lab7

October 11, 2016

1 EE 379K: Lab 7

1.1 Rohan Nagar and Wenyang Fu

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import linear_model
from sklearn import cross_validation
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.grid_search import GridSearchCV

from statsmodels.formula.api import ols

%matplotlib inline
```

```
C:\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning: This module was depre
  "This module will be removed in 0.20.", DeprecationWarning)
C:\Anaconda3\lib\site-packages\sklearn\grid_search.py:43: DeprecationWarning: This module was deprecated
  DeprecationWarning)
```

2 Problem 1

This question should be answered using the `Carseats` data set.

In [2]:
```
carseats = pd.read_csv('Data/Carseats.csv')
carseats.drop(carseats.columns[0], axis=1, inplace=True)
carseats.head()
```

Out[2]:
```
    Sales  CompPrice  Income  Advertising  Population  Price ShelveLoc  Age  \
0    9.50        138      73           11         276    120       Bad   42
1   11.22        111      48           16         260     83      Good   65
2   10.06        113      35           10         269     80    Medium   59
3    7.40        117     100            4         466     97    Medium   55
4    4.15        141      64            3         340    128       Bad   38

   Education Urban   US
0         17   Yes  Yes
1         10   Yes  Yes
2         12   Yes  Yes
3         14   Yes  Yes
4         13   Yes   No
```

2.0.1 Part A

Fit a multiple regression model to predict `Sales` using `Price`, `Urban`, and `US`.

```
In [3]: carseats = carseats.replace(['Yes', 'No'], [1, 0])

        regr = linear_model.LinearRegression()
        X = carseats[['Price', 'Urban', 'US']]
        y = carseats['Sales']
        regr.fit(X, y)

        print('Coefficients: {}'.format(regr.coef_))
        print('Intercept: {}'.format(regr.intercept_))
        print('R^2: {}'.format(regr.score(X, y)))

        model = ols("Sales ~ Price + Urban + US", carseats).fit()
        model.summary()
```

```
Coefficients: [-0.05445885 -0.02191615  1.2005727 ]
Intercept: 13.043468936764896
R^2: 0.23927539218405547
```

```
Out[3]: <class 'statsmodels.iolib.summary.Summary'>
        """
                                    OLS Regression Results
        ==============================================================================
        Dep. Variable:                  Sales   R-squared:                       0.239
        Model:                            OLS   Adj. R-squared:                  0.234
        Method:                 Least Squares   F-statistic:                     41.52
        Date:                Tue, 11 Oct 2016   Prob (F-statistic):           2.39e-23
        Time:                        22:05:41   Log-Likelihood:                -927.66
        No. Observations:                 400   AIC:                             1863.
        Df Residuals:                     396   BIC:                             1879.
        Df Model:                           3
        Covariance Type:            nonrobust
        ==============================================================================
                         coef    std err          t      P>|t|      [95.0% Conf. Int.]
        ------------------------------------------------------------------------------
        Intercept     13.0435      0.651     20.036      0.000        11.764    14.323
        Price         -0.0545      0.005    -10.389      0.000        -0.065    -0.044
        Urban         -0.0219      0.272     -0.081      0.936        -0.556     0.512
        US             1.2006      0.259      4.635      0.000         0.691     1.710
        ==============================================================================
        Omnibus:                        0.676   Durbin-Watson:                   1.912
        Prob(Omnibus):                  0.713   Jarque-Bera (JB):                0.758
        Skew:                           0.093   Prob(JB):                        0.684
        Kurtosis:                       2.897   Cond. No.                         628.
        ==============================================================================

