stat4500notes

Yi Wang

2023-09-19

Table of contents

# Preface

This is a Lecture note book written for the course STAT 4500: Machine Learning offered at Auburn University at Montgomery.

This is a book wrtieen by Quarto book.

To learn more about Quarto books visit <https://quarto.org/docs/books>.

# 1. Setting up Python Computing Environment

## 1.1 on Your own computer

1. you can either git clone or download a zipped file containing the codes from the site: <https://github.com/intro-stat-learning/ISLP_labs/tree/stable>. If downloaded a zipped file of the codes, unzipped the file to a folder, for example, named islp. If git clone (preferred, you need to have Git installed on your computer, check this link for how to install Git <https://ywanglab.github.io/stat1010/git.html>), do git clone https://github.com/intro-stat-learning/ISLP\_labs.git
2. Download and install the following software:
   * **Anaconda**: Download anaconda and install using default installation options
   * **Visual Studio Code** (VSC): Download VSC and install
   * start VSC and install VSC extensions in VSC: Python, Jupyter, intellicode
   * (optional) **Quarto** for authoring: Download Quarto and install
3. Create a virtual environment named islp for Python. Start an anaconda terminal.

* conda create -n islp python==3.10  
   conda activate islp  
   conda install pip ipykernel  
   pip install -r https://raw.githubusercontent.com/intro-stat-learning/ISLP\_labs/v2.1.2/requirements.txt

1. You are ready to run the codes using VSC or jupyter lab.
   * Activate the venv: conda activate islp
   * Start a Anaconda terminal, navigate to the folder using the command cd path/to/islp, where path/to/islp means the file path to the folder islp, such as \Users\ywang2\islp. Start VSC by typing code . in the anaconda terminal.
   * open/create a .ipynb or .py file.
   * Select the kernel islp
   * Run a code cell by pressing Shift+Enter or click the triangular play button.
   * Continue to run other cells.
   * After finishing using VSC, close the VSC, and deactivate the virtual environment in a conda terminal: conda deactivate

## 1.2 Use Google Colab

All you need is a Google account. Sign in your Google account in a browser, and navigate to Google Colab. Google Colab supports both Python and R. Python is the default engine. Change the engine to R in Connect->change runtime type. Then you are all set. Your file will be saved to your Google Drive or you can choose to send it to your GitHub account (recommended).

### 1.2.1 How to run a project file from your Google Drive?

Many times, when you run a python file in Colab, it needs to access other files, such as data files in a subdirectory. In this case, it would be convenient to have the same file structure in the Google Colab user home directory. To do this, you can use Google Drive to store your project folder, and then mount the Google Drive in Colab.

Let’s assume the project folder name, islp/.Here are the steps:

1. git clone the project folder (example: git clone https://github.com/intro-stat-learning/ISLP\_labs.git) to your local folder. This step is only needed when you want to clone some remote repo from GitHub.
2. **Upload** the folder (ex: islp) to Google Drive.
3. **Open the file using Colab**. In Google Drive, double click on the ipynb file, example, ch06.ipynb (or click on the three dots on the right end, and choose open with, then Google Colaborotary), the file will be opened by Google Colab.
4. **Mount the Google Drive**. In Google Colab, with the specific file (example, ch06.ipynb) being opened, move your cursor to the first code cell, and then click on the folder icon (this should be the fourth icon) on the upper left border in the Colab browser. This will open the file explorer pane. Typically you would see a folder named sample\_data shown. On the top of the pane, click on the Google Drive icon to mount the Google Drive. Google Colab will insert the following code below the cursor in your opened ipynb file:

* from google.colab import drive  
  drive.mount('/content/drive')
* Run this code cell by pressing SHIFT+ENTER, and follow the prompts to complete the authentication. Wait for ~10 seconds, your Google Drive will be mounted in Colab, and it will be displayed as a folder named drive in the file explorer pane. You might need to click on the Refresh folder icon to see the folder drive.

1. Open a new code cell below the above code cell, and type the code

* %cd /content/drive/MyDrive/islp/
* This is to change the directory to the project directory on the Google Drive. Run this code cell, and you are ready to run the file ch06.ipynb from the folder islp on your personal Google Drive, just like it’s on your local computer.

# 2. Chapter 2: Statistical Learning

## 2.1 What is statistical learning?

For the input variable and response variable , assume that

where is a random variable representing **irreducible error**. We assume is *independent* of and . may include *unmeasured variables* or *unmeasurable variation*.

Statistical learning is to estimate using various methods. Denote the estimate by .

* regression problem: when is a continuous (quantitative) variable . In this case is the population *regression* function, that is, regression finds a conditional expectation of .
* classification problem: when only takes small number of discrete values, i.e., qualitative (categorical).

**Logistic regression** is a classification problem, but since it estimates class probability, it may be considered as a regression problem.

