Regression and Regularization

Generative AI

Reza Rawassizadeh

Classification vs Regression

- Classification is about whether something will happen, whereas regression predicts how much something will happen.

- Regression is used to understand “how a variable’s value changes when values of other corresponding variables change”.

- Nevertheless, the border is blurred, and a machine learning algorithm can belong to both groups. For example, we explain Logistic regression as a regression method, but it is a classification method.

Outline

- Cost/objective function Concepts
- Linear Regressions (linear, polynomial, piecewise regression)
- Non-Linear Regressions (Logistic, Softmax)
- Evaluating Regression Models
- Devils of Model Building (Overfitting and Underfitting)
- Regularization Methods (Ridge, LASSO, ElasticNet)

Outline

- **Cost/objective function Concepts**
- Linear Regressions (linear, polynomial, piecewise regression)
- Non-Linear Regressions (Logistic, Softmax)
- Evaluating Regression Models
- Devils of Model Building (Overfitting and Underfitting)
- Regularization Methods (Ridge, LASSO, ElasticNet, Non-Negative Garrote)

Objective, Cost and Loss

- Anything we do in our life is associated with a cost. For example, you decided to get an MSc and perhaps move to US/Boston. There are costs associated with spending your precious time, staying separated from your family, etc.

- You may have decided or will decide to get married. The cost is your free time, which will be dedicated to another person. Also, to have a successful marriage, we (and everybody else) should make some changes to our behaviors as well, and it is impossible to keep the same behavior as we were single.

Objective Function

- A mathematical function that *tries to maximize or minimize a variable* (e.g. accuracy of an algorithm, battery use of an algorithm, …) is called an **objective function**.

- In the context of machine learning, we can call any function an objective function, if the model is trying to optimize this function.

Cost Function / Loss Score

- If the objective function seeks to minimize the cost of a variable (where the higher its value, the more its cost). We may refer to this objective function as the **cost function.**

- **Loss score** is a score that defines the cost of a *single data point* and measures the cost of that single data point.

- In layman's terms, we can say loss functions measures how many mistakes we make.

- The cost function measures the sum of the losses (mistakes) of all data points, but the loss function focuses on the costs of a single data point.

- In simple words, the differences between actual output values (y_{actual}) and predicted output values ($y_{predicted}$), specify the cost. One single difference is the loss, and all loss values are constructing the cost.

Epoch

- Every iteration through the entire training set for calculating the cost function is called **epoch.** If you encounter a text that said after 200 epochs, this means that the algorithm reports the result of cost function after 200 times it has iterated the entire training dataset.

Outline

- Cost/objective function Concepts
- **Linear Regressions (linear, polynomial, piecewise regression)**
- Non-Linear Regressions (Logistic, Softmax)
- Evaluating Regression Models
- Devils of Model Building (Overfitting and Underfitting)
- Regularization Methods (Ridge, LASSO, ElasticNet, Non-Negative Garrote)

Correlation Coefficient vs Regression

- Correlation (Pearson, Kendall Tau, Spearman) assigns a score that specifies the relationship between two variables.

- Regression describes how changes in one variable affect the other variable.

Linear Regression

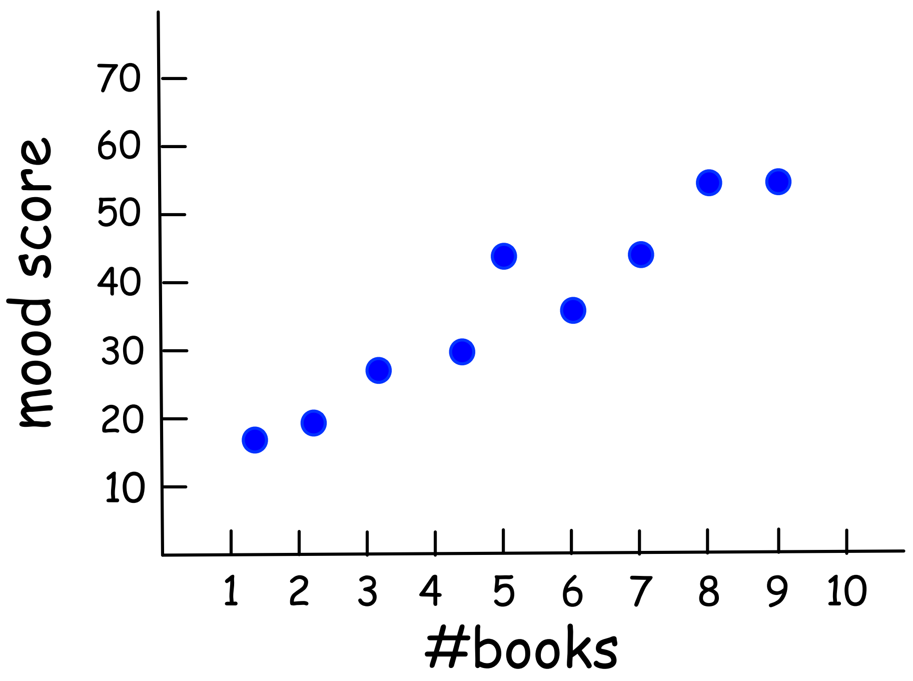
- Linear regression predicts the **quantitative** variable ***Y*** based on the single **predictor** variable ***X*** (or more than one single predictor variable). *Y* is also called a **response**, **dependent,** or **output** variable. *X* is called **predicator**, **input**, **explanatory** or **independent** variable.

- We have two model parameters that a simple linear regression model should identify their best optimal values, i.e. β_0 **(slope)** and β_1 **(intercept)**.

$$\overset{\text{what we want to model}}{Y} = \overset{\text{predictor variable}}{X}\underset{\text{slope}}{\beta_1} + \underset{\text{intercept}}{\beta_0}$$

Linear Regression Example

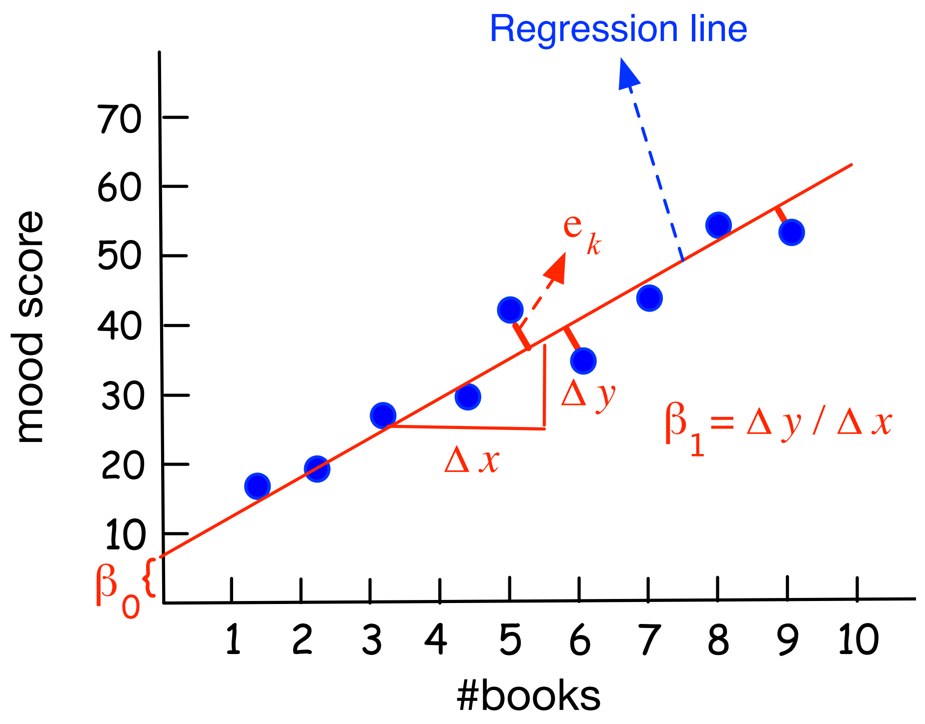
- The objective of linear regression is to find the optimal values for β_0 and β_1. The best value that a cost function of linear regression found is β_0 (intercept) = 7.369 and β_1(slope) = 5.58.



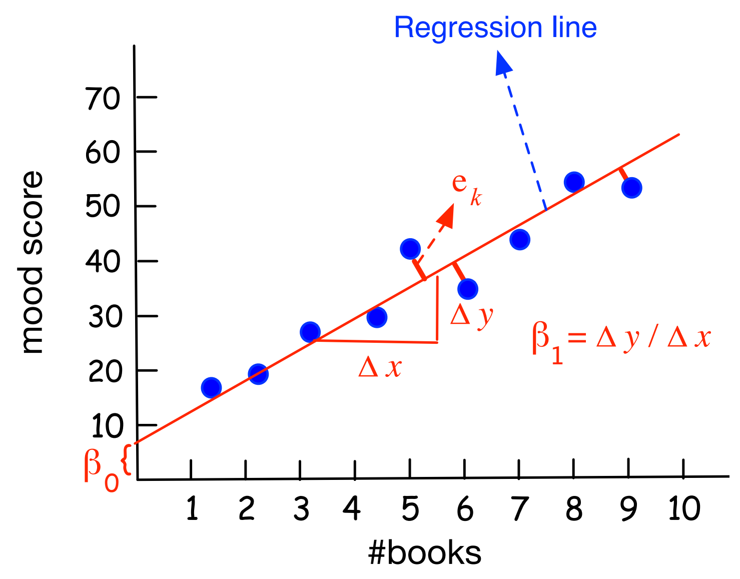
mood score	#books
10	1.5
20	2.1
28	3.2
30	4.2
45	5
38	6
42	7
55	8
55	9

- Therefore, assuming that there is no error, the algorithm can easily predict the mood score achieved by reading 20 books, via replacing these parameters in the linear equation. It will be calculated as follows:

$$mood - score_{(20th-book)} \approx 20 \times 7.369 + 5.58 = 152.96$$



How the cost function identifies β_1(slope) and β_0 (intercept) parameters?



Model Parameter Estimation (Residual Sum of Squares)

- **Residual Sum of Squares (RSS)** is a cost function commonly used in linear regression.
- $RSS = e_1^2 + e_2^2 + \ldots + e_n^2$
- Or it could be written as follows:
  $RSS = (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2 + (y_2 - \hat{\beta}_0 - \hat{\beta}_1 x_2)^2 + \ldots = \Sigma_{i=1}^n (y_i - \hat{y}_i)^2$
- The mean of x is $\bar{x}$ and mean of y is $\bar{y}$, the predicted values are presented with a hat sign "^", e.g. the predicted value of y is presented as $\hat{y}$.
- To calculate the $\hat{\beta}_0$ and $\hat{\beta}_1$ which minimizes RSS we can use the following equation.

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^n (x_i - \overline{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \beta_1 \bar{x}$$

Other Notations of Linear Regression

- To be more accurate, we can write the linear regression equation as follows: $y = \beta_1 x + \beta_0 + e$, assuming that *e* is the mean of error values. $y \approx \beta_1 x + \beta_0 + e$.

- We can also write a linear model as a transpose of matrix *x* (it is a vector) times the vector of coefficients ($\hat{\beta}$), as follows: $y = x^T \hat{\beta}$.

- β_0 can be written as *b* and other intercepts (βs) are vectors of weights *w, i.e.,* $y = wx + b$*.* Because mapping them is easier than the neural network concept.

A Short Summary

Model parameter estimation will be done through a cost function, and not by the user of the algorithm.

Parameters that are defined by the user of the algorithm are referred to as hyperparameters.

Multi Linear Regression

- Most of the time we have multiple input (predictor) variables. In these cases, we can use multiple linear regression to predict the output, i.e., multilinear regression.

$$Y = \beta_0 + x_1\beta_1 + x_2\beta_2 + \ldots + \epsilon$$

or we can summarize the summation as follows:

$$y = \beta_0 + \Sigma_{k=1}^{n} x_k\beta_k + \epsilon$$

Multi Linear Regression Example

- Assume social media activity, physical pain and family issues also contributing to a person's stress and to model them, we can write:

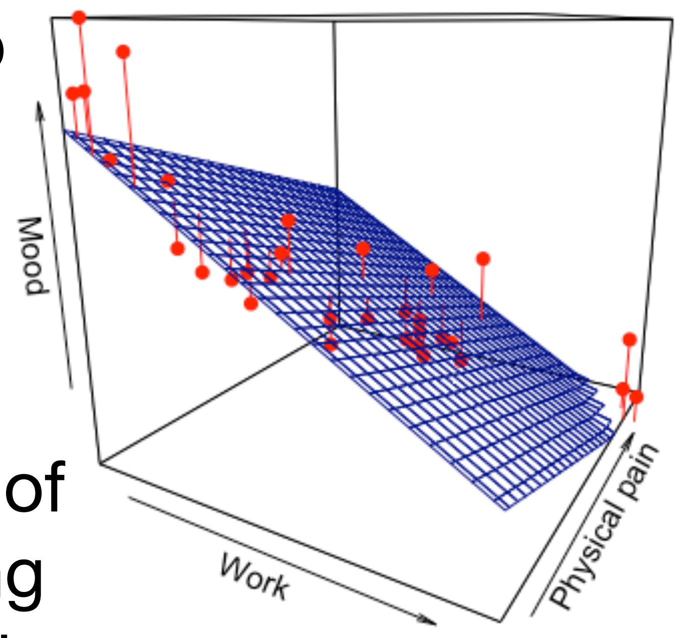
$$Stress = \beta_0 + \beta_1 . work + \beta_2 . social\ media + \beta_3 . physcial\ pain + \beta_4 . family - issues + \ldots + e$$

- While employing multiple linear regression, usually, we need to answer some questions. For example:

- Do all *Xs* (input, independent, or predictor variables) help to explain *Y* (output or dependent variable)?

- how well does our model fit the data?

- To answer these questions we need to rely on the output of our software which implements linear regression, including the *p*-value given for each input variable or F-statistic test.



