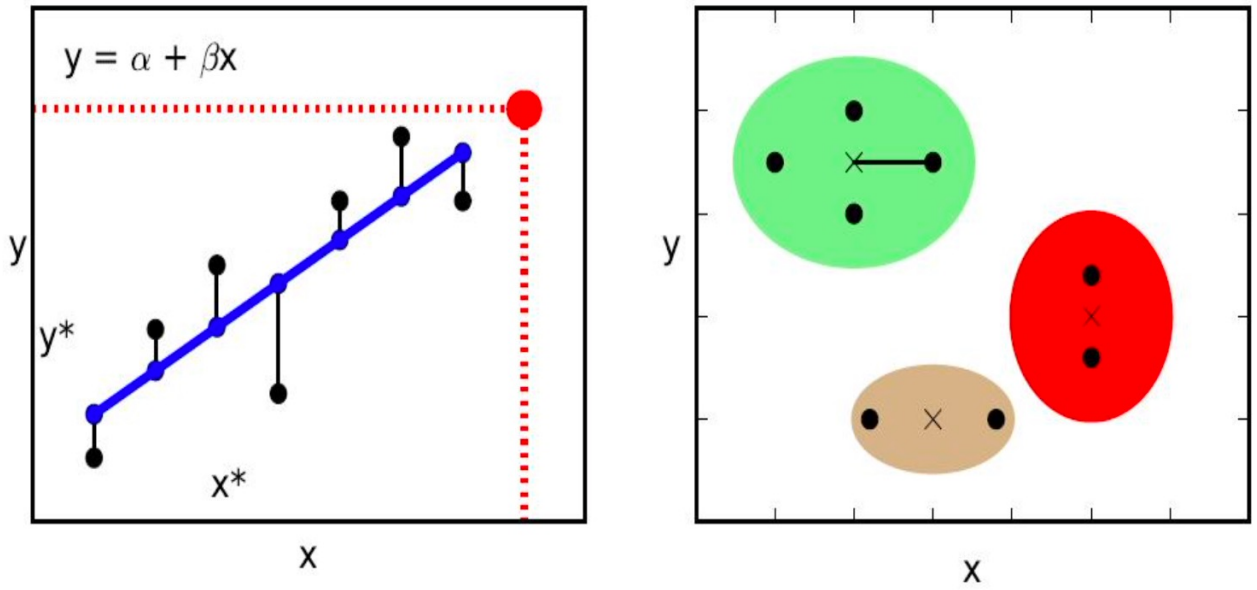
Intro to Classification – Logistic Regression

Data Science with Python CS677

Farshid Alizadeh-Shabdiz, PhD, MBA
Alizadeh@bu.edu
Fall 2021

Logistic Regression

Prediction vs. Classification



- two main goals in machine learning

- Classification is predicting a qualitative output (plus quality of estimate)

Classification Examples

- Face detection and face recognition
- Detecting credit card fraud
- Article classification – politics vs sport vs finance vs etc
- User segmentation
- Object detection

Classification

- Output: Probability of belonging to a class
  - One (1): if it belongs to the class
  - Zero (0): if not belongs to the class

Ex: Logistic Regression

- assume probability P
- define *odds* as

$$\text{odds}(P) = \frac{P}{1-P}$$

- $P = 0.25 \mapsto \text{odds}(P) = 1/3$
- $P = 0.50 \mapsto \text{odds}(P) = 1$
- $P = 0.75 \mapsto \text{odds}(P) = 3/1$
- want to model

$$f(\text{odds}(P)) = ag(x) + b$$

Logistic Function

- Probability is between (0,1)
- Odd function is between (0,∞)
- Log of odd function is between $(-\infty, \infty)$

