ps3_problem3

May 14, 2020

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

0.1.1 PS3, Problem 3: Best Subset Selection

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]:
```
import numpy as np
import numpy.matlib
import scipy.stats
import itertools
import matplotlib.pyplot as plt

def standardize_col(column):
    """
    column - an np array of values from a population

    returns the standardized column with mean 0 and std = 1
    """
    mean = np.mean(column)
    std = np.std(column, ddof=1)

    return (column - mean) / std

def predict(ols, data):
    """
    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

    returns the predictions for the observations in data
    """
```

```
    x_with_bias_term = np.concatenate((np.matlib.repmat(1, len(data), 1), data[:
↪,:-1]), axis=1)
    return np.matmul(x_with_bias_term, ols)

def reduce_data(data, indices):
    """
    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
    indices - which indices to use from the data

    returns the reduced dataset containing only the predictors in indices
    """
    return np.append(data[:,indices], data[:,-1][...,None], 1)

def find_beta(data):
    """
    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

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

    # add bias term to training data
    bias = np.matlib.repmat(1, len(x), 1)
    x = np.concatenate((bias, x), axis=1)

    # calculate beta
    intermediate = np.matmul(x.transpose(), x)
    inverse_intermediate = np.linalg.inv(np.array(intermediate))
    pseudo_x = np.matmul(inverse_intermediate, x.transpose())

    return np.matmul(pseudo_x, y)

def rss(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 residual sum of squares for the values
    """
    return np.sum((data[:,-1] - preds)**2)
```

```
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)
```

[2]:
```
data = np.genfromtxt('prostate_cancer.csv', delimiter=',', skip_header=1)

standardized_data = data.copy()

for i in range(len(data[0])-2):
    standardized_data[:,i] = standardize_col(data[:,i])

# split the data into train and test
train_data = np.array([observation[:-1] for observation in standardized_data if␣
↪observation[-1] == 1])
test_data = np.array([observation[:-1] for observation in standardized_data if␣
↪observation[-1] == 0])
```

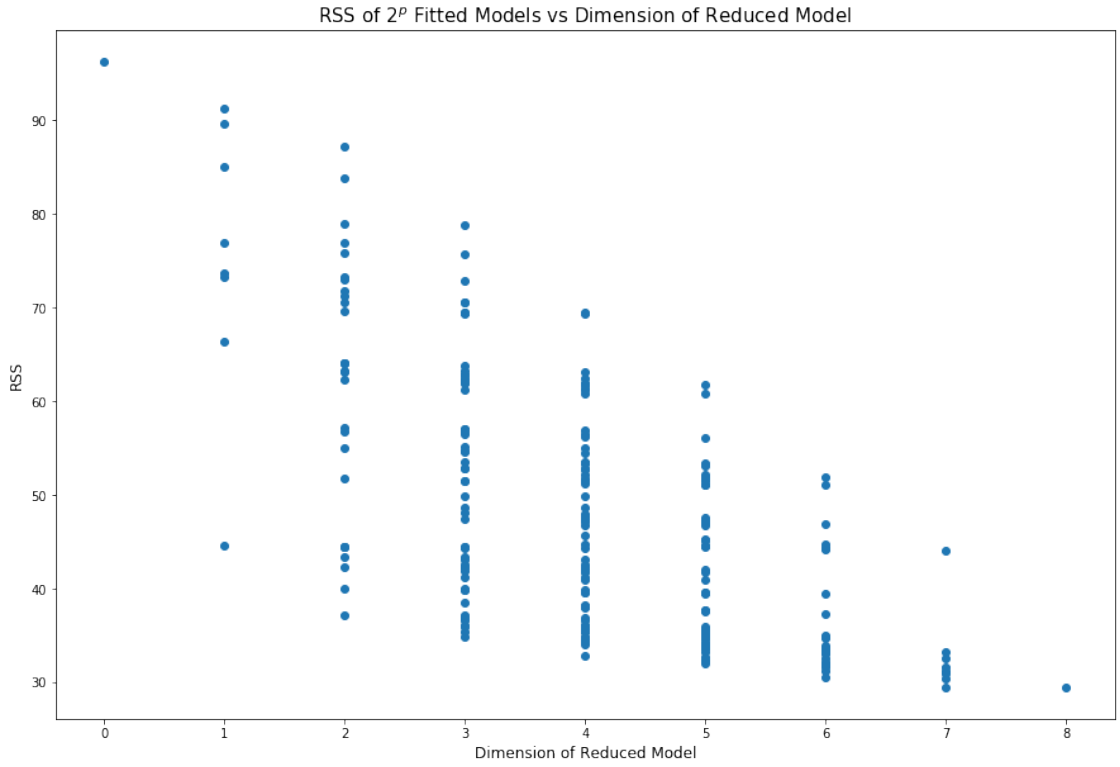
- Part A

[3]:
```
models = {
    0:[],
    1:[],
    2:[],
    3:[],
    4:[],
    5:[],
    6:[],
    7:[],
    8:[]
}
```