        Warnings:
        [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
        """
```

2.0.2 Part B

Provide an interpretation of each coefficient in the model. Be careful - some of the variables in the model are qualitative!

2.0.3 Answer

1. The first coefficient denotes that more expensive car seats are less likely to be sold.
2. The second coefficient denotes that a rural car buyer is slightly more likely to purchase a car seat.
3. The third coefficient denotes that a car buyer in the US is far more likely to purchase a car seat than buyers outside the US.

2.0.4 Part C

Write out the model in equation form, being careful to handle the qualitative variables properly.

$Sales = 13.0435 - .0544x_0 - .02192x_1 + 1.2005x_2$

2.0.5 Part D

For which of the predictors can you reject the null hypothesis $H_0 : \beta_j = 0$?

2.0.6 Answer

Price and US, based on the p-values.

2.0.7 Part E

On the basis of your response to the previous quesion, fit a smaller model that only uses the predictors for which there is evidence of association with the outcome.

In [4]:
```
regr = linear_model.LinearRegression()
X = carseats[['Price', 'US']]
y = carseats['Sales']
regr.fit(X, y)

print('Coefficients: {}'.format(regr.coef_))
print('Intercept: {}'.format(regr.intercept_))
print('R^2: {}'.format(regr.score(X, y)))

model = ols("Sales ~ Price + US", carseats).fit()
model.summary()
```

```
Coefficients: [-0.05447763  1.19964294]
Intercept: 13.03079275461576
R^2: 0.23926288842678567
```

Out[4]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.239
Model:                            OLS   Adj. R-squared:                  0.235
Method:                 Least Squares   F-statistic:                     62.43
Date:                Tue, 11 Oct 2016   Prob (F-statistic):           2.66e-24
Time:                        22:05:41   Log-Likelihood:                -927.66
No. Observations:                 400   AIC:                             1861.
Df Residuals:                     397   BIC:                             1873.
```

```
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept     13.0308      0.631     20.652      0.000        11.790    14.271
Price         -0.0545      0.005    -10.416      0.000        -0.065    -0.044
US             1.1996      0.258      4.641      0.000         0.692     1.708
==============================================================================
Omnibus:                        0.666   Durbin-Watson:                   1.912
Prob(Omnibus):                  0.717   Jarque-Bera (JB):                0.749
Skew:                           0.092   Prob(JB):                        0.688
Kurtosis:                       2.895   Cond. No.                         607.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

2.0.8 Part F

How well do the models in (a) and (e) fit the data?

2.0.9 Answer

The two models fit similarly, with the model in (e) doing ever so slightly better.

2.0.10 Part G

Using the model from (e), obtain 95% confidence intervals for the coefficients.

2.0.11 Answer

1. Price: [-0.065, -0.044]
2. US: [0.692, 1.708]
3. Intercept: [11.790, 14.271]

2.0.12 Part H

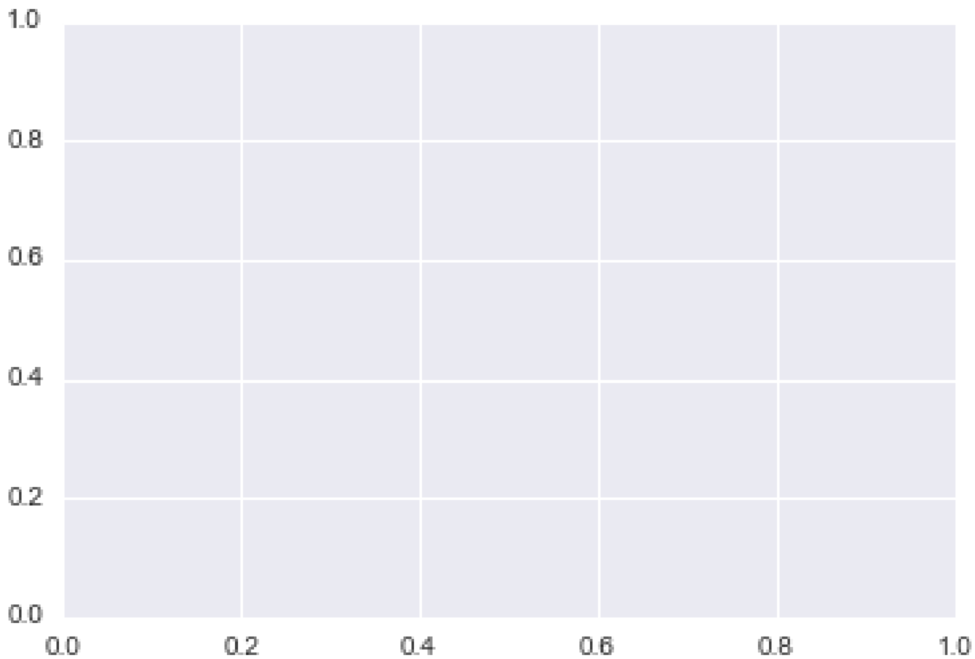
Is there evidence of outliers or high leverage observations in the model from (e)?

2.0.13 Answer

No

In [5]:
```
fig, ax = plt.subplots()
len(regr.predict(X))
# ax.scatter(regr.predict(X), regr.residues_)
```

Out[5]:
```
400
```



3 Problem 2

This problem involves the `Boston` data set, which we saw in the lab for this chapter. We will now try to predict per capita crime rate using the other variables in this data set. In other words, per capita crime rate is the response, and the other variables are the predictors.

