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/
   ```

4. 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_l` 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`.

5. 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 $X \in \mathbb{R}^p$ and response variable $Y \in \mathbb{R}$, assume that

$$Y = f(X) + \epsilon,$$

where ϵ is the random variable representing **irreducible error**. We assume ϵ is *independent* of X and $E[x] = 0$. ϵ may include *unmeasured variables* or *unmeasurable variation*.

Statistical learning is to estimate f using various methods. Denote the estimate by $\hat{f}$.

- regression problem: when Y is a continuous variable (quantitative). In this case $f(x) = E(Y|X = x)$ is the *regression* function, that is, regression finds a conditional expectation of Y.
- classification problem: when Y 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 $\mathcal{T}r = \{(x_i, y_i) : i \in \mathbb{Z}_n\}$: linear regression, logistic regression
- unsupervised learning: when only x_i are available. clustering analysis, PCA
- semi-supervised learning: some data with labels (y_i), some do not.
- reinforcement learning: learn a state-action policy function for an agent to interacting with an environment.

2.2 Why estimate f?

We can use estimated $\hat{f}$ to

- make predictions for a new X,
$$\hat{Y} = \hat{f}(X).$$

The prediction error may be quantified as

$$E[(Y - \hat{Y})^2] = (f(X) - \hat{f})^2 + \text{Var}[\epsilon].$$

The first term of the error is *reducible* by trying to improve $\hat{f}$, where we assume f, $\hat{f}$ and X are fixed.

- make inference:
  - Which predictors are associated with the response?
  - which is the relationship between the response and each predictor?
  - is the assumed relationship adequate? (linear or more complicated?)

2.3 How to estimate f

We use obtained observations called **training data** $\{(x_k, y_k) : k \in \mathbb{Z}_n\}$ to train an algorithm to obtain the estimate $\hat{f}$.

- 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 f. They seek to estimate f directly using data points, can be quite flexible and accurate.

  **Disadvantage: need large number of data points

  Example: KNN (breakdown for higher dimention. Typically only for $p \leq 4$), spline fit.

2.4 How to assess model accuracy

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

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

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

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

The accuracy on a training set can be arbitrarily increased by increasing the model flexibility. Since 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 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 x_0, the expected test MSE can be decomposed into

$$E\left[(y_0 - \hat{f}(x_0))^2\right] = \text{Var}[\hat{f}(x_0)] + (\text{Bias}(\hat{f}(x_0)))^2 + \text{Var}[\epsilon]$$

where the expectation is over different $\hat{f}$ on a different training set or on a different training step if the training process is stochastic, and

$$\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$$

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

$$\text{Ave}(I(y_0 \neq \hat{y}_0)).$$

A Bayes Classifier assigns a test observation with predictor x_0 to the class for which

$$\Pr(Y = j | X = x_0)$$

is largest. It's error rate is given by

$$1 - E[\max_j \Pr(Y = j|X)]$$

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

Bayes Classifier is not attainable as we do not know $\Pr(Y|X)$. We only can estimate $\Pr(Y|X)$. One way to do this is by KNN. KNN estimate the conditional probability simply with a majority vote. The flexibility of KNN increases as $1/K$ increases with $K = 1$ being the most flexible KNN. The training error is 0 for $K = 1$. A suitable K should be chosen for an appropriate trade off between bias and variance. The KNN classifier will classify the test point x_0 based on the probability calculated from the k nearest points. KNN regression on the other hand will assign the test point x_0 the average value of the k 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

```
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]`

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
A[:,[0,2]] # cols 0 and 2
A[[1,3], [0,2]] # 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
A[np.ix_([1,3],keep_cols)] # np.ix_()can be applied to mixture of integer array and boolean a
```

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
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 teh 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` dt
# 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']
                          right=True,# True: right-inclusive for each bin ( ]; False:rigth-ex
                          )   # bin labels (names)
college['Elite'].value_counts() # frequency counts
auto.columns.tolist() # or  auto.columns.format() (rarely used way)
```

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-basd
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 c
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
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 `cylinder
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 Y and X. 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 model

$$Y = \beta_0 + \beta_1 X + \epsilon.$$

After training using the training data, we can obtain the parameter estimates $\hat{\beta}_0$ and $\hat{\beta}_1$. The we can obtain the prediction fro x as

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

The error at a data point x_i is given by $e_i = y_i - \hat{y}_i$, and the *residual sum of squares* (RSS) is

$$\text{RSS} = e_1^2 + \cdots + e_n^2.$$

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

$$\hat{\beta}_1 = \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = r_{xy} \frac{\sigma_y}{\sigma_x}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where, $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ and $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$, and the correlation

$$r_{xy} = \frac{\text{cov(x,y)}}{\sigma_x \sigma_y} = \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}.$$

Note $-1 \leq r_{xy} \leq 1$. When there is no intercept, that is $\beta_0 = 0$, then

$$\hat{y}_i = x_i \hat{\beta} = \sum_{i=1}^n a_i y_i$$

where,

$$\hat{\beta} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}$$

That is, the fitted values are linear combinations of the response values when there is no intercept. ### Assessing the accuracy of the coefficients Let $\sigma^2 = \text{Var}(\epsilon)$. Then the standard errors under repeated sampling

$$(\text{SE}[\hat{\beta}_1])^2 = \frac{1}{\sigma_x^2} \cdot \frac{\sigma^2}{n}$$

$$(\text{SE}[\hat{\beta}_0])^2 = \left[1 + \frac{\bar{x}^2}{\sigma_x^2}\right] \cdot \frac{\sigma^2}{n}$$