[4]:
```
best_models = []

for p_reduced in range(9):
    # keep track of the best model for each p_reduced (rss, indices, beta)
    p_reduced_best = (10**100, None)

    for indices in itertools.combinations(range(len(train_data[0][:-1])),␣
↪p_reduced):
```

```
        # get new dataset and calculate OLS using only indices
        reduced_data = reduce_data(train_data, indices)
        beta = find_beta(reduced_data)

        # find residuals
        preds = predict(beta, reduced_data)
        residuals = rss(reduced_data, preds)
        models[p_reduced].append(residuals)

        # update if model has lower residuals
        if residuals < p_reduced_best[0]:
            p_reduced_best = (residuals, indices, beta)

    # keep track of best model for each p_reduced
    best_models.append((p_reduced_best[1], p_reduced_best[2]))
```

[5]:
```
subset_sizes = []
residuals = []

for key in models:
    for _ in range(len(models[key])):
        subset_sizes.append(key)

for key in models:
    for r in models[key]:
        residuals.append(r)

plt.rcParams['figure.figsize'] = [15, 10]
plt.scatter(subset_sizes, residuals)
plt.xlabel('Dimension of Reduced Model', fontsize=12)
plt.ylabel('RSS', fontsize=12)
plt.title('RSS of $2^p$ Fitted Models vs Dimension of Reduced Model',␣
 ↪fontsize=15)
plt.show()
```



[6]:
```
np.array(best_models)[:,0]
```

[6]:
```
array([(), (0,), (0, 1), (0, 1, 4), (0, 1, 3, 4), (0, 1, 3, 4, 7),
       (0, 1, 3, 4, 5, 7), (0, 1, 2, 3, 4, 5, 7),
       (0, 1, 2, 3, 4, 5, 6, 7)], dtype=object)
```

Best model for $\tilde{p}$=0 includes no inputs

Best model for $\tilde{p}$=1 includes lcavol

Best model for $\tilde{p}$=2 includes lcavol and lweight

Best model for $\tilde{p}$=3 includes lcavol, lweight, and svi

Best model for $\tilde{p}$=4 includes lcavol, lweight, lbph, and svi

Best model for $\tilde{p}$=5 includes lcavol, lweight, lbph, svi, and pgg45

Best model for $\tilde{p}$=6 includes lcavol, lweight, lbph, svi, lcp, and pgg45

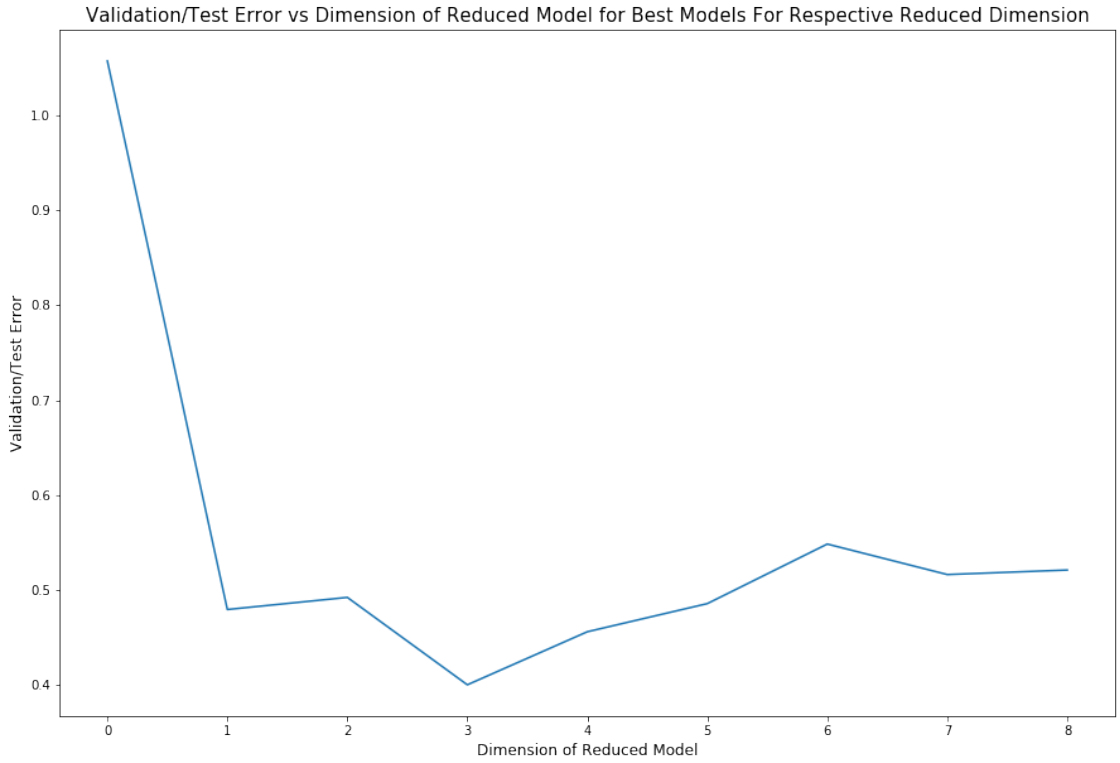
Best model for $\tilde{p}$=7 includes lcavol, lweight, age, lbph, svi, lcp, and pgg45

