Model Fit in Linear Regression - Lab

Introduction

In this lab, you'll learn how to evaluate your model results, and you'll learn methods to select the appropriate features using stepwise selection.

Objectives

You will be able to:

- Analyze the results of regression and R-squared and adjusted-R-squared
- Understand and apply forward and backward predictor selection

The Boston Housing Data once more

We pre-processed the Boston Housing Data the same way we did before:

- We dropped "ZN" and "NOX" completely
- We categorized "RAD" in 3 bins and "TAX" in 4 bins
- We used min-max-scaling on "B", "CRIM" and "DIS" (and logtransformed all of them first, except "B")
- We used standardization on "AGE", "INDUS", "LSTAT" and "PTRATIO" (and logtransformed all of them first, except for "AGE")

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
boston = load_boston()

boston_features = pd.DataFrame(boston.data, columns = boston.feature_names)
boston_features = boston_features.drop(["NOX","ZN"],axis=1)

# first, create bins for based on the values observed. 3 values will result in 2 bins
bins = [0,6,  24]
bins_rad = pd.cut(boston_features['RAD'], bins)
bins_rad = bins_rad.cat.as_unordered()

# first, create bins for based on the values observed. 4 values will result in 3 bins
bins = [0, 270, 360, 712]
bins_tax = pd.cut(boston_features['TAX'], bins)
bins_tax = bins_tax.cat.as_unordered()

tax_dummy = pd.get_dummies(bins_tax, prefix="TAX")
rad_dummy = pd.get_dummies(bins_rad, prefix="RAD")
boston_features = boston_features.drop(["RAD","TAX"], axis=1)
boston_features = pd.concat([boston_features, rad_dummy, tax_dummy], axis=1)

age = boston_features["AGE"]
b = boston_features["B"]
logcrim = np.log(boston_features["CRIM"])
logdis = np.log(boston_features["DIS"])
logindus = np.log(boston_features["INDUS"])
loglstat = np.log(boston_features["LSTAT"])
logptratio = np.log(boston_features["PTRATIO"])

# minmax scaling
boston_features["B"] = (b-min(b))/(max(b)-min(b))
boston_features["CRIM"] = (logcrim-min(logcrim))/(max(logcrim)-min(logcrim))
boston_features["DIS"] = (logdis-min(logdis))/(max(logdis)-min(logdis))

#standardization
boston_features["AGE"] = (age-np.mean(age))/np.sqrt(np.var(age))
boston_features["INDUS"] = (logindus-np.mean(logindus))/np.sqrt(np.var(logindus))
boston_features["LSTAT"] = (loglstat-np.mean(loglstat))/np.sqrt(np.var(loglstat))
boston_features["PTRATIO"] = (logptratio-np.mean(logptratio))/(np.sqrt(np.var(logptratio)))
```

Perform stepwise selection

The code for stepwise selection is copied below.

In [2]:

```
import statsmodels.api as sm

def stepwise_selection(X, y,
                       initial_list=[],
                       threshold_in=0.01,
                       threshold_out = 0.05,
                       verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
```

```
        X - pandas.DataFrame with candidate features
        y - list-like with the target
        initial_list - list of features to start with (column names of X)
        threshold_in - include a feature if its p-value < threshold_in
        threshold_out - exclude a feature if its p-value > threshold_out
        verbose - whether to print the sequence of inclusions and exclusions
    Returns: list of selected features
    Always set threshold_in < threshold_out to avoid infinite looping.
    See https://en.wikipedia.org/wiki/Stepwise_regression for the details
    """
    included = list(initial_list)
    while True:
        changed=False
        # forward step
        excluded = list(set(X.columns)-set(included))
        new_pval = pd.Series(index=excluded)
        for new_column in excluded:
            model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included+[new_column]]))).fit()
            new_pval[new_column] = model.pvalues[new_column]
        best_pval = new_pval.min()
        if best_pval < threshold_in:
            best_feature = new_pval.idxmin()
            included.append(best_feature)
            changed=True
            if verbose:
                print('Add  {:30} with p-value {:.6}'.format(best_feature, best_pval))

        # backward step
        model = sm.OLS(y, sm.add_constant(pd.DataFrame(X[included]))).fit()
        # use all coefs except intercept
        pvalues = model.pvalues.iloc[1:]
        worst_pval = pvalues.max() # null if pvalues is empty
        if worst_pval > threshold_out:
            changed=True
            worst_feature = pvalues.argmax()
            included.remove(worst_feature)
            if verbose:
                print('Drop {:30} with p-value {:.6}'.format(worst_feature, worst_pval))
        if not changed:
            break
    return included
```

```
/Users/lore.dirick/anaconda3/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.dat
etools module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