- Linear function

$$b_0 + b_1 X_1 + \cdots + bnXn$$

- It is between $(-\infty, \infty)$

Main Idea

- use $f = \log()$ as *link* for the *odds*
- use regression

$$\log\left(\frac{P}{1-P}\right) = b_0 + b_1 x$$

$$\frac{P}{1-P} = \exp(b_0 + b_1 x)$$

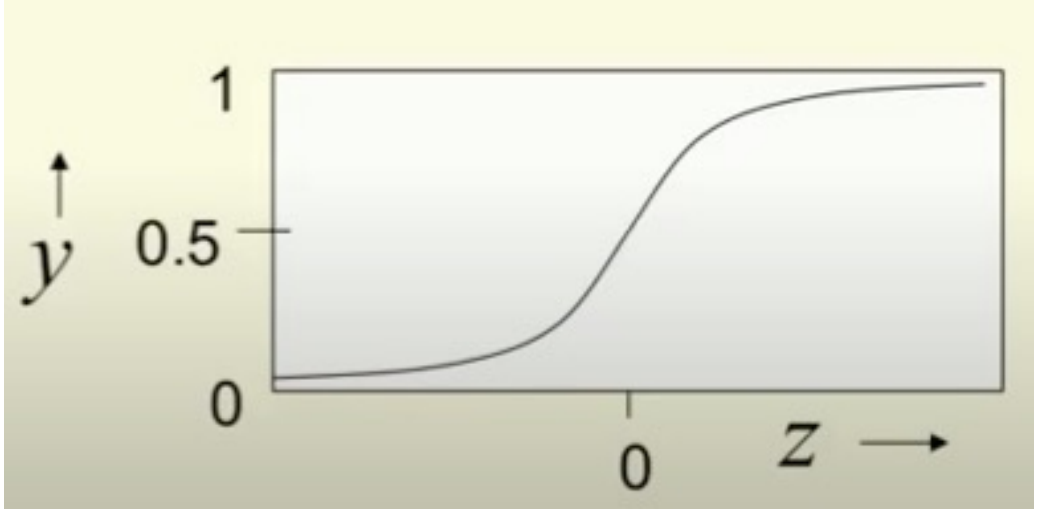
$$P = \frac{\exp(b_0 + b_1 x)}{1 + \exp(b_0 + b_1 x)}$$

- note:

$$\frac{\exp(b_0 + b_1 x)}{1 + \exp(b_0 + b_1 x)} = \frac{1}{1 + \exp(-(b_0 + b_1 x))}$$

Logistic Function

$$y = \frac{1}{1+\exp(-Z)} = \frac{e^Z}{1+eZ}$$



Prediction that a person will default

- Answer of "default" as a function of "balance"

	Coefficient	Std. Error	Z-statistic	P-value
Intercept	-10.6513	0.3612	-29.5	< 0.0001
balance	0.0055	0.0002	24.9	< 0.0001

- Default probability if a person has $1000 balance
- What if the balance is $2000?