Best model for $\tilde{p}$=8 includes lcavol, lweight, age, lbph, svi, lcp, gleason, pgg45

[7]:
```
test_errs = []

for indices, beta in best_models:
```

```
    # calculate test error using best reduced models
    reduced_test = reduce_data(test_data, indices)
    preds = predict(beta, reduced_test)
    test_errs.append(l2_loss(reduced_test, preds))
```

[8]:
```
plt.plot(np.arange(len(train_data[0])), test_errs)
plt.xlabel('Dimension of Reduced Model', fontsize=12)
plt.ylabel('Validation/Test Error', fontsize=12)
plt.title('Validation/Test Error vs Dimension of Reduced Model for Best Models␣
 ↪For Respective Reduced Dimension', fontsize=15)
plt.show()
```



[9]:
```
best_models[3]
```

[9]:
```
((0, 1, 4), array([2.46944993, 0.61286858, 0.31565144, 0.22268916]))
```

The best final model is $f(X) = \sum_{i=0}^{p} \hat{\beta} X_i$ where $\hat{\beta}$ = [2.46944993, 0.61286858, 0.31565144, 0, 0, 0.22268916, 0, 0, 0]. Specifically, lcavol, lweight, and svi are inputs.

- Part B

[10]:
```
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 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 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
```

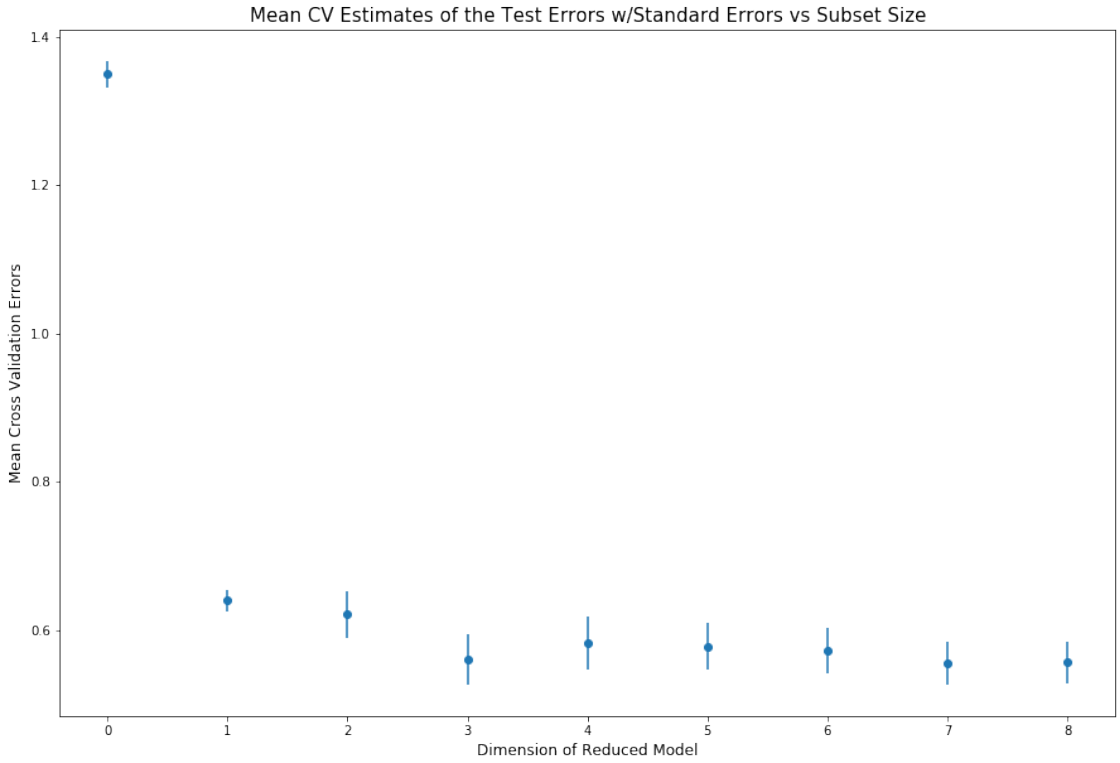
[11]:
```
r_cvs = {
    0:[],
    1:[],
    2:[],
    3:[],
    4:[],
    5:[],
    6:[],
    7:[],
    8:[]
}
```