Deciding About Model Variables

- Assume we build a model with three variables (x_1, x_2, x_3) as follows:
  $Y = \beta_1 \,.\, x_1 + \beta_2 \,.\, x_2 + \beta_3 \,.\, x_3 + \beta_0$

- If the model fails with x_1, x_2, x_3 variables, we can remove x_1 and test another model x_2, x_3. Or we can remove x_2, x_3 and test with x_1, or …

- For *m* number of variables, we could have 2^m different models.

- In some cases, two more parameters of a model together have a very strong impact on the output variable. This phenomenon is called the **synergy effect** or **interaction effect.**

- For example, a popular social media influencer (*x_1*) is advertising a product on popular social media (*x_2*). In this scenario, the popularity of these two variables boosts product sales, and we can extend our model to emphasize their importance by writing $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + e$ instead of $Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + e$ because we want to emphasize the combination of *x_1* and *x_2.*

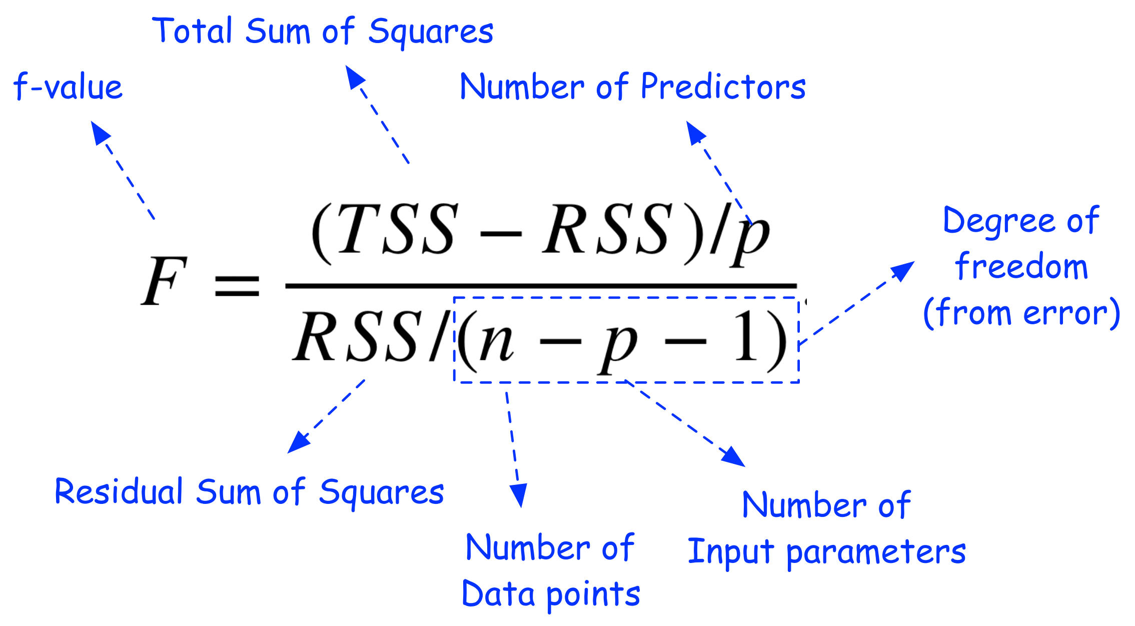
- Usually, we need to experiment with both models (using interaction effect and not using interaction effect). Then, we can compare the *RSS* or other evaluation metrics of these two models and decide to use one for the upcoming data (i.e., test dataset).

F-Statistic

- If there is no relation between input variables, their βs should all be equal to zero. If there is a relation at least of one the βs is not zero.

  - H$_0$ = There is no relation between input and output variables ($\beta_1 = \beta_2 = \ldots = \beta_p = 0$).
  - H$_1$ = There is a relation between input and output variables (at least one $\beta_x \neq 0$).

- This hypothesis test will be performed by the **F-statistic test** or **F-test**, which is written as follows:



$$F = \frac{(TSS - RSS)/p}{RSS/\boxed{(n - p - 1)}}$$

$$TSS = \Sigma_{i=1}^{n}(y_i - \bar{y}_i)^2$$

The *f-value larger than 1, means the null hypothesis is rejected (the alternate hypothesis is correct)*. Otherwise, the *f*-value ≤ 1 means the null hypothesis is correct and thus the model has no predictive capability.

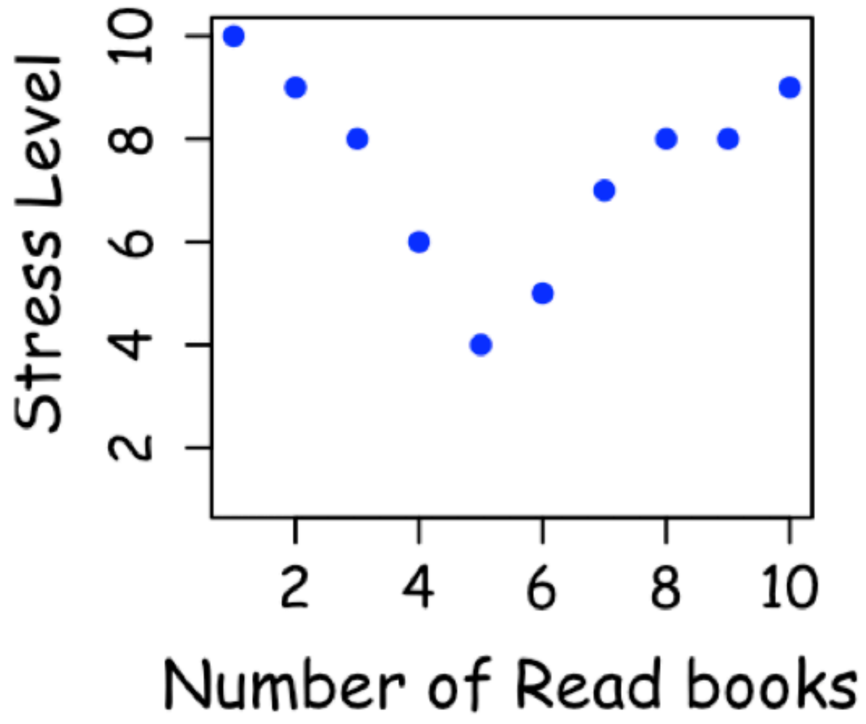
F-Statistic

```
#--------- F-value ---------------
import numpy as np
import scipy

x = [18, 19, 22, 25, 27, 28, 41, 45, 51, 55]
y = [14, 15, 15, 17, 18, 22, 25, 25, 27, 34]
#define F-test function
def f_test(x, y):
    x = np.array(x)
    y = np.array(y)
    f = np.var(x, ddof=1)/np.var(y, ddof=1) #calculate F test statistic
    dfn = x.size-1 #define degrees of freedom numerator
    dfd = y.size-1 #define degrees of freedom denominator
    p = 1 - scipy.stats.f.cdf(f, dfn, dfd) #find p-value of F test statistic
    return f, p



#perform F-test

f_test(x, y)

# The F-test statistic is [first output] and the corresponding p-value is [second output]
```

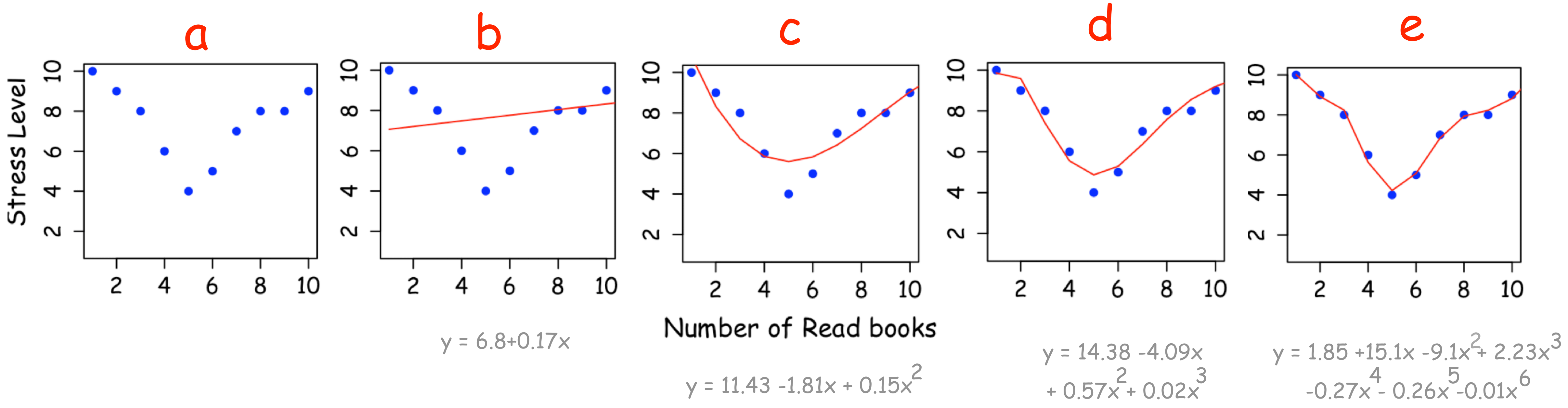
Linear Regression Example

```
#-------- Linear Regression --------
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([10, 20, 28, 30, 45, 38, 42, 55,
55]).reshape((-1, 1))
Y = np.array([1.5, 2.1, 3.2, 4.2, 5, 6, 7, 8,9])
#print (X)
#print (Y)
model = LinearRegression().fit(X, Y)

x_new = np.array([100]).reshape((-1, 1))
#print(x_new)

y_new = model.predict(x_new)
print(y_new)
```

Can we make a linear regression to model this data?



Polynomial Regression



- A series of mathematical terms (coefficient, e.g. βs, multiplied by a variable) which are joined together by addition or subtraction is called a **polynomial.**

- Polynomial regression can produce a non-linear curve*.* The polynomial equation for linear regression is written as follows:

$$y = \beta_0 + x\beta_1 + x^2\beta_2 + x^3\beta_3 + \ldots + \beta_d x^d + e$$

Why do you think Polynomial Regression is not as popular as linear regression?

Why do you think Polynomial regression is as popular as linear regression?

Assuming we have *n* data points and *d* degree, polynomial regression builds $((n + d)!)/(n!\,.\,d!)$ features.

Therefore, because of its computational complexity, we can not increase the degree of polynomial regression generously.

Model Parameters (Degrees and Coefficients) Estimation

- There are two types of parameters required to be identified in polynomial regression.
- First, is the regression “coefficients” $(\beta_0, \beta_1, \beta_2, \ldots)$ and the second one is to identify the minimum number of “degrees” for the model $(x, x^2, x^3, \ldots)$, which need to be identified.

Model Parameters: Coefficients Estimation

- **Coefficients:** A well-known method to estimate polynomial coefficients is to use the **least square** cost function or **least square fitting**. A polynomial regression can be written as a multiplication of matrices, and solving the equation can provide the values for βs.

- For example, let's say we have a four degree polynomial regression. By substituting the values for *x* and *y* (from our real dataset) and by solving the following matrix, we can identify $\beta_0, \beta_1\ and\ \beta_2$ (we learned in school how to resolve three equations that have three unknown variables):

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 \\ \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i^3 & \sum_{i=1}^{n} x_i^4 \end{bmatrix} \cdot \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} x_i y_i^2 \end{bmatrix}$$

Model/Hyper Parameters: Degree of Ploynomiality Estimation

- One way to identify a reasonable degree is to perform the following two steps:

1. Leave out some data points from the dataset and then calculate some polynomial regressions with different degrees, e.g. with x^2, x^3, x^4 and x^5. There is a method called **leave-one-out** and one by one data points will be removed.

2. Add back those removed data points and check how much an error value (e.g. RSS) changes. For example, assume x^2 error = 0.28, x^3 error = 0.23, x^4 error = 0.21 and x^5 error = 0.46. Then we can say x^4 or quadratic polynomial regression is good, because it has the least amount of error while encountering new data.

- This approach is very similar to the test and train that we use in all supervised machine learning, but it is not exactly the same. In this case, we play with the training dataset and not the test dataset.

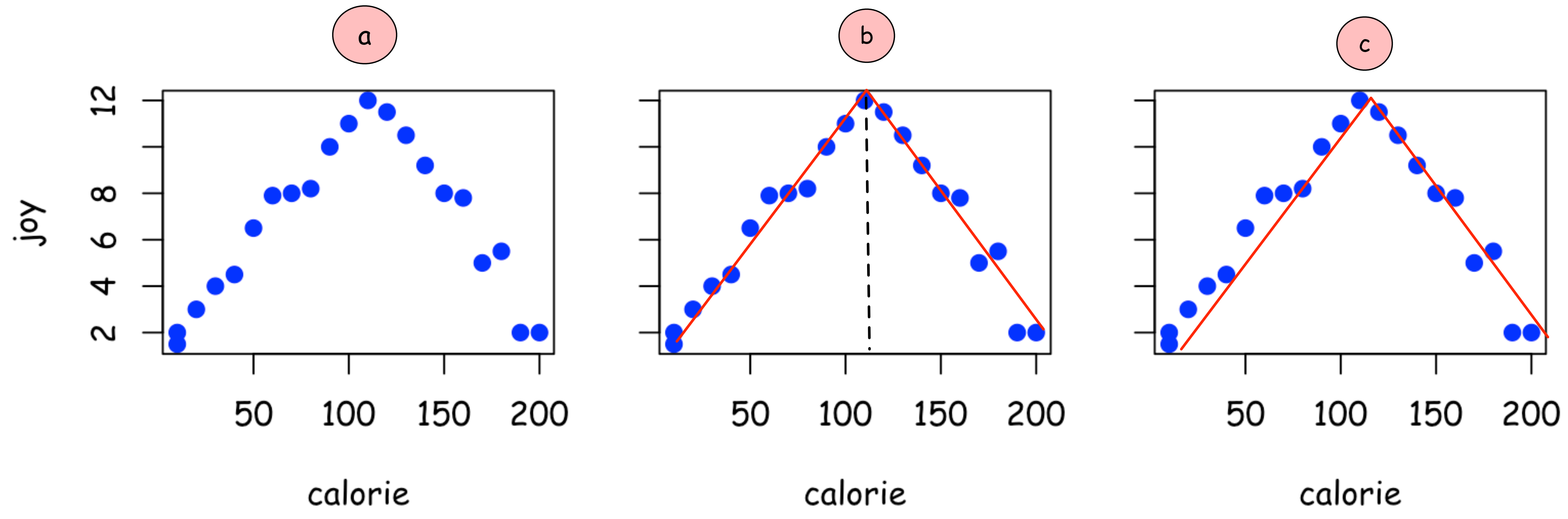
- However, this method is very resource intensive, and it is recommended not to use it.

