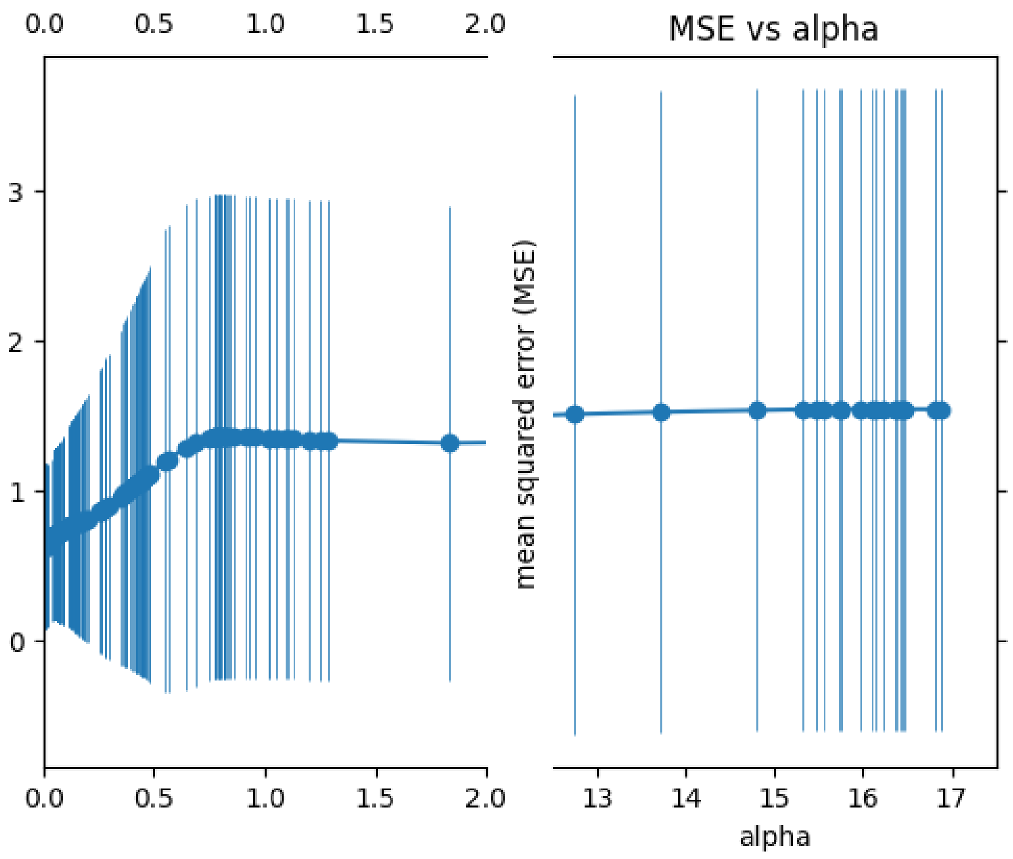
1.4.2 (b)

Now we will find a good value of alpha using cross-validation. Due to differences in how the LASSO model is optimized, there are dedicated methods for performing cross-validation on LASSO. Scikit-Learn's LassoLarsCV class performs LASSO-specific cross-validation using an optimized Least Angle Regression (LARS) algorithm. In the cell below do the following, * Using LassoLarsCV perform 20-fold cross validation to solve all solution paths for Lasso * Plot mean +/- standard error of **mean squared error** versus regularization coefficient α * Title the plot and axes * Report the best alpha value and the corresponding average mean squared error from cross-validation

Note: LassoLarsCV returns mean squared error, rather than R^2. It also determines the set of α values automatically, which are stored in the cv_alphas_ attribute.

Documentation - Scikit-Learn - LassoLarsCV

[174]:
```
# Insert code here
from sklearn.linear_model import LassoLarsCV
llcv = LassoLarsCV(cv=20).fit(X_train, Y_train)
# print(llcv.alpha_)
# print(llcv.cv_alphas_)
# print(llcv.mse_path_)
# print(llcv)
# plt.errorbar(llcv.cv_alphas_, [np.mean(mse) for mse in llcv.mse_path_],␣
 ↪fmt='o-')

fig,(ax,ax2) = plt.subplots(1, 2, sharey=True)

# plot the same data on both axes
ax.errorbar(llcv.cv_alphas_, [np.mean(mse) for mse in llcv.mse_path_], [np.
 ↪std(mse) for mse in llcv.mse_path_], fmt='o-', elinewidth=0.5)
ax2.errorbar(llcv.cv_alphas_, [np.mean(mse) for mse in llcv.mse_path_], [np.
 ↪std(mse) for mse in llcv.mse_path_], fmt='o-', elinewidth=0.5)

# zoom-in / limit the view to different portions of the data
ax.set_xlim(0, 2) # most of the data
ax2.set_xlim(12.5,17.5) # outliers only

# hide the spines between ax and ax2
ax.spines['right'].set_visible(False)
ax2.spines['left'].set_visible(False)
ax.yaxis.tick_left()
ax.tick_params(labeltop='off') # don't put tick labels at the top
ax2.yaxis.tick_right()


# Make the spacing between the two axes a bit smaller
plt.subplots_adjust(wspace=0.15)

plt.title('MSE vs alpha')
plt.xlabel('alpha')
plt.ylabel('mean squared error (MSE)')
plt.show()


print('best alpha:', llcv.alpha_)
print('corresponding mse:', np.min([np.mean(mse) for mse in llcv.mse_path_]))
```



```
best alpha: 0.011311646934499086
corresponding mse: 0.6322147354954939
```