[12]:
```
for _ in range(100):
    folds = kfolds(standardized_data[:,:-1])
    cvs = {
            0:[],
            1:[],
            2:[],
            3:[],
            4:[],
            5:[],
            6:[],
            7:[],
            8:[]
    }

    for k in range(len(folds)):
        # organize our train and test from the fold split
        train, test = split_folds(folds, k)

        for p_reduced in range((len(train[0]))):
            # keep track of the best model for each p_reduced (rss, indices,␣
↪beta)
            p_reduced_best = (10**100, None)

            for indices in itertools.combinations(range(len(train[0][:-1])),␣
↪p_reduced):
                # reduce the data and calculate the residuals
                reduced_data = reduce_data(train, indices)
                beta = find_beta(reduced_data)
                preds = predict(beta, reduced_data)
                residuals = rss(reduced_data, preds)

                # update the best model if residuals are smaller
                if residuals < p_reduced_best[0]:
                    p_reduced_best = (residuals, indices, beta)

            # retrain best model and calculate the test error
            reduced_train = reduce_data(train, p_reduced_best[1])
            reduced_test = reduce_data(test, p_reduced_best[1])
            preds = predict(p_reduced_best[2], reduced_test)
            cvs[p_reduced].append(l2_loss(reduced_test, preds))

    # keep track of cv err for each run
    for key in cvs:
        r_cvs[key].append(np.mean(cvs[key]))
```

[13]:
```
means = []
ses = []
```

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

[14]:
```
plt.errorbar(np.arange(9), means, ses, linestyle='None', marker='o')
plt.xlabel('Dimension of Reduced Model', fontsize=12)
plt.ylabel('Mean Cross Validation Errors', fontsize=12)
plt.title('Mean CV Estimates of the Test Errors w/Standard Errors vs Subset␣
 ↪Size', fontsize=15)
plt.show()
```



[15]:
```
p_reduced_min = np.argmin(means)
p_reduced_min
```

[15]: 7

[16]:
```
p_reduced_best = None

for i in range(len(means)):
    if means[i] < means[p_reduced_min] + ses[p_reduced_min]:
```

```
        p_reduced_best = i
        break
```

[17]:
```
p_reduced_best
```

[17]:
```
3
```

$\tilde{p}_{min} = 7$ $\tilde{p}^* = 3$

[18]:
```
best_model = (10**100, None)

for indices in itertools.combinations(range(len(standardized_data[0][:-2])),␣
↪p_reduced_best):
    # find all models with only p* predictors and find best one with lowest rss
    reduced_data = reduce_data(standardized_data[:,:-1], indices)
    beta = find_beta(reduced_data)
    preds = predict(beta, reduced_data)
    residuals = rss(reduced_data, preds)

    if residuals < best_model[0]:
        best_model = (residuals, indices, beta)
```

[19]:
```
best_model[1:]
```

[19]:
```
((0, 1, 4), array([2.47838688, 0.61978211, 0.28350966, 0.27558254]))
```

The best final model is $f(X) = \sum_{i=0}^{p} \hat{\beta} X_i$ where $\hat{\beta} =$ [2.47838688, 0.61978211, 0.28350966, 0, 0, 0.27558254, 0, 0, 0]. Specifically, lcavol, lweight, and svi are inputs.