Polynomial Regression Example

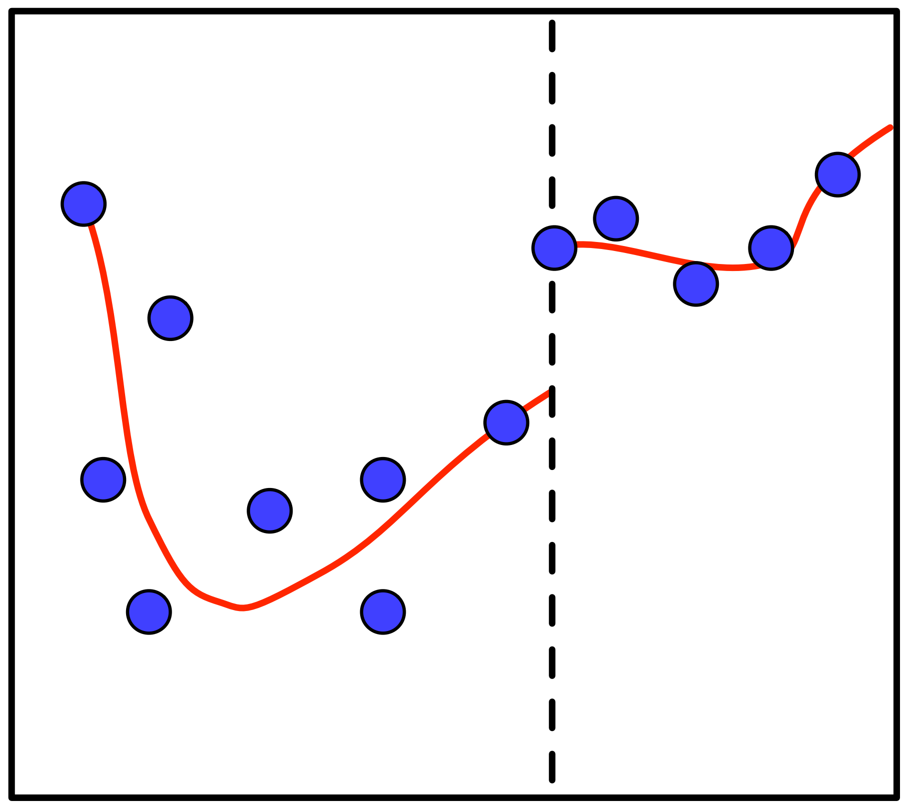
```
# ------- Polynomial Regression ---------
# source: https://www.w3schools.com/python/python_ml_polynomial_regression.asp
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

plt.scatter(x, y)
plt.show()
#-------
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3)) # TODO: change "degree parameter" 3 to 8
and see the result of new regression line
# print model coefficients
print(mymodel.coefficients)

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Piecewise, Segmented, or Non-Additive Regression

Instead of using polynomial regression which is not as efficient as linear regression, we use two linear regressions to get rid of the **additive assumption** of linear regression



Piecewise, Segmented, or Non-Additive Regression



Even sometimes, we can use more than one polynomial regression to model our dataset and it is not just limited to linear regression.

Piecewise, Segmented, or Non-Additive Regression

- Now, the question is “at what point do we need to separate the dataset?”

- We can start by randomly separating the dataset from one data point into two subsets. Then, we draw a linear regression line for each subset and calculate R, SSE or F-statistics, or other evaluation metrics for each of these regression lines.

- We change the random datapoint and use evaluation metrics. The minimum sum of an evaluation metric will be the best discriminatory point to separate the dataset into two subsets and use two linear models.

- For example, assuming among the three following lines (*d_1, d_2, d_3*), *d_2* is the best one because it has the minimum sum of F-statistics.

$$d_1 : f\_statistics_1 + f\_statistics_2 = 3.1 + 2.5$$
$$d_2 : f\_statistics_1 + f\_statistics_2 = 3.2 + 2.1$$
$$d_3 : f\_statistics_1 + f\_statistics_2 = 3.1 + 2.6$$

- The point at which one linear regression is broken and another is started is called **knot**.

- Each of these regression models in a piecewise regression is called **spline**.

Regression Result Evaluation (Fitting a model)

- Evaluating regression algorithms is referred to as **model fitness** or **fitting a model**.

- There are approaches used for evaluating linear models' fitness:

  - **Residual Standard Error (RSE)**

  - Coefficient of Determination **R-squared (R^2)**

  - **Root Mean Square Error (RMSE)** for Polynomial Regression

  - **Mean Square Error (MSE)**

Residual Standard Error (RSE)

- RSE measures the differences between y_i and $\hat{y}_i$ (y_i is the original data point and $\hat{y}_i$ is predicted value which is a data point located on the regression line). As these two variables (y_i and $\hat{y}_i$) get closer, this will result in having better model at the end. Thus, a smaller RSE is better. (*p* is degree of freedom)

$$RSE = \sqrt{\frac{RSS}{n-p-1}} = \sqrt{\frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-p-1}} \quad y_i$$

- RSE measures the differences between y_i , the original data point and $\hat{y}_i$, the predicted value (which is a data point located on the regression line).

Residual Standard Error (RSE)

- RSE is reported with a Degree of Freedom
- The degrees of freedom is equal to the number of data points minus the number of parameters that will be estimated by the model. For example, if our sample size is 12 and our model has 3 parameters. The degree of freedom is $12 - 3 = 9$.
- In the context of regression analysis, a parameter is estimated for every model's variable, and each parameter costs one degree of freedom.
    - Therefore, including lots of variables in a regression model reduces the degrees of freedom available to estimate the parameters' variability.

R²

- **Coefficient of determination or R² (R squared).** RSE depends on the value of *Y,* strongly. Therefore, it is not clear what part of the linear regression equation contributes more to the RSE score.

- *R²* tries to mitigate this challenge by using the **total sum of squares (TSS)**, i.e., $\Sigma_{i=1}^{n}(y_i - \bar{y}_i)^2$ divided by **regression (or residual) sum of squares (RSS)**, i.e. $\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2$. Therefore, *R²* will be written as follows:

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\Sigma_{i=1}^{n}(y_i - \hat{y}_i)^2}{\Sigma_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Mean Square Error and Root Mean Square Error

- **Root Mean Square Error (RMSE):** For polynomial regressions, we can use RMSE to measure the accuracy. It is recommended to do the experiment for several different polynomial degrees and select the lowest RMSE. RMSE is the standard deviation of residuals or prediction errors.

- Residuals are the distance of data points to the regression line.

- Assuming $\hat{y}_i$ as a predicted value and y_i is the *i*th observed data points, the formula is written as follows:

$$RMSE = \sqrt{\frac{\Sigma_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

- When we encounter **Mean Square Error (MSE)** it is the same formula as RMSE, just its square root has been removed. In some implementation libraries, MSE is supported and not RMSE.

RSME, MSE and R^2

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

import numpy as np
# Given values
Y_true = [1,4,5,6,7]  # Y_true = Y (original values)

# predicted values
Y_pred = [0.6,3,4,5.1,5.9]  # Y_pred = Y


print ('MSE:' ,mean_squared_error(Y_true,Y_pred))
rmse = np.sqrt(mean_squared_error(Y_true,Y_pred))
print ('RSME: ',rmse)
print ('R^2: ',r2_score(Y_true, Y_pred))
```

Plotting Residuals

- Sometimes to identify which model, from a set of models, can fit better to our dataset we need to plot their residuals.

- Residuals are useful to identify the linearity of the model. After plotting residuals, we can check if the residuals are linear or not, and we can find a pattern in residuals. If there is a pattern in residuals this is a sign that there is a non-linearity in the dataset.

- Linear regression is amazingly efficient and popular. To enforce linearity in the data we can apply a transformation on the data, e.g., log transformation or L_2 norm transformation.

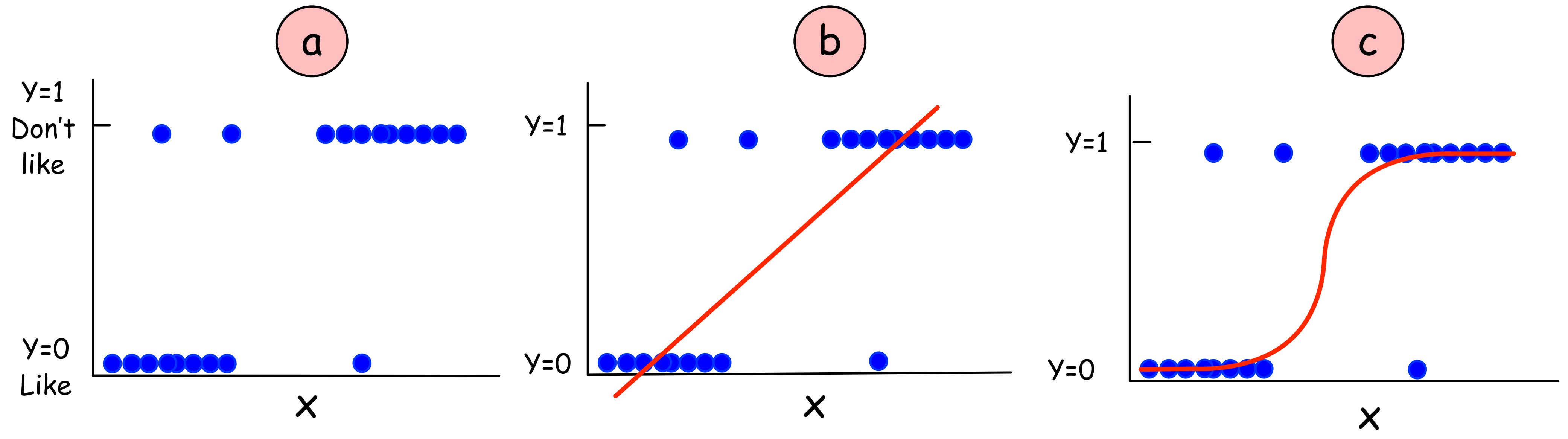
- Another useful application of plotting residuals is plotting them based on time to ensure that errors are not correlated.

- Correlated residuals, based on time, is a phenomenon called **tracking** [James ’13]. If there is tracking exists in our residuals, then we have no guarantee about the confidence of our model, and thus we should think about another model. Tracking phenomena is common in time series analysis.

Outline

- Cost/objective function Concepts
- Linear Regressions (Linear, Polynomial, Piecewise Regression)
- **Non-Linear Regressions (Logistic, Softmax)**
- Evaluating Regression Models
- Devils of Model Building (Overfitting and Underfitting)
- Regularization Methods (Ridge, LASSO, ElasticNet, Non-Negative Garrote)

Logistic Regression



- By using an S-shaped line, as it is shown in Figure c., we can cover more data points than a straight line (linear regression).
- Sigmoid function is used to model S-shape data.

Logistic Regression (Classification)

- Logistic (or Logit) regression is used to *model binary outputs*. It has a binary output (e.g. yes/no, open/close, die/survive), as opposed to linear regression, which has a numeric range.

- This means it is used for classification.

- Logistic regression specifies the probability of a data point belonging to a class or not, e.g., the probability of an email being spam or the probability of not being spam. For example, if the probability of an email being spam is higher than 50%, then that particular email (data point) will be marked as spam.

Sigmoid Function

There is a function called **Sigmoid function** or sigmoid curve. It is written as follows and its output is the S-shaped curve that we have been looking for it in our example.

$$\sigma(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}}$$

For example, $x = 0, \quad x = 1, \quad x = 2, \quad x = 3$, we will have the following

$$\sigma(0) = \frac{1}{1 + e^0} = 0.5,\ \sigma(1) = 0.73,\ \sigma(2) = 0.88,\ \sigma(2) = 0.95.$$

When we write linear regression inside a sigmoid function, it presents a logistic regression, therefore we can write $\hat{y} = \sigma(\beta_0 + x\beta_1)$

- You can plot them yourself, and use more data points, then you will notice that this function returns something between 0 and 1 and its shape is very similar to an S-shaped curve.

Logistic Regression (Classification)

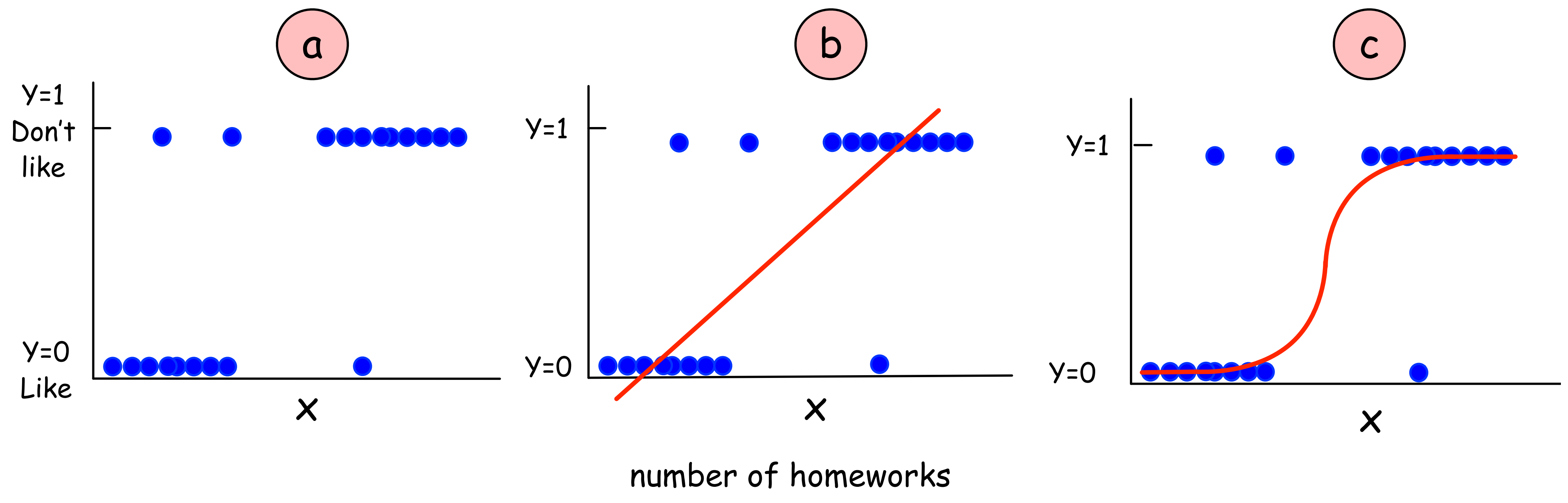
With some mathematical calculations, we can derive the equation of logistic regression as follows:

$$\hat{Y} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

$\hat{Y}$ presents what we want to predict (similar to $\hat{Y}$, in linear regression), and *X* is a set of the predictor (input) variables. In linear regression, we had $Y = \beta_1 X + \beta_0$.