Default Probability for Different Balance

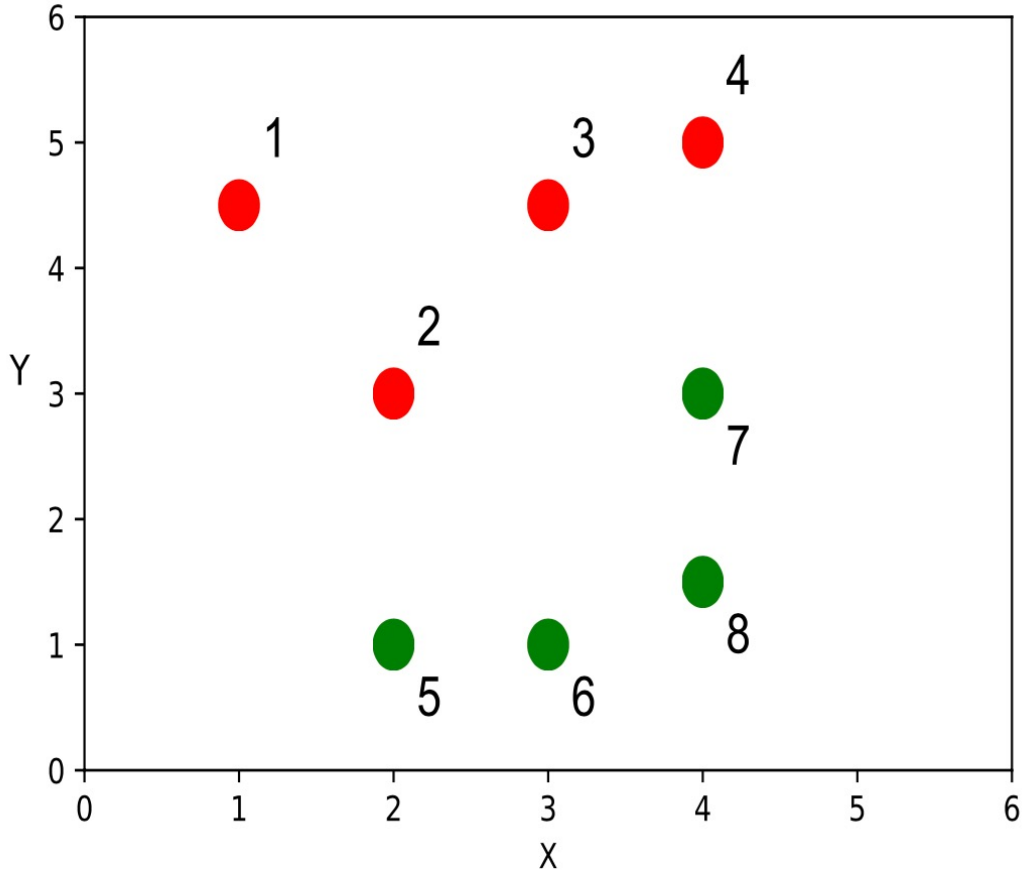
- Default probability if a person' balance = $1000

$$y = \frac{e^{\beta 0 + \beta 1 X}}{1 + e^{\beta 0 + \beta 1 X}} = \frac{e^{-10{\cdot}6 + 0{\cdot}0055 \times 1000}}{1 + e^{-10{\cdot}6 + 0{\cdot}0055 \times 1000}} = 0.006$$

- What if balance is $2000

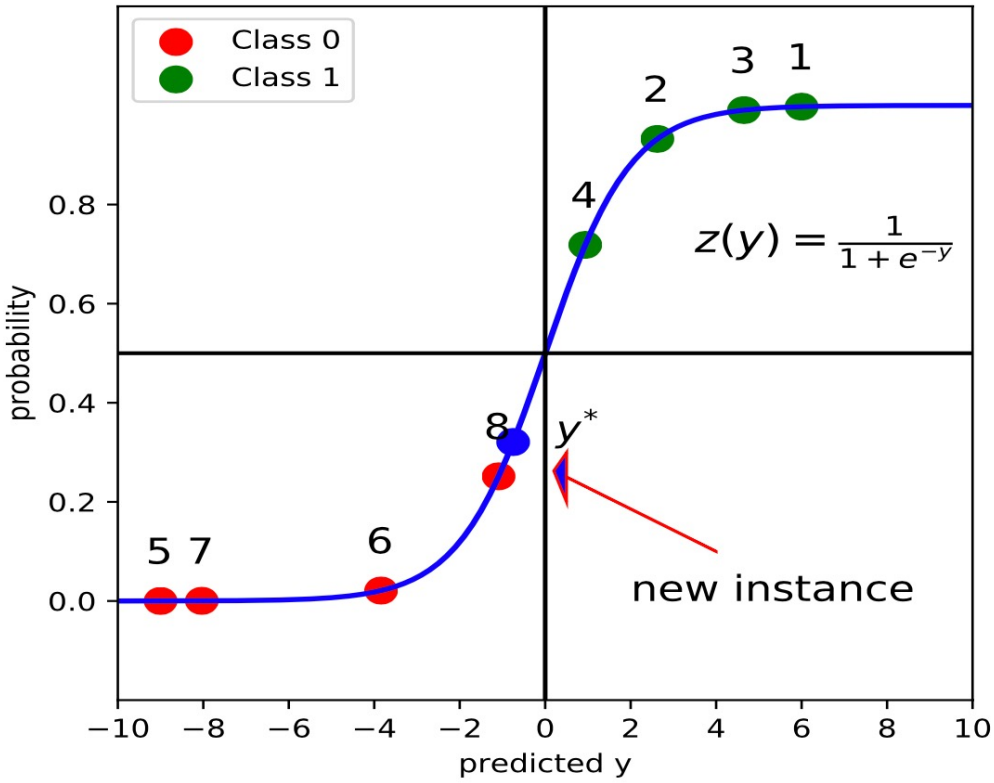
$$y = \frac{e^{\beta 0 + \beta 1 X}}{1 + e^{\beta 0 + \beta 1 X}} = \frac{e^{-10{\cdot}6 + 0{\cdot}0055 \times 2000}}{1 + e^{-10{\cdot}6 + 0{\cdot}0055 \times 2000}} = 0.586$$

