# 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 $\epsilon$ is a random variable representing **irreducible error**. We assume $\epsilon$ is *independent* of $X$ and $E[\epsilon] = 0$. $\epsilon$ 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 (quantitative) variable . In this case $f(x) = E(Y|X = x)$ is the population *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 to maximize a reward function.

## 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, 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 $f$

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 (but breakdown for higher dimention. Typically only for $p \leq 4$), 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

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

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. 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 $x_0$, 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 $\epsilon$.

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 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
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
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 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` 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'],  # bin labels (names)
                          right=True,# True: right-inclusive (default) for each bin ( ]; Fals
                          )
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 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**. Note *multivariate regression* refer to the $Y$ variable is a vector.

## 3.1 Simple Linear Regression

Assumes the *population regression line* model

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

where, $\beta_0$ is the *expected* value of $Y$ when $X = 0$, and $\beta_1$ is the *average* change in $Y$ with a one-unit increase in $X$. $\epsilon$ is a "catch all" error term.

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 for $x$ given by the *least square line*:

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

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

$$r_{xy} = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y} = \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2}\sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}. \tag{3.1}$$

is the normalized convariance. 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.

### 3.1.1 Assessing the accuracy of the coefficients

Let $\sigma^2 = \text{Var}(\epsilon)$, that is, $\sigma^2$ is the variance of $Y$, (estimated by $\sigma^2 \approx= \text{RSE} = \text{RSS}/(n-p-1)$. ) Assume each observation have *common variance* (homoscedasticity) and are *uncorrelated*, then the standard errors under repeated sampling

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

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

- when $x_i$ are more spread out (with large $\sigma_x^2$), then $\text{SE}[\hat{\beta}_1]$ is small. This is because there are more *leverage* (of $x$ values) to estimate the slope.
- when $\bar{x} = 0$ , then $\text{SE}[\hat{\beta}_0] = \text{SE}[\bar{y}]$. In this case, $\hat{\beta}_0 = \bar{y}$.

Standard errors are used to construct CI and perform hypothesis test for the estimated $\hat{\beta}_0$ or $\hat{\beta}_1$. Under the assumption of **Gaussian error**, One can construct the CI of significance level $\alpha$ (e.g., $\alpha = 0.05$) as

$$\hat{\beta}_j = [\hat{\beta}_j - t_{1-\alpha/2,n-p-1} \cdot \text{SE}[\hat{\beta}_j], \hat{\beta}_j + t_{1-\alpha/2,n-p-1} \cdot \text{SE}[\hat{\beta}_j]]$$

Where $j = 0, 1$. Large interval including zero indicates $\beta_j$ is not statistically significant from 0. 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 $n-p-1$, given by

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

shows how far away $\hat{\beta}_j$ is away from zero, normalized by its error $\text{SE}[\hat{\beta}_j]$. One can then compute the $p$-value corresponding to this $t$ and test the hypothesis. Small $p$-value indicates **strong** relationship.

## 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 $\beta_j$ as the *expected* (average) effect on $Y$ with 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.* When 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.

### 3.2.1 Model Assumption

- linearity: $Y$ is linear in $X$. The change in Y associated with one unit of change in $X_j$ is constant, regardless of the value of $X_j$. This can be examined visually by plotting the *residual plot* ($e_i$ vs. $x_i$ for $p = 1$ or $e_i$ vs $\hat{y}_i$ for multiple regression). If 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 $X$, such as $\log X$, $\sqrt{X}$, or $X^2$.
- additive: The association between $X_j$ and $Y$ is independent of other predictors.
- Errors $\epsilon_i$ are uncorrelated. This means $\epsilon_i$ provides no information for $\epsilon_{i+1}$. 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 take similar values), the estimated standard error will tend to be underestimated, hence leading less confidence in the estimated model.
- Homoscedasticity: $\text{Var}(\epsilon_i) = \sigma^2$. The error terms have constant variance. If not (heteroscedasticity), one may use transformed $Y$, such as $\sqrt{Y}$, or $\log(Y)$ to mitigate this; or use *weighted least squares* if it's known that for example $\sigma_i^2 = \sigma^2/n_i$.

