BT2101 Tutorial 2
Logistic Regression

Agenda

- Understand Logistic Regression

- Discussion about Programming Assignment 2
    - Logistic Regression
    - Gradient Ascent

- Python Implementation

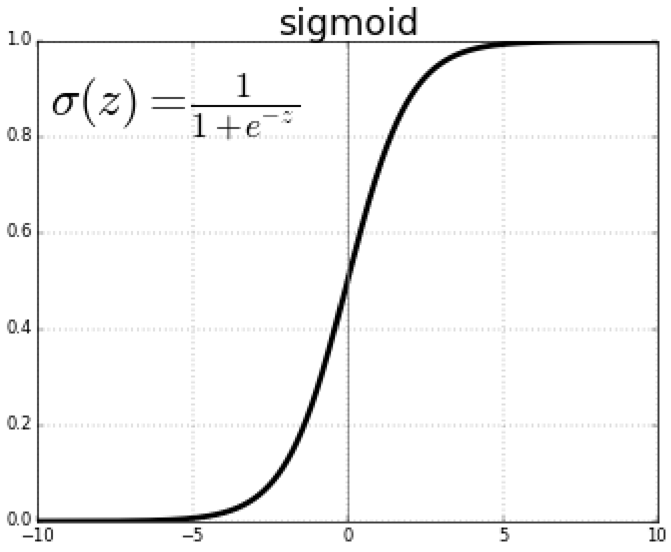
Logistic Regression

- Think about
  - General specification: $$Logodds(\Pr(y_i = 1)) = \log\left(\frac{\Pr(y_i = 1)}{1 - \Pr(y_i = 1)}\right) = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$$

$$\Pr(y_i = 1) = \frac{\exp(\beta_0 + \sum_{j=1}^{p} x_j \beta_j)}{1 + \exp(\beta_0 + \sum_{j=1}^{p} x_j \beta_j)}$$

  - Sigmoid function:

(Very Important Function)



Estimation

- Gradient Ascent
  - Maximizing (Log-)Likelihood:

$$l(\beta)=\prod_{i=1}^{N}[\frac{e^{\beta_0+\sum_{j=1}^{p}x_j\beta_j}}{1+e^{\beta_0+\sum_{j=1}^{p}x_j\beta_j}}]^{y_i}[\frac{1}{1+e^{\beta_0+\sum_{j=1}^{p}x_j\beta_j}}]^{(1-y_i)}$$

$$ll(\beta)=\sum_{i=1}^{N}[-\log(1+e^{\beta_0+\sum_{j=1}^{p}x_j\beta_j})+y_i(\beta_0+\sum_{j=1}^{p}x_j\beta_j)]$$

  - Question: How to maximize this complex objective function?
    - Let first-order derivatives = 0 ?
    - Maybe you can take steps by steps (iteratively) to approach the (local) optimal value