Original Dataset



- use *logit*() function

Computing Class Labels



- $z(y^*) < 0.5$ - "red" (class 0)

Example: Numerical Dataset

object x_i	Height (H)	Weight (W)	Foot (F)	Label (L)
x_1	5.00	100	6	green
x_2	5.50	150	8	green
x_3	5.33	130	7	green
x_4	5.75	150	9	green
x_5	6.00	180	13	red
x_6	5.92	190	11	red
x_7	5.58	170	12	red
x_8	5.92	165	10	red

Example Continue

Code for the Dataset

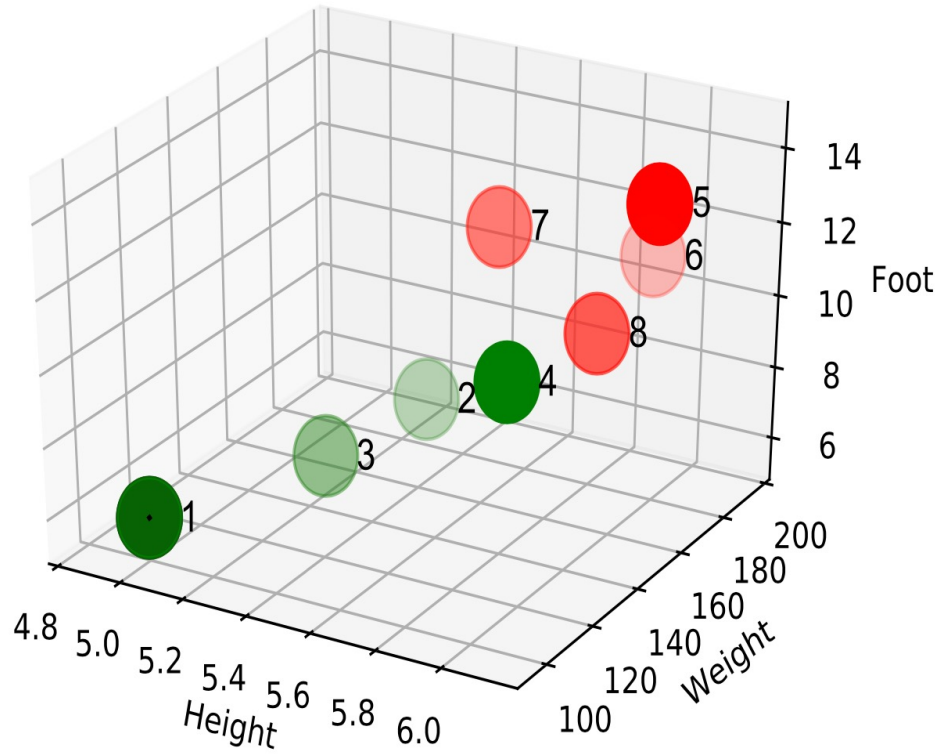
```
import pandas as pd
data = pd.DataFrame(
      {"id": [ 1,2,3,4,5,6,7,8],
       "Label": ["green","green",
                 "green","green",
                 "red","red",
                 "red","red"],
       "Height": [5, 5.5, 5.33,5.75,
                  6.00,5.92,5.58,5.92],
       "Weight": [100, 150, 130, 150,
                    180, 190, 170, 165],
       "Foot": [6, 8, 7, 9,
                13, 11, 12, 10]},
        columns = ["id", "Height",
                   "Weight", "Foot",
                   "Label"] )
```