```
In [6]: boston = pd.read_csv('Data/Boston.csv')
        boston.dropna()
        boston.head()
```

```
Out[6]:       crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
        0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
        1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
        2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
        3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
        4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

            black  lstat  medv
        0  396.90   4.98  24.0
        1  396.90   9.14  21.6
        2  392.83   4.03  34.7
        3  394.63   2.94  33.4
        4  396.90   5.33  36.2
```

3.0.1 Part A

For each predictor, fit a simple linear regression model to predict the response. Describe your results. In which of the models is there a statistically significant associate between the predictor and the response? Create some plots to back up your assertions.

3.0.2 Answer

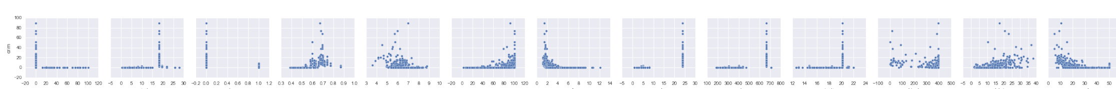
Everything except *chas* is statistically significant.

```
In [7]: X = boston.drop(boston.columns[0], axis=1)
        y = boston.crim
        cols = X.columns
        scores = []
        coefficients = []
        for col in cols:
            regr = linear_model.LinearRegression()
            X_train = X[col].reshape(-1, 1)
            regr.fit(X_train, y)
            coefficients.append(regr.coef_)
            scores.append((col, regr.score(X_train, y)))
            print("Feature {}’s R^2 is: {}".format(col, regr.score(X_train, y)))
            # print(’Coefficients: {}’.format(regr.coef_))
            # print(’Intercept: {}\n’.format(regr.intercept_))
        print(sorted(scores, key=lambda x: x[1], reverse=True))
```

```
Feature zn’s R^2 is: 0.04018790803211081
Feature indus’s R^2 is: 0.16531007043075163
Feature chas’s R^2 is: 0.0031238689633057426
Feature nox’s R^2 is: 0.17721718179269375
Feature rm’s R^2 is: 0.048069116716083604
Feature age’s R^2 is: 0.12442145175894635
Feature dis’s R^2 is: 0.1441493749253987
Feature rad’s R^2 is: 0.39125668674998915
Feature tax’s R^2 is: 0.3396142433788123
Feature ptratio’s R^2 is: 0.0840684389437365
Feature black’s R^2 is: 0.1482742394241312
Feature lstat’s R^2 is: 0.20759093253433558
Feature medv’s R^2 is: 0.15078046904975717
[(’rad’, 0.39125668674998915), (’tax’, 0.3396142433788123), (’lstat’, 0.20759093253433558), (’nox’, 0.1
```