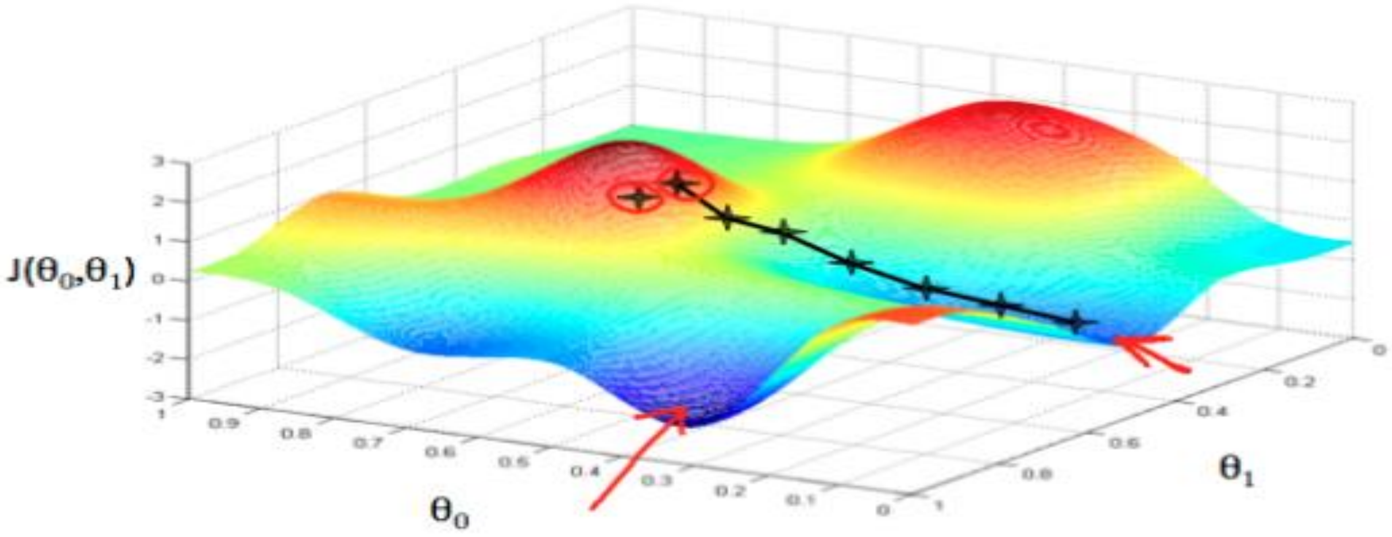
Remember BT1101

Gradient Descent

(For Minimizing)

Gradient Ascent

(For Maximizing)



Correct: Simultaneous update

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

$\theta_0 := \text{temp0}$

$\theta_1 := \text{temp1}$

Incorrect:

$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

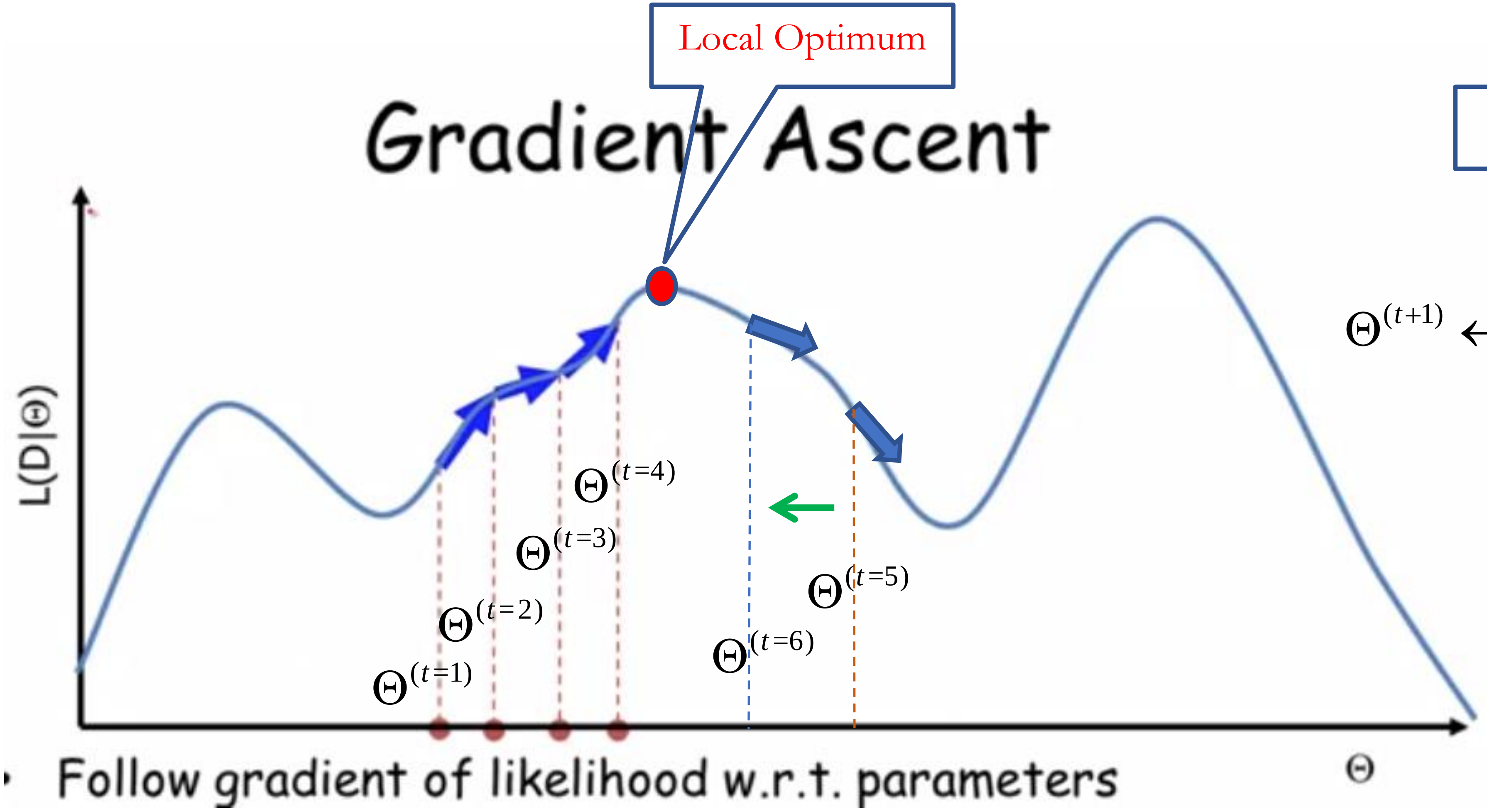
$\theta_0 := \text{temp0}$

$\text{temp1} := \theta_1 - \boxed{\alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)}$

$\theta_1 := \text{temp1}$

- gradient *descent* aims at *minimizing* some objective function: $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
- gradient *ascent* aims at *maximizing* some objective function: $\theta_j \leftarrow \theta_j + \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

Overview of Gradient Ascent



Step size

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \boxed{stepsize} \times \boxed{\frac{\partial L}{\partial \Theta^{(t)}}}$$

Gradient

Estimation

- Gradient Ascent
  - Maximizing (Log-)Likelihood:

$$ll(\beta)=\sum_{i=1}^{N}[-\log(1+e^{\beta_0+\sum_{j=1}^{p}x_j\beta_j})+y_i(\beta_0+\sum_{j=1}^{p}x_j\beta_j)]$$

  - Gradient Ascent:
    - Remember **Taylor Expansion**
    - Newton-Raphson Method
      - Hard to get $-H_t^{-1}$
    - Steepest Ascent Method:
      - Let: $-H_t^{-1} = \lambda I$
      - I: Identity Matrix

8.3.1. Newton–Raphson

To determine the best value of β_{t+1}, take a second-order Taylor's approximation of $LL(\beta_{t+1})$ around $LL(\beta_t)$:

(8.1)

$$LL(\beta_{t+1}) = LL(\beta_t) + (\beta_{t+1}-\beta_t)'g_t + \tfrac{1}{2}(\beta_{t+1}-\beta_t)'H_t(\beta_{t+1}-\beta_t).$$

Now find the value of β_{t+1} that maximizes this approximation to $LL(\beta_{t+1})$:

$$\frac{\partial LL(\beta_{t+1})}{\partial \beta_{t+1}} = g_t + H_t(\beta_{t+1}-\beta_t) = 0,$$

$$H_t(\beta_{t+1}-\beta_t) = -g_t,$$

$$\beta_{t+1}-\beta_t = -H_t^{-1}g_t,$$

$$\beta_{t+1} = \beta_t + \boxed{(-H_t^{-1})}g_t.$$

Gradient Ascent

$$\beta_{t+1} \leftarrow \beta_t + \lambda \times g_t$$

Gradient Ascent

- Gradient Ascent
  - Objective Function: $ll(\beta) = \sum_{i=1}^{N} [-\log(1 + e^{\beta_0 + \sum_{j=1}^{p} x_j \beta_j}) + y_i(\beta_0 + \sum_{j=1}^{p} x_j \beta_j)]$
  - Gradient Ascent:
    - (1) Initialize $\beta^{(0)} = (\beta_0^{(0)}, \beta_1^{(0)}, ..., \beta_j^{(0)}) = (0, 0, ..., 0), t = 1$
    - (2) In step t, update coefficients:

$$\frac{\partial ll}{\partial \beta_j} \leftarrow \sum_{i=1}^{N} (y_i - \frac{e^{\beta_0^{(t-1)} + \sum_{j=1}^{p} x_j \beta_j^{(t-1)}}}{1 + e^{\beta_0^{(t-1)} + \sum_{j=1}^{p} x_j \beta_j^{(t-1)}}}) x_{ij}$$

Calculate gradients or first-order derivatives of coefficients

$$\beta_j^{(t)} \leftarrow \beta_j^{(t-1)} + stepsize \times \frac{\partial ll}{\partial \beta_j}$$

Update coefficients with Steepest Ascent Method

$$t \leftarrow t + 1$$

    - (3) Check convergence condition $\| \nabla ll(\beta^{(t)}) \| < tolerance$. If not, go back to (2) until (3) is satisfied

Binary Classifier Performance

Performance of Binary Classifier

- Confusion Matrix

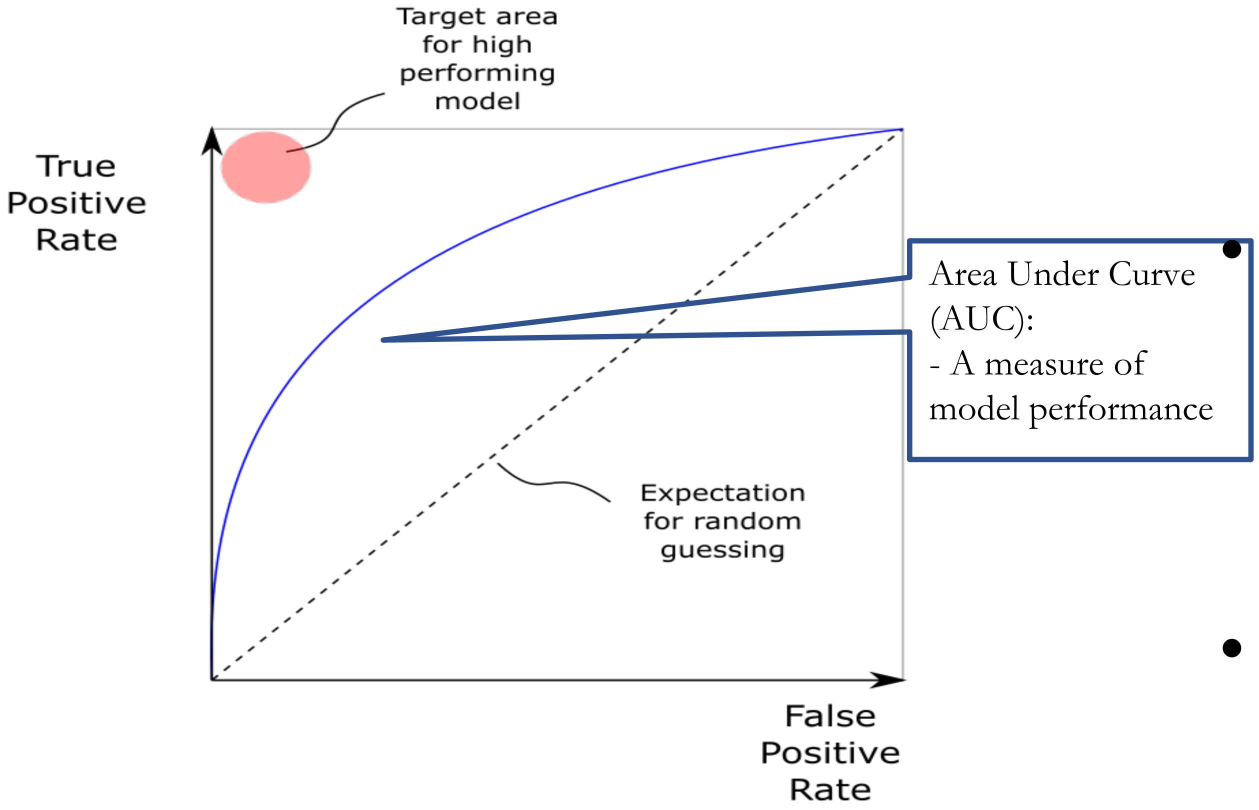
Correct classification	Classified as	
	+	−
+	true positives	false negatives
−	false positives	true negatives