* supervised learning: training data : linear regression, logistic regression
* unsupervised learning: when only are available. clustering analysis, PCA
* semi-supervised learning: some data with labels (), some do not.
* reinforcement learning: learn a state-action policy function for an agent to interacting with an environment to maximize a reward function.

## 2.2 Why estimate ?

We can use estimated to

* make predictions for a new ,
* The prediction error may be quantified as
* The first term of the error is *reducible* by trying to improve , where we assume , and are fixed.
* make inference, such as
  + Which predictors are associated with the response?
  + what is the relationship between the response and each predictor?
  + is the assumed relationship adequate? (linear or more complicated?)

## 2.3 How to estimate

We use obtained observations called **training data** to train an algorithm to obtain the estimate .

* Parametric methods: first assume there is a function form (shape) with some parameters. For example, a linear regression model with two parameters. Then use the *training data* to **train** or **fit** the model to determine the values of the parameters.
* **Advantages**: simplify the problem of fit an arbitrary function to estimate a set of parameters.
* **Disadvantages**: may not be flexible unless with large number of parameters and/or complex function shapes.
* Example: linear regression,
* Non-parametric methods: Do not explicitly assume a function form of . They seek to estimate directly using data points, can be quite flexible and accurate.
* \*\*Disadvantage: need large number of data points
* Example: KNN (but breakdown for higher dimention. Typically only for ), spline fit.

## 2.4 How to assess model accuracy

For regression problems, the most commonly used measure is the *mean squared error* (MSE), given by

For classification problems, typically the following **error rate** (classifications error) is calculated:

The accuracy on a training set can be arbitrarily increased by increasing the model flexibility. However, we are in general interested in the error on the test set rather on the training set, the model accuracy should be assessed on a test set.

Flexible models tend to overfit the data, which essentially means they follow the error or *noise* too closely in the training set, therefore cannot be generalized to *unseen cases* (test set).

## 2.5 Model Selection:

**No free lunch theorem**

There is no single best method for all data sets, which means some method works better than other methods for a particular dataset. Therefore, one needs to perform model selections. Here are some principles.

### 2.5.1 Trade-off between Model flexibility and Model Interpretability

More flexible models have higher *degree of freedom* and are less interpretable because it’s difficult to interpret the relationship between a predictor and the response.

LASSO is less flexible than linear regression. GAM (generalized additive model) allows some non-linearity. Full non-linear models have higher flexibility, such as *bagging, boosting, SVM*, etc.

When *inference* is the goal, then there are advantages to using simple and less flexible models for interpretability.

When *prediction* is the main goal, more flexible model may be a choice. But sometimes, we obtain more accurate prediction using a simpler model because the underlying dataset has a simpler structure. Therefore, it is not necessarily true that a more flexible model has a higher prediction accuracy.

**Occam’s Razor**: Among competing hypotheses that perform equally well, the one with the fewest assumptions should be selected.

### 2.5.2 Model Selection: the Bias-Variance Trade-off

As the model flexibility increases, the training MSE (or error rate for classificiton) will decrease, but the test MSE (error rate) in general will not and will show a characteristic **U-shape**. This is because when evaluated at a test point , the expected test MSE can be decomposed into

where the expectation is over different on a different training set or on a different training step if the training process is stochastic, and

To obtain the least test MSE, one must trade off between variance and bias. Less flexible model tendes to have higher bias, and more flexible models tend to have higher variance. An optimal flexibility for the least test MSE varies with different data sets. Non-linear data tends to require higher optimal flexibility.

## 2.6 Bayes Classifier

It can be shown that Bayes Classifier minimizes the classification test error

A Bayes Classifier assigns a test observation with predictor to the class for which

is largest. It’s error rate is given by

where the expectation is over . The Bayes error is analogous to the irreducible error .

Bayes Classifier is not attainable as we do not know . We only can estimate . One way to do this is by KNN. KNN estimate the conditional probability simply with a majority vote. The flexibility of KNN increases as increases with being the most flexible KNN. The training error is 0 for . A suitable should be chosen for an appropriate trade off between bias and variance. The KNN classifier will classify the test point based on the probability calculated from the nearest points. KNN regression on the other hand will assign the test point the average value of the nearest neighbors.