```
In [8]: sns.pairplot(boston, x_vars=X.columns ,y_vars=[’crim’])
```

```
Out[8]: <seaborn.axisgrid.PairGrid at 0x20bf7fdf400>
```



3.0.3 Part B

Fit a multiple regression model to predict the response using all of the predictors. Describe your results. For which predictors can we reject the null hypothesis $H_0 : \beta_j = 0$?

3.0.4 Answer

zn, dis, rad, black, medv

```
In [9]: all_columns = "+".join(boston.columns.difference(["crim"]))
        formula = "crim ~" + all_columns
```

```
model = ols(formula, boston).fit()
model.summary()

regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

3.0.5 Part C

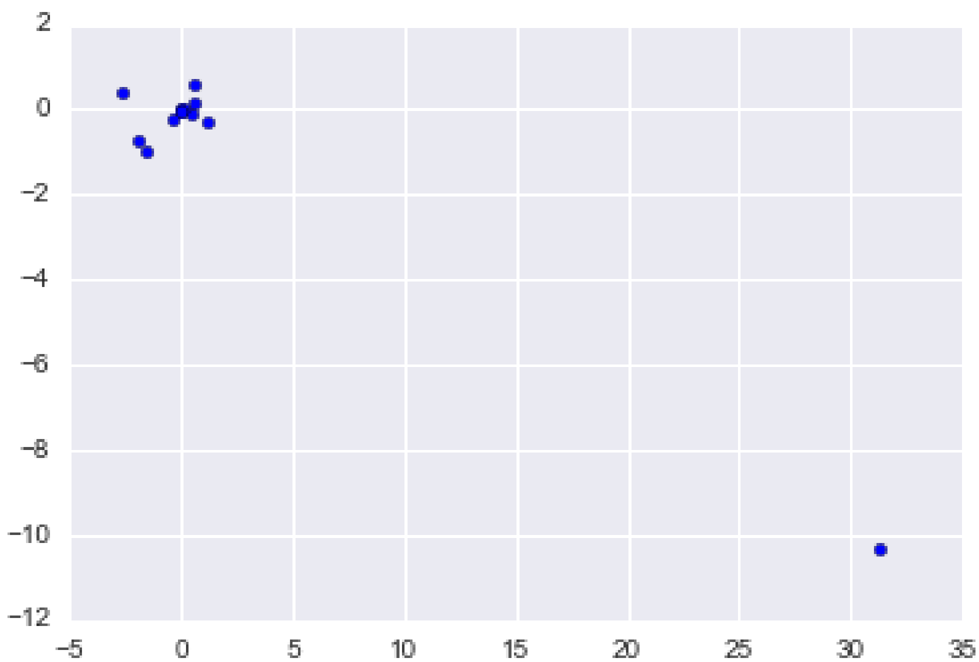
How do your results from (a) compare to your results from (b)? Create a plot displaying the univariate regression coefficients from (a) on the x-axis, and the multiple regression coefficients from (b) on the y-axis. That is, each predictor is displayed as a single point in the plot. Its coefficient in a simple linear regression model is shown on the x-axis, and its coefficient estimate in the multiple linear regression model is shown on the y-axis.

3.0.6 Answer

Coefficients are fairly correlated with the sole exception of *nox*, which is an outlier.

In [10]:
```
fig, ax = plt.subplots()
ax.scatter(coefficients, regr.coef_)
```

Out[10]: <matplotlib.collections.PathCollection at 0x20bfe8ab5c0>



3.0.7 Part D

Is there evidence of non-linear associate between any of the predictors and the response? To answer this question, for each predictor X, fit a model of the form

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$$

Yes, using some R code, there was some non-linear association for all features except black and chas.

In [ ]:

4 Problem 3

We will now try to predict per capita crime rate in the `Boston` data set.

