ISYE 7406 Homework #1

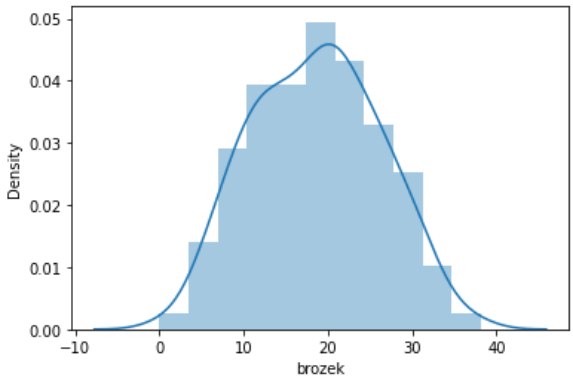
Fall 2022

1. Introduction

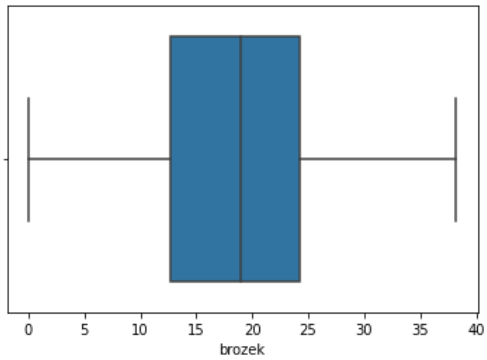
For this homework problem, we are asked to compare various linear regression models on a sample dataset. The dataset in particular is 17 predicting variables and 1 response variable denoted as "brozek". This actually represents the body fat percentage calculated using Brozek's equation. This report will contain preliminary exploratory data analysis, followed by comparison of various methods and their results, then concluded by findings.

2. Exploratory Data Analysis

Firstly, we separate our data into train and test portions where the test set is a specified 25-index set from the homework instructions. After splitting, we take a look at the response variable's distribution.



In this distribution plot, we notice that the response variable follows a normal distribution, which is favored by linear regression.



Next, a boxplot shows whether any of the data points of the response variable are outside of the inter-quartile range. In this case, there are none in the train dataset.

Finally, we look at the correlation between the different predicting variables with the response.

	brozek	siri	density	age	weight	height	adipos	free	neck	chest	abdom	hip	thigh	knee
brozek	1.000000	0.999710	-0.986440	0.279714	0.630109	-0.064712	0.726189	0.052401	0.505780	0.705517	0.820042	0.632590	0.595457	0.532855

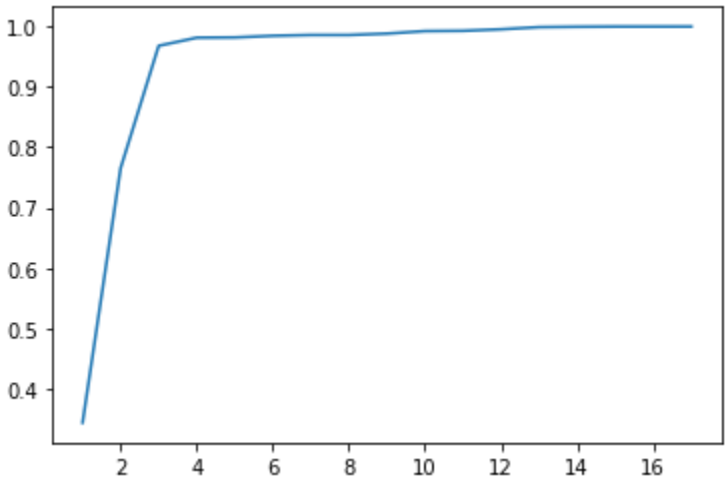
According to the figure above, "siri" and "density" have a correlation of almost 1.0. And upon further investigation of the dataset description (https://cran.r-project.org/web/packages/faraway/faraway.pdf) , we see that "siri" is also percent body fat, but just measured with a different equation, and "density" is essentially, the main variable used in both equations for calculating percentage body fat, hence explaining the large correlation.