## 2.7 Homework (\* indicates optional):

* Conceptual: 1,2,3,4\*,5,6,7
* Applied: 8, 9\*, 10\*

## 2.8 Code Gist

### 2.8.1 OS

import os  
os.chdir(path) # change dir

### 2.8.2 Python:

Concatenation using +

"hello" + " " + "world" # 'hello world'  
[3,4,5] + [4,9,7] # [3,4,5, 4,9,7]

String formatting using string.format()

print('Total is: {0}'.format(total))

zip to loop over a sequence of tuples

for value, weight in zip([2,3,19],  
 [0.2,0.3,0.5]):  
 total += weight \* value

### 2.8.3 Numpy

#### 2.8.3.1 Numpy functions:

np.sum(x), np.sqrt(x) (entry wise). x\*\*2 (entry wise power), np.corrcoef(x,y) (find the correlation coefficient of array x and array y)

np.mean(axis=None): axis could be None (all entries), 0(along row), 1(along column)

np.var(x, ddof=0), np.std(x, ddof=0), # Note both np.var and np.std accepts an argument ddof, the divisor is N-ddof.

np.linspace(-np.pi, np.pi, 50) # start, end, number of points 50

np.multiply.outer(row,col) # calculate the product over the mesh with vectors row and col.

np.zeros(shape or int, dtype) #eg: np.zeros(5,bool)

np.ones(Boston.shape[0])

np.all(x), np.any(x): check if all or any entry of x is true.

np.unique(x): find unique values in x. np.isnan(x): return a boolean array of len(x). np.isnan(x).mean(): find the percentage of np.nan values in x.

#### 2.8.3.2 Array Slicing and indexing

np.arange(start, stop, step) # numpy version of range

x[slice(3:6)] # equivalent to x[3:6]

Indexing an array using [row, col] format. If col is missing, then index the entire rows. len(row) must be equal to len(col). Otherwise use iterative indexing or use np.ix\_(x\_idx, y\_idx) function, or use Boolean indexing, see below.

A[1,2]: index entry at row 1 and col 2 (recall Python index start from 0)  
A[[1,3]] # row 1 and 3. Note the outer [] is considered as the operator, so only row indices are provided.   
A[:,[0,2]] # cols 0 and 2  
A[[1,3], [0,2,3]] # entry A[1,0] and A[3,2]  
A[1:4:2, 0:3:2] # entries in rows 1 and 3, cols 0 and 2  
A[[1,3], [0,2,3]] # syntax error  
# instead one can use the following two methods   
A[[1,3]][:,[0,2]] # iterative subsetting  
A[np.ix\_([1,3],[0,2,3])] # use .ix\_ function to create an index mesh  
A[keep\_rows, keep\_cols] # keep\_rows, keep\_cols are boolean arrays of the same length of rows or cols, respectively  
A[np.ix\_([1,3],keep\_cols)] # np.ix\_()can be applied to mixture of integer array and boolean array

#### 2.8.3.3 Random numbers and generators

np.random.normal(loc=0.0, scale=1.0,size=None) # size can be an integer or a tuple.  
#   
rng = np.random.default\_rng(1303) # set random generator seed  
rng.normal(loc=0, scale=5, size=2) #   
rng.standard\_normal(10) # standard normal distribution of size 10  
rng.choice([0, np.nan], p=[0.8,0.2], size=A.shape)

#### 2.8.3.4 Numpy array atributes

.dtype, .ndim, .shape

#### 2.8.3.5 Numpy array methods

x.sum(axis=None) (equivalent to np.sum(x)), x.T (transpose),

x.reshape((2,3)) # x.reshape() is a reference to x.

x.min(), x.max()

### 2.8.4 Graphics

#### 2.8.4.1 2-D figure

# Using the subplots + ax methods  
fig, ax = subplots(nrows=2, ncols=3, figsize=(8, 8))   
# explicitly name each axis in the grid   
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(10,10))  
  
ax[0,1].plot(x, y,marker='o', 'r--', linewidth=3); #line plot. `;` suppresses the text output. pick ax[0,1] when there are multiple axes  
ax.plot([min(fitted),max(fitted)],[0,0],color = 'k',linestyle = ':', alpha = .3)  
ax.scatter(x, y, marker='o'); #scatter plot  
ax.scatter(fitted, residuals, edgecolors = 'k', facecolors = 'none')  
ax.set\_xlabel("this is the x-axis")  
ax.set\_ylabel("this is the y-axis")  
ax.set\_title("Plot of X vs Y");  
axes[0,1].set\_xlim([-1,1]) # set x\_lim. similarly `set\_ylim()`  
  
fig = ax.figure # get the figure object from an axes object  
fig.set\_size\_inches(12,3) # access the fig object to change fig size (width, height)  
fig # re-render the figure  
fig.savefig("Figure.pdf", dpi=200); #save a figure into pdf. Other formats: .jpg, .png, etc

#### 2.8.4.2 Contour and image

fig, ax = subplots(figsize=(8, 8))  
x = np.linspace(-np.pi, np.pi, 50)  
y = x  
f = np.multiply.outer(np.cos(y), 1 / (1 + x\*\*2))  
ax.contour(x, y, f, levels=None); # numbre of levels. if None, automatically choose  
ax.imshow(f); # heatmap colorcoded by f

### 2.8.5 Pandas

#### 2.8.5.1 loading data

pd.read\_csv('Auto.csv') # read csv  
pd.read\_csv('Auto.data',   
 na\_values =['?'], #specifying the na\_values in the datafile.   
 delim\_whitespace=True) # read whitespaced text file  
pd.read\_csv('College.csv', index\_col=0) # use column `0` as the row labels