4.0.1 Part A

Try out some of the regression methods explored in this chapter, such as the best subset selection, the lasso, ridge regression, and PCR. Present and discuss results for the approaches that you consider.

4.0.2 Approach 1: Lasso

Lasso does pretty well with a $R^2 = .538$. As you can see, it ends up shrinking the importance of 5 features, and retains the rest.

In [27]:
```
X = boston.drop(boston.columns[0], axis=1)
y = boston.crim
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

regr = linear_model.LassoCV(cv=5)
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
regr.coef_
```

```
0.537588283341
```

Out[27]:
```
array([  2.24761117e-02,  -0.00000000e+00,  -0.00000000e+00,
        -0.00000000e+00,   0.00000000e+00,   2.23614202e-02,
        -1.83611966e-02,   5.12322719e-01,   4.26053805e-05,
        -0.00000000e+00,  -7.46120467e-03,   2.83184483e-02,
        -1.21316678e-01])
```

4.0.3 Approach 2: Ridge

Ridge does slightly better than lasso with a $R^2 = .557$. Unlike LASSO, it doesn't aggressively zero out as many features, but it does scatter the feature weights exponentially across four orders of magnitude.

In [28]:
```
regr = linear_model.RidgeCV(cv=5)
regr.fit(X_train, y_train)
print(regr.score(X_test, y_test))
regr.coef_
```

```
0.557024626663
```

Out[28]:
```
array([  4.36165878e-02,  -3.27519656e-02,  -9.88111269e-01,
        -1.08919102e+01,   5.24808970e-01,   1.00542481e-02,
        -9.92226506e-01,   6.20592963e-01,  -3.75232375e-03,
        -2.92647312e-01,  -5.21585510e-03,   3.22843876e-02,
        -2.31279174e-01])
```

4.0.4 Approach 3: Principal Components Regression (via Partial Least Squares)

PCR seems to do the worst with a $R^2 = .236$. By embedding the original data matrix onto a lower-dimensional subspace, PCR seems to have left out some important information.

```
In [29]: from sklearn.cross_decomposition import PLSRegression

         regr = PLSRegression(n_components=2)
         regr.fit(X_train, y_train)
         print(regr.score(X_test, y_test))
         regr.coef_
```

```
0.235949756809
```

```
Out[29]: array([[ 0.0283097 ],
                [ 0.01922917],
                [-1.54016339],
                [ 1.70274105],
                [ 0.33906391],
                [ 0.00185609],
                [-0.12586492],
                [ 0.26927111],
                [ 0.01053107],
                [ 0.19701884],
                [-0.01176867],
                [ 0.07584012],
                [-0.07348777]])
```

4.0.5 Part B

Propose a model (or set of models) that seem to perform well on this data set, and justify your answer. Make sure that you are evaluating model performance using validation set error, cross-validation, or some other reasonable alternative, as opposed to using training error.

4.0.6 Proposal: Ridge Regression

Ridge regression seems to do well without any tuning, and it has built-in feature selection (so to speak), so we decided to tune it further.The best model selected by Grid Search performed slightly better than the base RidgeCV model we selected.

```
In [54]: from sklearn.model_selection import GridSearchCV

         alphas = list(np.power(10., np.arange(-2, 5)))

         param_grid = [
           {'alpha': alphas, 'normalize': [True, False]},
          ]

         model = linear_model.RidgeCV(alphas, cv=5)
         model.fit(X_train, y_train)
         print(model.alpha_)
         print(model.score(X_test, y_test))

         greg = GridSearchCV(linear_model.Ridge(), param_grid=param_grid,cv=5)
         greg.fit(X_train, y_train)
         print(greg.best_params_)
```

```
print(greg.best_score_ )
print(greg.score(X_test, y_test))
print(greg.bestr_)
```

```
1000.0
0.556817916986
{'alpha': 0.10000000000000001, 'normalize': True}
0.463140350198
0.559753705589
GridSearchCV(cv=5, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params={}, iid=True, n_jobs=1,
       param_grid=[{'alpha': [0.01, 0.10000000000000001, 1.0, 10.0, 100.0, 1000.0, 10000.0], 'normalize'
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

4.0.7 Part C

Does your chosen model involve all of the features in the data set? Why or why not?

4.0.8 Answer

Yes, our model did involve all of the features in the dataset. All of the features were deemed important enough by Ridge Regression (when $\alpha = 1000$) that they each had a final impact on the target variable.