- 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 $\beta_j$ to grow, and hence smaller $t$-statistic. As a result, we may fail to reject $H_0 : \beta_j = 0$. 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 $\geq 1$) 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 $\hat{\beta}_j$ when fitting the full model divided by the variance of $\hat{\beta}_j$ if fit on its own. It can be calculated by

  $$\text{VIF}(\hat{\beta}_j) = \frac{1}{1 - R^2_{X_j|X_{-j}}}$$

  Where $R^2_{X_j|X_{-j}}$ is the $R^2$ from a regression of $X_j$ onto all of the other predictors. A VIF value exceeds 5 or 10 (i.e., $R^2_{X_j|X_{-j}}$ 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)

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

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

  using $F$-statistic

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

If $H_0$ is true, $F \approx 1$; if $H_a$ is true, $F >> 1$. $F$-statistic adjust with $p$. Note that one **cannot conclude** if an individual $t$-statistic is significant, then there is at least one predictor is related to the response, especially when $p$ is large. This is related to *multiple testing*. The reason is that when $p$ is large, there is $\alpha$ (eg 5%) chance that a predictor will have a small $p$-value by chance. When $p > n$, $F$-statistic cannot be used.

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

$$H_0 : \beta_{p-q+1} = \beta_{p-q+2} = \cdots = \beta_p = 0 \tag{3.2}$$

In this case, use

$$F = \frac{(\text{RSS}_0 - \text{RSS})/q}{\text{RSS}/(n-p-1)} \sim F_{q,n-p-1}$$

where, $\text{RSS}_0$ is the residual sum of squares of a second model that uses all variables *except* those last $q$ variables. When $q = 1$, $F$-statistic in Equation 3.2 is the square of the $t$-statistic of that variable.

### 3.2.3 Assess the accuracy of the future prediciton

- confidence interval: Indicate how far away $\hat{Y} = \hat{f}(X)$ is from the population average $f(X)$ because the coefficients $\hat{\beta}_j$ are estimated, It quantifies *reducible error* around the predicted average response $\hat{f}(X)$, does-not include $\epsilon$.
- prediction interval: Indicate how far away $\hat{Y} = \hat{f}(X)$ is from $Y$. predict an individual response $Y \approx \hat{f}(X) + \epsilon$. Prediction interval is always wider than the confidence interval, because it includes *irreducible error* $\epsilon$.

### 3.2.4 Assessing the overall accuracy of the model

- RSE. To this end, first define the *lack of fit* measure **Residual Standard Error**

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

  It is the *average amount* in $\hat{Y}$ that a response deviates from the *true regression line* $(\beta_0 + \beta_1 X)$. Note, RSE can increase with more variables if the decrease of RSS doesnot offset the increase of $p$.

- Approach 2: Using *R-squared* (fraction of variance in $Y$ explained by $X$), which is independent of of the scale of $Y$, and $0 \leq R^2 \leq 1$:

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

  where, $\text{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})$. When $R^2$ is near 0 indicates that 1) either the linear model is wrong 2) or th error variance $\sigma^2$ is high, or both. $R^2$ measures the linear relationship between $X$ and $Y$. If computed on the training set, when adding more variables, the RSS always decrease, hence $R^2$ will always increase.

For simple linear regression, $R^2 = r_{xy}^2$, where the *sample correlation* measures the linear relationship between variables $X$ and $Y$. See the formula $r_{xy}$ above Equation 3.1. For multiple linear regression, $R^2 = (\text{Cor}(Y, \hat{Y}))^2$. 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 $2^p$ 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 $\beta_0$. Then find the best model containing one predictor that minimizing RSS. Denote the variable by $\beta_1$. 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.
- **Mixed selection**: Start with forward selection. Since the $p$-value for variables can become larger as new predictors are added, at any point if the $p$-value of a variable in the model rises above a certain threshold, then remove that variable. Continue to perform these forward and backward steps until all variables in the model have a sufficiently low $p$-value, and all variables outside the model would have a large $p$-value if added to the model.

  Backward selection cannot be used if $p > n$. 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 $C_p$, AIC (Akaike Informaton Criterion), BIC, adjusted $R^2$, Cross-validation, test set performance.
- **not valid**: we could look at individual $p$-values, but when the number of variables $p$ is large, we likely to make a false discoveries.

## 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

### 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 terms of transformed predictors

1) *Polynomial regression*: Add a term involving $X_i^k$ for some $k > 1$.
2) other forms: Adding root or logarithm terms of the predictors.