#### 2.8.5.2 Pandas Dataframe attributes and methods

Auto.shape  
Auto.columns # gets the list of column names  
Auto.index #return the index (labels) objects  
Auto['horsepower'].to\_numpy() # convert to numpy array  
Auto['horsepower'].sum()  
  
Auto.dropna() # drop the rows containing na values.   
df.drop('B', axis=1, inplace=True) # drop a column 'B' inplace.   
#equivalent to df.drop(columns=['B'], inplace=True)  
df.drop(index=['Ohio','Colorado']) #eqivalent to: df.drop(['Ohio','Colorado'], axis=0)  
auto\_df.drop(auto\_df.index[10:86]) # drop rows with index[10:86] not including 86  
  
Auto.set\_index('name')# rename the index using the column 'name'.  
  
pd.Series(Auto.cylinders, dtype='category') # convert the column `cylinders` to 'category` dtype  
# the convertison can be done using `astype()` method  
Auto.cylinders.astype('category')  
Auto.describe() # statistics summary of all columns  
Auto['mpg'].describe() # for selected columns  
  
college.rename({'Unnamed: 0': 'College'}, axis=1): # change column name,   
# alternavie way  
college\_df.rename(columns={college\_df.columns[0] : "College"}, inplace=True) #  
  
college['Elite'] = pd.cut(college['Top10perc'], # binning a column  
 [0,0.5,1], #bin edges  
 labels=['No', 'Yes'], # bin labels (names)  
 right=True,# True: right-inclusive (default) for each bin ( ]; False:rigth-exclusive   
 )   
college['Elite'].value\_counts() # frequency counts  
auto.columns.tolist() # equivalent to auto.columns.format() (rarely used)

#### 2.8.5.3 Selecting rows and columns

Select Rows:

Auto[:3] # the first 3 rows.   
Auto[Auto['year'] > 80] # select rows with boolean array  
Auto\_re.loc[['amc rebel sst', 'ford torino']] #label\_based row selection  
Auto\_re.iloc[[3,4]] #integer-based row seleciton: rows 3 and 4 (index starting from 0)

Select Columns

Auto['horsepower'] # select the column 'horsepower', resulting a pd.Series.  
Auto[['horsepower']] #obtain a dataframe of the column 'horsepower'.   
Auto\_re.iloc[:,[0,2,3]] # intger-based selection  
auto\_df.select\_dtypes(include=['int16','int32']) # select columns by dtype

Select a subset

Auto\_re.iloc[[3,4],[0,2,3]] # integer-based   
Auto\_re.loc['ford galaxie 500', ['mpg', 'origin']] #label-based   
Auto\_re.loc[Auto\_re['year'] > 80, ['weight', 'origin']] # mix bolean indexing with labels  
  
Auto\_re.loc[lambda df: (df['year'] > 80) & (df['mpg'] > 30),  
 ['weight', 'origin']  
 ] # using labmda function with loc[]

#### 2.8.5.4 Pandas graphics

Without using subplots to get axes and figure objects

ax = Auto.plot.scatter('horsepower', 'mpg') #scatter plot of 'horsepower' vs 'mpg' from the dataframe Auto  
ax.set\_title('Horsepower vs. MPG');  
fig = ax.figure  
fig.savefig('horsepower\_mpg.png');  
  
plt.gcf().subplots\_adjust(bottom=0.05, left=0.1, top=0.95, right=0.95) #in percentage of the figure size.   
ax1.fig.suptitle('College Scatter Matrix', fontsize=35)

Using subplots

fig, axes = subplots( ncols=3, figsize=(15, 5))  
Auto.plot.scatter('horsepower', 'mpg', ax=axes[1]);  
Auto.hist('mpg', ax=ax);  
Auto.hist('mpg', color='red', bins=12, ax=ax); # more customized

Boxplot using subplots

Auto.cylinders = pd.Series(Auto.cylinders, dtype='category') # needs to convert the `cylinders` column to categorical dtype  
fig, ax = subplots(figsize=(8, 8))  
Auto.boxplot('mpg', by='cylinders', ax=ax);

Scatter matrix

pd.plotting.scatter\_matrix(Auto); # all columns  
pd.plotting.scatter\_matrix(Auto[['mpg',  
 'displacement',  
 'weight']]); # selected columns  
   
   
#Alternatively with sns.pairplot

Sns Graphic

# Scatter matrix  
ax1 = sns.pairplot(college\_df[college\_df.columns[0:11]])  
  
# Boxplot  
sns.boxplot(ax=ax, x="Private", y="Outstate", data=college\_df)

# 3. Chapter 3: Linear Regression

Linear regression is a simple supervised learning assuming a linear relation between and . When there is only one predictor, it’s a **simple linear regression**. When there are more than one predictors, it’s called **multiple linear regression**.