```
In [40]: print(model.coef_)
```

```
[ 0.02674592 -0.06658358 -0.02817974 -0.00879101  0.03970368  0.01769596
 -0.24646743  0.47776279  0.00222968 -0.07355824 -0.0071971   0.05144703
 -0.1390575 ]
```

5 Problem 4

In this exercise, we will predict the number of applications recived using the other variables in the `College` data set.

```
In [15]: college = pd.read_csv('Data/College.csv')
         college.head()
```

Out[15]:

	Unnamed: 0	Private	Apps	Accept	Enroll	Top10perc
0	Abilene Christian University	Yes	1660	1232	721	23
1	Adelphi University	Yes	2186	1924	512	16
2	Adrian College	Yes	1428	1097	336	22
3	Agnes Scott College	Yes	417	349	137	60
4	Alaska Pacific University	Yes	193	146	55	16

	Top25perc	F.Undergrad	P.Undergrad	Outstate	Room.Board	Books	Personal
0	52	2885	537	7440	3300	450	2200
1	29	2683	1227	12280	6450	750	1500
2	50	1036	99	11250	3750	400	1165
3	89	510	63	12960	5450	450	875
4	44	249	869	7560	4120	800	1500

PhD   Terminal   S.F.Ratio   perc.alumni   Expend   Grad.Rate

```
0  70   78   18.1       12   7041    60
1  29   30   12.2       16  10527    56
2  53   66   12.9       30   8735    54
3  92   97    7.7       37  19016    59
4  76   72   11.9        2  10922    15
```

5.0.1 Part A

Split the data set into a training set and a test set.

```
In [16]: college = college.replace(['Yes', 'No'], [1, 0])
```

```
In [17]: features = ['Private', 'Accept', 'Enroll', 'Top10perc', 'F.Undergrad',
                     'P.Undergrad', 'Outstate', 'Room.Board', 'Books', 'Personal',
                     'PhD', 'Terminal', 'S.F.Ratio', 'perc.alumni', 'Expend', 'Grad.Rate']
         X_train, X_test, Y_train, Y_test = train_test_split(college[features], college['Apps'], test_s
```

5.0.2 Part B

Fit a linear model using least squares on the training set, and report the test error obtained.

```
In [18]: from sklearn.metrics import mean_squared_error

         reg = linear_model.LinearRegression()
         reg.fit(X_train, Y_train)

         score = mean_squared_error(Y_test, reg.predict(X_test))
         print("Error on test data: {}".format(score))
```

```
Error on test data: 1793667.3647683186
```

5.0.3 Part C

Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```
In [19]: ridge_reg = linear_model.RidgeCV(alphas=[0.01, 0.1, 0.5, 1, 1.5, 2, 5, 10])
         ridge_reg.fit(X_train, Y_train)

         score = mean_squared_error(Y_test, ridge_reg.predict(X_test))
         print("Error on test data: {}".format(score))
```

```
Error on test data: 1790804.8706922578
```

5.0.4 Part D

Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
In [20]: lasso_reg = linear_model.LassoCV(alphas=[0.01, 0.1, 0.5, 1, 1.5, 2, 5, 10])
         lasso_reg.fit(X_train, Y_train)

         score = mean_squared_error(Y_test, lasso_reg.predict(X_test))
         print("Error on test data: {}".format(score))
```

