stat4500notes

Yi Wang

2023-09-19

Table of contents

# Preface

This is a Lecture note written for the course STAT 4500: Machine Learning offered at Auburn University at Montgomery. The course uses the textbook James et al. (2023).

This is a book wrtieen by Quarto book.

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

# 1. Setting up Python Computing Environment

## 1.1 on Your own computer

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

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

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

## 1.2 Use Google Colab

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

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

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

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

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

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

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

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

# 2. Chapter 2: Statistical Learning

## 2.1 What is statistical learning?

For the input variable and response variable , assume that

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

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

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

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

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

## 2.2 Why estimate ?

We can use estimated to

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

## 2.3 How to estimate

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

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

## 2.4 How to assess model accuracy

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

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

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

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

## 2.5 Model Selection:

**No free lunch theorem**

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

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

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

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

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

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

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

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

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

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

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

## 2.6 Bayes Classifier

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

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

is largest. It’s error rate is given by

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

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

## 2.7 Homework (\* indicates optional):

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

## 2.8 Code Gist

### 2.8.1 OS

import os  
os.chdir(path) # change dir

### 2.8.2 Python:

Concatenation using +

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

String formatting using string.format()

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

zip to loop over a sequence of tuples

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

### 2.8.3 Numpy

#### 2.8.3.1 Numpy functions:

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

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

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

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

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

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

np.ones(Boston.shape[0])

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

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

#### 2.8.3.2 Array Slicing and indexing

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

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

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

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

#### 2.8.3.3 Random numbers and generators

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

#### 2.8.3.4 Numpy array atributes

.dtype, .ndim, .shape

#### 2.8.3.5 Numpy array methods

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

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

x.min(), x.max()

### 2.8.4 Graphics

#### 2.8.4.1 2-D figure

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

#### 2.8.4.2 Contour and image

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

### 2.8.5 Pandas

#### 2.8.5.1 loading data

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

#### 2.8.5.2 Pandas Dataframe attributes and methods

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

#### 2.8.5.3 Selecting rows and columns

Select Rows:

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

Select Columns

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

Select a subset

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

#### 2.8.5.4 Pandas graphics

Without using subplots to get axes and figure objects

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

Using subplots

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

Boxplot using subplots

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

Scatter matrix

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

Sns Graphic

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

# 3. Chapter 3: Linear Regression

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

## 3.1 Simple Linear Regression

Assumes the *population regression line* model

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

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

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

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

where, and , and the correlation

is the normalized convariance. Note . When there is no intercept, that is , then

where,

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

### 3.1.1 Assessing the accuracy of the coefficients

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

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

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

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

The -statistic of degree , given by

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

## 3.2 Multiple Linear Regression

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

### 3.2.1 **Model Assumption**

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

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

### 3.2.2 Assessing existence of linear relationship

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

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

In this case, use

where, is the residual sum of squares of a second model that uses all variables *except* those last variables. When , -statistic in [Equation 3.2](#eq-subset-hypothesis) is the square of the -statistic of that variable. The -statistic reported in a regression model gives the *partial effect* of adding that variable, while holding other variables fixed.

### 3.2.3 Assess the accuracy of the future prediciton

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

### 3.2.4 Assessing the overall accuracy of the model

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

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

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

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

## 3.4 Handle categorical variables (factor variables)

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

## 3.5 Adding non-linearity

### 3.5.1 Modeling interactions (synergy)

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

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

### 3.5.2 Adding terms of transformed predictors

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

## 3.6 Outliers (Unusual that is far from )

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

An outlier can be identified by computing the

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

## 3.7 High leverage points (unusual )

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

and . A large value of this statistic (for example, great than ) indicates an observation with high leverage. The leverage , reflects the amount an observation influences its own fit.

## 3.8 Compared to KNN Regression

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

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

## 3.9 Homework (\* indicates optional):

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

## 3.10 Code Gist

### 3.10.1 Python

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

### 3.10.2 Numpy

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

### 3.10.3 Pandas

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

### 3.10.4 Graphics

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

### 3.10.5 Using Sns

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

### 3.10.6 Using Sklearn

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

### 3.10.7 Using statsmodels and ISLP

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

Useful Code Snippets

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

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

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

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

# 4. Chapter 4: Classification

Given a feature vector and a *qualitative* (categorical) response taking finite values in a set , the classification task is to build a classifier that takes an input and predicts its class . This is often done by model for each .

## 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
* However, linear regression may not represent a probability as it may give a value outside the interval .
* When there are more than two classes, linear regression is not appropriate, because any chosen coding of the 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 to make classification.