## 3.6 Outliers (Unusual $y_i$ that is far from $\widehat{y}_i$)

It is typical for an outlier that does not have an unusual predictor value (with low levarage) to have little effect on the least squares fit, but it will increase RSE, hence deteriorate CI, $p$-value and $R^2$, thus affecting interpreting the model.

An outlier can be identified by computing the

$$\text{studentized residual} = \frac{e_i}{\text{RSE}_i}$$

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

## 3.7 High leverage points (unusual $x_i$)

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**

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{n}(x_j - \bar{x})^2}.$$

$1/n \leq h_i \leq 1$ and $\text{Ave}(h_i) = (p+1)/n$. A large value of this statistic (for example, great than $(p+1)/n$) indicates an observation with high leverage.

## 3.8 Compared to KNN Regression

KNN regression is a non-parametric method that makes prediction at $x_0$ by taking the average in a $K$-point neightborhood

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in \mathcal{N}_{x_0}} y_i$$

A small value of $K$ provides more flexible model with low bias but high variance while a larger value of $K$ provides smoother fit with less variance. An optimal value of $K$ 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., $p \geq 4$), even for nonlinear data set, KNN may perform much inferior to linear regression, because of the **curse of dimensionality**, as the $K$ observations that are nearest to $x_0$ may in fact far away from $x_0$.

## 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 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(numeric_only=True) # 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)

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, 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
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
results.pvalues
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 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', 

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 Chapter 4: Classification

Given a feature vector $X$ and a *qualitative* (categorical) response $Y$ taking finite values in a set $\mathcal{C}$, the classification task is to build a classifier $C(X)$ that takes an input $X$ and predicts its class $Y = C(X) \in \mathcal{C}$. This is often done by model $P(Y = k|X = x)$ for each $k \in \mathcal{C}$.

## 4.1 Linear regression and Classification

- For a *binary* classification, one can use linear regression and does a good job. In this case, the linear regression classifier is equivalent to LDA, because

  $$P(Y = 1|X = x) = E[Y|X = x]$$

  However, linear regression may not represent a probability as it may give a value outside the interval $[0, 1]$.
- When there are more than two classes, linear regression is not appropriate, because any chosen coding of the $Y$ variable imposes an ordering and fixed differences among categories, which may not be implied by the data set. If the coding changes, a dramatic function will be fitted, which is not reasonable. One should turn to *multiclass logistic regression* or *Discriminant Analysis*.

## 4.2 Logistic Regression

Logistic regression is a *discriminative learning*, because it directly calculates the conditional probability $P(Y|X)$ to make classification.

### 4.2.1 Binary classification

with a single variable Logistic regression simply convert the linear regression to probability by

$$p(X) = Pr(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}.$$

Note the *logit* or *log odds* is linear

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

Increasing $X$ by one unit, changes the log odds by $\beta_1$. Equivalently, it multiplied the odds by $e^{\beta_1}$. The rate of change of $p(X)$ is no longer a constant, but depends on the current value of $X$. Positive $\beta_1$ implies increasing $p(X)$, and vice vesa.

The parameters are estimated by maximizing the *liklihood*

$$\ell(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i:y_i=0} (1 - p(x_i))$$

With the estimated parameters $\hat{\beta}_j, j = 0, 1$, one can calculate the probability

$$p(X) = Pr(Y = 1|X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1 + e^{\hat{\beta}_0+\hat{\beta}_1 X}}$$

### 4.2.2 with multiple variables

In this case, simply let the logit be a linear function of $p$ variables.

Note when there are multiple variables, it's possible to have variables confounding (especially when two variables are correlated): the coefficient of a variable may changes significantly or may change sign, this is because the coefficient represents the rate of change in $Y$ of that variable when holding other variable constants. The coefficient reflects the effect when other variables are hold constant, how the variable affects $Y$, and this effect may be different than when only this variable is used in the model.

> **Note**
>
> One can include a nonlinear term such as a quadratic term in the logit model, similar to a linear regression that includes a non-linear term.