```
Error on test data: 1793660.783999553
```

5.0.5 Part E

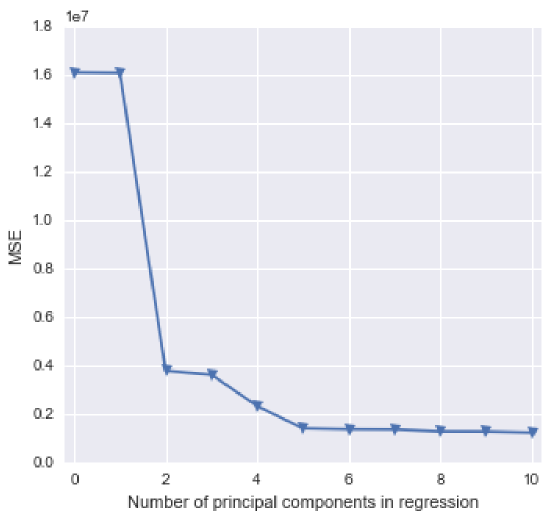
Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

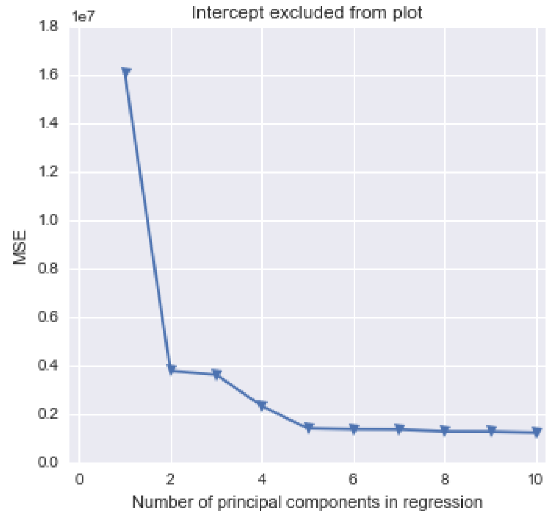
```
In [21]: from sklearn.decomposition import PCA

         pca = PCA()
         X_reduced = pca.fit_transform(X_train)

         # Show how much variance is explained
         np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
Out[21]: array([  49.94,   87.47,   95.73,   97.5 ,   98.71,   99.45,   99.92,
                  99.98,  100.01,  100.01,  100.01,  100.01,  100.01,  100.01,
                 100.01,  100.01])
```

```
In [22]: n = len(X_reduced)
         kf_10 = cross_validation.KFold(n, n_folds=10, shuffle=True, random_state=2)

         # Use Linear Regression with increasing number of pricipal components
         regr = linear_model.LinearRegression()
         mse = []

         # Without any components
         score = -1*cross_validation.cross_val_score(regr, np.ones((n,1)), Y_train.ravel(), cv=kf_10, sc
         mse.append(score)

         # Use 10 components adding one at a time
         for i in np.arange(1,11):
             score = -1*cross_validation.cross_val_score(regr, X_reduced[:,:i], Y_train.ravel(), cv=kf_1
             mse.append(score)

         fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
         ax1.plot(mse, '-v')
         ax2.plot([1,2,3,4,5,6,7,8,9,10], mse[1:11], '-v')
         ax2.set_title('Intercept excluded from plot')

         for ax in fig.axes:
             ax.set_xlabel('Number of principal components in regression')
             ax.set_ylabel('MSE')
             ax.set_xlim((-0.2,10.2))

         fig.show()
```

```
C:\Anaconda3\lib\site-packages\matplotlib\figure.py:397: UserWarning: matplotlib is currently using a no
  "matplotlib is currently using a non-GUI backend, "
```






```
In [23]: # Let's use 5 components, gives us the best score with the least number of components.
         pcr_regr = linear_model.LinearRegression()
         pcr_regr.fit(X_reduced[:,:5], Y_train)

         score = mean_squared_error(Y_test, ridge_reg.predict(X_test))
         print('Error on test data: {}'.format(score))
         print('Value of M selected by CV: 5')
```