### 4.2.1 Binary classification

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

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

Increasing by one unit, changes the log odds by . Equivalently, it multiplied the odds by . The rate of change of is no longer a constant, but depends on the current value of . Positive implies increasing , and vice vesa.

The parameters are estimated by maximizing the *liklihood*

With the estimated parameters , one can calculate the probability

### 4.2.2 with multiple variables

In this case, simply let the logit be a linear function of 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 of that variable when holding other variable constants. The coefficient reflects the effect when other variables are hold constant, how the variable affects , and this effect may be different than when only this variable is used in the model.

|  |  |
| --- | --- |
|  | 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

for each class . Note and the cross-entropy loss function is given by , where represents all the parameters.

The log odds between th and th classes equals

## 4.3 Discriminant Classifier: Approximating Optimal Bayes Classifier

Apply the Bayes Theorem, the model

where is the *marginal* or *prior* probability for class , and ) is the *density* for in class . Note the denominator is a *normalizing constant*. So when making decisions, effectively we compare , and assign to a class with the largest .

Discriminant uses the full liklihood to calculate to make a classification, so it’s known as *generative learning*.

* when is chosen as a normal distribution with constant variance () for or correlation matrix for , this leads to the LDA. For , the *discriminant score* is given by
* when and m then the *decision boundary* is given by
* When , assume that is drawn from a multivariate Gaussian distribution , with a class-specific mean vector and a a common variance matrix.
* The score function (posterior probability) is *linear* in . With for each , it can be converted to the class probability by the *softmax* function
* The , and are estimate the follwing way:
* where is the estimated variance for the -th class.

|  |  |
| --- | --- |
|  | 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 , then it’s QDA. It assumes an observation from the -th class is .The score function has a *quadratic* term
* QDA has much more parameters to estimate compared to LDA (), 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 predictors, , the method is *naive Bayes*, and 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 , one can
  + assume that , that is, a class specific covariance but is diagonal. QDA’s is not diagonal. If we model (Note is shared among clases), In this case NB is a special case of LDA that has a diagonal 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 is qualitative, then one can simply count the proportion of training observations for the 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 is not large enough relative to , 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 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 , while NB is purely additive, in the sense that a function of is added to a function of . 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 is small and the distribution of is approximately normal in each of the classes, then LDA is more stable than logistic regression. Also used when .
* when there are more than two classes, LDA provides low-dimensional views of the data hence popular. Specifically, when there are classes, LDA can be viewed exactly in dimensional plot. This is because it essentially classifies to the closest centroid, and they span a dimensional plane.
* For a two-class problem, the logit of ) by LDA (generative learning) is a linear function in , 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 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 in a -neighborhood.

* KNN dominates LDA and Logistic Regression when the decision boundary is highly non-linear, provided is large and is small. As KNN breaks down when is large.
* KNN requires large , this is because KNN is non-parametric and tends to reduce bias but increase variance.
* When the decision boundary is non-linear but is only modest and 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 is discrete and non-negative, a linear regression model is not satisfactory, even with the transformation of , because does not allow .

* Poisson Regression: typically used to model counts,
* where, . This means that if follows a Poissson distribution, the larger the mean of , the larger its variance. Posisson regression can handle this when variance changes with mean, but linear regression cannot, because it assumes contant variance.

Assume

Then one can use maximum likelihood

to estimate the parameters.

* Interpretation: An increase in by one unit is associated with a change in by a *factor* (percentage) of .

## 4.6 Generalized Linear Models (GLM)

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

* Use predictors to predict . Assume conditional on follow some distribution: For linear regression, assume follows a normal distribution; for logistic regression, assume follows a Bernoulli (multinomial distribution for multi-class logistic regression) distribution; For poisson distribution, assume follows a poisson distribution.
* Each approach models the mean of as a function of using a *linking function* to transform to a linear function.
  + for linear regression
  + for logistic regression
  + for Poisson regression
  + .
  + Gamma regression and negative binomial regression.

## 4.7 Assessment of a classifier

* Confusion matrix
* Overall error rate: equals to
* 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
  + this is equal to , where, FNR is The fraction of positive examples that are classified as negatives
  + percentage of TN detected among all negatives
  + This is equal to , where, False positive rate (FPR): the fraction of negative examples (N) that are classified as positive:
  + 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

## 4.8 Homework:

* Conceptual: 1,2,3,4, 5,6,7,8, 9, 10, 12
* 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 new index, introducing missing values if any index values were not already present.

