# ps3_problem4

May 14, 2020

## 0.1 IDS/ACM/CS 158: Fundamentals of Statistical Learning

### 0.1.1 PS3, Problem 4: Ridge Regression

Name: Li, Michael

Email address: mlli@caltech.edu

Notes: Please use python 3.6

You are required to properly comment and organize your code.

- Helper functions (add/remove part label according to the specific question requirements)

[1]:
```python
import numpy as np
import numpy.matlib
import scipy.stats
import itertools
import matplotlib.pyplot as plt
from collections import defaultdict

def predict(ols, data, y_avg):
    """
    ols - ols estimate of the regression parameter
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    y_avg - average y prediction from the training data
            to add back when we predict

    returns the predictions for the observations in data
    """
    return np.matmul(data[:,:-1], ols) + y_avg

def ridge_regression(data, lamb):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    lamb - lambda to fit the ridge estimates with
```

```
    returns the ridge estimate of the regression parameter
    """
    x = data[:,:-1]
    y = data[:,-1]

    intermediate = np.matmul(x.transpose(), x)
    regularization = lamb * np.identity(len(x[0]))
    inverse_intermediate = np.linalg.inv(np.array(intermediate) +␣
 ↪regularization)
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    return np.matmul(pseudo_x, y)

def l2_loss(data, preds):
    """
    data - a matrix where each row corresponds to the
           p predictors in the first p columns and
           the observed output y in the final column
    preds - the predictions for the observations in data

    returns the L2 loss of the values
    """
    return np.mean((data[:,-1] - preds)**2)

def split_folds(folds, index):
    """
    folds - list of K folds of data
    index - which of the folds to use for test data

    returns train and test of the data
    """
    test = folds[index]
    train_temp = np.delete(folds, index, axis=0)
    train = []

    for fold in train_temp:
        for row in fold:
            train.append(row)

    return np.array(train), test

def kfolds(data):
    """
    data - data to split into 5 folds

    returns 5 different folds of data
```

```
    """
    np.random.shuffle(data)
    return [data[:19], data[19:38], data[38:57], data[57:77], data[77:]]

def mean_and_se(data):
    """
    data - a column of data

    returns the mean of the data and standard error
    """
    mean = np.mean(data)
    se = np.sqrt(np.mean((data-mean)**2))

    return mean, se
```

[2]:

```
class RidgePreprocessor:
    """
    Object that keeps track of the training preprocesing step

    Initialize with data and object keeps track of
    mean of each column, standard deviations of each column, and
    the average y of the data
    """
    def __init__(self, data):
        self.means = np.mean(data[:,:-1], axis=0)
        self.stds = np.std(data[:,:-1], axis=0, ddof=1)
        self.y_avg= np.mean(data[:,-1])

    def _standardize_col(self, column, mean, std):
        """
        column - an np array of values from a population

        returns the standardized column with mean 0 and std = 1
        """
        return (column - mean) / std

    def preprocess(self, data):
        """
        given a dataset, standardize it using the saved means, stds, and y_avg
        """
        standardized_data = data.copy()

        for i in range(len(data[0])-1):
            standardized_data[:,i] = self._standardize_col(data[:,i], self.
↪means[i], self.stds[i])

        standardized_data[:,-1] -= self.y_avg
```