3. Methods

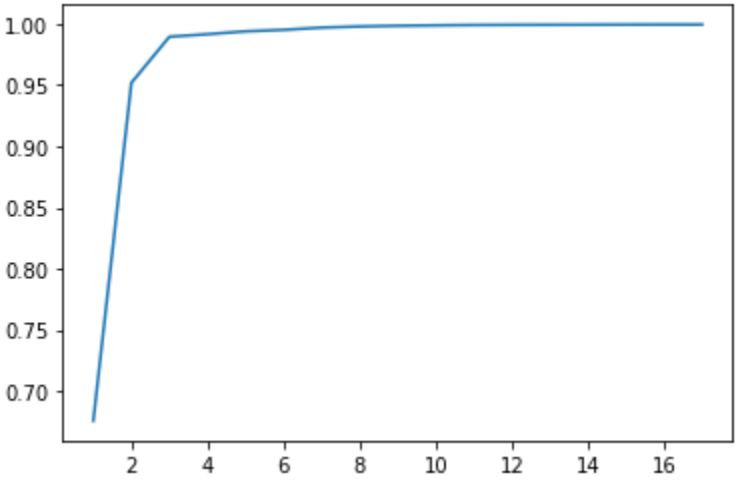
In this assignment, we compare different linear regression methods on the dataset:

- Linear Regression: For this case, we train a linear regression model on the train dataset using all predicting variables. This will serve as our baseline to the other methods.
- Linear Regression with best k=5 predictors: In this case, we take all combinations of 5 predicting variables and train a model on each subset. Afterwards, we compare the R-squared score for each model and take the best subset with the highest score. For our best subset, we are using "siri", "density", "age", "weight", and "height".
- Stepwise Regression using AIC: For this process, we implemented in Python a forward selection process that adds a predicting variable on each iteration until the AIC score no longer improves. The best subset in this case used "siri", "density", "thigh", "knee", "wrist", "biceps", and "forarm".
- Ridge Regression: For ridge, we tried a range from 0.01 to 1 in steps 0.001. Interestly, our best alpha value was 0.01 indicating that the best performance is closest to that of simple linear regression.
- Lasso Regression: Our best alpha value for the LASSO method was 0.137, once again indicating that the best performance is close to simple linear regression.
- Principal component regression: Before fitting data to the linear regression model, we tried transforming the data by its principal components first to compare

performance. However, as the plot shows below, the best R-squared value of the model is achieved by 17 components, which is using all the predicting variables.



- Partial Least Squares: Similar to PCA, our best component was also 17, indicating the best performance is similar to simple linear regression.

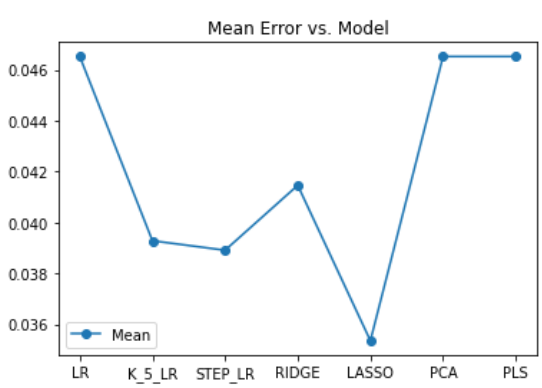


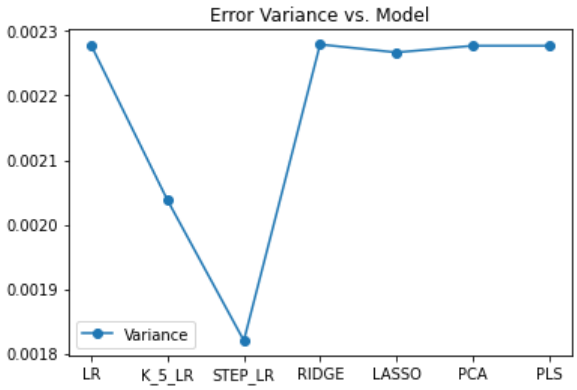
4. Results

We will now compare all linear regression methods by their mean squared test error, as well as the monte carlo cross-validation test error.

Model	MSE on test set	Average MSE on CV	Variance of MSE on CV
Linear Regression	0.012996	0.046526	0.002277
Best k=5 subset	0.007875	0.039286	0.002037
Step Regression	0.013213	0.038914	0.**001820**
Ridge	0.013056	0.041464	0.002279
Lasso	**0.007282**	**0.035359**	0.002267
PC Regression	0.012996	0.046526	0.002277
PLS Regression	0.012996	0.046526	0.002277

From our results, we find that in both comparing the mean squared error values for the specified test dataset, as well in 100 loops of monte-carlo cross-validation that the LASSO method outperformed the other models. However, for our step-wise regression model, the subset chosen had the least variance.