## 3.1 Simple Linear Regression

Assumes the *populaton regression line* model

where, is the *expected* value of when , and is the *average* change in with a one-unit increase in . is a “catch all” error term.

After training using the training data, we can obtain the parameter estimates and . The we can obtain the prediction for given by the *least square line*:

The error at a data point is given by , and the *residual sum of squares* (RSS) is

One can use the least square approach to minimize RSS to obtain

where, and , and the correlation

Note . When there is no intercept, that is , then

where,

That is, the fitted values are linear combinations of the response values when there is no intercept.

### 3.1.1 Assessing the accuracy of the coefficients

Let , that is, is the variance of , (estimated by . ) Assume each observation have *common variance* (homoscedasticity) and are *uncorrelated*, then the standard errors under repeated sampling

* when are more spread out (with large ), then is small. This is because there are more *leverage* (of values) to estimate the slope.
* when , then . In this case, .

Standard errors are used to construct CI and perform hypothesis test for the estimated or . Under the assumption of **Gaussian error**, One can construct the CI as

Where . Large interval including zero indicates is not statistically significant from 0$. When is sufficient large, . With the standard errors of the coefficients, one can also perform **hypothesis test** on the coefficients. For ,

The -statistic of degree , given by

shows how far away is away from zero, normalized by its error . One can then compute the -value corresponding to this and test the hypothesis. Small -value indicates **strong** relationship.

## 3.2 Multiple Linear Regression

The estimate of the coefficients , are found by using the same least square method to minimize RSS. we interpret as the *expected* (average) effect on on one unit increase in , **holding all other predictors fixed**. This interpretation is based on the assumptions that *the predictors are uncorrelated*, so *each predictor can be estimated and tested separately*. Where there are correlations among predictors, the variance of all coefficients tends to increase, sometimes dramatically, and the previous interpretation becomes hazardous because when changes, everything else changes.

### 3.2.1 **Model Assumption**

* linearity: is linear in . The change in Y associated with one unit of change in is constant, regardless of the value of . This can be examined visually plotting the *residual plot* vs.  when or vs for multiple regression. Then the linear assumption is true, then the residual plot should not exhibit obvious pattern. If there is a nonlinear relationship suggested by by the residual plot, then a simple approach is to include transformed , such as , , or .
* additive: The association between and is independent of other predictors.
* Errors are uncorrelated. This means provides no information for . Otherwise (for example, frequently observed in a time series, where error terms are positively correlated, and *tracking* is observed in the residuals, i.e., adjacent error terms similar values), the estimated standard error will tend to be underestimated, hence leading less confidence in the estimated model.
* Homoscedasticity: . The error terms have constant variance. If not (heteroscedasticity), one may use transformed , such as , or to mitigate this; or use *weighted least squares* if it’s known that for example .
* Non-colinearity: two variables are colinear if they are highly correlated with each other. Co-linearity causes a great deal of *uncertainty* in the coefficient estimates, that is, reducing the accuracy of the coefficient estimates, thus cause the standard error of to grow, and hence smaller -statistic. As a result, we may fail to reject . This in turn means the power of Hypothesis test, the probability of correctly detecting a *non-zero* coefficient is reduced by colinearity. To detect colinearity,
  + use the correlation matrix of predictors. large value of the matrix in absolute value indicates highly correlated variable pairs. But this approach cannnot detect *multicolinearity*.
  + Use VIF (Variance inflation factor, VIF ) to detect multicolinearity. It is possible for colinearity exists between three or more variables even if no pair of variables has a particularly high correlation. This is the *multicolinearity* situation.
* VIF is the ratio of the variance of when fitting the full model divided by the variance of if fit on its own. It can be calculated by
* Where is the from a regression of onto all of the other predictors. A VIF value exceeds 5 or 10 ( close to 1) indicates colinearity.
* To remedy a colinearity problem:
  + drop a redundant variable (variables with colinearity should have similar VIF values. )
  + Combine the colinear variables into a single predictor, e.g., taking the average of the standardized versions of those variables.

**Claims of causality should be avoided for observational data**.

### 3.2.2 Assessing existence of linear relationship

* test Hypothesis (test if there is a linear relationship between the response and predictors)
* using -statistic
* If is true, , if is true, . -statistic adjust with . Note that one **cannot conclude** if an individual -statistic is significant, then there is at least one predictor is related to the response, especially when is large. This is related to *multiple testing*. The reason is that when is large, there is (eg 5%) of chance that a predictor will have a small -value by chance. When , -statistic cannot be used.

If the goal is to test that a particular subset of of the coefficients are zero, that is, (for convenience, we put the variables chosen at the end of the variabale list)

In this case, use

where, is the residual sum of squares of a seond model that uses all variables *except* those last variables.