Using the estimated $\hat{beta}_0$ or $\hat{beta}_1$ or one can construct the CI as

$$\hat{\beta}_j = [\hat{\beta}_j - t_{0.975,n-p-1} \cdot \text{SE}[\hat{\beta}_j], \hat{\beta}_j + t_{0.975,n-p-1} \cdot \text{SE}[\hat{\beta}_j]]$$

Where $j = 0, 1$. When n is sufficient large, $t_{0.975,n-p-1} \approx 2$. With the standard errors of the coefficients, one can also perform **hypothesis test** on the coefficients. For $j = 0, 1$,

$$H_0 : \beta_j = 0$$

$$H_A : \beta + j \neq 0$$

The t-statistic of degree $n - p - 1$, is given by

$$t = \frac{\hat{\beta}_j - 0}{\text{SE}[\hat{\beta}_j]}$$

One can then compute the p-value corresponding to this t and test the hypothesis.

3.2 Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon.$$

The estimate of the coefficients $\hat{\beta}_j$, $j \in \mathbb{Z}_{p+1}$ are found by using the same least square method to minimize RSS. we interpret β_j as the *expected* (average) effect on Y on one unit increase in X_j, **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 X_j changes, everything else changes.

Claims of causality should be avoided for observational data.

3.2.1 Assess the accuracy of the future prediciton

- prediction interval: predict an individual response $Y = f(X) + \epsilon$. Always wide than the confidence interval, because it includes ϵ.
- confidence interval: predict an average response $f(X)$, doesnot include ϵ

3.2.2 Assessing the overall accuracy of the model

To this end, first define the *Residual Standard Error*

$$\text{RSE} = \sqrt{\frac{1}{n-p-1}\text{RSS}} = \sqrt{\frac{1}{n-p-1}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \approx \sigma = \sqrt{\text{Var}(\epsilon)}$$

We will use two approaches:

- Approach 1: Using *R-squared* (fraction of variance explained):

  $$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

  where, $TSS = \sum_{i=1}^{n}(y_i - \bar{y})$.
- Approach 2: test Hypothesis

  $$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p$$

  $$H_a : \text{at least one } \beta_j \text{ is non-zero.}$$

  using F-statistic

  $$F = \frac{\text{SSB/df(B)}}{\text{SSW/df(W)}} = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n-p-1)} \sim F_{p,n-p-1}$$

  If H_0 is true, $F \approx 1$, if H_a is true, $F >> 1$.

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

- **All subsets (best subsets) regression**: compute the least square fit for all 2^p 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 β_0. Then find the best model containing one predictor that minimizing RSS. Denote the variable by β_1. 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 p-value greater than some threshold.

- **Backward selection**: start with all variables in the model. Remove the variable with the largest p-value (least statistically significant). The new $(p-1)$ model is fit, and remove the variable with the largest p-value. Continue until a stopping rule is satisfied, e.g., all remaining variables have p-value less than some threshold.
- **others**: including Mallow's C_p, AIC (Akaike Informaton Criterion), BIC, adjusted R^2, Cross-validation, test set performance.

3.4 Handle categorical variables (factor variables)

For a categorical variable X_i with m levels, create one fewer dummy variables $(x_ij, 1 \leq j \leq m-1)>$. The level with no dummy variable is called the *baseline*. The coefficient corresponding to a dummy variable is the expected difference in change in Y when compared to the baseline, while holding other predictors fixed.

3.5 Adding non-linearity (to Polynomial Regression)

3.5.1 Modeling interactions (synergy)

When two variables have interaction, then their product X_iX_j can be added into the regression model, and the product maybe considered as a single variable for inference, for example, compute its SE, t-statistics, p-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 p-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 higher power of a predictor

Add a term involving X_i^k for some $k > 1$.

3.6 Outliers (Unusual y_i)

3.7 Non-constant variance of error terms

3.8 High leverage points (unusual x_i)

To quantify the observation's leverage, one needs to compute the **leverage statistic**. A large value of this statistic indicates an observation with high leverage.

3.9 Colinearity

Variance inflection factors (VIF) is useful to assess the effect of colinearity. VIF exceeds 5 or 10 indicates a problematic amount of colinearity.

3.10 Homework (* indicates optional):

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

3.11 Code Gist

3.11.1 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 "+"
```

3.11.2 Numpy

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

3.11.3 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 colu

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

auto_df.drop('name', axis=1, inplace=True)
```

3.11.4 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
```

3.11.5 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)
```

3.11.6 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, addi
design = design.fit(Boston) # do intial computation as specified in the model object design l

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

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 pr
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, # ac
# Given a qualitative variable, `ModelSpec()` generates dummy
variables automatically, to avoid collinearity with an intercept, the first column is droppe

# Compare nested models using ANOVA
anova_lm(results1, results3) # result1 is the result of linear model, an result3 is the resul

# 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)
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', i

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