### 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 given slope.

### 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 coefficient for the missing level equals to the negative ofthe sum of the coefficients of all other variables. In this a coefficient for a level may be interpreted as the differnece from the mean level of response.   
  
#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 available for ‘svd’ and ‘eigen’ solvers.  
  
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 warning that the alg doesnot converge.   
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. Chapter 5: Resampling Methods

Resampling methods are mainly used to estimate the test error by resampling the training set. Two methods: cross-validation and bootstrap. Theses methods refit a model to samples from the training set, in order to obtain additional information (eg. prediction error on the test set, standard deviation and bias of estimated parameters) about the fitted model.

Recall training error in general *dramatically underestimate* the test error, and in general, training error decreases as the model flexibility increases, but the test error shows a characteristic U-curve due to the bias-variance trade-off of the test error.

*Model Assessment*: evaluating a model’s performance *Model selection*: selecting the proper level of flexibility.

## 5.1 how to estimate test error

* use a large designated test set, but often not available.
* make adjustment to the training error to estimate the test error, e.g., Cp statistic, AIC and BIC.
* validation set approach: estimate the test error by *holding out* a subset of the training set, also called a *validation set*.
  + the estimate of the test error can be highly variable, depending on the random train-validation split.
  + Only a subset of the training set is used to fit the model. Since statistical methods tend to perform worse when trained on a smaler data set, which suggests the validation error tends to *overestimate* the test error compared to the model that uses the entire training set.
* K-fold Cross-Validation: randomly divide the data into equal-sized parts . For each , leave out part , fit the model on the remaining parts (combined, of the original traning set), and then evaluate the model on the part . Then repeat this for each , and weighted average of the errors is computed:
* where .

For classification problem, simply replace with the misclassificaiton rate .

The estimated standard error of can be calculated by

The estimated error tends bias upward because it uses only of the training set. This *bias is minimized with (LOOCV)*, but LOOCV estimate has *high variance* due to the high correlation between folds.

* LOOCV: it’s a special case of K-fold CV with . For least squares linear or polynomial regression, the LOOCV error can be computed by
* Where is the leverage statistic of . There is no randomness in the error. The leverage , reflects the amount an observation influences its own fit. The above formula doesn’t hold in genearl, in which case the model has to refit times to estimate the test error.
* for LOOCV, the estimate from each fold are highly correlated, hence their average can have high variance.
* better choice is or for bias-variance trade-off, because large leads to low bias but high variance due to the increased correlation between models. Despite the estimated test error sometimes *underestimate* the true test error, they then to be close to identify the correct flexibility where the test error is minimum.
* Bootstrap: Primarily used to estimate the standard error, or a CI (called *bootstrap percentile*) of an estimate . Repeatedly sampling the training set with replacement and obtain a *bootstrap set* of the *the same size* as the original training set. One can fit a model and estimate a parameter with each bootstrap data set, and then estimate the *standard error* using the estimated parameters by the bootstrap model, assuming there are bootstrap data sets:
  + Note sometimes sampling with replacement must take caution, for example, one can’t simply sample a time series with replacement because the data are sequential.
  + Estimate prediction error: Each bootstrap sample has significant overlap with the original data, in fact, about 2/3 of the original data points appear in each bootstrap sample. If we use the original data set as the validation set, This will cause the bootstrap to seriously *underestimate* the true prediction error. To fix this, one can only use predictions on those samples that do not occur (by chance) in a bootstrap sample.
  + Bootstrap vs. Permutation test: permutation methods sample from an estimated *null* distribution for the data, and use this to estimate -values and *False Discovery Rates* for hypothesis tests.
  + The bootstrap can be used to test a null hypothesis in simple situation. Eg. If , we can check whether the confidence interval for contains zero.

## 5.2 Homework

* Conceptual: 1,2,3,4
* Applied: 5–9, at least one.

## 5.3 Code Gist

### 5.3.1 Python

np.empty(1000) #create an array without initializing  
quartiles = np.percentile(arr, [25, 50, 75])

### 5.3.2 Numpy

c = np.power.outer(row, col) # mesh of row[i]^col[j] power.   
# random choice   
rng = np.random.default\_rng(0)  
alpha\_func(Portfolio,  
 rng.choice(100, # random numbers are selected from arange(100)  
 100, #size  
 replace=True))

### 5.3.3 Pandas

np.cov(D[['X','Y']].loc[idx], rowvar=False) #cov compute corr of variables. rowvar-False: cols are vars.