Example Continue

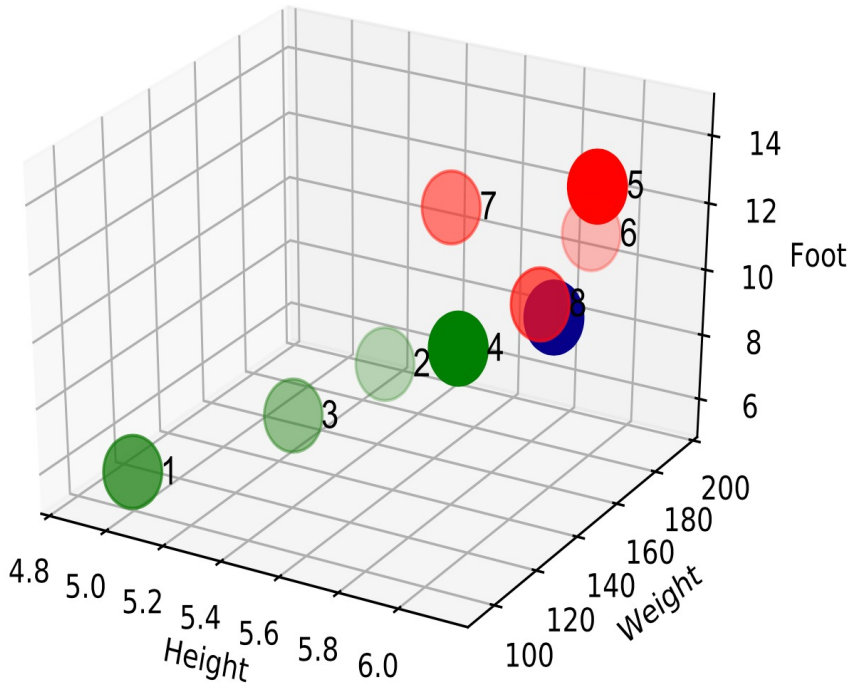
Code for the Dataset (cont'd)

```
ipdb> data
   id Height Weight Foot Label
0   1   5.00    100    6  green
1   2   5.50    150    8  green
2   3   5.33    130    7  green
3   4   5.75    150    9  green
4   5   6.00    180   13    red
5   6   5.92    190   11    red
6   7   5.58    170   12    red
7   8   5.92    165   10    red
```

Example Continue



Example Continue – A New Data Point



$$(\mathrm{H}{=}6,\ \mathrm{W}{=}160,\ \mathrm{F}{=}10) \mapsto\ ?$$

Examples of Objectives

1. binary classification: compute label (green or red) on new instance

$$x \mapsto \phi(\cdot) \mapsto y \in \{0, 1\}$$

2. regression: predict foot size given height and weight

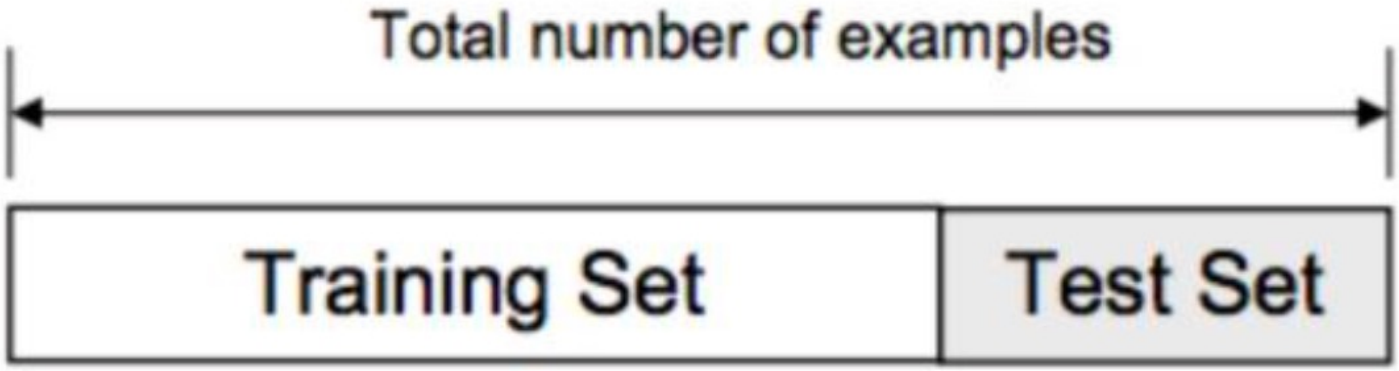
$$x \mapsto \phi(\cdot) \mapsto y \in R$$