Here we call β_0 a “balance”, but in linear regression, it is called “intercept”.

Logistic Regression Example



- A student is attending a course at the university and due to a pandemic (Covid-19 started in 2019), the grade of the course is based on assignments. She either likes a course or doesn't like it (only two possible classes). If a course has more than 10 assignments, it's very unlikely that she will like it, and she doesn't subscribe to that course. Accordingly, we can formalize the student course-taking procedure as follows:

=> *10 assignments: don't like*

< *10 assignments: like*

Logistic Regression Example

- To summarize, in this example, the output of all of the above questions is provided as a **probability value** of “like” or “don’t like”, and the input variable of the logistic regression is the number of pages.

- $Y = 0$ presents that she likes the course and $Y = 1$ presents that she does not like the course. For example, we give data for her courses (likes and dislikes based on the number of assignments) to a *logit* cost function, which identifies β_0 = 0.4 and β_1 = 0.3 coefficients. Now, we can calculate the estimated probability of her liking a new course, which has 8 assignments, as follows:

- $\hat{Y} = \dfrac{e^{0.4+0.3\times 8}}{1 + e^{0.4+0.3\times 8}} = 0.94$, which is more than 50%, and thus she likes it. Also, the probability of not liking it is: $1 - 0.94 = 0.06$.

Logistic Regression

```
# Source: https://datatofish.com/logistic-regression-python/
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

candidates = {'gmat': 
[780,750,690,710,680,730,690,720,740,690,610,690,710,680,770,610,580,650,540,590,620,600,550,550,570,670,660,580,650,660,
640,620,660,660,680,650,670,580,590,690],
              'gpa': 
[4,3.9,3.3,3.7,3.9,3.7,2.3,3.3,3.3,1.7,2.7,3.7,3.7,3.3,3.3,3,2.7,3.7,2.7,2.3,3.3,2,2.3,2.7,3,3.3,3.7,2.3,3.7,3.3,3,2.7,4,
3.3,3.3,2.3,2.7,3.3,1.7,3.7],
              'work_experience': [3,4,3,5,4,6,1,4,5,1,3,5,6,4,3,1,4,6,2,3,2,1,4,1,2,6,4,2,6,5,1,2,4,6,5,1,2,1,4,5],
              'admitted': [1,1,0,1,0,1,0,1,1,0,0,1,1,0,1,0,0,1,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,1,1,0,0,0,0,1]
              }

df = pd.DataFrame(candidates,columns= ['gmat', 'gpa','work_experience','admitted'])

X = df[['gmat', 'gpa','work_experience']]
y = df['admitted']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)  #in this case, you may choose to set 
the test_size=0. You should get the same prediction here

logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)

new_candidates = {'gmat': [590,740,680,610,710],
                  'gpa': [2,3.7,3.3,2.3,3],
                  'work_experience': [3,4,6,1,5]
                  }

df2 = pd.DataFrame(new_candidates,columns= ['gmat', 'gpa','work_experience'])
y_pred=logistic_regression.predict(df2)

print (df2)
print (y_pred)
```

Model Parameter Estimation

- Similar to linear regression, the objective of logistic regression is to identify β_0 (intercept) and other β parameters that fit the sigmoid line.

- Logistic regression uses a **Maximum Likelihood Estimation (MLE),** to estimate the optimal model parameters.

- Here, the algorithm that implements MLE tries to identify parameter values closest to the output probability (either 0 or 1). In other words, the algorithm tries to find a value for $\hat{\beta}_0$, $\hat{\beta}_1$, …$\hat{\beta}_k$. Then it substitutes them to estimate $\hat{Y}$, which yields a number close to 1 or close to 0.

Model Parameter Estimation

- Assuming θ presents parameters of our model we have $\ell(\theta; X) = log\ L(\theta; X)$. Now, we have $\ell(\beta_0, \beta_1, \ldots \beta_k)$, because here the objective of this MLE is to find the optimal values for Logistic regression model parameters.

- We know that the output of logistic regression is a probability value (between 0 and 1). However, the linear regression output does not have a range limitation. This flexibility in linear regression allows it to identify all linear model parameters easily.

- Therefore, if we make Logistic regression similar to linear regression we can determine its model parameters.

Model Parameter Estimation

- Assuming θ presents parameters of our model we have $\ell(\theta$ … ecause here … s for Log…

- We … bility valu … utput doe … regr … meters. The … regression we can determine its model parameters.

We stated that to solve non-linear models, scientists try to convert them into a linear model and solve it. This is another similar scenario.

We use MLE to transfer the Logistic regression into linear regression and identify optimal parameters for it.

Model Parameter Estimation

- To make Logistic regression similar to linear regression and be able to determine a value for each model parameter, the output variable is transformed from probability to **Logarithm of Odds (Logit)**.

- What is Odds? In statistics, **the likelihood of an event happening is called the “odds” of that event**. Assuming *P* is the probability, the odds is calculated as follows:

$$odds = \frac{p(event)}{1 - p(event)} = \frac{probability \quad of \quad success}{probability \quad of \quad failure}$$

- Odds is not probability; odds is a ratio of an event happening to that event not happening. For example, the probability of rolling a dice and getting the number 6 is $1/6$, but the odds is $1/6 \div 5/6 = 1/5$, and 5/6 here presents the probability of not getting 6 (failure).

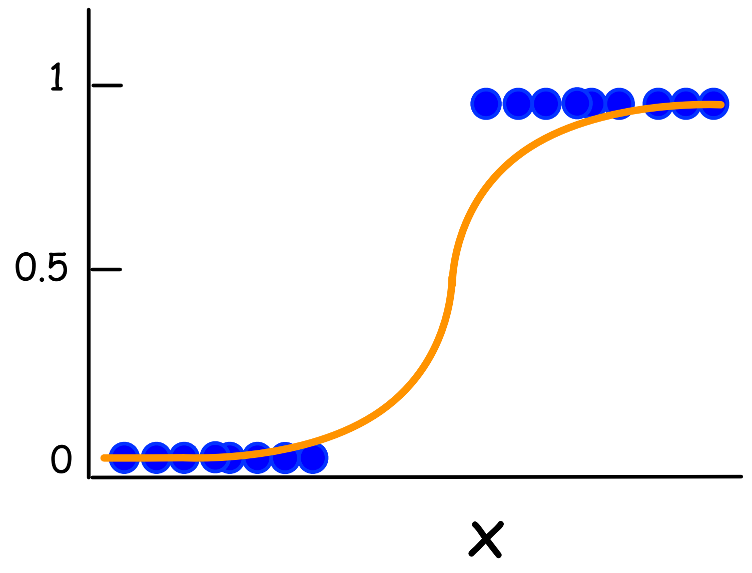
- A **Logit** function, which is the **logarithm of odds** or **Log-Odds** is written as follows, for the sake of simplicity, we write *p* instead of $p(event)$:

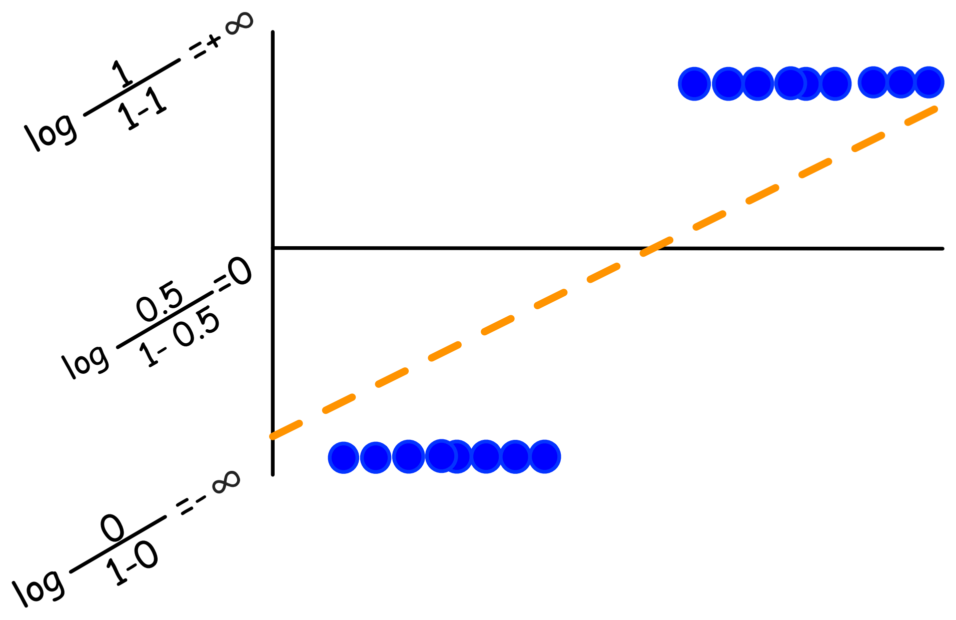
$$log(odds) \ = log(\frac{p}{1-p}) \text{ or } \ ln(\frac{p}{1-p}) \ \ \text{for } 0 \leq p \leq 1$$

Model Parameter Estimation

- In other words, to transform Logistic regression output into a linear model, the output of Logistic Regression $[(e^{\beta_0+\beta_1 X})/(1+e^{\beta_0+\beta_1 X})]$, which is in probability will be fed into the Logit function.

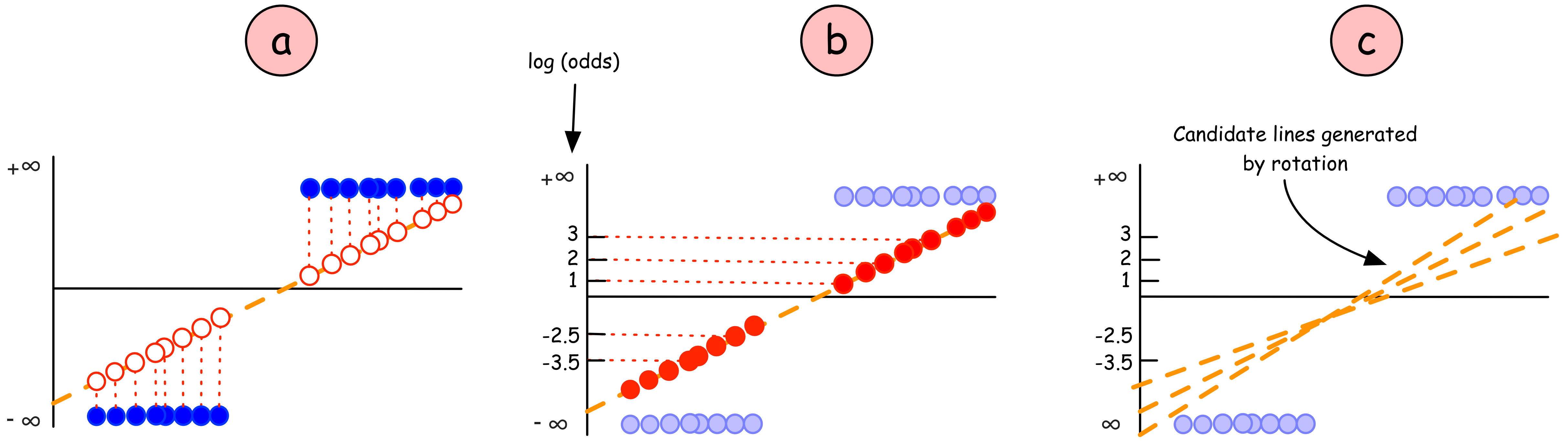
- This makes the output variable of logistic regression (probability value) similar to the output of linear regression (numerical value).





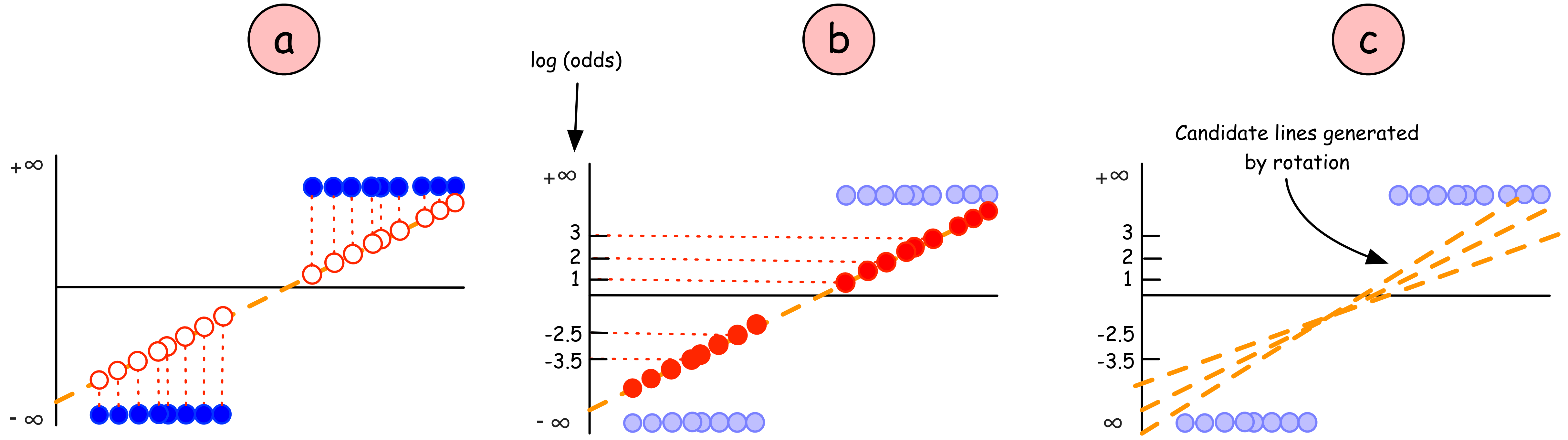
Model Parameter Estimation

After such a transformation, we will have a range between -∞ and +∞, instead of a range between 0 and 1. Now, our data points are located at -∞ and +∞ and still we cannot identify their coordinates. Nevertheless, we can think of an imaginary linear regression line, and project our data points on that line, the orange dotted line in Figure a. and b. present that line.