### 4.2.3 Multi-class logistic regression (multinomial regression) with more than two classes

in this case, we use the *softmax* function to model

$$\Pr(Y = k|X) = \frac{e^{\beta_{0k}+\beta_{1k}X_1+\cdots+\beta_{pk}X_p}}{\sum_{\ell=1}^{K} e^{\beta_{0\ell}+\beta_{1\ell}X_1+\cdots+\beta_{p\ell}X_p}} = a_k$$

for each class $k$. Note $\Sigma_k a_k = 1$ and the cross-entropy loss function is given by $-\log \ell(\beta) = -\Sigma_k \mathbb{1}_k \log a_k$, where $\beta$ represents all the parameters.

The log odds between $k$th and $k'$th classes equals

$$\log(\frac{\Pr(Y=k|X=x)}{\Pr(Y=k'|X=x)}) = (\beta_{k0} - \beta_{k'0}) + (\beta_{k1} - \beta_{k'1}) + \cdots + (\beta_{kp} - \beta_{k'p})$$

## 4.3 Discriminant Classifier: Approximating Optimal Bayes Classifier

Apply the Bayes Theorem, the model

$$\Pr(Y=k|X=x) = \frac{\Pr(X=x|Y=k)\cdot\Pr(Y=k)}{\Pr(X=x)} = \frac{\pi_k f_k(x)}{\sum_{\ell=}^{K} \pi_\ell f_\ell(x)}$$

where $\pi_k$ = Pr(Y=k) is the *marginal* or *prior* probability for class $k$, and $f_k(x) = \Pr(X = x|Y=k)$ is the *density* for $X$ in class $k$. Note the denominator is a *normalizing constant*. So when making decisions, effectively we compare $\pi_k f_k(x)$, and assign $x$ to a class $k$ with the largest $\pi_k f_k(x)$.

Discriminant uses the full liklihood $P(X,Y)$ to calculate $P(Y|X)$ to make a classification, so it's known as *generative learning*.

- when $f_k$ is chosen as a normal distribution with constant variance ($\sigma^2$) for $p = 1$ or correlation matrix $\Sigma$ for $p > 1$, this leads to the LDA. For $p = 1$, the *discriminant score* is given by

  $$\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$

  when $K = 2$ and $\pi_1 = \pi_2 = 0.5$m then the *decision boundary* is given by

  $$x = \frac{\mu_1 + \mu_2}{2}.$$

  When $p \geq 2$, assume that $X = (X_1, X_2, \cdots, X_p)$ is drawn from a multivariate Gaussian distribution $X \sim N(\mu_k, \Sigma)$, with a class-specific mean vector and a a common variance matrix.

  $$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log\pi_k = c_{k0} + c_{k1}x_1 + \cdots + c_{kp}x_p.$$

  The score function (posterior probability) is *linear* in $x$. With $\hat{\delta}_k(x)$ for each $k$, it can be converted to the class probability by the *softmax* function

  $$\hat{\Pr}(Y=k|X=x) = \frac{e^{\hat{\delta}_k(x)}}{\sum_{\ell=1}^{K} e^{\hat{\delta}_\ell(x)}}$$

The $\pi_k$, $\mu_k$ and $\sigma$ are estimate the follwing way:

$$\hat{\pi}_k = \frac{n_k}{n}$$

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$$

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 = \sum_{k=1}^{K} \frac{n_k - 1}{n-K} \hat{\sigma}_k^2$$

where $\hat{\sigma}_k^2 = \frac{1}{n_k - 1} \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$ is the estimated variance for the $k$-th class.

> **Note**
>
> One can include a nonlinear term such as a quadratic term in the LDA model, similar to a linear regression that includes a non-linear term.

- when each class chooses a different $\Sigma_k$, then it's QDA. It assumes an observation from the $k$-th class is $X \sim N(\mu_k, \Sigma_k)$.The score function has a *quadratic* term

  $$\delta_k(x) = -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) + \log \pi_k - \frac{1}{2} \log |\Sigma_k|$$

  QDA has much more parameters $Kp(p+1)/2$ to estimate compared to LDA ($Kp$), hence has higher flexibility and may lead to higher variance. When there are few training examples, LDA tend to perform better and reducing variance is crucial. When there is a large traning set, QDA is recommended as variance is not a major concern. LDA is a special case of QDA.