### 3.2.3 Assess the accuracy of the future prediciton

* confidence interval: Indicate how far away is from the population average because the coefficients are estimated, It quantifies *reducible error* around the predicted average response , does-not include .
* prediction interval: Indicate how far away is from . predict an individual response . Prediction interval is always wider than the confidence interval, because it includes *irreducible error*. .

### 3.2.4 Assessing the overall accuracy of the model

* RSE. To this end, first define the *lack of fit* measure **Residual Standard Error**
* It is the *average amount* in the unit of that a response deviates from the *true regression line* (). Note, RSE can increase with more variables if the drecrease of RSS doesnot offset the increase of .
* Approach 2: Using *R-squared* (fraction of variance in explained by ), which is independent of of the scale of , and :
* where, . When is near 0 indicates that 1) either the linear model is wrong 2) or th error variance is high, or both. measures the linear relationship between and . If computed on the training set, when adding more variables, the RSS always decrease, hence will always increase.

For simple linear regression, , where the *sample correlation* measures the linear relationship between variables and . See the formula above [Equation 3.1](#eq-correlation-rxy). For multiple linear regression, . The fitted linear model maximizes this correlation among all possible linear models.

## 3.3 Model Selection/Variable Selections: balance training errors with model size

* **All subsets (best subsets) regression**: compute the least square fit for all possible subsets and then choose among them based on certain criterion that balance training error and model size
* **Forward selection**: Start from the *null model* that only contains . Then find the best model containing one predictor that minimizing RSS. Denote the variable by . Then continue to find the best model with the lowest RSS by adding one variable from the remaining predictors, and so on. Continue until some stopping rule is met: e.g., when all remaining variables have a -value greater than some threshold.
* **Backward selection**: start with all variables in the model. Remove the variable with the largest -value (least statistically significant). The new model is fit, and remove the variable with the largest -value. Continue until a stopping rule is satisfied, e.g., all remaining variables have -value less than some threshold.
* **Mixed selection**: Start with forward selection. Since the -value for variables can become larger as new predictors are added, at any point if the -value of a variable in the model rises above a certain threshold, then we remove that variable. Continue to perform these forward and backward steps until all variables in the model have a sufficiently low -value, and all variables outside the model would have a large -value if added to the model.
* Backward selection cannot be used if . Forward selection can always be used, but might include variables early that later become redundant. Mixed selection can remedy this problem.
* **others** (Chapter 6): including Mallow’s , AIC (Akaike Informaton Criterion), BIC, adjusted , Cross-validation, test set performance.
* **not valid**: we could look at individual -values, but when the number of variables is large, we likely to make a false discoveries.

## 3.4 Handle categorical variables (factor variables)

For a categorical variable with levels, create one fewer dummy variables ()>. The level with no dummy variable is called the *baseline*. The coefficient corresponding to a dummy variable is the expected difference in change in when compared to the baseline, while holding other predictors fixed.

## 3.5 Adding non-linearity

### 3.5.1 Modeling interactions (synergy)

When two variables have interaction, then their product can be added into the regression model, and the product maybe considered as a single variable for inference, for example, compute its SE, -statistics, -value, Hypothesis test, etc.

If we include an interaction in a model, then the **Hierarchy principle** should be followed: always include the main effects, even if the -values associated with their coefficients are not significant. This is because without the main effects, the interactions are hard to interpret, as they would also contain the main effect.

### 3.5.2 Adding terms of transformed predictors

1. *Polynomial regression*: Add a term involving for some .
2. other forms: Adding root or logarithm terms of the predictors.

## 3.6 Outliers (Unusual that is far from )

It is typical for an outlier that does not have an unusual predictor value to have little effect on the least squares fit, but it will increase RSE, hence deteriorate CI, -value and , thus affecting interpreting the model.

An outlier can be identified by computing the

A studentized residual great than 3 may be considered as an outlier.

## 3.7 High leverage points (unusual )

High leverage points tend to have sizeable impact on the regression line. To quantify the observation’s leverage, one needs to compute the **leverage statistic**

and . A large value of this statistic (for example, great than ) indicates an observation with high leverage.

## 3.8 Compared to KNN Regression

KNN regression is a non-parametric method that makes prediction at by taking the average in a -point neightborhood

A small value of provides more flexible model with low bias but high variance while a larger value of provides smoother fit with less variance. An optimal value of depend on the *bias-variance tradeoff*. For non-linear data set, KNN may provides better fit than a linear regression model. However, in higher dimension (e.g., ), even for nonlinear data set, KNN may perform much inferior to linear regression, because of the **curse of dimension**, as the observations that are nearest to may in fact far away from .

## 3.9 Homework (\* indicates optional):

* Conceptual: 1–6
* Applied: 8–15. at least one.

## 3.10 Code Gist

### 3.10.1 Python

dir() # provides a list of objects at the top level name space  
dir(A) # display addtributes and methods for the object A  
' + '.join(X.columns) # form a string by joining the list of column names by "+"

### 3.10.2 Numpy

np.argmax(x) # identify the location of the largest element  
np.concatenate([x,y],axis=0) # concatenate two arrays x and y.