- how do we compute $\phi(\cdot)$?

Testing/Training Sets

- Q: how do we compute model parameters?
- Q: how do we estimate accuracy?
- A: split known objects into two subsets
  1. X_train: for parameters
  2. X_testing: for accuracy

Testing/Training Sets (cont'd)



- need to minimize any biases in the data

Bias in Training Data

```
import pandas as pd
data = pd.DataFrame(
       {"id": [ 1,2,3,4,5,6,7,8],
        "Label": ["green","green",
                  "green","green",
                  "red","red",
                  "red","red"],
        "Height": [5, 5.5, 5.33,5.75,
                   6.00,5.92,5.58,5.92],
        "Weight": [100, 150, 130, 150,
                     180, 190, 170, 165],
        "Foot": [6, 8, 7, 9,
                 13, 11, 12, 10]},
         columns = ["id", "Height",
                    "Weight", "Foot",
                    "Label"] )
X = data[["Height","Weight","Foot"]]
X_train = X[ : 4]
X_test  = X[ 4 : ]
```

Bias in Training Data (cont'd)

```
ipdb> X_train
0    5.00      100      6  green
1    5.50      150      8  green
2    5.33      130      7  green
3    5.75      150      9  green
ipdb> X_test
   Height  Weight  Foot Label
4    6.00      180    13   red
5    5.92      190    11   red
6    5.58      170    12   red
7    5.92      165    10   red
ipdb> X_test
```

- need to randomize data

Eliminating Bias

```
import pandas as pd
from sklearn.model_selection \
      import train_test_split

data = pd.DataFrame(
      {"id": [ 1,2,3,4,5,6,7,8],
       "Label":["green","green","green","green",
                "red","red","red","red"],
       "Height":[5, 5.5, 5.33,5.75,
                 6.00,5.92,5.58,5.92],
       "Weight":[100, 150, 130, 150,
                   180, 190, 170, 165],
       "Foot":[6, 8, 7, 9, 13, 11, 12, 10]},
        columns = ["id", "Height",
                   "Weight", "Foot","Label"] )

X = data[["Height","Weight","Foot"]]
y = data["Label"]

X_train,X_test,y_train,y_test=\
      train_test_split(X, y, train_size=0.5)
```

Eliminating Bias

```
ipdb> X_train
   Height  Weight  Foot  Label
3    5.75     150     9  green
1    5.50     150     8  green
4    6.00     180    13    red
5    5.92     190    11    red
ipdb> X_test
   Height  Weight  Foot  Label
2    5.33     130     7  green
0    5.00     100     6  green
7    5.92     165    10    red
6    5.58     170    12    red
ipdb> X_test
```

- data is now "shuffled"

Stratifying Classes

```
import pandas as pd
from sklearn.model_selection \
      import train_test_split

data = pd.DataFrame(
      {"id": [ 1,2,3,4,5,6,7,8],
       "Label":["green","green","green","green",
                "red","red","red","red"],
       "Height":[5, 5.5, 5.33,5.75,
                 6.00,5.92,5.58,5.92],
       "Weight":[100, 150, 130, 150,
                   180, 190, 170, 165],
       "Foot":[6, 8, 7, 9, 13, 11, 12, 10]},
        columns = ["id", "Height",
                   "Weight", "Foot","Label"] )

X = data[["Height","Weight","Foot"]]
y = data["Label"]

X_train,X_test,y_train,y_test=\
      train_test_split(X, y, train_size=0.75)
```