True Positive Rate or Hit Rate or Recall or Sensitivity or TP Rate	TP/P	The proportion of positive instances that are correctly classified as positive
False Positive Rate or False Alarm Rate or FP Rate	FP/N	The proportion of negative instances that are erroneously classified as positive
False Negative Rate or FN Rate	FN/P	The proportion of positive instances that are erroneously classified as negative = 1 − True Positive Rate

True Negative Rate or Specificity or TN Rate	TN/N	The proportion of negative instances that are correctly classified as negative
Precision or Positive Predictive Value	TP/(TP+FP)	Proportion of instances classified as positive that are really positive
F1 Score	(2 × Precision × Recall) /(Precision + Recall)	A measure that combines Precision and Recall
Accuracy or Predictive Accuracy	(TP + TN)/(P + N)	The proportion of instances that are correctly classified
Error Rate	(FP + FN)/(P + N)	The proportion of instances that are incorrectly classified

Performance of Binary Classifier

- ROC and AUC



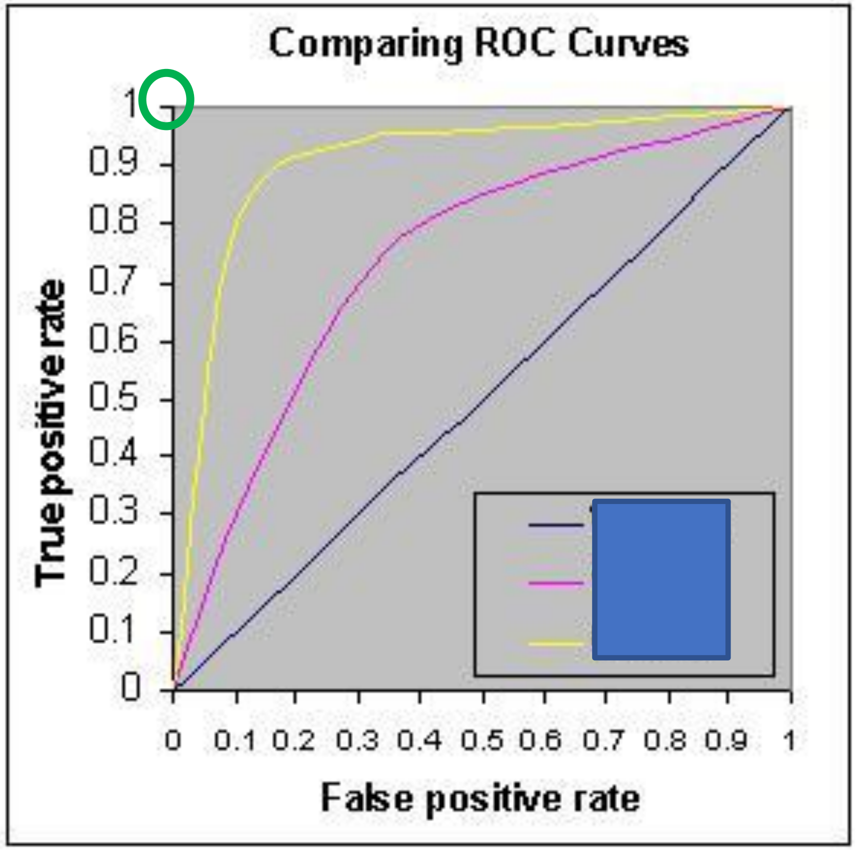
- The **upper left-hand triangle** corresponds to classifiers that are better than random guessing. The **lower right-hand triangle** corresponds to classifiers that are worse than random guessing

- The closer the curve follows the left-hand border and the top border of the ROC space (i.e., closer to the upper left-hand circle), the more accurate the test.

- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Performance of Binary Classifier

- ROC and AUC



Quiz. Which model is better?

A. Yellow Curve

B. Pink Curve

C. Reference Line

Multi-Class Classifier Performance

Confusion Matrix

	Predicted					
Ground Truth		Class1	Class2	Class3	Class4	Class5
	Class1	92	3	2	2	1
	Class2	2	92	2	2	2
	Class3	1	1	92	6	0
	Class4	0	1	1	92	6
	Class5	1	4	2	1	92

- Accuracy
- Misclassification Rate = 1 - Accuracy

Cross Validation

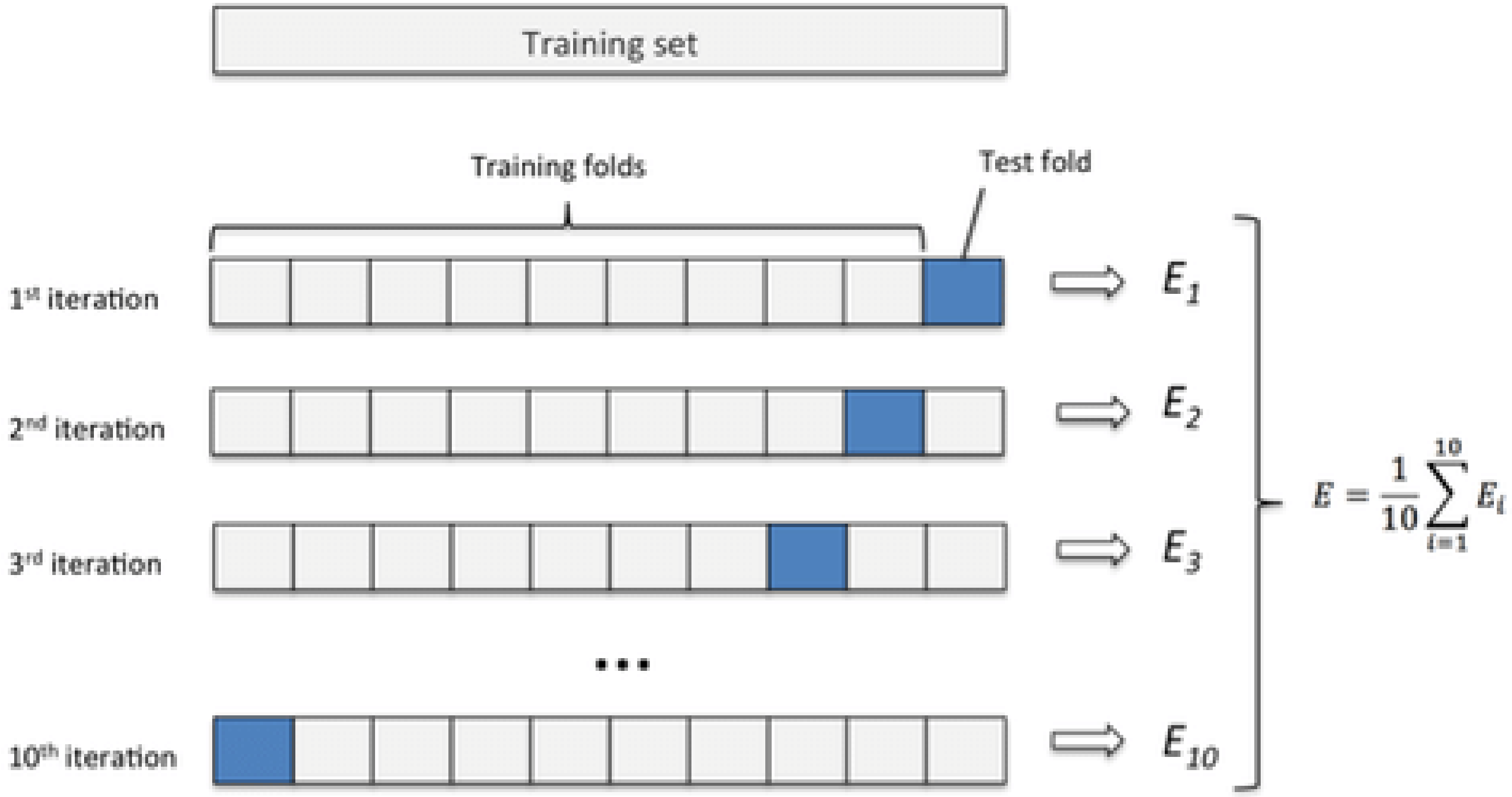
- Cross Validation (e.g., K-Fold Cross Validation)