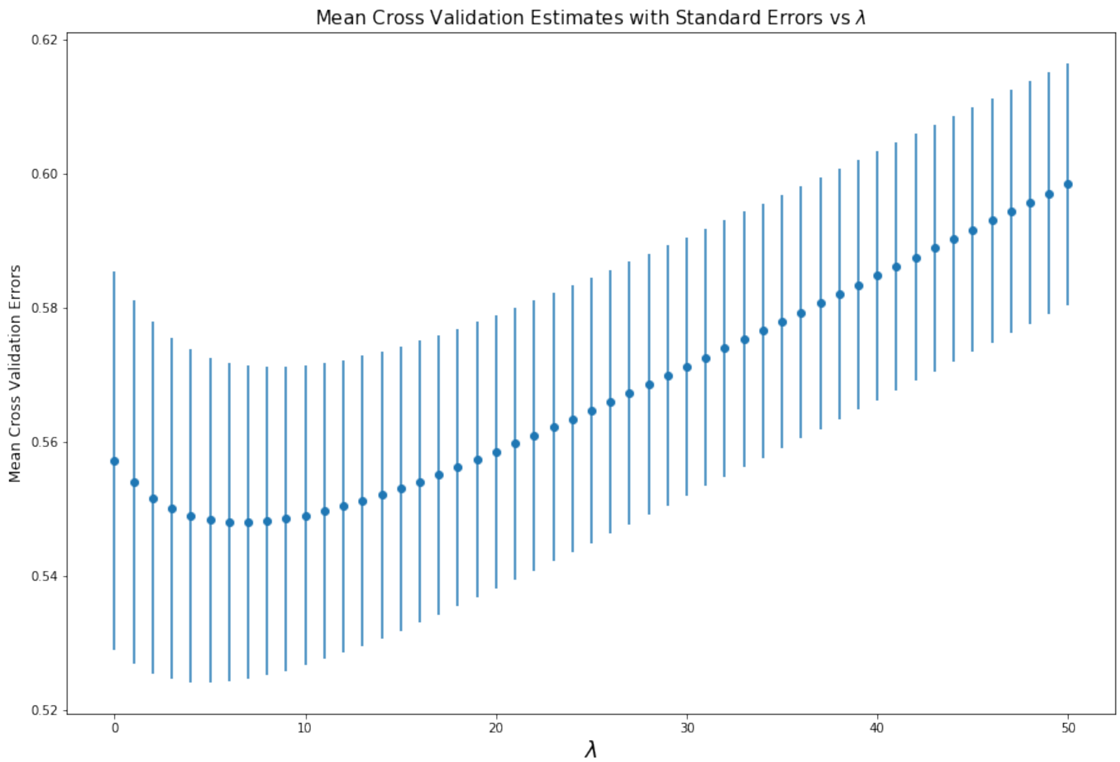
```python
        return standardized_data

    def get_y_avg(self):
        """
        Getter method to get the y_avg value
        """
        return self.y_avg
```

[3]:
```python
data = np.genfromtxt('prostate_cancer.csv', delimiter=',', skip_header=1)[:,:-1]
```

[24]:
```python
cvs = defaultdict(list)

for _ in range(100):
    # split up our data into folds
    folds = kfolds(data)

    for lamb in range(51):
        cv_err = []

        for k in range(len(folds)):
            # organize our train and test and preprocess everything according␣
 ↪to train
            train, test = split_folds(folds, k)
            preprocessor = RidgePreprocessor(train)
            train_processed = preprocessor.preprocess(train)
            test_processed = preprocessor.preprocess(test)

            # calculate ridge estimates and calculate error for fold
            ols_ridge = ridge_regression(train_processed, lamb)
            test_preds = predict(ols_ridge, test_processed, preprocessor.
 ↪get_y_avg())
            cv_err.append(l2_loss(test, test_preds))

        # keep track of average cross validation error for 5 folds
        cvs[lamb].append(np.mean(cv_err))
```

[25]:
```python
means = []
ses = []

for key in cvs:
    mean, se = mean_and_se(cvs[key])
    means.append(mean)
    ses.append(se)

plt.rcParams['figure.figsize'] = [15, 10]
```

```python
plt.xlabel('$\lambda$', fontsize=18)
plt.ylabel('Mean Cross Validation Errors', fontsize=12)
plt.title('Mean Cross Validation Estimates with Standard Errors vs $\lambda$',␣
 ↪fontsize=15)
plt.errorbar(np.arange(51), means, ses, linestyle='None', marker='o')
plt.show()
```




[26]:
```python
lamb_min = np.argmin(means)
lamb_min
```

[26]: 7

[27]:
```python
lamb_best = None

for i in range(len(means)-1, -1, -1):
    if means[i] < means[lamb_min] + ses[lamb_min]:
        lamb_best = i
        break

lamb_best
```

[27]: 30

[41]:

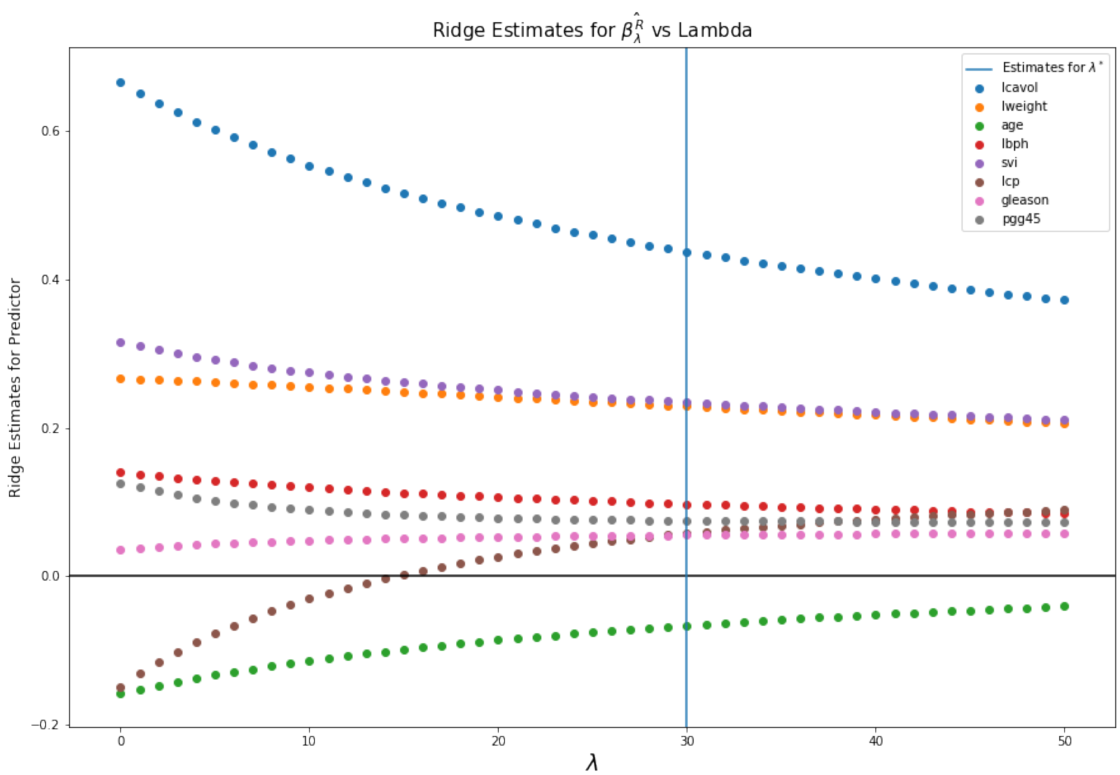
```
data = np.genfromtxt('prostate_cancer.csv', delimiter=',', skip_header=1)[:,:-1]
preprocessor = RidgePreprocessor(data)
data = preprocessor.preprocess(data)
```

[42]:

```
betas = []

for lamb in range(51):
    betas.append(ridge_regression(data, lamb))
```

[43]:

```
labels = ["lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason", "pgg45"]
plt.rcParams['figure.figsize'] = [15, 10]
for i in range(len(labels)):
    plt.scatter(np.arange(51), np.array(betas)[:,i], label=labels[i])

plt.axhline(y=0, color='k')
plt.axvline(lamb_best, label='Estimates for $\lambda^*$')
plt.legend()
plt.xlabel('$\lambda$', fontsize=18)
plt.ylabel('Ridge Estimates for Predictor', fontsize=12)
plt.title(r'Ridge Estimates for $\hat{\beta_\lambda^R}$ vs Lambda', fontsize=15)
plt.show()
```



[44]:
```
betas[lamb_best], preprocessor.get_y_avg()
```

[44]:
```
(array([ 0.43733146,  0.2288155 , -0.0662534 ,  0.09746244,  0.23424639,
         0.05798012,  0.0554223 ,  0.07499681]),
 2.478386878350515)
```

The best final model is $f(X) = \bar{y} + \sum_{i=1}^{p} \beta_{i,\lambda^*}^{\hat{R}} X_i$ where $\beta_{i,\lambda^*}^{\hat{R}}$ = [ 0.43733146, 0.2288155 , -0.0662534 , 0.09746244, 0.23424639, 0.05798012, 0.0554223 , 0.07499681] and $\bar{y} = 2.478$