Stratifying Classes

```
ipdb> X_train
   Height  Weight  Foot  Label
6    5.58     170    12    red
2    5.33     130     7  green
3    5.75     150     9  green
0    5.00     100     6  green
1    5.50     150     8  green
4    6.00     180    13    red
ipdb> X_test
   Height  Weight  Foot Label
7    5.92     165    10   red
5    5.92     190    11   red
```

- label counts are not evenly distributed

Split and Stratify

```
import pandas as pd
from sklearn.model_selection \
      import train_test_split

data = pd.DataFrame(
      {"id": [ 1,2,3,4,5,6,7,8],
       "Label":["green","green","green","green",
                "red","red","red","red"],
       "Height":[5, 5.5, 5.33,5.75,
                 6.00,5.92,5.58,5.92],
       "Weight":[100, 150, 130, 150,
                   180, 190, 170, 165],
       "Foot":[6, 8, 7, 9, 13, 11, 12, 10]},
        columns = ["id", "Height",
                   "Weight", "Foot","Label"] )

X = data[["Height","Weight","Foot"]]
y = data["Label"]

X_train,X_test,y_train,y_test=\
      train_test_split(X, y,
          train_size=0.75,stratify=y)
```

Stratifying Classes

```
ipdb> X_train
1    5.50     150     8  green
6    5.58     170    12    red
3    5.75     150     9  green
5    5.92     190    11    red
2    5.33     130     7  green
7    5.92     165    10    red
ipdb> X_test
   Height  Weight  Foot  Label
0     5.0     100     6  green
4     6.0     180    13    red
```

- label counts are now evenly distributed

Cross Validation

- we use only a portion of data for testing and training
- can use more with n-fold cross validation

1. split data randomly into n parts
2. use $n-1$ for training
3. use 1 part for testing
4. repeat n times using different part for testing
5. average results

k-fold Validation

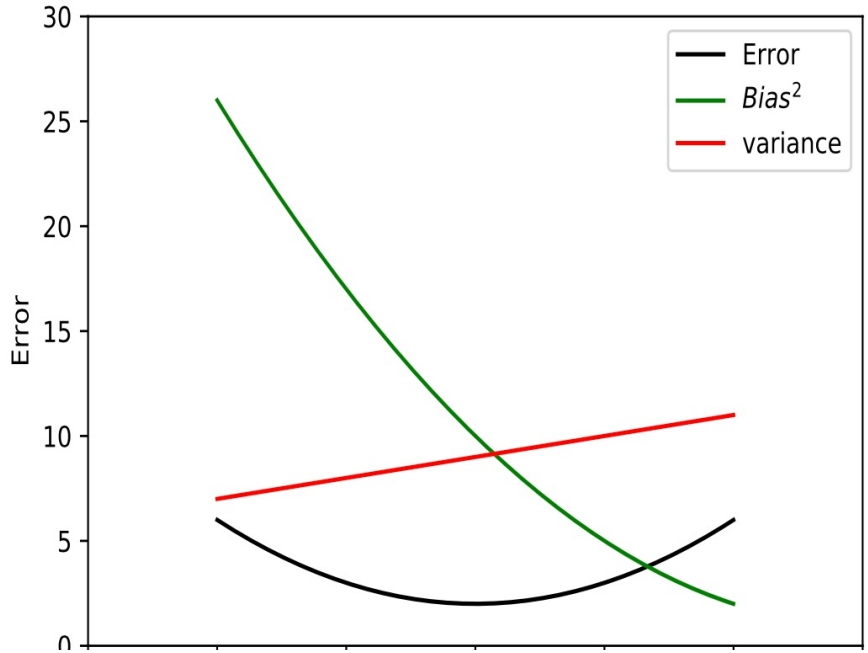
- expensive for large n

Bias-Variance Decomposition

- bias - average difference between prediction and correct value
- variance - variability of prediction for a given point

$$\textbf{Error} = \textbf{Bias}^2 + \textbf{Variance} + \textbf{Irreducible Error}$$