### 5.3.4 Graphics

### 5.3.5 ISLP and statsmodels

# function that evalues MSE for training a model  
def evalMSE(terms, #predictor variables  
 response, #response variable  
 train,  
 test):  
  
 mm = MS(terms)  
 X\_train = mm.fit\_transform(train)  
 y\_train = train[response]  
  
 X\_test = mm.transform(test)  
 y\_test = test[response]  
  
 results = sm.OLS(y\_train, X\_train).fit()  
 test\_pred = results.predict(X\_test)  
  
 return np.mean((y\_test - test\_pred)\*\*2)  
  
# Compare polynomial models of different degrees  
MSE = np.zeros(3)  
for idx, degree in enumerate(range(1, 4)):  
 MSE[idx] = evalMSE([poly('horsepower', degree)],  
 'mpg',  
 Auto\_train,  
 Auto\_valid)  
MSE  
  
# Estimating the accuracy of a LR model using bootstrap  
  
# Compute the SE of the boostraped values computed by func   
def boot\_SE(func,  
 D,  
 n=None,  
 B=1000,  
 seed=0):  
 rng = np.random.default\_rng(seed)  
 first\_, second\_ = 0, 0  
 n = n or D.shape[0]  
 for \_ in range(B):  
 idx = rng.choice(D.index,  
 n,  
 replace=True)  
 value = func(D, idx)  
 first\_ += value  
 second\_ += value\*\*2  
 return np.sqrt(second\_ / B - (first\_ / B)\*\*2) #compute var.   
def boot\_OLS(model\_matrix, response, D, idx):  
 D\_ = D.loc[idx]  
 Y\_ = D\_[response]  
 X\_ = clone(model\_matrix).fit\_transform(D\_) #clone create a deep copy.   
 return sm.OLS(Y\_, X\_).fit().params  
   
quad\_model = MS([poly('horsepower', 2, raw=True)]) #raw=True: not normalize the feature  
quad\_func = partial(boot\_OLS,  
 quad\_model,  
 'mpg')  
boot\_SE(quad\_func, Auto, B=1000)

### 5.3.6 sklearn

from functools import partial  
from sklearn.model\_selection import \  
 (cross\_validate,  
 KFold,  
 ShuffleSplit)  
from sklearn.base import clone  
from ISLP.models import sklearn\_sm #wrapper to feed a sm model to sklearn  
  
#Cross Validation  
hp\_model = sklearn\_sm(sm.OLS,  
 MS(['horsepower']))  
X, Y = Auto.drop(columns=['mpg']), Auto['mpg']  
cv\_results = cross\_validate(hp\_model,  
 X,  
 Y,  
 cv=Auto.shape[0]) #cv=K.loocv. Can use cv=KFold()object  
cv\_err = np.mean(cv\_results['test\_score']) # test\_score: MSE  
cv\_err  
  
# Use KFold to partition instead of using an integer.   
cv\_error = np.zeros(5)  
cv = KFold(n\_splits=10,  
 shuffle=True,#shuffle before splitting  
 random\_state=0) # use same splits for each degree  
for i, d in enumerate(range(1,6)):  
 X = np.power.outer(H, np.arange(d+1))  
 M\_CV = cross\_validate(M,  
 X,  
 Y,  
 cv=cv)  
 cv\_error[i] = np.mean(M\_CV['test\_score'])  
cv\_error  
  
# using ShuffleSplit() method   
validation = ShuffleSplit(n\_splits=10,  
 test\_size=196,  
 random\_state=0)  
results = cross\_validate(hp\_model,  
 Auto.drop(['mpg'], axis=1),  
 Auto['mpg'],  
 cv=validation)  
results['test\_score'].mean(), results['test\_score'].std()  
  
  
#View skleanrn fitting results using model.results\_  
hp\_model.fit(Auto, Auto['mpg']) # hp\_model is a sklearn model sk\_model.fit(X, Y) for trainning  
model\_se = summarize(hp\_model.results\_)['std err'] #summarize is an ISLP function  
model\_se

### 5.3.7 Useful code snippet

# 6. Chapter 6: Linear Model Selection and Regrularization

Linear models are *interpretable* and often shows small variance. They are fitted by *OLS*. There are other methods that can either provide alternatives or improve linear regression models in terms of *prediction accuracy* (especially when ) and automatic *feature selection*.