```
Error on test data: 1790804.8706922578
Value of M selected by CV: 5
```

6 Part F

Fit a PLS model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
In [24]: from sklearn.cross_decomposition import PLSRegression

         params = {'n_components':[2, 3, 4, 5, 7, 10]}

         pls = PLSRegression()
         pls_reg = GridSearchCV(pls, params, scoring='neg_mean_squared_error')
         pls_reg.fit(X_train, Y_train)

         score = mean_squared_error(Y_test, pls_reg.predict(X_test))
         print("Error on test data: {}".format(score))
         print("Value of M selected by CV: {}".format(pls_reg.best_params_['n_components']))
```

```
Error on test data: 10911489.731786955
Value of M selected by CV: 10
```

6.0.1 Part G

Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

6.0.2 Answer

After trying all 5 approaches, they seemed to result in very similar scores on the test data except for PLS, which did much worse. The best we did was a score of about 1790788, which came from both Ridge Regression and PCR. This looks like the most accurate we can predict the number of college applications recieved. As stated earlier, all 5 models performed similarly save for PLS, which was much worse (about a factor of 5).

7 Problem 5

Generate data of the form $y = X\beta + \epsilon$, where X is a n x p matrix where $n = 51$, $p = 50$, and each X_{ij} ~ $N(0, 1)$. Also, generate the noise according to ϵ_i ~ $N(0, \frac{1}{4}$. Let β be the all ones vector (for simplicity).

By repeatedly doing this experiment and generating fresh data (fresh X, y, and hence ϵ), but keeping β fixed, you will estimate many different solutions $\hat{\beta}$. Estimate the mean and variance of $\hat{\beta}$. Note that $\hat{\beta}$ is a vector, so for this exercise simply estimate the variance of a single component.

Choose regularization coefficients $\lambda = 0.01, 0.1, 1, 10, 100$ and repeat the above experiment. What do you observe? How do you explain this?

```
In [25]: def generate_data():
             X = np.random.randn(51, 50)
             epsilon = (1/4)*np.random.randn(51)
             beta = np.ones(50)
             y = np.dot(X, beta) + epsilon

             return (X, y)

         def estimate_beta_hat(X, y, l=0):
             X_t = X.transpose()
             first_term = np.power(np.dot(X_t, X) + l*np.identity(50), -1)
             temp = np.dot(first_term, X_t)
             return np.dot(temp, y)

         lambdas = [0, 0.01, 0.1, 1, 10, 100]
         for l in lambdas:
             beta_hat_zeros = []
             for i in range(5000):
                 X, y = generate_data()
                 beta_hat = estimate_beta_hat(X, y, l=l)
                 beta_hat_zeros.append(beta_hat[0])

             print('With Lambda = {}'.format(l))
             print('Mean: {}'.format(np.mean(beta_hat_zeros)))
             print('Variance: {}'.format(np.var(beta_hat_zeros)))
             print()
```

```
With Lambda = 0
Mean: -557.0247249562078
Variance: 2041609410.5309222

With Lambda = 0.01
Mean: 582.5252144966123
Variance: 1132450700.985211

With Lambda = 0.1
Mean: 133.53251112850842
```

```
Variance: 237240137.79508066

With Lambda = 1
Mean: -1711.9628437159895
Variance: 16216570457.940891

With Lambda = 10
Mean: -1295.1044065392912
Variance: 5935390658.723576

With Lambda = 100
Mean: 4939.4450696976055
Variance: 68305244969.84221
```

7.0.1 Answer

When introducing a higher regularization coefficient, the mean and variance of the estimated beta value should both shrink. Because a higher value of λ adds more penalty to higher beta values, the beta values should naturally tend toward zero. Especially in our example, since our true betas are all ones, this should help. The mean should shrink as well as the variance.