Bias-Variance: Ideal Model



- "overfitting" - low bias, high variance
- 'underfitting" - low variance, high bias

Logistic Regression Loss Function

Output of logistic regression is $\hat{p}$ and the target output is p

$$Cost_{bits} = p \times log_2(\hat{p}) + (1-p) \times log_2(1-\hat{p})$$

or

$$Cost_{nats} = p \times ln(\hat{p}) + (1-p) \times ln\,(1-\hat{p})$$

Concepts Check:

(a) prediction vs. classification

(b) numerical vs. categorical data

(c) loss function

(d) testing and training data

(e) bias elimination and stratification

(f) cross and k-fold validation

(g) bias vs. variance trade-offs

Multivariable and Multivariate Logistic Regression

Multivariable Logistic Regression

- One variable can be extended to multivariable as follows:

$$y = \frac{exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}{1 + exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}$$

and

$$(0 \leq y \leq 1),$$

$$y = \begin{cases} 1, & Belong\ to\ the\ class \\ 0, & Doesn't\ belong \end{cases}$$

For a rare event we cheat! we use none proportional ratio

When Logistic regression is trained for a none proportional ratio of samples (like rare events)

- The model will calculate probabilities wrong.
  - E.G. rare event occurs 1%. We train with 40% of the rare event

<u>Solution:</u>

- Case control sampling
  - Regression parameters β_i are accurate, and only the intercept β_0 needs to get corrected by

$$\beta_0{}^{*} = \beta_0 + \log\left(\frac{p_{rare}}{1 - prare}\right) - \log(\frac{p_{set}}{1 - pset})$$

- P_{rare}: actual probability of the rare event
- P_{set}: probability of the rare event in the training set

Control vs Case Sample Size

- Control to case ratio: In order to have smaller variance in the coefficients it is good to have more control samples.
- Question is how much more?
  - Rule of thumb is five to six times is sufficient

Multiclass Logistic Regression or Multinomial Regression

- Logistic regression can be extended to more than two class prediction by.

$$\Pr(y = k|X) = \frac{e^{\beta 0k + \beta 1kX1 + \cdots + \beta nkXn}}{\sum_{j=1}^{K} e^{\beta 0j + \beta 1jX1 + \cdots + \beta njXn}}$$

K : capital K is total number of classes

k: small K is one of the classes

- Select the class with the highest probability

- This is also called “softmax” function

Interesting Math about Logistic Regression

- *log odds* or *logit* transformation of *y*

$$\ln\left(\frac{y}{1-y}\right) = \beta_0 + \beta_1 X_1 + \cdots + BnXn$$

- Derivative

$$\frac{\partial y}{\partial z} = y(1-y)$$

- Odd ratio

$$\frac{y}{1-y} = \exp(\beta_0 + \beta_1 X_1 + \cdots + BnXn)$$

Pros and Cons of Logistic Regression

- Logistic regression is easy to implement and interpret.
- If number of observations is less than the number of features, it will overfit
- No assumption on distribution of classes in feature space
- The boundaries constructed are linear
- Multinomial regression/multiple classes implementation and interpretation is easy
- Assuming linear dependency of dependent and independent variables
- Measures importance of each independent variable and also positive or negative correlation
- Not appropriate for non-linear problems and complex relationships
- Mostly doesn't overfit
- Extracts linear dependency of independent variables and log odd of dependent variable.

Important note:
Logistic Regression is not Stable for fully separable classes.

In this case, other classification methods like Discriminant analysis has to be used.

Design considerations

- Threshold doesn't need to be set at 0.5
- Multivariate logistic regression (Softmax) is easy to implement
- Multivariable logistic regression is easy to implement

- Discriminate analysis is a good option when
  - Number of samples is small
  - Classes are fully separatable