5. Findings

It is interesting to note that Principal Component Regression, Partial least squares, and

Linear Regression with all predictors all have the same mean squared test error. This is due to principal components selected as 17 (all variables). However, with multiple runs, it's possible there is a principal component that provides the lowest MSE for that seed. Although the LASSO method may have seemed superior from our results, this is largely due to our variables used as predictors with "siri" and "density". We should repeat this experiment without using these two predictors because they are actually related to the response variable, and hence, have a large correlation. Additionally, one of the assumptions of a multi linear regression model holds that the predicting variables used are not highly correlated with one another. Finally, all of our train and test splits were chosen with the test dataset at 10%, while traditionally, this value is closer to 20% or 30%, so our results could change drastically.

6. Appendix: Python code

```
#Import statements
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from itertools import combinations
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
import statsmodels.api as sm

#Read in csv file
fat_df = pd.read_csv("fat.csv")

#Split data into train and test
test_idx = [1, 21, 22, 57, 70, 88, 91, 94, 121, 127, 149, 151, 159, 162,
            164, 177, 179, 194, 206, 214, 215, 221, 240, 241, 243]
fat1test = fat_df.loc[test_idx,:]
fat1train = fat_df.drop(test_idx, axis=0)

#Show correlation between all predictors
fat1train.corr()

#Show box plot to check for outliers
sns.boxplot(x = fat1train['brozek'])

#Show distribution of response var
sns.distplot(fat1train['brozek'])
```

```
#Show quick summary of data
fat1train.describe()
# Train linear regression model with all predictors
X_train = fat1train.drop('brozek',axis=1)
y_train = fat1train['brozek']
X_test = fat1test.drop('brozek',axis=1)
y_test = fat1test['brozek']
lm = LinearRegression().fit(X_train, y_train)
train_mse = mean_squared_error(y_train, lm.predict(X_train))
test_mse = mean_squared_error(y_test, lm.predict(X_test))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.028777733601962237  | Test MSE:  0.012996870362070588

#Find best subset of k=5
combs = list(combinations(X_train.columns, 5))
best_rss = 0
best_comb = None

for comb in combs:
    tmp_x = X_train[list(combs[0])]
    tmp_lm = LinearRegression().fit(tmp_x, y_train)
    tmp_score = tmp_lm.score(tmp_x,y_train)
    if tmp_score > best_rss:
        best_rss = tmp_score
        best_comb = comb
k_5_lm = LinearRegression().fit(X_train[list(best_comb)], y_train)
train_mse = mean_squared_error(y_train,
k_5_lm.predict(X_train[list(best_comb)]))
test_mse = mean_squared_error(y_test, k_5_lm.predict(X_test[list(best_comb)]))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.03198079231134047  | Test MSE:  0.00787544871755143

#function inspired and refactored from
#https://www.datasklr.com/ols-least-squares-regression/variable-selection
def forward_regression(X, y,
                       initial_list=[],
                       verbose=True):
    initial_list = []
    included = list(initial_list)
    curr_aic = 1e9
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_aic = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y,
sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
            new_aic[new_column] = model.aic
        best_aic = new_aic.min()
        if best_aic < curr_aic:
            best_feature = new_aic.index[new_aic == best_aic].tolist()[0]
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add ',best_feature,' with aic ',best_aic)
        curr_aic = best_aic
```

```
        if not changed:
            break

    return included
#Train linear regression with selected variables
selected_vars = forward_regression(X_train,y_train)
step_lm = LinearRegression().fit(X_train[selected_vars], y_train)
train_mse = mean_squared_error(y_train,
step_lm.predict(X_train[selected_vars]))
test_mse = mean_squared_error(y_test, step_lm.predict(X_test[selected_vars]))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.029638212480354354  | Test MSE:  0.013213014750610026

#Train ridge regression model
ridge_clf = RidgeCV(alphas=np.arange(0.01,1,0.001),
store_cv_values=True).fit(X_train, y_train)
train_mse = mean_squared_error(y_train, ridge_clf.predict(X_train))