### 3.10.3 Pandas

X = pd.DataFrame(data=X, columns=['a','b'])  
  
pd.DataFrame({'intercept': np.ones(Boston.shape[0]),  
 'lstat': Boston['lstat']}) # make a dataframe using a dictionary  
Boston.columns.drop('medv','age') # drop the elements 'medv' and 'age' from the list of column names  
  
pd.DataFrame({'vif':vals},  
 index=X.columns[1:]) # form a df by specifying index labels  
  
X.values # Convert dataframe X to numpy array  
X.to\_numpy() # recommended to replace the above method  
DataFrame.corr() # correlations between columns   
x.sort\_values(ascending=False)  
pd.to\_numeric(auto\_df['horsepower'], errors='coerce') # if error, denote it by "NaN".  
auto\_df.dropna(subset= ['horsepower', 'mpg',], inplace=True) # looking for NaN in the columns in subset, otherwise, all columns  
  
auto\_df.drop('name', axis=1, inplace=True)  
  
left2.join(right2, how="left") #join two databases by index.   
left1.join(right1, on="key") # left-join by left1["key"] and the index of right1.   
pd.concat([s1, s4], axis="columns", join="outer")

### 3.10.4 Graphics

xlim = ax.get\_xlim() # get the x\_limit values xlim[0], xlim[1]  
ax.axline() # add a line to a plot  
ax.axhline(0, c='k', ls='--'); # horizontal line  
line, = ax.plot(x,y,label="line 1") # "line 1" is the legend  
# alternatively the label can be set by   
line.set\_label("line 1")  
ax.scatter(fitted, residuals, edgecolors = 'k', facecolors = 'none')  
ax.plot([min(fitted),max(fitted)],[0,0],color = 'k',linestyle = ':', alpha = .3)  
ax.legend(loc="upper left", fontsize=25) # adding legendes  
ax.annotate(i,xy=(fitted[i],residuals[i])) # annote at the xy position with i.   
  
  
plt.style.use('seaborn') # pretty matplotlib plots  
plt.rcParams.update({'font.size': 16})  
plt.rcParams["figure.figsize"] = (8,7)  
  
plt.rc('font', size=10)  
plt.rc('figure', titlesize=13)  
plt.rc('axes', labelsize=10)  
plt.rc('axes', titlesize=13)  
plt.rc('legend', fontsize=8) # adjust legend globally

### 3.10.5 Using Sns

sns.set(font\_scale=1.25) # set font size 25% larger than default  
sns.heatmap(corr, cbar=True, annot=True, square=True, fmt='.2f', annot\_kws={'size': 10})  
ax = sns.regplot(x=x, y=y)

### 3.10.6 Using Sklearn

from sklearn.linear\_model import LinearRegression  
## Set the target and predictors  
X = auto\_df['horsepower']  
  
### To get polynomial features  
poly = PolynomialFeatures(interaction\_only=True,include\_bias = False)  
X = poly.fit\_transform(X)  
  
y = auto\_df['mpg']  
  
## Reshape the columns in the required dimensions for sklearn  
length = X.values.shape[0]  
X = X.values.reshape(length, 1) #both X and y needs to be 2-D  
y = y.values.reshape(length, 1)  
  
## Initiate the linear regressor and fit it to data using sklearn  
regr = LinearRegression()  
regr.fit(X, y)  
regr.intercept\_  
regr.coef\_  
  
pred\_y = regr.predict(X)

### 3.10.7 Using statsmodels and ISLP

from ISLP import load\_data  
from ISLP.models import (ModelSpec as MS,  
 summarize,  
 poly)  
   
import statsmodels.api as sm  
import statsmodels.formula.api as smf  
from statsmodels.stats.outliers\_influence \  
 import variance\_inflation\_factor as VIF  
from statsmodels.stats.anova import anova\_lm  
  
#Training  
Boston = load\_data("Boston")   
#hand-craft the design matrix X  
X = pd.DataFrame({'intercept': np.ones(Boston.shape[0]), #design matrix. intercept column  
 'lstat': Boston['lstat']})   
#the following is the preferred method to create X  
design = MS(['lstat']) # specifying the model variables. Automatically add an intercept, adding "intercept=False" if no intercept.   
design = design.fit(Boston) # do intial computation as specified in the model object design by MS(), such as means or sd. This attached some statistics to the `design` object, and need to be applied to the new data for prediciton  
  