- when the features are modeled independently, i.e., there is no association between the $p$ predictors, $f_k(x) = \prod_{j=1}^{p} f_{jk}(x_j)$, the method is *naive Bayes*, and $\Sigma_k$ are diagonal. Any classifier with a linear decision boundary is a special case of NB. So LDA is a special case of NB. To estimate $f_kj$, one can

  - assume that $X_j|Y = k \sim N(\mu_{jk,\sigma_{jk}^2})$, that is, a class specific covariance but is diagonal. QDA's $\Sigma_k$ is not diagonal. If we model $f_{kj}(x_j) \sim N(\mu_{kj} + \sigma_j^2)$ (Note $\sigma_j^2$ is shared among clases), In this case NB is a special case of LDA that has a diagonal $\Sigma$ and
  - use a non-parametric estimate such as histogram (or a smooth kernel density estimator) for the observations of the jth Predictor within each class.
  - If $X_j$ is qualitative, then one can simply count the proportion of training observations for the $j$th predictor corresponding to each class.

- Can applied to *mixed* feature vectors (qualitative and quantitative). NB does not assume normally distributed predictors.
- Despite strong assumptions, performs well, especially when $n$ is not large enough relative to $p$, when estimating the joint distribution is difficult. It introduces some biases but reduces variance, leading to a classifier that works quite well as a result of bias-variance trade-off.
- Useful when $p$ is very large.
- NB is a *generalized additive model*.
- Neigher NB nor QDA is a special case of the other. Because QDA contains interaction term $x_i x_j$, while NB is purely additive, in the sense that a function of $x_i$ is added to a function of $x_j$. Therefore, QDA potentially is a better fit when the interactions among predictors are important.

### 4.3.1 Why discriminant analysis

- When the classes are well-separated, the parameter estimation of logistic regression is unstable, while LDA does not suffer from this problem.
- if the data size $n$ is small and the distribution of $X$ is approximately normal in each of the classes, then LDA is more stable than logistic regression. Also used when $K > 2$.
- when there are more than two classes, LDA provides low-dimensional views of the data hence popular. Specifically, when there are $K$ classes, LDA can be viewed exactly in $K-1$ dimensional plot. This is because it essentially classifies to the closest centroid, and they span a $K-1$ dimensional plane.
- For a two-class problem, the logit of $p(Y=1|X=x)$ by LDA (generative learning) is a linear function in $X$, the same as a logistic regression (discriminative learning). The difference lies in how the parameters are estimated. But in practice, they are similar.
- LDA assumes the predictors follow a multivariable normal distribution with a shared $\Sigma$ among classes. So when this assumption holds, we expect LDA performs better; and Logistic regress performs better when this asuumption does not hold.

## 4.4 KNN

KNN is a non-parametric method and doesnot assume a shape for the decision boundary. KNN assign the class of popularity to $X = x$ in a $K$-neighborhood.

- KNN dominates LDA and Logistic Regression when the decision boundary is highly non-linear, provided $n$ is large and $p$ is small. As KNN breaks down when $p$ is large.

- KNN requires large $n >> p$, this is because KNN is non-parametric and tends to reduce bias but increase variance.
- When the decision boundary is non-linear but $n$ is only modest and $p$ is not very small, QDA may outperform KNN. This is because QDA provides a non-linear boundary while taking advantage of a parametric form, which means that if requires smaller size for accurate classification.
- Unlike logistic regression, KNN does not tell which predictors are more importnat: We dont get a table of coefficients.
- When the decision boundary is linear, LDA or logistic regression may perform better, when the boundary is moderately non-linear, QDA or NB may perform better; For a much more complicated decision boundary, KNN may perform better.

## 4.5 Poisson Regression

When $Y$ is discrete and non-negative, a linear regression model is not satisfactory, even with the transformation of $\log(Y)$, because log does not allow $Y = 0$.

- Poisson Regression: typically used to model counts,

$$\Pr(Y = k) = \frac{e^{-\lambda}\lambda^k}{k!}, \qquad k = 0, 1, 2, \cdots,$$

  where, $\lambda = E(Y) = \text{Var}(Y)$. This means that if $Y$ follows a Poissson distribution, the larger the mean of $Y$, the larger its variance. Posisson regression can handle this when variance changes with mean, but linear regression cannot, because it assumes contant variance.

Assume

$$\log(\lambda(X_1, X_2, \cdots, X_p)) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

Then one can use maximum likelihood

$$\ell(\beta_0, \beta_1, \cdots, \beta_p) = \prod_{i=1}^{n} \frac{e^{-\lambda(x_i)}\lambda(x_i)^{y_i}}{y_i!}$$

to estimate the parameters.