test_mse = mean_squared_error(y_test, ridge_clf.predict(X_test))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.029372567430054525  | Test MSE:  0.013056345004719774

#Train lasso regression model
lasso_clf = LassoCV(cv=5, random_state=7406).fit(X_train, y_train)
train_mse = mean_squared_error(y_train, lasso_clf.predict(X_train))
test_mse = mean_squared_error(y_test, lasso_clf.predict(X_test))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.03323077629269723  | Test MSE:  0.0072823790778779665

#Principal component regression
pca_scores=[]
best_component = None
best_score = 0
for i in range(1,18):
    x_pca = PCA(n_components=i).fit_transform(X_train)
    pca_clf = LinearRegression().fit(x_pca, y_train)
    score = pca_clf.score(x_pca,y_train)
    pca_scores.append(score)
    if score > best_score:
        best_score = score
        best_component = i
print(best_score)
print(best_component)
#0.9995001801291656
#17

plt.plot(range(1,18), pca_scores)
#Train PCA regression with best component
pca_model = PCA(n_components=best_component)
x_pca = pca_model.fit_transform(X_train)
pca_clf = LinearRegression().fit(x_pca, y_train)
train_mse = mean_squared_error(y_train,
pca_clf.predict(pca_model.transform(X_train)))
test_mse = mean_squared_error(y_test,
pca_clf.predict(pca_model.transform(X_test)))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.028777733601962122  | Test MSE:  0.012996870362068908
```

```
#Partial least squares
pls_scores=[]
best_component = None
best_score = 0
for i in range(1,18):
    pls = PLSRegression(n_components=i).fit(X_train, y_train)
    score = pls.score(X_train,y_train)
    pls_scores.append(score)
    if score > best_score:
        best_score = score
        best_component = i
print(best_score)
print(best_component)
#0.9995001801291656
#17

plt.plot(range(1,18), pls_scores)
pls = PLSRegression(n_components=17).fit(X_train, y_train)
train_mse = mean_squared_error(y_train, pls.predict(X_train))
test_mse = mean_squared_error(y_test, pls.predict(X_test))
print("Train MSE: ", train_mse, " | Test MSE: ", test_mse)
#Train MSE:  0.028777733601962046  | Test MSE:  0.01299687036205577

X_full = fat_df.iloc[:,1:]
y_full = fat_df.iloc[:,0]

#Monte carlo CV
B = 100 #total loop times
all_test_errors = []
k_range = [1,3,5,7,9,11,13,15]

for b in range(B):
    test_errors = []
    #Split train and test dataset based on random state
    X_train, X_test, y_train, y_test = train_test_split(X_full, y_full,
test_size=0.1, random_state=b)

    #1. Train LinearRegression on all predictors
    lr_model = LinearRegression().fit(X_train, y_train)
    test_errors.append(mean_squared_error(y_test, lr_model.predict(X_test)))

    #2. Train linear regression on best subset
    k_5_lm = LinearRegression().fit(X_train[list(best_comb)], y_train)
    test_errors.append(mean_squared_error(y_test,
k_5_lm.predict(X_test[list(best_comb)])))

    #3. Train linear regression with stepwise predictors
    step_lm = LinearRegression().fit(X_train[selected_vars], y_train)
    test_errors.append(mean_squared_error(y_test,
step_lm.predict(X_test[selected_vars])))

    #4. Train with ridge regression model
    ridge_clf = RidgeCV(alphas=np.arange(0.01,1,0.001)).fit(X_train, y_train)
    test_errors.append(mean_squared_error(y_test, ridge_clf.predict(X_test)))

    #5. Train with lasso regression model
    lasso_clf = LassoCV(cv=5, random_state=7406).fit(X_train, y_train)
```

```
    test_errors.append(mean_squared_error(y_test, lasso_clf.predict(X_test)))

    #6. Train with principal component regression
    pca_model = PCA(n_components=17)
    x_pca = pca_model.fit_transform(X_train)
    pca_clf = LinearRegression().fit(x_pca, y_train)
    test_errors.append(mean_squared_error(y_test,
pca_clf.predict(pca_model.transform(X_test))))

    #7. Train with PLS Regression
    pls = PLSRegression(n_components=17).fit(X_train, y_train)
    test_errors.append(mean_squared_error(y_test, pls.predict(X_test)))

    all_test_errors.append(test_errors)

all_test_errors = np.array(all_test_errors)
index_names = ['LR','K_5_LR','STEP_LR','RIDGE','LASSO','PCA','PLS']
all_test_err_df = pd.DataFrame({'Mean':np.mean(all_test_errors,axis=0),
                                'Variance':np.var(all_test_errors, axis=0)},
                               index=index_names)
all_test_err_df
all_test_err_df.plot.line(y='Mean',title='Mean Error vs. Model', marker='o')
all_test_err_df.plot.line(y='Variance',title='Error Variance vs. Model',
marker='o')
```