X = design.transform(Boston) # apply the fitted transformation to the data to create X  
#alternatiely,   
X = design.fit\_transform(Boston) # this combines the .fit() and .transform() two lines  
  
y = Boston['medv']  
model = sm.OLS(y, X) # setup the model  
model = smf.ols('mpg ~ horsepower', data=auto\_df) # alternatively use smf formula, y~x  
smf.ols("y ~ x -1" , data=df).fit() # "-1" not inclding the intercept  
results = model.fit() # results is a dictionary:.summary(), .params   
  
results.summary()  
results.params # coefficients  
results.resid # reisdual array  
results.rsquared # R^2  
np.sqrt(results.scale) # RSE  
results.fittedvalues # fitted \hat(y)\_i at x\_i in the traning set  
  
  
summarize(results) # summzrize() is from ISLP to show the esstial results from model.fit()  
  
# Makding prediciton   
new\_df = pd.DataFrame({'lstat':[5, 10, 15]}) # new test-set containing data where to make predicitons  
newX = design.transform(new\_df) # apply the same transform to the test-set  
new\_predictions = results.get\_prediction(newX);  
new\_predictions.predicted\_mean #predicted values  
new\_predictions.conf\_int(alpha=0.05) #for the predicted values  
  
new\_predictions.conf\_int(obs=True, alpha=0.05) # prediction intervals by setting obs=True  
  
# Including an interaction term  
X = MS(['lstat',  
 'age',  
 ('lstat', 'age')]).fit\_transform(Boston) #interaction term ('lstat', 'age')  
  
# Adding a polynomial term of higher degree  
X = MS([poly('lstat', degree=2), 'age']).fit\_transform(Boston) # Note poly is from ISLP, # adding deg1 and deg2 terms. by default poly creates ortho. poly. not including an intercept.   
# Given a qualitative variable, `ModelSpec()` generates dummy  
variables automatically, to avoid collinearity with an intercept, the first column is dropped in the design matrix generated by 'ModelSpec()` by default.  
  
# Compare nested models using ANOVA  
anova\_lm(results1, results3) # result1 is the result of linear model, an result3 is the result of a larger model  
  
# Identify high leverage x  
infl = results.get\_influence()   
# hat\_matrix\_diag calculate the leverate statistics  
np.argmax(infl.hat\_matrix\_diag) # identify the location of the largest levarage  
  
# Calculate VIF  
vals = [VIF(X, i)  
 for i in range(1, X.shape[1])] #excluding column 0 because it's all 1's in X.  
vif = pd.DataFrame({'vif':vals},  
 index=X.columns[1:])  
vif # VIF exceeds 5 or 10 indicates a problematic amount of colinearity

Useful Code Snippets

def abline(ax, b, m, \*args, \*\*kwargs):  
 "Add a line with slope m and intercept b to ax"  
 xlim = ax.get\_xlim()  
 ylim = [m \* xlim[0] + b, m \* xlim[1] + b]  
 ax.plot(xlim, ylim, \*args, \*\*kwargs)

# Plot scatter plot with a regression line  
ax = Boston.plot.scatter('lstat', 'medv')  
abline(ax,  
 results.params[0],  
 results.params[1],  
 'r--',  
 linewidth=3)

# Plot residuals vs. fitted values (note, not vs x, therefore works for multiple regression)  
ax = subplots(figsize=(8,8))[1]  
ax.scatter(results.fittedvalues, results.resid)  
ax.set\_xlabel('Fitted value')  
ax.set\_ylabel('Residual')  
ax.axhline(0, c='k', ls='--');  
  
# Alternatively  
sns.residplot(x=X, y=y, lowess=True, color="g", ax=ax)  
  
# Plot the smoothed residuals~fitted by LOWESS  
from statsmodels.nonparametric.smoothers\_lowess import lowess  
smoothed = lowess(residuals,fitted) # Note the order (y,x)  
ax.plot(smoothed[:,0],smoothed[:,1],color = 'r')  
  
# QQ plot for the residuas (obtain studentized residuals for identifying outliers)  
import scipy.stats as stats  
sorted\_student\_residuals = pd.Series(smf\_model.get\_influence().resid\_studentized\_internal)  
sorted\_student\_residuals.index = smf\_model.resid.index  
sorted\_student\_residuals = sorted\_student\_residuals.sort\_values(ascending = True)  
df = pd.DataFrame(sorted\_student\_residuals)  
df.columns = ['sorted\_student\_residuals']  
  
#stats.probplot() #assess whether a dataset follows a specified distribution  
df['theoretical\_quantiles'] = stats.probplot(df['sorted\_student\_residuals'], dist = 'norm', fit = False)[0]   
   
x = df['theoretical\_quantiles']  
y = df['sorted\_student\_residuals']  
ax.scatter(x,y, edgecolor = 'k',facecolor = 'none')

# Plot leverage statistics  
infl = results.get\_influence()  
ax = subplots(figsize=(8,8))[1]  
ax.scatter(np.arange(X.shape[0]), infl.hat\_matrix\_diag)  
ax.set\_xlabel('Index')  
ax.set\_ylabel('Leverage')  
np.argmax(infl.hat\_matrix\_diag) # identify the location of the largest levarage

# 4. Summary

In summary, this book has no content whatsoever.

# References