If we can project our data points on this line, then, their coordinate can be in a range that linear regression can use to calculate θ parameters (we refer to all β parameters as θ while working with MLE approach).

Model Parameter Estimation



Our data points are located at - ∞ and +∞. Therefore, their regression line will start from -infinity to +infinity, and thus we cannot identify model parameters. The MLE can resolve the infinity problem, by projecting the original data points into an imaginary (candidate) line.

These values on Y-axis are *log(odds)* of the original data points. Then, we can transform these *log(odds)* back to probabilities by the following equation:

$$Y = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

Model Parameter Estimation

For example, let's take a look at Figure b. there is one data point its value is -3.5, its projected probability (p) will be calculated as: $y = e^{-3.5}/(1 + e^{-3.5}) = 0.29$.

This is the number that is used by MLE to **sum** all likelihoods together. Projected probabilities are between 0 to 1 classes. If the data point belongs to 0 classes (the probability is less than 0.5) its likelihood will be $1 - p$. If it belongs to the 1 class (its probably is larger than 0.5) its likelihood will be the p.

The likelihood of all data points on a candidate line is the product of all projected values, or log-likelihood of all projected data points is the sum of projected data.

It is better to convert a product into a sum, and the logarithm function converts the products into a sum, e.g. $log(xy) = log(x) + log(y)$. Nevertheless, since the logarithm of numbers smaller than one is negative, the maximum log-likelihood negates the result at the end.

Check the following example:

$$log\ likelihood\ = -(\ldots + 0.73 + 0.56 + 0.92... + (1 - 0.35) + (1 - .42) + \ldots) = 3.21$$

This means that the log-likelihood of this particular candidate line is 3.21. Then, the MLE rotates a bit this candidate line and creates a new candidate line, then it calculates the log-likelihood for the new candidate line, e.g. 2.99, and another rotation and thus another line, e.g. 2.19, and so forth. In the end, the candidate line with the highest likelihood (3.21 in our example) will be used as the best candidate line.

Summary of Model Param. Estimation for Logistic Regression

- (1) Convert logistic regression to log(odds); using log(odds) makes a linear regression out of given logistic regression. However, now we have a problem with the range of the new linear regression. It has a range from $-\infty$ to $+\infty$.

- (2) The MLE resolves the range problem, but it reports a log-likelihood. Its result is negative because the logarithm of numbers between 0 and 1 is negative.

- (3) We use the negative log-likelihood. Therefore the implementation of the MLE experiment with different linear regression lines and choose the one that has the highest negative likelihood.

Any guess how can we handle more than two classes of data with Logistic regression?

Softmax Regression (Classifier)

- By using logistic regression, we have one output variable that is binary (e.g. die/survive, like/dislike, true/false, etc). To handle more than two outputs, we use **multinomial logistic regression**, **multi-class logistic regression,** or **polytomous regression.**

- Four known approaches are being used for the multinomial logistic regression, including the **baseline logit model**, **adjacent category logit**, **proportional odds cumulative logit**, and **Softmax regression.**

- We describe **softmax regression** or **maximum entropy regression**, which is the most important one.

Softmax Regression (Classifier)

- Softmax regression is a generalization of logistic regression for *K* categories, so instead of two binary probabilities, we have *K* different probabilities.

- For example, the transportation mode could “walk”, “bike”, “public transport” & “private vehicle” (*K=4*), or a medical treatment type could be “surgical”, “pharmaceuticals”, “diet” and “none” (*K=4*).

- The output variable of softmax regression is shown as $Y^{(i)} = \{1,2,...,K\}$ (in logistic regression, it was binary, i.e. $Y^{(i)} = \{0,1\}$). The Softmax output is a set of probabilities (values between 0 to 1), and their sum is equal to 1.

- *The softmax regression predicts one output class at a time, it is a multi-class classifier but not multi-output.* Therefore, we can not use it to predict more than one output variable, e.g. we can identify a chicken in a picture, but we can not identify a chicken and a cat in the same image with one softmax.

Softmax Regression (Classifier)

- The logistic regression function (sigmoid function) as follows:
- $\sigma(z)_i = \frac{1}{1 + e^{-(z)}}$ assuming that $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_k x_k = \Sigma_{i=0}^{m} \beta_i x_i = \theta^T x$
- In many resources, $\sigma(z)_i$ is written as $h_\theta(x)$ or $\hat{P}$, which $h_\theta(x)$ stays for a hypothesis based on a given input vector of x and θ (recall that θ presents all parameters of the model).

Softmax Regression (Classifier)

- We see a transpose sign used for θ, why? Because θ is a $1 \times n$ matrix (which is a vector) and x is $1 \times n$ matrix (which is again a vector). Therefore, to multiply these two matrices, we need to multiply $n \times 1$ matrix to $1 \times n$ matrix, and from matrix algebra, we knew that $(1 \times n)^T = (n \times 1)$.

- The output result of softmax regression is a vector of probabilities and their sum is equal to 1.

- In softmax function, *Each output data object depends on the entire vector of input data objects*.

k = number of possible classes

vector of transpose of parameters (θ) multiplied by input values (x)

$$P(y=j|x,\theta)= \frac{e^{\theta^T x_j}}{\Sigma_{i=1}^{k} e^{\theta^T x_k}}$$

A vector of what we want to predict

$$\hat{P}_k \; or \; h_\theta(x) = \begin{bmatrix} P(y=1\,|\,x,\theta) \\ P(y=2\,|\,x,\theta) \\ \cdots \\ P(y=k\,|\,x,\theta) \end{bmatrix} = \frac{1}{exp(\theta^{(1)T}x) + exp(\theta^{(2)T}x) + \ldots + exp(\theta^{(k)T}x)} \cdot \begin{bmatrix} exp(\theta^{(1)T}x_1) \\ exp(\theta^{(2)T}x_2) \\ \cdots \\ exp(\theta^{(k)T}x_j) \end{bmatrix}$$

Softmax Regression (Classifier)

- Expansion of the equation:
- The output result of softmax regression is a vector of probabilities and their sum is equal to 1.
- In softmax function, *Each output data object depends on the entire vector of input data objects*.

vector of transpose of parameters (θ) multiplied by input values (x)

k = number of possible classes

$$P(y=j|x,\theta)=\frac{e^{\theta^T x_j}}{\Sigma_{i=1}^{k} e^{\theta^T x_k}}$$

A vector of what we want to predict

$$\hat{P}_k \; or \; h_\theta(x) = \begin{bmatrix} P(y=1\,|\,x,\theta) \\ P(y=2\,|\,x,\theta) \\ \cdots \\ P(y=k\,|\,x,\theta) \end{bmatrix} = \frac{1}{exp(\theta^{(1)T}x) + exp(\theta^{(2)T}x) + \ldots + exp(\theta^{(k)T}x)} \cdot \begin{bmatrix} exp(\theta^{(1)T}x_1) \\ exp(\theta^{(2)T}x_2) \\ \cdots \\ exp(\theta^{(k)T}x_j) \end{bmatrix}$$

Softmax Example

- Since it requires training time, and GPU we can not run it in real-time and thus you can check it on Google Colab
- https://colab.research.google.com/drive/1I_3Bz3KyNRDYzZs666prh9sJmlxneEwf?usp=sharing

Model Parameter Estimation

- Assuming $X = \{x_1, x_2, \ldots, x_n\}$ are input variables of the model, $\theta = \{\beta_0, \beta_1, \ldots \beta_n\}$ presents all model parameters, and $Y = \{0,1\}$ presents output classes for our input variables, the following equation presents the likelihood function for logistic regression, which we have described:

(i) $L(\theta) = \Pi_{i=1}^{n} p(x_i)(1 - p(x_i))$

The sign Π represents a set of products (multiple multiplications). The goal is to find θ (or $\hat{\beta}$) that maximizes the $L(\theta)$ function. Since there are only two possible variables for *Y* we can write the cost function of logistic regression as follows:

(ii) $L(\theta) = \Pi_{i=1}^{n} p(Y = 1 \,|\, x_i)(p(Y = 0 \,|\, 1 - x_i))$

We can extend the above equation and incorporate more than two *Y* values as follows:

(iii) $L(\theta) = P(Y \,|\, X, \theta) = \Pi_{n=1}^{N} P(y_n \,|\, x_n, \theta)$

Model Parameter Estimation

- We have explained that for the logistic regression, we are using the log-likelihood function. Therefore the described equation can be written as follows. Note that instead of $\ell(\theta)$ we use $J(\theta)$, which is common while using Softmax refer to the cost function as $J(\theta)$:

(iv) $J(\theta) = -\, log\ P(Y\,|\,X, \theta) = -\, \Sigma_{n=1}^{N}\ log\ P(y_n\,|\,x_n, \theta)$

It is better to convert a product into a sum and the *logarithm function converts the products into a sum*, e.g.

$$log(xy) = log(x) + log(y)$$

We use the negative value for $J(\theta)$ as a function for model parameter estimation, which is called the **negative log-likelihood** or **cross-entropy loss**. Because the logarithm of any variable smaller than one is negative, we use the negative sign at the end to make it positive.

Since **cross-entropy** is a cost function used to measure the accuracy of the softmax regressions, our goal should be minimizing the cross-entropy loss.

Assuming we have ***M* different output classes (in logistic reg. we have only two)**, and ***N* number of input variables,** the cross-entropy cost function is written as equation (v). This equation is a generalization of equation (i) for more than two output variables:

(v) $J(\theta) = -\, \Sigma_{j=1}^{M} \Sigma_{i=1}^{N}\ y_{ij}\ log(p_{ij})$

The cross-entropy objective function is to reduce the cross-entropy. Thus a model that has a smaller cross-entropy is favored over the model that has a larger cross-entropy.

Outline

- Cost/objective function Concepts
- Linear Regressions (linear, polynomial, piecewise regression)
- Non-Linear Regressions (Logistic, Softmax)
- **Evaluating Regression Model Fitness**
- Devils of Model Building (Overfitting and Underfitting)
- Regularization Methods (Ridge, LASSO, ElasticNet, Non-Negative Garrote)
- Optimization Methods (Algebra required for Optimization, Gradient Descent and Newton Raphson)

Model Fitness Evaluation

- The output of softmax regression is *k* number of distinct classes. The output of logistic regression is a classification result, which could be 0 or 1, unlike linear and polynomial regression there is no traditional residual existed to calculate the accuracy.
- *k*-fold cross validation
- Learning Curve
- ROC Curve
- Information Criterion

k Fold Cross Validation

- To implement *k*-fold cross validation, we should make *k* subsets of the dataset, one set will be used as testing and the other *k-1* sets will be used as training. We redo this process *k* times, and in each iteration, we change the testing set to another subset. Then we evaluate the accuracy of each iteration, next we take an average of these accuracies and report the final accuracy.

		Predicted	
		0	1
Actual	0	12	2
	1	0	14

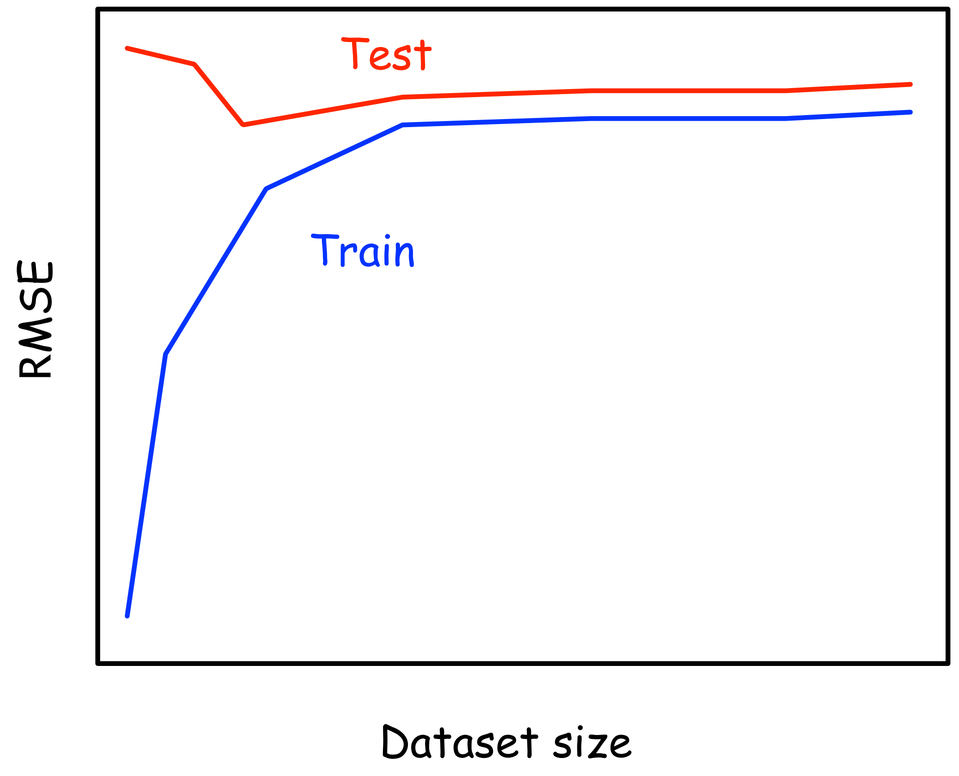
		Predicted			
		a	b	c	d
Actual	a	5	1	0	1
	b	1	7	2	3
	c	0	3	12	1
	d	2	2	1	6

Learning Curve

- A fairly easy approach to determine the optimal model size is to study the error changes with different dataset sizes and experiment the model.