There are three classes of methods:

* subset selection: pick a subset of the predictors that best explains the response.
* Shrinkage (regularization). With an added regularizing term, estimated parameters are shrunken to zero relative the OLS estimates. If regularization is used, all coefficients are shrunk toward zero; while if #L^1$ is used, then some coefficients will become zero, leading to actual *variable selection* or *sparse representation*.
* Dimension reduction. Project -predictors to a () dimensional subspace. Each new direction is a linear combination (or projection) of the -variables. These -projections can then be used to fit a linear regression model(**PCR** if the -projections are obtained in an unsupervised way; or **PLR** if thses -projections are obtained in a supervised way))

## 6.1 Best Subset Selecttion

**Algorithm**

1. Fit the data with the *null model* , which contains no predictors. This model simply set .
2. for : fit models containing exactly predictors. Pick the best one that having the smallest RSS or largest on the training set, called . Note for each categorical variable with -level, there are dummy variables.
3. Select the best one among using cross-validation or other measures such as , or adjusted .

Best subset selection suffers - high computation: needs to compute models - overfitting due to the large search space of models

## 6.2 Stepwise selection

Both Forward and Backward selection are stepwise selection. They are used when is large. They searches over models and are *greedy* algorithm and is not guaranteed to find the best possible model out of all models.

* Backward selection requires (so that the full model can be fit);
* Forwarad selection can be used when .

### 6.2.1 Forward Stepwise Selection

Adding one variable at at time that offers the greatest addtional improvement.

**Algorithm**

1. Fit the data with the *null model* , which contains no predictors. This model simply set .
2. for :

* fit all models that augment the predictors in with one additional predictor.
* Pick the best one that having the smallest RSS or largest on the training set, called .

1. Select the best one among using cross-validation or other measures such as , or adjusted .

### 6.2.2 Backward Stepwise Selection

It begins with the full model with all variables, and iteratively removing one variable at at time.

**Algorithm**

1. Fit the data with the *full model* , which contains all predictors.
2. for :

* fit all models that contains all but one of the predictors in .
* Pick the best one that having the smallest RSS or largest on the training set, called .

1. Select the best one among using cross-validation or other measures such as , or adjusted .

## 6.3 Model selection

Models with all predictors always have the smallest or largest on the training set. Therefore they are not suitable to choose the best one among models with different number of predictors.

We ought to *estimate the test error* on a test set. This may be done - indirectly by adjusting the training error to account for the bias due to overfitting: (equivalently, AIC in case of linear model with Gaussian errors), BIC and adjusted . \* Mallow’s :

where, us the number of parameters and .

* AIC
* where, is the maximum likelihood function for the estimated model.
* BIC
* Since for , the BIC places a higher penatlty on models with many variables, and hence select smalller models than .
* Adjusted (larger value is better)
* may increase or decrease depends on . Unlike , adjusted pays a price for the inclusion of unnecessary variables in a model.
* directly by cross-validation (or validation). It doesnot require estimate . It has a wide range of usage, as it may difficult to estimate or . Choose the model that has the smallest test error.

## 6.4 Shrinkage methods for Variable selection

The shrinkage offers an altertivate to selecting variables by adjusting a hyperparameter that trades-off RSS and the model parameter magnitudes. Cross-validation may be used to select the best . After the is selected, one can fit a final model using the entire traning data set.

### 6.4.1 Ridge regression: minimize the following objective

- It encourages the model parameters to shrink toward zero and find a balance between RSS and model parameter magnitudes. Cross-validation is used to find the best tuning parameter . When is large, . Note, Ridge shrinks all coefficients and include all variables.

* The OLS coefficients estimates are *scale equivariant*: regardless of how is scaled, remain the same: if is multiplied by , this will simply leads to be scaled by a factor of .
* In contrast, when multiplying by a factor, this may significantly change the ridge coefficients. Therefore, it is best practice to *standardize the predictors* before fitting a ridge model:
* \*\*\* The Lasso (Least Absolute Shrinkage and Selection Operator)
  + The Lasso replaces the error with penalty. Lasso can force some coefficients to become exactly zero when is large enough. Thus it can actually performs *variable selection*. Agian, cross-validaton is employed to select .
  + The reason Lasso can perform variable selection is because the objective function is equivalent to
* $$, \_{j=1}^p |\_j| s
* $$ for some . The contour of RSS in general only touch the ball at its vertex, at which a minimum is obtained with some variables vanishes. In contrast, the ball is round, and in general, the countour of the RSS function only touches the sphere at a surface point where a minimum is obtained with no variable vanishes.
* Neither ridge nor the lasso will universally dominate the other. When the response is a small number of predictors, one may expect lasso performs better, better in practice, this is never known in advance.
* Combining ridge and lasso leads to *elastic net* method.