- Interpretation: An increase in $X_j$ by one unit is associated with a change in $E(Y) = \lambda$ by a *factor* (percentage) of $\exp(\beta_j)$.

## 4.6 Generalized Linear Models (GLM)

Perform a regression by modeling $Y$ from a particular member of the *exponential family* (Gaussian, Bernoulli, Poisson, Gamma, negative binomial), and then transform the mean of $Y$ to a linear function.

- Use predictors $X_1, \cdots, X_p$ to predict $Y$. Assume $Y$ conditional on $X$ follow some distribution: For linear regression, assume $Y$ follows a normal distribution; for logistic regression, assume $Y$ follows a Bernoulli (multinomial distribution for multi-class logistic regression) distribution; For poisson distribution, assume $Y$ follows a poisson distribution.
- Each approach models the mean of $Y$ as a function of $X$ using a *linking function* $\eta$ to transform $E[Y|X]$ to a linear function.
  - for linear regression

    $$E(Y|X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$$

    $\eta(\mu)\mu$
  - for logistic regression

    $$E(Y|X) = P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}$$

    $\eta(\mu) = \log(\mu/(1-\mu))$
  - for Poisson regression

    $$E(Y|X) = \lambda(X) = e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}$$

    $\eta(\mu) = \log(\mu)$.
  - Gamma regression and negative binomial regression.

## 4.7 Assessment of a classifier

- Confusion matrix
- Overall error rate: equals to

  $$\frac{FP + FN}{N + P}$$

- Class-specific performance: One can adjust the decision boundary (posterior probability threshold) to improve class specific performance at the expense of lowered overall performance.

- percentage of TP detected among all positives

  $$\text{sensitivity (recall, power)} = TPR = \frac{TP}{TP+FN} = \frac{TP}{P} = 1 - \text{Type II error} = 1-\beta$$

  this is equal to $1-FNR$, where, FNR is The fraction of positive examples that are classified as negatives

  $$FNR = \frac{FN}{FN+TP} = \frac{FN}{P}$$

- percentage of TN detected among all negatives

  $$\text{specificity} = TNR = \frac{TN}{TN+FP} = \frac{TN}{N}$$

  This is equal to $1-FPR$, where, False positive rate (FPR): the fraction of negative examples (N) that are classified as positive:

  $$FPR = \frac{FP}{FP+TN} = \frac{FP}{N} = \text{Type I error}(\alpha)$$

- ROC (receiver operating characteristic curve): plot true positive rate (TPR=1-Type II error) ~ false positive rate (FPR= 1- specificity=Type I error) as a threshold for the posterior probability of positive class changes from 0 to 1. The point on the ROC curve closest to the point (0,1) corresponds to the best classifier.
- AUC (area under the ROC): Overall performance of a classified summarized over all thresholds. AUC measures the probability a random positive example is ranked higher than a random negative example. A larger AUC indicates a better classifier.

- class-specific prediction performance

  -

  $$\text{precision} = \frac{TP}{TP+FP} = \frac{TP}{\text{predicted postives}} = 1 - \text{false discovery proportion}$$

## 4.8 Homework:

- Conceptual: 1,2,3,5,6,7,8, 9
- Applied: 13, 14*,15*,16*

## 4.9 Code Gist

### 4.9.1 Python

### 4.9.2 Numpy

```
np.where(lda_prob[:,1] >= 0.5, 'Up','Down')
np.argmax(lda_prob, 1) #argmax along axis=1 (col)
np.asarray(feature_std) # convert to np array
np.allclose(M_lm.fittedvalues, M2_lm.fittedvalues)
#check if corresponding elts are equal within rtol=1e-5 and atol=-1e08
```

### 4.9.3 Pandas

```
Smarket.corr(numeric_only=True)
train = (Smarket.Year < 2005)
Smarket_train = Smarket.loc[train] # equivalent to Smarket[train]
Purchase.value_counts() # frequency table
feature_std.std() #calculate column std
S2.index.str.contains('mnth')
Bike['mnth'].dtype.categories # get the categories of the categorical data
obj2 = obj.reindex(["a", "b", "c", "d", "e"])# rearrange the entries in obj according to the
```