- In the example, the error rate will converge at some point and the model will be stable. There is the place that we can claim the dataset size is stable.

This test is good for dataset size estimation

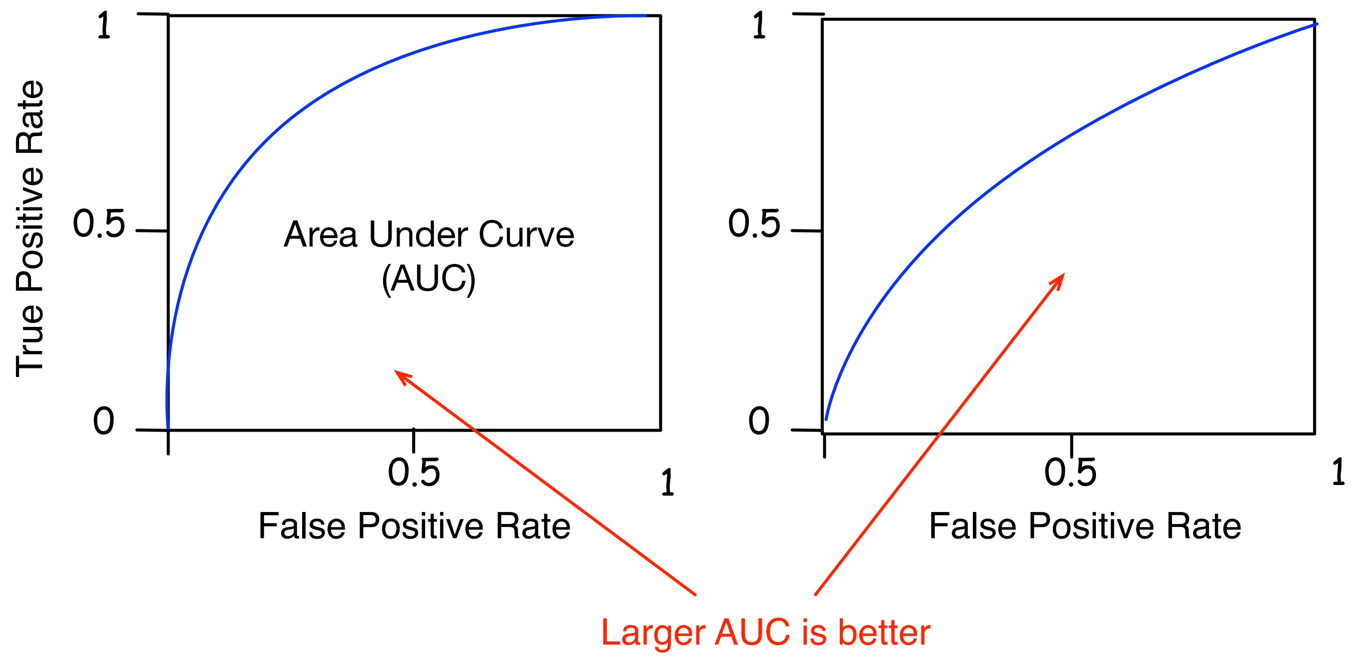


ROC Curve

- **Receiver Operating Characteristic (ROC) curve:** is not limited to the described regressions and it can be used to measure the accuracy of any classification algorithm.

- The ROC curve is plotting True Positive Rate (TPR or sensitivity or recall) against the False Positive Rate (FPR or fall-out), at different threshold settings. Note that *FPR = 1 - Specificity (or True Negative Rate)* as it has been shown in the following:

$$FPR = \frac{FP}{FP+FN} = 1 - \frac{TN}{TN+FP}$$



Information Criterion

- Information criterion methods are used to compare a set of models together and identify the best model among others.

- **Akaike Information Criterion** (**AIC**) [Alkaline '74] and **Bayesian Information Criterion** (**BIC**), which is also called **Schwarz Information Criterion** (**SIC**) [Schwarz '78].

- Both BIC and AIC are based on the law of **Occam's Razor** principle or the law of briefness. It indicates that *we should use only necessary things*.

- Assuming $log(L(\hat{\theta}))$ presents the log-likelihood of the current model params, and $\underline{k}$ <u>is the number of adjustable parameters</u> in that model, AIC is written as follows:

$$AIC = -2(log(L(\hat{\theta}))) + 2k$$

- Assuming we have n number of data points (sample size), it can also be used for linear regressions by using the following equation:

$$AIC = n\ log(\frac{RSS}{n}) + 2k$$

Information Criterion

- The output of AIC is a numerical score, and the lower AIC score is better. Therefore, if we compare the two models' AIC scores, we should choose the model with the lowest AIC score.

- BIC has an advantage over AIC, because it penalizes the number of parameters as a penalty more severely than AIC.

- BIC also penalizes the model by the number of data points as well, while AIC doesn't penalize the number of data points.

- As the number of model parameters increases, the BIC score increases as well, and a good model is the one with the lowest BIC score.

Information Criterion

- Assuming n is the number of data objects, k is the number of parameters, θ is a set of all model parameters and $L(\hat{\theta})$ is the maximum likelihood of the model, BIC is written as follows:

$$BIC = k\ log(n) - 2log(L(\hat{\theta}))$$

Similar to AIC, while comparing BIC scores of several models, the lowest BIC score presents the best model.

For linear types of regressions, we use the following equation:

$$BIC = k\,.\,log(n) + n\,.\,log(\frac{RSS}{n})$$

n is the number of data points

k Fold Crossvalidation

```
#------------ k Fold cross validation ------------------------
# source: https://machinelearningmastery.com/how-to-configure-k-
fold-cross-validation/

# test classification dataset
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=100, n_features=20,
n_informative=15, n_redundant=5, random_state=1)
# summarize the dataset
print(X.shape, y.shape)
```

k Fold Crossvalidation

```
# evaluate a logistic regression model using k-fold cross-validation
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
# create dataset
X, y = make_classification(n_samples=100, n_features=20, n_informative=15, n_redundant=5,
random_state=1)

# prepare the cross-validation procedure
cv = KFold(n_splits=10, random_state=1, shuffle=True)
# create model
model = LogisticRegression()
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

ROC curve

```
# ------------------- ROC curve -----------------------------
# source:  https://www.statology.org/plot-roc-curve-python/
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import matplotlib.pyplot as plt


#import dataset from CSV file on Github
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/default.csv"
data = pd.read_csv(url)

#define the predictor variables and the response variable
X = data[['student', 'balance', 'income']]
y = data['default']

#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)

#instantiate the Logistic Regression model
log_regression = LogisticRegression()

#fit the model using the training data
log_regression.fit(X_train,y_train)

#import dataset from CSV file on Github
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/default.csv"
data = pd.read_csv(url)

#define the predictor variables and the response variable
X = data[['student', 'balance', 'income']]
y = data['default']
```

ROC curve

```
#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.3,random_state=0)

#instantiate the model
log_regression = LogisticRegression()

#fit the model using the training data
log_regression.fit(X_train,y_train)

# --- Plot ROC
#define metrics
y_pred_proba = log_regression.predict_proba(X_test)[::,1]
fpr, tpr, threshold = metrics.roc_curve(y_test,  y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

AIC

```
# ------- AIC ---------------
# source: https://machinelearningmastery.com/probabilistic-model-selection-measures/
# calculate akaike information criterion for a linear regression model
from math import log
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# calculate aic for regression
def calculate_aic(n, mse, num_params):
    aic = n * log(mse) + 2 * num_params
    return aic

# generate dataset
X, y = make_regression(n_samples=100, n_features=2, noise=0.1)
# define and fit a linear model on all data
model = LinearRegression()
model.fit(X, y)
# number of parameters
num_params = len(model.coef_) + 1
print('Number of parameters: %d' % (num_params))
# predict the training set
yhat = model.predict(X)
# calculate the error
mse = mean_squared_error(y, yhat)
print('MSE: %.3f' % mse)
# calculate the aic
aic = calculate_aic(len(y), mse, num_params)
print('AIC: %.3f' % aic)
```

Outline

- Cost/objective function Concepts
- Linear Regressions (linear, polynomial, piecewise regression)
- Non-Linear Regressions (Logistic, Softmax)
- Evaluating Regression Model Fitness
- **Devils of Model Building (Overfitting and Underfitting)**
- Regularization Methods (Ridge, LASSO, ElasticNet, Non-Negative Garrote)
- Optimization Methods (Algebra required for Optimization, Gradient Descent and Newton Raphson)

Generalized Linear Model

- The ability to match future data with observed data is called **generalization**. Therefore, an umbrella term that can be used for regression algorithms is called Generalized Linear Models (GLM).

- GLM models do not necessarily assume a linear relationship between input and output variables. Instead, GLM models assume there is a linear relationship between output variables (a.k.a. transformed responses), in terms of link function (e.g. logit), and input (a.k.a explanatory) variables, which covers logistic and softmax regressions as well.

- In other words, they incorporate link function as well, and not just linear relationship between input and output).

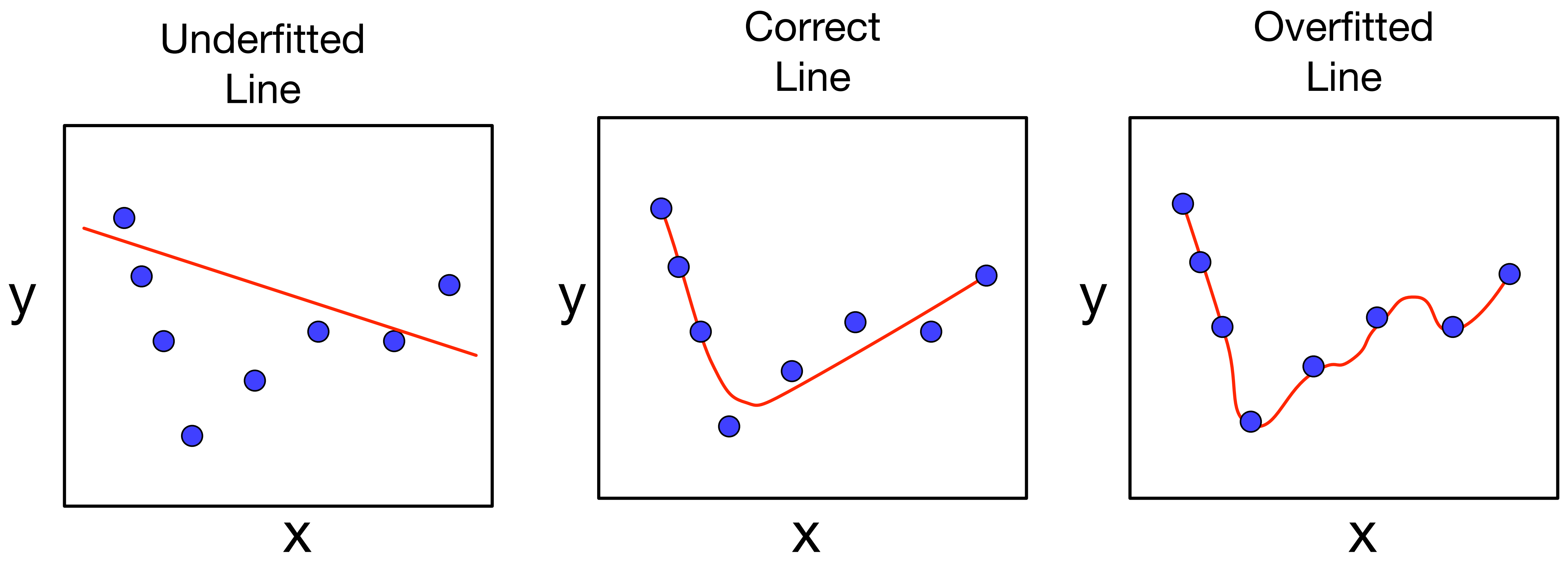
- There are two challenges associated with all GLM methods and also other supervised learning algorithms, underfitting and overfitting.

Overfitting and Underfitting

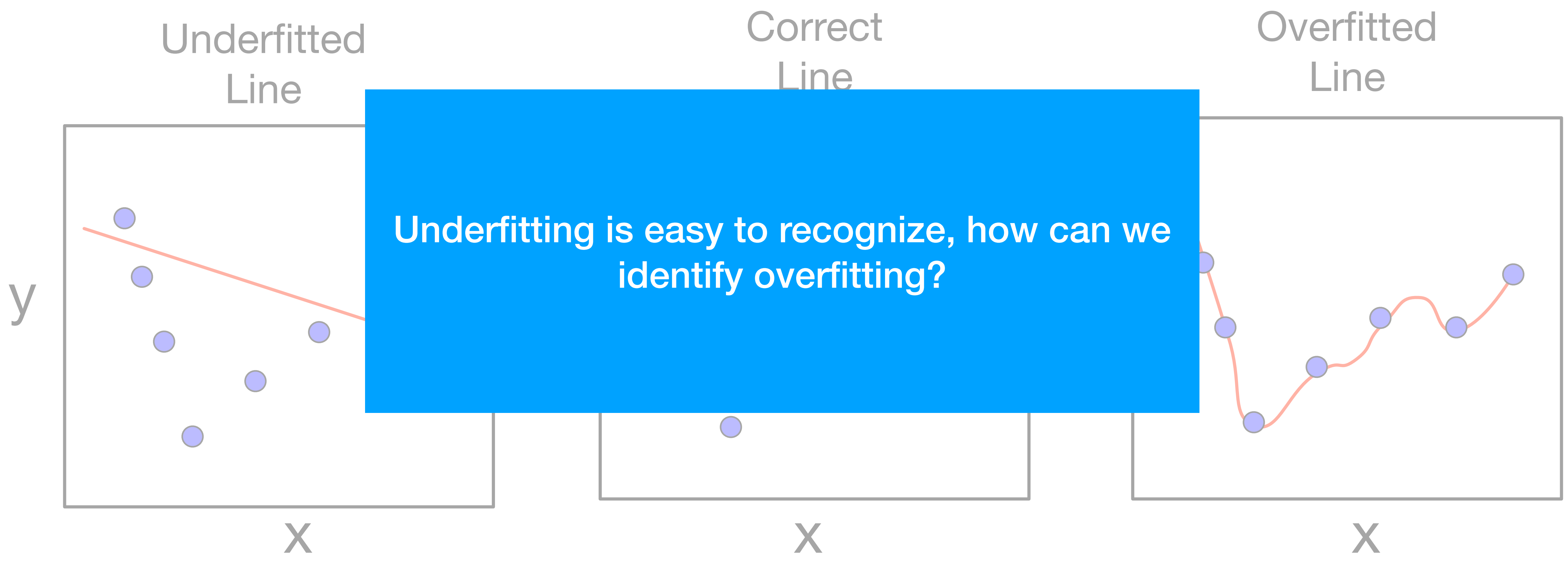
- **Overfitting** refers to creating a model that it performs very well on the data that we have observed (train dataset), but it performs very weakly on the newly arrived data (test dataset). In other words, overfitting makes a too specific model on the training dataset and poor generalization on any new data points (test dataset).