## 6.5 Dimension reduction methods: transforming .

There are two types of dimension reduction methods for regression: a) PCA regression, b) Partial list squares PLS.

### 6.5.1 PCA regression: first use PCA to obtain - PCA as linear combinations (directions) of the original predictors:

The first PCA contains the largest variance in , and minimize the sum of squared perpendicular distances to each point (the projection error on the PCA); The second PCA is orthogonal to the first PCA and has the second largest variance and is uncorrelated with the first, and so on. These directions are obtained in an *unsupervised way*, as is not used to obtain these components. Consequently, there is no guarantee that the directions that best explain the predictors will also be the best directions to use for predicting the response.

We then use OLS to fit a linear regression model

After substitute [Equation 6.1](#eq-PCA) into equation [Equation 6.2](#eq-PCAR), one can find that

with

So model [Equation 6.2](#eq-PCAR) is a special case of linear regression subject to the constants [Equation 6.3](#eq-pcar-beta).

### 6.5.2 Partial Least Squares

Similar to PCAR, PLS also first identifies a new set of features , each of which is a linear combinations of the orginnal features, and then fits a linear model via OLS with these new features.

But PLS identifies these new features in a *supervised way*, that is, PLS uses in order to identify the new features that not only approximate the old features well, but also are *related to the response*, i.e., these new features explain both the response and the predictors.

Firt PLS standarizes the predictors. PLS identifies the first component by choosing equals to the coefficient from the simple linear regression of onto . Since this coefficient is equal to , PLS places teh highest weight on the variables that are most strongly related to .

Next, PLS takes the residuals and then repeat the same process.

## 6.6 Code Snippet

### 6.6.1 Python

np.isnan(Hitters['Salary']).sum()

### 6.6.2 Numpy

np.linalg.norm(beta\_hat) #L2 norm. ord=1: L1 ord='inf': max norm.

### 6.6.3 Pandas

Hitters.dropna();  
soln\_path = pd.DataFrame(soln\_array.T,  
 columns=D.columns,  
 index=-np.log(lambdas))  
soln\_path.index.name = 'negative log(lambda)'

### 6.6.4 Graphics

ax.errorbar(np.arange(n\_steps),   
 cv\_mse.mean(1), #mean of each row (model)  
 cv\_mse.std(1) / np.sqrt(K), #estimate standard error of the mean  
 label='Cross-validated',  
 c='r') # color red  
   
ax.axvline(-np.log(tuned\_ridge.alpha\_), c='k', ls='--') # plot a verticalline

### 6.6.5 ISLP and statsmodels

#Estimate Var(epsilon)  
  
design = MS(Hitters.columns.drop('Salary')).fit(Hitters)  
design.terms # to see the variable names in the design matrix  
Y = np.array(Hitters['Salary'])  
X = design.transform(Hitters)  
sigma2 = OLS(Y,X).fit().scale #.scale: RSE: residual standard error estimating   
  
# Forward Selection using ISLP.models and a scoring function  
from ISLP.models import \  
 (Stepwise,  
 sklearn\_selected,  
 sklearn\_selection\_path)  
strategy = Stepwise.first\_peak(design,  
 direction='forward',  
 max\_terms=len(design.terms))  
hitters\_Cp = sklearn\_selected(OLS,  
 strategy,  
 scoring=neg\_Cp)  
 #default scoring MSE, will choose all variables  
hitters\_Cp.fit(Hitters, Y) # the same as hitters\_Cp.fit(Hitters.drop('Salary', axis=1), Y)  
hitters\_Cp.selected\_state\_  
  
#Forward selection using cross-validation  
strategy = Stepwise.fixed\_steps(design,  
 len(design.terms),  
 direction='forward')  
full\_path = sklearn\_selection\_path(OLS, strategy) #using default scoring MSE  
full\_path.fit(Hitters, Y) # there are , 19 variables, 20 models  
Yhat\_in = full\_path.predict(Hitters)  
  
#calculate in-sample mse  
  
mse\_fig, ax = subplots(figsize=(8,8))  
insample\_mse = ((Yhat\_in - Y[:,None])\*\*2).mean(0) #Y[:,None]: add a second axis, create a column vector  
 #[yw] mean(0): calculate mean along row, i.e., for each col. mean(1): calculate mean for each row  
  