### 4.9.4 Graphics

```
ax_month.set_xticks(x_month) # set_xticks at the place given by x_month
ax_month.set_xticklabels([l[5] for l in coef_month.index], fontsize=20)
ax.axline([0,0], c='black', linewidth=3,
          linestyle='--', slope=1);#axline method draw a line passing a given point with a g
```

### 4.9.5 ISLP and Statsmodels

```
from ISLP import confusion_table
from ISLP.models import contrast

# Logistic Regression using sm.GLM() syntax similar to sm.OLS()
design = MS(allvars)
X = design.fit_transform(Smarket)
y = Smarket.Direction == 'Up'
glm = sm.GLM(y,
```

```
            X,
            family=sm.families.Binomial())
results = glm.fit()
summarize(results)
results.pvalues
probs = results.predict() #without data set, calculate predictions on the training set.
results.predict(exog=X_test) # on test set
# Prediction on a new dataset
newdata = pd.DataFrame({'Lag1':[1.2, 1.5],
                        'Lag2':[1.1, -0.8]});
newX = model.transform(newdata)
results.predict(newX)
confusion_table(labels, Smarket.Direction) #(predicted_labels, true_labels)
np.mean(labels == Smarket.Direction) # calculate the accuracy

hr_encode = contrast('hr', 'sum') #coding scheme for categorical data: the unreported coeffi

#Poisson Regression
M_pois = sm.GLM(Y, X2, family=sm.families.Poisson()).fit()
#`family=sm.families.Gamma()` fits a Gamma regression
model.
```

### 4.9.6 sklearn

```
from sklearn.discriminant_analysis import \
     (LinearDiscriminantAnalysis as LDA,
      QuadraticDiscriminantAnalysis as QDA)
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#LDA
lda = LDA(store_covariance=True) #store the covariance of each class
X_train, X_test = [M.drop(columns=['intercept']) # drop the intercept column
                   for M in [X_train, X_test]]
lda.fit(X_train, L_train) # LDA() model will automatically add a intercept term

lda.means_ # mu_k (n_classes, n_features)
lda.classes_
lda.priors_ # prior probability of each class
```

```
#Linear discrimnant vectors
lda.scalings_ #Scaling of the features in the space spanned by the class centroids. Only ava

lda_pred = lda.predict(X_test) #predict class labels
lda_prob = lda.predict_proba(X_test) #ndarray of shape (n_samples, n_classes)

#QDA
qda = QDA(store_covariance=True)
qda.fit(X_train, L_train)
qda.covariance_[0] #estimated covariance for the first class

# Naive Bayes
NB = GaussianNB()
NB.fit(X_train, L_train)
NB.class_prior_
NB.theta_ #means for (#classes, #features)
NB.var_ #variances (#classes, #features)
NB.predict_proba(X_test)[:5]

# KNN
knn1 = KNeighborsClassifier(n_neighbors=1)
X_train, X_test = [np.asarray(X) for X in [X_train, X_test]]
knn1.fit(X_train, L_train)
knn1_pred = knn1.predict(X_test)

# When using KNN one should standarize each varaibles
scaler = StandardScaler(with_mean=True,
                        with_std=True,
                        copy=True) # do calculaton on a copy of the dataset
scaler.fit(feature_df)

#train test split
X_std = scaler.transform(feature_df)
(X_train,
 X_test,
 y_train,
 y_test) = train_test_split(np.asarray(feature_std),
                            Purchase,
                            test_size=1000,
                            random_state=0)

# Logistic Regression
logit = LogisticRegression(C=1e10, solver='liblinear') #use solver='liblinear'to avoid warni
```

```
logit.fit(X_train, y_train)
logit_pred = logit.predict_proba(X_test)
```

### 4.9.7 Useful code snippet

```
# Tuning KNN
for K in range(1,6):
    knn = KNeighborsClassifier(n_neighbors=K)
    knn_pred = knn.fit(X_train, y_train).predict(X_test)
    C = confusion_table(knn_pred, y_test)
    templ = ('K={0:d}: # predicted to rent: {1:>2},' +  # > for right alighment
            '  # who did rent {2:d}, accuracy {3:.1%}')
    pred = C.loc['Yes'].sum()
    did_rent = C.loc['Yes','Yes']
    print(templ.format(
          K,
          pred,
          did_rent,
          did_rent / pred))
```

# 5 Summary

In summary, this book has no content whatsoever.

# References