In [3]:

```
X = boston_features
y = pd.DataFrame(boston.target, columns= ["price"])
```

In [4]:

```
result = stepwise_selection(X, y, verbose = True)
print('resulting features:')
print(result)
```

```
Add  LSTAT                          with p-value 9.27989e-122
Add  RM                             with p-value 1.98621e-16
Add  PTRATIO                        with p-value 2.5977e-12
Add  DIS                            with p-value 2.85496e-09
Add  B                              with p-value 2.77572e-06
Add  TAX_(0, 270]                   with p-value 0.000855799
Add  CHAS                           with p-value 0.00151282
Add  INDUS                          with p-value 0.00588575
resulting features:
['LSTAT', 'RM', 'PTRATIO', 'DIS', 'B', 'TAX_(0, 270]', 'CHAS', 'INDUS']
```

Build the final model again in Statsmodels

In [5]:

```
import statsmodels.api as sm
X_fin = X[["LSTAT", "RM", "PTRATIO", "DIS", "B", "TAX_(0, 270]", "CHAS", "INDUS"]]
X_int = sm.add_constant(X_fin)
model = sm.OLS(y,X_int).fit()
model.summary()
```

Out[5]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.776
Model:	OLS	**Adj. R-squared:**	0.773
Method:	Least Squares	**F-statistic:**	215.7
Date:	Mon, 15 Oct 2018	**Prob (F-statistic):**	2.69e-156
Time:	21:15:33	**Log-Likelihood:**	-1461.3
No. Observations:	506	**AIC:**	2941.
Df Residuals:	497	**BIC:**	2979.
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.8980	2.813	1.742	0.082	-0.628	10.424
LSTAT	-5.5932	0.319	-17.538	0.000	-6.220	-4.967
RM	2.8294	0.386	7.333	0.000	2.071	3.587
PTRATIO	-1.3265	0.226	-5.878	0.000	-1.770	-0.883

DIS	-9.1984	1.333	-6.898	0.000	-11.818	-6.579
B	3.9052	0.931	4.195	0.000	2.076	5.734
TAX_(0, 270]	1.4418	0.552	2.614	0.009	0.358	2.526
CHAS	2.7988	0.791	3.539	0.000	1.245	4.353
INDUS	-0.9574	0.346	-2.766	0.006	-1.637	-0.277

Omnibus:	114.307	**Durbin-Watson:**	1.088
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	482.579
Skew:	0.945	**Prob(JB):**	1.62e-105
Kurtosis:	7.395	**Cond. No.**	96.8

Where our stepwise procedure mentions that "CHAS" was added with a p-value of 0.00151282, but our statsmodels output returns a p-value of 0.000. What is the intuition behind this?

Use Feature ranking with recursive feature elimination

Use feature ranking to select the 5 most important features

In [6]:
```python
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()
selector = RFE(linreg, n_features_to_select = 5)
selector = selector.fit(X, y)
```

```
/Users/lore.dirick/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:578: DataConversionWarning: A column-ve
ctor y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [7]:
```python
selector.support_
```

Out[7]:
```
array([False, False,  True,  True, False,  True, False,  True,  True,
       False, False, False, False, False])
```

Fit the linear regression model again using the 5 columns selected

In [8]:
```python
selected_columns = X.columns[selector.support_ ]
linreg.fit(X[selected_columns],y)
```

Out[8]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Now, predict $\hat{y}$ using your model. you can use `.predict()` in scikit-learn

In [9]:
```python
yhat = linreg.predict(X[selected_columns])
```

Now, using the formulas of R-squared and adjusted-R-squared below, and your Python/numpy knowledge, compute them and contrast them with the R-squared and adjusted-R-squared in your statsmodels output using stepwise selection. Which of the two models would you prefer?

$$SS_{residual} = \sum (y - \hat{y})^2$$

$$SS_{total} = \sum (y - \bar{y})^2$$

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

In [10]:
```python
SS_Residual = np.sum((y-yhat)**2)
SS_Total = np.sum((y-np.mean(y))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y)-1)/(len(y)-X.shape[1]-1)
```

In [11]:
```python
r_squared
```

Out[11]:
```
price    0.742981
dtype: float64
```

In [12]:
```python
adjusted_r_squared
```

Out[12]:
```
price    0.735652
dtype: float64
```

Level up - Optional

- Perform variable selection using forward selection, using this resource: https://planspace.org/20150423-forward_selection_with_statsmodels/. Note that this time features are added based on the adjusted-R-squared!
- Tweak the code in the `stepwise_selection()` -function written above to just perform forward selection based on the p-value.

Summary

Great! You now performed your own feature selection methods!