#Cross-validation  
K = 5  
kfold = skm.KFold(K,  
 random\_state=0,  
 shuffle=True)  
Yhat\_cv = skm.cross\_val\_predict(full\_path,  
 Hitters,  
 Y,  
 cv=kfold)  
# Cross-validation mse  
cv\_mse = []  
for train\_idx, test\_idx in kfold.split(Y):  
 errors = (Yhat\_cv[test\_idx] - Y[test\_idx,None])\*\*2  
 cv\_mse.append(errors.mean(0)) # column means  
cv\_mse = np.array(cv\_mse).T  
  
#validation approach using ShuffleSplit  
validation = skm.ShuffleSplit(n\_splits=1, # only split one time.   
 test\_size=0.2,  
 random\_state=0)  
for train\_idx, test\_idx in validation.split(Y):  
 full\_path.fit(Hitters.iloc[train\_idx], #note needing to use iloc  
 Y[train\_idx])  
 Yhat\_val = full\_path.predict(Hitters.iloc[test\_idx])  
 errors = (Yhat\_val - Y[test\_idx,None])\*\*2  
 validation\_mse = errors.mean(0)

### 6.6.6 sklearn

rom sklearn.pipeline import Pipeline  
from sklearn.decomposition import PCA  
from sklearn.cross\_decomposition import PLSRegression  
  
#Best subset selection using 10bnb  
  
D = design.fit\_transform(Hitters)  
D = D.drop('intercept', axis=1) #needs to drop intercept  
X = np.asarray(D)  
path = fit\_path(X,   
 Y,  
 max\_nonzeros=X.shape[1]) #fit\_path: a funciton from l0nb. use all variables  
 # max\_nonzeros: max nonzero coefficients in the fitted model.  
  
# Ridge Regression  
soln\_array = skl.ElasticNet.path(Xs, # standardized, no intercept  
 Y,  
 l1\_ratio=0., #ridge  
 alphas=lambdas)  
# Using pipline  
ridge = skl.ElasticNet(alpha=lambdas[59], l1\_ratio=0)  
scaler = StandardScaler(with\_mean=True, with\_std=True)  
pipe = Pipeline(steps=[('scaler', scaler), ('ridge', ridge)])  
pipe.fit(X, Y)  
ridge.coef\_  
  
# Validation  
  
validation = skm.ShuffleSplit(n\_splits=1,  
 test\_size=0.5,  
 random\_state=0) # validation is a generator  
ridge.alpha = 0.01  
results = skm.cross\_validate(ridge,  
 X,  
 Y,  
 scoring='neg\_mean\_squared\_error',  
 cv=validation) # using the strategy defined in validation  
-results['test\_score']  
  
# GridSearchCV()  
param\_grid = {'ridge\_\_alpha': lambdas}  
grid = skm.GridSearchCV(pipe,  
 param\_grid,  
 cv=validation, # or use cv=kfold (5-fold CV defined separately)  
 scoring='neg\_mean\_squared\_error') #default scoring=R^2  
grid.fit(X, Y)  
grid.best\_params\_['ridge\_\_alpha']  
grid.best\_estimator\_  
grid.cv\_results\_['mean\_test\_score']  
grid.cv\_results\_['std\_test\_score']  
  
  
#Plot CV MSE  
ridge\_fig, ax = subplots(figsize=(8,8))  
ax.errorbar(-np.log(lambdas),  
 -grid.cv\_results\_['mean\_test\_score'],  
 yerr=grid.cv\_results\_['std\_test\_score'] / np.sqrt(K))  
ax.set\_ylim([50000,250000])  
ax.set\_xlabel('$-\log(\lambda)$', fontsize=20)  
ax.set\_ylabel('Cross-validated MSE', fontsize=20);  
  
# Use ElasticNetCV()  
ridgeCV = skl.ElasticNetCV(alphas=lambdas, # ElasticNetCV accepts a sequence of alphas  
 l1\_ratio=0,  
 cv=kfold)  
pipeCV = Pipeline(steps=[('scaler', scaler), # scaling is done once.   
 ('ridge', ridgeCV)])  
pipeCV.fit(X, Y)  
tuned\_ridge = pipeCV.named\_steps['ridge']  
tuned\_ridge.mse\_path\_  
tuned\_ridge.alpha\_ # best alpha  
tuned\_ridge.coef\_  
  
# Evaluating test Error of Cross-validated Ridge   
  
outer\_valid = skm.ShuffleSplit(n\_splits=1,   
 test\_size=0.25,  
 random\_state=1)  
inner\_cv = skm.KFold(n\_splits=5,  
 shuffle=True,  
 random\_state=2)  
ridgeCV = skl.ElasticNetCV(alphas=lambdas, # a sequence of lambdas  
 l1\_ratio=0,  
 cv=inner\_cv) # K-fold validation  
pipeCV = Pipeline(steps=[('scaler', scaler),  
 ('ridge', ridgeCV)]);  
   
   
results = skm.cross\_validate(pipeCV,   
 X,  
 Y,  
 cv=outer\_valid,  
 scoring='neg\_mean\_squared\_error')  
-results['test\_score']  
  
