This lab on Subset Selection is a Python adaptation of p. 244-247 of "Introduction to Statistical Learning with Applications in R" by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Adapted by R. Jordan Crouser at Smith College for SDS293: Machine Learning (Spring 2016).

Want to follow along on your own machine? Download the .py (lab8.py) or Jupyter Notebook (Lab 8 - Subset Selection in Python.ipynb) version.

```
%matplotlib inline
import pandas as pd
import numpy as np
import itertools
import time
import statsmodels.api as sm
import matplotlib.pyplot as plt
```

6.5.1 Best Subset Selection

Here we apply the best subset selection approach to the Hitters data. We wish to predict a baseball player's Salary on the basis of various statistics associated with performance in the previous year. Let's take a quick look:

```
hitters_df = pd.read_csv('Hitters.csv')
hitters_df.head()
```

First of all, we note that the `Salary` variable is missing for some of the players. The `isnull()` function can be used to identify the missing observations. It returns a vector of the same length as the input vector, with a `TRUE` value for any elements that are missing, and a `FALSE` value for non-missing elements. The `sum()` function can then be used to count all of the missing elements:

```
print("Number of null values:", hitters_df["Salary"].isnull().sum())
```

We see that `Salary` is missing for 59 players. The `dropna()` function removes all of the rows that have missing values in any variable:

```
# Print the dimensions of the original Hitters data (322 rows x 20 columns)
print("Dimensions of original data:", hitters_df.shape)

# Drop any rows the contain missing values, along with the player names
hitters_df_clean = hitters_df.dropna().drop('Player', axis=1)

# Print the dimensions of the modified Hitters data (263 rows x 20 columns)
print("Dimensions of modified data:", hitters_df_clean.shape)

# One last check: should return 0
print("Number of null values:", hitters_df_clean["Salary"].isnull().sum())
```

Some of our predictors are categorical, so we'll want to clean those up as well. We'll ask `pandas` to generate dummy variables for them, separate out the response variable, and stick everything back together again:

```
dummies = pd.get_dummies(hitters_df_clean[['League', 'Division', 'NewLeague']])

y = hitters_df_clean.Salary

# Drop the column with the independent variable (Salary), and columns for which we
created dummy variables
X_ = hitters_df_clean.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).
astype('float64')

# Define the feature set X.
X = pd.concat([X_, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
```

We can perform best subset selection by identifying the best model that contains a given number of predictors, where **best** is quantified using RSS. We'll define a helper function to outputs the best set of variables for each model size:

```
def processSubset(feature_set):
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y,X[list(feature_set)])
    regr = model.fit()
    RSS = ((regr.predict(X[list(feature_set)]) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}
```

```
def getBest(k):

    tic = time.time()

    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print("Processed", models.shape[0], "models on", k, "predictors in", (toc-tic
), "seconds.")

    # Return the best model, along with some other useful information about the mo
del
    return best_model
```

This returns a `DataFrame` containing the best model that we generated, along with some extra information about the model. Now we want to call that function for each number of predictors k:

```
# Could take quite awhile to complete...

models_best = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
for i in range(1,8):
    models_best.loc[i] = getBest(i)

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

Now we have one big `DataFrame` that contains the best models we've generated along with their RSS:

```
models_best
```

If we want to access the details of each model, no problem! We can get a full rundown of a single model using the `summary()` function:

```
print(models_best.loc[2, "model"].summary())
```

This output indicates that the best two-variable model contains only `Hits` and `CRBI`. To save time, we only generated results up to the best 7-variable model. You can use the functions we defined above to explore as many variables as are desired.

```
# Show the best 19-variable model (there's actually only one)
print(getBest(19)["model"].summary())
```

Rather than letting the results of our call to the `summary()` function print to the screen, we can access just the parts we need using the model's attributes. For example, if we want the R^2 value:

```
models_best.loc[2, "model"].rsquared
```

Excellent! In addition to the verbose output we get when we print the summary to the screen, fitting the `OLM` also produced many other useful statistics such as adjusted R^2, AIC, and BIC. We can examine these to try to select the best overall model. Let's start by looking at R^2 across all our models:

```
# Gets the second element from each row ('model') and pulls out its rsquared attri
bute
models_best.apply(lambda row: row[1].rsquared, axis=1)
```

As expected, the R^2 statistic increases monotonically as more variables are included.

Plotting RSS, adjusted R^2, AIC, and BIC for all of the models at once will help us decide which model to select. Note the `type="l"` option tells R to connect the plotted points with lines:

```
plt.figure(figsize=(20,10))
plt.rcParams.update({'font.size': 18, 'lines.markersize': 10})

# Set up a 2x2 grid so we can look at 4 plots at once
plt.subplot(2, 2, 1)

# We will now plot a red dot to indicate the model with the largest adjusted R^2 s
tatistic.
# The argmax() function can be used to identify the location of the maximum point
 of a vector
plt.plot(models_best["RSS"])
plt.xlabel('# Predictors')
plt.ylabel('RSS')

# We will now plot a red dot to indicate the model with the largest adjusted R^2 s
tatistic.
# The argmax() function can be used to identify the location of the maximum point
 of a vector

rsquared_adj = models_best.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.subplot(2, 2, 2)
plt.plot(rsquared_adj)
plt.plot(rsquared_adj.argmax(), rsquared_adj.max(), "or")
plt.xlabel('# Predictors')
plt.ylabel('adjusted rsquared')

# We'll do the same for AIC and BIC, this time looking for the models with the SMA
LLEST statistic
aic = models_best.apply(lambda row: row[1].aic, axis=1)

plt.subplot(2, 2, 3)
plt.plot(aic)
plt.plot(aic.argmin(), aic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('AIC')

bic = models_best.apply(lambda row: row[1].bic, axis=1)

plt.subplot(2, 2, 4)
plt.plot(bic)
plt.plot(bic.argmin(), bic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('BIC')
```

Recall that in the second step of our selection process, we narrowed the field down to just one model on any $k <= p$ predictors. We see that according to BIC, the best performer is the model with 6 variables. According to AIC and adjusted R^2 something a bit more complex might be better. Again, no one measure is going to give us an entirely accurate picture... but they all agree that a model with 5 or fewer predictors is insufficient.

6.5.2 Forward and Backward Stepwise Selection

We can also use a similar approach to perform forward stepwise or backward stepwise selection, using a slight modification of the functions we defined above:

```
def forward(predictors):

    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X.columns if p not in predictors]

    tic = time.time()

    results = []

    for p in remaining_predictors:
        results.append(processSubset(predictors+[p]))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print("Processed ", models.shape[0], "models on", len(predictors)+1, "predicto
rs in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the mo
del
    return best_model
```

Now let's see how much faster it runs!

```
models_fwd = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
predictors = []

for i in range(1,len(X.columns)+1):
    models_fwd.loc[i] = forward(predictors)
    predictors = models_fwd.loc[i]["model"].model.exog_names

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

Phew! That's a lot better. Let's take a look:

```
print(models_fwd.loc[1, "model"].summary())
print(models_fwd.loc[2, "model"].summary())
```

We see that using forward stepwise selection, the best one-variable model contains only `Hits`, and the best two-variable model additionally includes `CRBI`. Let's see how the models stack up against best subset selection:

```
print(models_best.loc[6, "model"].summary())
print(models_fwd.loc[6, "model"].summary())
```

For this data, the best one-variable through six-variable models are each identical for best subset and forward selection.

Backward Selection

Not much has to change to implement backward selection... just looping through the predictors in reverse!

```
def backward(predictors):

    tic = time.time()

    results = []

    for combo in itertools.combinations(predictors, len(predictors)-1):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print("Processed ", models.shape[0], "models on", len(predictors)-1, "predicto
rs in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the mo
del
    return best_model
```

```
models_bwd = pd.DataFrame(columns=["RSS", "model"], index = range(1,len(X.columns
)))

tic = time.time()
predictors = X.columns

while(len(predictors) > 1):
    models_bwd.loc[len(predictors)-1] = backward(predictors)
    predictors = models_bwd.loc[len(predictors)-1]["model"].model.exog_names

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")
```

For this data, the best one-variable through six-variable models are each identical for best subset and forward selection. However, the best seven-variable models identified by forward stepwise selection, backward stepwise selection, and best subset selection are different:

```
print("------------")
print("Best Subset:")
print("------------")
print(models_best.loc[7, "model"].params)
```

```
print("-----------------")
print("Foward Selection:")
print("-----------------")
print(models_fwd.loc[7, "model"].params)
```

```
print("-------------------")
print("Backward Selection:")
print("-------------------")
print(models_bwd.loc[7, "model"].params)
```

Getting credit

To get credit for this lab, please post an example where you would choose to use each of the following:

- Best subset
- Forward selection
- Backward selection

to Moodle (https://moodle.smith.edu/mod/quiz/view.php?id=257886).