- **Underfitting** refers to creating a model that can neither fit the train nor test data properly. In other words, underfitting refers to *too much generalization*. Therefore, the model will have a poor performance on both our training and test datasets.

Overfitting and Underfitting



Overfitting and Underfitting



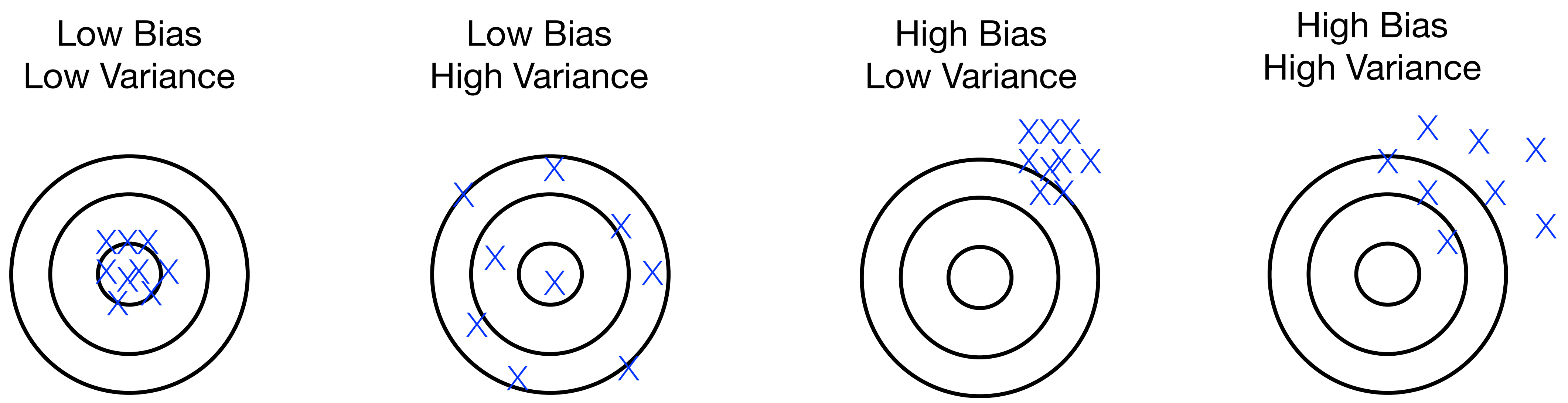
Underfitting is easy to recognize, how can we identify overfitting?

Bias and Variance Tradeoff

- A good approach to deal with overfitting is to decompose it into bias and variance problems.

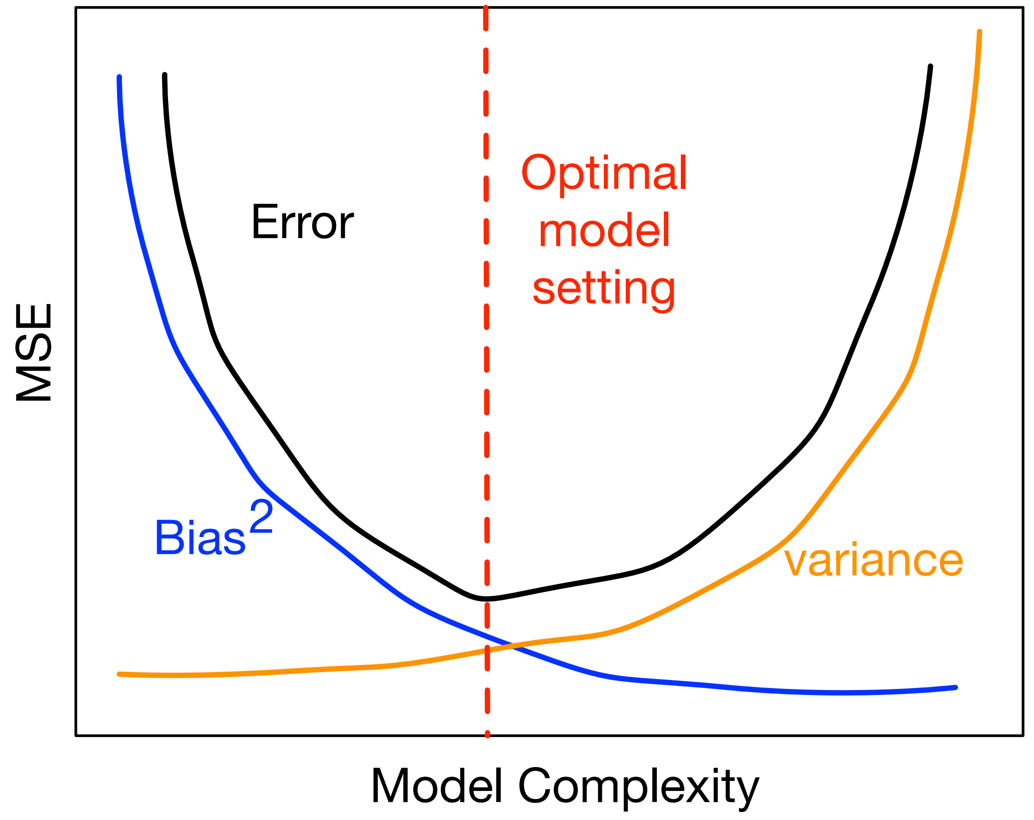
- **Variance** refers to how far our observed data (or training dataset) differs from the mean of the predicted data (output variables). *We need to be sure that changes in the output variables for different training sets are insignificant, i.e. the variety of output variables is low.*

- **Bias** refers to differences between the model output and correct output (what happens in the real world). In other words, bias *presents the differences between the model estimation and the true value*.



Bias and Variance Tradeoff

- Bias is about the Model‘s incorrectness, Variance is varieties of the model‘s outputs.
- A good model has both a low bias and low variance. However, there is a phenomenon known as **Bias-Variance trade off**, which means an increase in one of them, causes a decrease in the other one.



e presents **an irreducible error**, which is an error that can not be reduced by having a good model, and it is always there.

$$Error = \underbrace{bias(\hat{y}_i)^2 + variance(\hat{y}_i)}_{\text{reducible error}} + \underbrace{variance(e)}_{\text{irreducible error}}$$

How to mitigate Bias and Variance

- It is recommended [Grus '19], that to mitigate high bias, we can add more features to the model, and to mitigate high variance, we can remove some features from the model and add more sample data points (not features). It means more flexible models (which cover more features) have higher variance.

- High bias presents underfitting, and high variance presents overfitting. Usually, the more flexible the method is, the less bias will be.

- Underfitting usually causes high bias and low variance. Underfitting can not be resolved by adding more training data points. To resolve the underfitting we should create more complex models, such as adding more parameters.

- A small sample size usually results in high variance, and the simplest approach to mitigate high variance is to increase the sample size. However, usually, the dataset is available before we start building our model, and we can not easily increase the sample size. The data is a valuable source, and usually, we use it with lots of care.

Outline

- Cost/objective function Concepts
- Linear Regressions (linear, polynomial, piecewise regression)
- Non-Linear Regressions (Logistic, Softmax)
- Evaluating Regression Model Fitness
- Devils of Model Building (Overfitting and Underfitting)
- **Regularization Methods (Ridge, LASSO, ElasticNet)**

Why Regularization?

- Due to Occam Razor's principle, we are always looking for less complex (parsimonious) models.
- Besides, we should always consider reducing overfitting, and for these two needs, we can use **regularization**.
- Regularization is defined as *constraining or shrinking a model to reduce the risk of overfitting, on the given training set.*
- It applies to the <u>training phase only</u>, not the testing phase. Besides, we can use regularization to <u>penalize the high power coefficients</u> that make the <u>polynomial model very sensitive to noise</u>.

Again Why Shrinking the Model (Regularization) ?

- There are $(2^n - 1)$ possible linear models from a combination of *n* input variables. For example, assume we have three input parameters x_1, x_2 and x_3, we can have $(2^3 - 1)$ linear models combinations of these parameters: x_1, x_2, x_3, x_1x_2, x_2x_3, x_1x_3, and $x_1x_2x_3$.

- If we have 10 input variables, we can build 2^{10} linear models.

- If we have 100 input variables, which is common in real-world applications, we need a super powerful computer to choose the model from a set of $2^{100} - 1 =$ 1267650600228229401496703205375 models.

Regularization Methods

- Ridge
- LASSO
- ElasticNet

Ridge

- Ridge regression or regularization (**Tikhonov-Miller *regularization***, **L_2** regularization, or **weight decay**) [Tikhonov ‘43] can be used when the model’s *independent variables are highly correlated* (multicollinearity).

- We use it when the sample size is small and we have few training data points.

- It makes the prediction less sensitive to the training data by reducing the variance of the model.

How Ridge Reg. works?

- It penalizes the model based on the distances of predictor coefficients (weights) from zero. The higher the distance between coefficient and zero, the higher will be the penalty. If a coefficient β_k is zero, its predictor will get zero effect on the model as well. Assuming x_j is a model parameter $\beta_k \,.\, x_j = 0.x_j = 0$ and thus we can get rid of x_j.

$$Ridge\ \ Regression = RSS + penalty$$

Ridge Regression Cost Function

$$J(\theta) = RSS(\theta) + \lambda \Sigma_{i=1}^{n} \theta_i^2$$

λ is a hyperparameter used to define the **severity of the penalty** or **tuning parameter**. As λ increases, the flexibility of the ridge regression fit decreases, leading to a decrease in variance but an increase in bias.

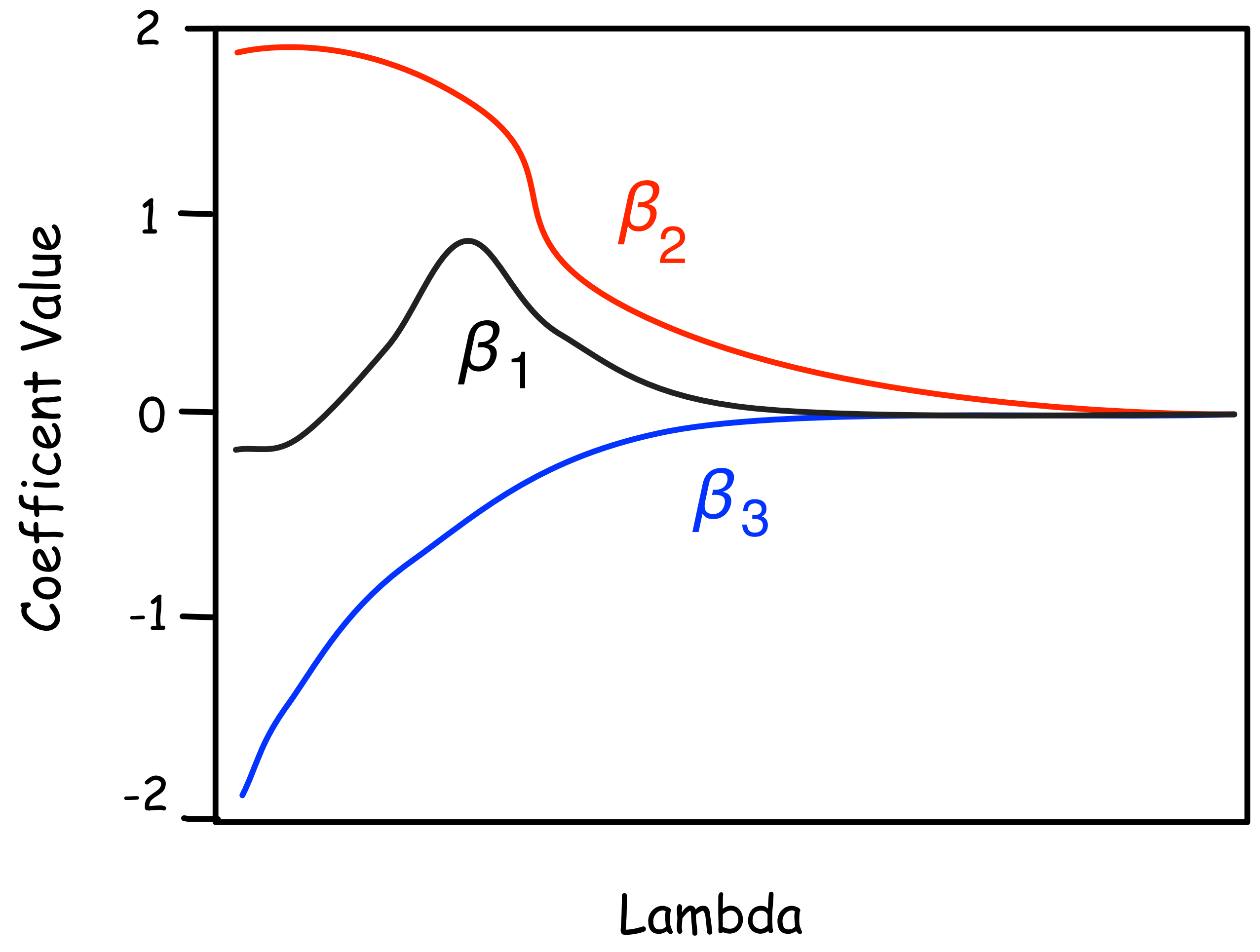
One question might arise now: how can we choose an appropriate λ? Typically, we use 10-fold cross-validation to choose the one that provides the lowest variance. In other words, we test with different λ values until we identify a λ that results in the lowest variance.