# Lasso regression  
lassoCV = skl.ElasticNetCV(n\_alphas=100, #test 100 alpha values  
 l1\_ratio=1,  
 cv=kfold)  
pipeCV = Pipeline(steps=[('scaler', scaler),  
 ('lasso', lassoCV)])  
pipeCV.fit(X, Y)  
tuned\_lasso = pipeCV.named\_steps['lasso']  
tuned\_lasso.alpha\_  
tuned\_lasso.coef\_  
np.min(tuned\_lasso.mse\_path\_.mean(1)) # miminum avg mse  
  
#to get the soln path  
lambdas, soln\_array = skl.Lasso.path(Xs, # standarsized, no -intercept  
 Y,  
 l1\_ratio=1,  
 n\_alphas=100)[:2]  
  
#PCA and PCR  
pca = PCA(n\_components=2)  
linreg = skl.LinearRegression()  
pipe = Pipeline([('scaler', scaler),   
 ('pca', pca),  
 ('linreg', linreg)])  
pipe.fit(X, Y)  
pipe.named\_steps['linreg'].coef\_  
pipe.named\_steps['pca'].explained\_variance\_ratio\_  
  
# perform Grid search  
param\_grid = {'pca\_\_n\_components': range(1, 20)} #PCA needs n\_components >0  
grid = skm.GridSearchCV(pipe,  
 param\_grid,  
 cv=kfold,  
 scoring='neg\_mean\_squared\_error')  
grid.fit(X, Y)  
  
# cross-validation a null model  
cv\_null = skm.cross\_validate(linreg,  
 Xn,  
 Y,  
 cv=kfold,  
 scoring='neg\_mean\_squared\_error')  
-cv\_null['test\_score'].mean()  
  
#PLS  
pls = PLSRegression(n\_components=2,   
 scale=True) # standarsize the data   
pls.fit(X, Y) # X has no-intercept   
  
# Cross-validation  
param\_grid = {'n\_components':range(1, 20)}  
grid = skm.GridSearchCV(pls,  
 param\_grid,  
 cv=kfold,  
 scoring='neg\_mean\_squared\_error')  
grid.fit(X, Y)

# 7. Class Project

**Goal**

to use various ML algorithms to predict a meaningful target by *classification algorithms* or *regression algorithms*. Present it at COS Research Sympsium in the end of April.

**Data set**: real-world stock price and volume data. We could start with just one stock, e.g. APAL, S&P 500, index, DJ index.

**Some ideas**: \* Create one model for each stock. \* create a single model for all stocks. Needs to embed each stock in a feature space. Research?

**Start-up code**: refer to the page <https://github.com/ywanglab/Predicting_stock_movement/blob/main/Time_series_stock_data_analysis_ver2.ipynb>. Perform some EDA to feel the data.

reference ticker symbols: <https://gist.github.com/quantra-go-algo/ac5180bf164a7894f70969fa563627b2>

**Questions**: Which are the variables? price, volume, return, day of week, month of year, etc. What is the variable? next-day price, next-day return, next-five-day average price, next-five-day-average return, etc.

**Models**

* Linear regression:
  + including continuous variables (price, volume), categorical variables (day-of-week, month-of-year)
  + transforming (for including non-linear relation between and ) or (when is heteroschedatic, i.e., with varied )
  + plot residual plot to see if is changing. If yes, may appeal to transforming , e.g., , .
  + investigate outliers (points with unusual -values) using the *residual plot* or looking at *studentized residual*.
  + investigate high leverage points (with unusual values), by calculating *leverage statistics*.
  + Investigate if there is *colinearity* among the variables by calculating *VIF*.
* classification: predicting directions of the stock price movement. binary (with two direction), or multinomial (more than two values: e.g., up, same, down), LDA, QDA
* regularization of the parameters
* selection of variables: forward, backward, mixture, regularization, cross-validation
* decision tree: random forest, boosting
* SVM: support vector machines
* NN

# 8. Summary

In summary, this book has no content whatsoever.

# References

James et al. (2023)

James, G., D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. 2023. *An Introduction to Statistical Learning*. USA: Springer. <https://hastie.su.domains/ISLP/ISLP_website.pdf>.