1.5 Problem 5 : Evaluate on Test

In this problem we will train all of the best performing models chosen by Best Subsets, Ridge Regression, and LASSO. We will evaluate and compare these models on the test data. This dataset uses a standard train / test split so we begin by loading test data below.

```
[175]: df_test = pd.read_csv('prostate_test.csv')
       df_test.head()
```

```
[175]:      lcavol   lweight  age      lbph  svi       lcp  gleason  pgg45      lpsa
       0  0.737164  3.473518   64  0.615186    0 -1.386294        6      0  0.765468
       1 -0.776529  3.539509   47 -1.386294    0 -1.386294        6      0  1.047319
       2  0.223144  3.244544   63 -1.386294    0 -1.386294        6      0  1.047319
       3  1.205971  3.442019   57 -1.386294    0 -0.430783        7      5  1.398717
       4  2.059239  3.501043   60  1.474763    0  1.348073        7     20  1.658228
```

1.5.1 (a)

Recall that all of the data are stored in a single table, with the final column being the output 'lpsa'. Before evaluating on test you must first create X_test and Y_test input/outputs where Y_test is the final column of the DataFrame, and X_test contains all other columns.

[176]:
```
# Insert code here
X_test = df_test[features]
Y_test = df_test[output]
```

1.5.2 (b) Best Subsets

In Problem 2 you found the best subset of features for an ordinary least squares regression model by enumerating all feature subsets. Using the best selected features train the model below and report mean squared error on the test set.

Documentation - Scikit-Learn - MeanSquaredError

[178]:
```
# Insert code here
from sklearn.metrics import mean_squared_error
reg_model = linear_model.LinearRegression().fit(X_test, Y_test)
mean_squared_error(Y_test, reg_model.predict(X_test))
```

[178]:
```
0.3231865312292136
```

1.5.3 (c) Ridge Regression

In the cell below, train a Ridge Regression model using the optimal regularization coefficient (α) found in Problem 2. Report mean squared error on the test set.

[180]:
```
# Insert code here
ridge_model = linear_model.Ridge(alpha=0.1).fit(X_test, Y_test)
mean_squared_error(Y_test, ridge_model.predict(X_test))
```

[180]:
```
0.32330219337541183
```

1.5.4 (d) LASSO Regression

Now, train and evaluate your final model. Train a Lasso regression using the optimal α parameters from Problem 3 and report MSE on the test set.

[182]:
```
# Insert code here
lasso = linear_model.Lasso(alpha=0.011311646934499086).fit(X_test, Y_test)
mean_squared_error(Y_test, lasso.predict(X_test))
```

[182]:
```
0.3271770465320337
```

1.5.5 (e) Compare feature weights for each model

Now let's compare the feature weight learned by each of the three models. In the cell below, report the regression weights for each feature under Best Subset, Ridge, and Lasso models evaluated above.

To make the output easier to read, please use a Pandas DataFrame to display the data. To do this, create a Pandas DataFrame where each column contains regression weights for one of the previous models, and then display that DataFrame in the standard fashion. You should also provide feature names on each of the rows.

Documentation - Pandas - DataFrame

[205]:
```
# Insert code here
models = ['Best Subsets', 'Ridge Regression', 'LASSO Regression']
# printFeatureWeights(reg_model.feature_names_in_, reg_model.coef_[0])
# print()
# printFeatureWeights(ridge_model.feature_names_in_, ridge_model.coef_[0])
# print()
# printFeatureWeights(lasso.feature_names_in_, lasso.coef_)
feature_weights = pd.DataFrame([reg_model.coef_[0], ridge_model.coef_[0], lasso.
 ↪coef_], models, features)
fw = pd.concat(
    [pd.concat(
        [feature_weights],
        keys=['Feature Weights'], axis=1)],
)

fw.T
```

[205]:
```
                                Best Subsets  Ridge Regression  LASSO Regression
Feature Weights lcavol              0.455704          0.453945          0.452811
                lweight             0.555361          0.521677          0.329346
                age                -0.008942         -0.008170         -0.005000
                lbph               -0.081018         -0.080513         -0.064796
                svi                 0.659736          0.631986          0.537037
                lcp                 0.169708          0.176789          0.192480
                gleason            -0.018438         -0.018719         -0.000000
                pgg45               0.000716          0.000733          0.000427
```