Ridge Regression Cost Function

$$J(\theta) = RSS(\theta) + \lambda \Sigma_{i=1}^{n} \theta_i^2$$

λ is a hyperparameter used to define the **severity of the penalty** or **tuning parameter**.

As λ increases, the flexibility of the ridge regression fit decreases, leading to a decrease in variance but an increase in bias.



The objective of Ridge regression is to minimize J(\theta). In some literature, MSE [Géron '19] has been used instead of RSS, both methods are correct.

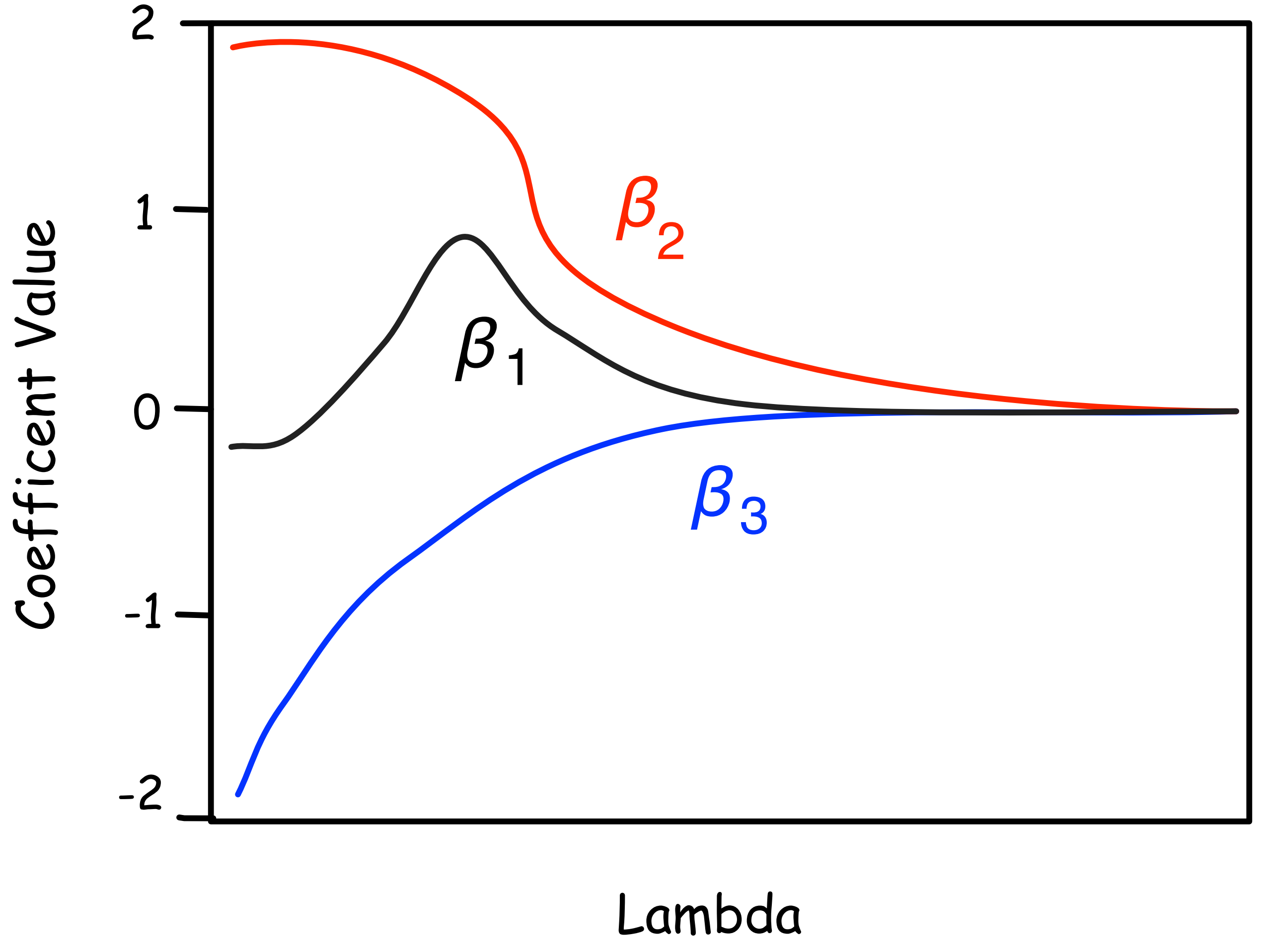
Ridge Regression Cost Function

$$J(\theta) = RSS(\theta) + \lambda \Sigma_{i=1}^{n} \theta_i^2$$

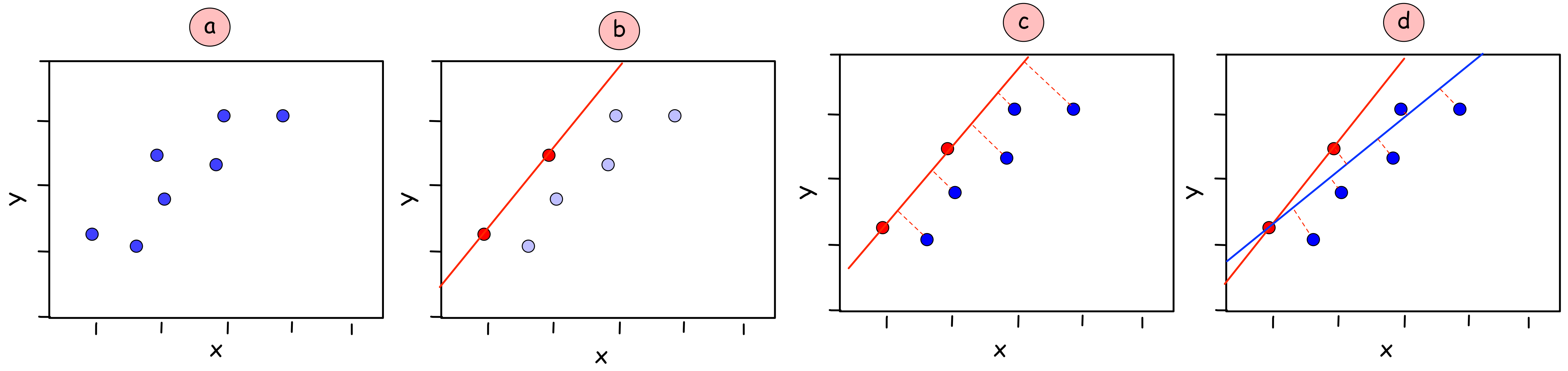
Note that the shrinkage penalty will not be applied on the intercept (β_0), it will be applied on other parameters ($\beta_1, \beta_2, \ldots$), because the intercept measures the mean value of responses, and we do not want to shrink that.

Increasing the λ causes a decrease in the slope. The larger the lambda gets, the prediction for *y* becomes less and less sensitive to *x*.

In other words, the introduction of the penalty moves the regression line more toward the horizon line (parallel to *x* Axis). When all coefficients are set to zero, the regression line will be a horizontal line, parallel to the *x* axis, which obviously is useless because $y = \beta_0 + 0.x_1 + 0.x_2 + \ldots = \beta_0$ and we do not have any predictor in the final model.



Regularization Example



- (a) shows a dataset composed of 7 data points and we would like to use linear regression to make a prediction model for this dataset.
- (b) shows two random data points that have been selected as training marked as red. By calculating their RSS we identify their RSS is zero, which is too low for a bias, and it makes us suspicious of overfitting.
- (c) calculates the RSS for other points (test set), which are marked with blue color. Test set data points (blue dots) are very different from the training set data points (red dots). This is a sign of high variance (overfitting) and we can say that this regression line is not a good model, because we observe overfitting.
- To solve the problem of high variance, we use a Ridge regression and increase in its penalty cause a rotation of the red regression line a bit toward X-axis, as a result, the blue regression line will be created, which has been shown in (d). The blue regression line has a lower variance, despite having a bit more bias. In such a simple approach, Ridge regression resolves the overfitting problem.
- Ridge regression can be applied to logistic regressions and other regression methods as well, but we used a linear regression example for the sake of simplicity.

LASSO (Least Absolute Shrinkage and Selection Operator)

- Ridge regression keeps <u>all coefficients in the final model and does not remove them completely</u>. It means that Ridge reduces all coefficients toward zero but didn’t set any of them exactly to zero.

- It does not exclude predictors that are unrelated to the model, it only reduces the magnitude of their coefficients, which is against Occam’s Razor principle. Therefore, all predictors will stay in the final model, even the unrelated ones, but with a low coefficient.

LASSO (Least Absolute Shrinkage and Selection Operator)

The LASSO regularization resolves this by substituting the Ridge's penalty $\Sigma_{i=1}^{n}\theta_i^2$ (L$_2$ norm) with $\Sigma_{i=1}^{n}|\theta_i|$ or $||\theta_i||_1$ (L$_1$ norm).

We have the following equations for these two regularizations:

$$Ridge\ \ Regression = RSS + \lambda\Sigma_{i=1}^{n}\theta_i^2$$

$$LASSO\ \ Regression = RSS + \lambda\Sigma_{i=1}^{n}|\theta_i|$$

LASSO Summary

- *LASSO shrinks the coefficients to zero, unlike Ridge which shrinks them toward zero but does not remove them completely*. Since LASSO can remove several features from the model, its result model is called a s*parse model.*

- LASSO is better than Ridge, but it is not performing very well for highly correlated predictors and it is also not scale-invariant. Besides, when the number of parameters are larger than the number of training set data points, it does not perform well.

Elastic Net

- Both Ridge and LASSO are useful when we have a good understanding of our parameters.

- However, when our model includes many parameters, like deep learning models which include millions of parameters (due to their black-box nature), it is not possible to know all of the parameters of the model.

- Besides, when the number of our parameters is larger than the available data points for training both LASSO and Ridge do not perform well.

- Elastic Net regression is a combination of Ridge (L_2) and LASSO (L_1) regressions. We write the equation of Elastic Net as follows:

$$Elastic\ Net\ Regression = RSS + \lambda_1 \Sigma_{i=1}^{n} \theta_i^2 + \lambda_2 \Sigma_{i=1}^{n} |\theta_i|$$

Elastic Net

- Both Ridge and LASSO are useful when we have a good understanding of our parameters.

- However, when our model includes many parameters, like deep learning models which include millions of parameters (due to their black-box nature), i … e model.

- Besides, … n the available … not perform well.

λ_1 is used for Ridge and λ_2 is used for LASSO. We should use cross validation to find the best values for both λ_1 and λ_2. By setting $\lambda_1=0$ we will have LASSO regression and by setting $\lambda_2=0$ we will have Ridge regression.

- Elastic Net regression is a combination of Ridge (L_2) and LASSO (L_1) regressions. We write the equation of Elastic Net as follows:

$$Elastic\ Net\ Regression = RSS + \lambda_1\Sigma_{i=1}^{n}\theta_i^2 + \lambda_2\Sigma_{i=1}^{n}|\theta_i|$$

Summary of Regularization

- LASSO shrinks the coefficients to zero, unlike Ridge which shrinks them toward zero but does not remove them completely.

- Regularization should be performed on the training set and not the test set. When the model is trained then we use the non-regularized model to measure the accuracy of our model.

- When the number of our parameters is larger than the available data points for training both LASSO and Ridge do not perform well. Therefore, we go for Elastic Net.

Regularization Example (1/5)

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import matplotlib

from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

Regularization Example (2/5)

```
boston = load_boston()
boston_df = pd.DataFrame(boston.data, columns=boston.feature_names)
print(boston_df.shape)
print(boston_df.columns)

# add another column that contains the house prices which in scikit learn datasets are
considered as target
boston_df['PRICE']= boston.target
print(boston_df.columns)


#print boston_df.head(3)
newX = boston_df.drop('PRICE',axis=1)
#print("--newX----",newX.shape)

newY=boston_df['PRICE']
#print("--newY.shape----",newY.shape)

#print type(newY)# pandas core frame
X_train,X_test,y_train,y_test = train_test_split(newX,newY, test_size=0.3, random_state=3)
#print (len(X_test), len(y_test))
```

Regularization Example (3/5)

```
#----- Linear Regression without any Regularization ------
lr = LinearRegression()
lr.fit(X_train, y_train)

train_score=lr.score(X_train, y_train)
test_score=lr.score(X_test, y_test)
print ('-----------------------')
print ('Linear Reg, train score', train_score)
print ('Linear Reg, test score', test_score)
print ('Linear Reg, coef.', lr.coef_)
print ('Linear Reg, coef.', len(lr.coef_))
```

Regularization Example (4/5)

```
# -------- Ridge Regularization -------

rr = Ridge(alpha=100)
rr.fit(X_train, y_train)

ridge_train_score=rr.score(X_train, y_train)
ridge_test_score=rr.score(X_test, y_test)

print ('Ridge Reg, train score', ridge_train_score)
print ('Ridge Reg, test score', ridge_test_score)
print ('Ridge Reg, coef.', rr.coef_)
print ('Ridge Reg, coef.', len(rr.coef_))
```

Regularization Example (5/5)

```
# -------- LASSO Regularization -------

la = Lasso(alpha=100)
la.fit(X_train, y_train)

lasso_train_score=rr.score(X_train, y_train)
lasso_test_score=rr.score(X_test, y_test)

print ('Lasso Reg, train score', lasso_train_score)
print ('Lasso  Reg, test score', lasso_test_score)
print ('Lasso Reg, coef.', la.coef_)
print ('Lasso Reg, coef.', len(la.coef_))
```