  In practice…

  - The cross-validation procedure is repeated K times
  - K random partitions of the original sample
  - The K results are again averaged (or otherwise combined) to produce a single estimation.
  - You can use cross validation for both binary classification and multi-class classification problems

Cross Validation

- K-Fold Cross Validation (e.g., K=10)



- Model Evaluation
- Model Comparison
- Model Tuning

All observations are used for both training and validation, and each observation is used for validation exactly once.

Cross Validation

- Scikit-learn: KFold

Examples

```
>>> from sklearn.model_selection import KFold
>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([1, 2, 3, 4])
>>> kf = KFold(n_splits=2)
>>> kf.get_n_splits(X)
2
>>> print(kf)
KFold(n_splits=2, random_state=None, shuffle=False)
>>> for train_index, test_index in kf.split(X):
...    print("TRAIN:", train_index, "TEST:", test_index)
...    X_train, X_test = X[train_index], X[test_index]
...    y_train, y_test = y[train_index], y[test_index]
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
```

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

Implementation in Python

BT2101 Introduction to Logistic Regression

Version: Python 3

1 Goal:

In this notebook, we will explore logistic regression using:

- Gradient ascent method (because you cannot get closed-form solutions)
- Open-source package: `scikit-learn`

For the gradient descent method, you will:

- Use numpy to write functions
- Write a likelihood function
- Write a derivative function
- Write an output function
- Write a gradient ascent function
- Add a constant column of 1's as intercept term
- Use the gradient ascent function to get regression estimators

In [ ]:
```
# -*- coding:utf-8 -*-
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import sqrt
from __future__ import division
%matplotlib inline
```

1.1 Summary of Logistic Regression

Programming Assignment 2

Using the BT2101 Tutorial 2 Programming code (Logistic Regression.ipynb), please answer the questions in the jupyter notebook

Answer all in the jupyter notebook.

Instructions

Submit Python Notebook to IVLE folder, and Naming the file:
AXXXX_T2_program.ipynb

Include your answers in the jupyter notebook

- You need to show outputs, instead of just showing functions.

Submit a FINAL program by Sep-11 Tuesday (by lunchtime)

- Based on Logistic Regression.ipynb

Thank you!

Appendix

Threshold p:
If Score>=p, Predict 1; If Score<p, Predict 0

ID	True Output	Score: P(y=1|Data, β)	Predicted Output (Threshold 90%)	Predicted Output (Threshold 80%)	...	Predicted Output (Threshold 3%)
1	1	0.9	1	1	...	1
2	1	0.8	0	1	...	1
3	0	0.7	0	0	...	1
4	1	0.6	0	0	...	1
5	1	0.55	0	0	...	1
6	1	0.47	0	0	...	1
7	0	0.39	0	0	...	1
8	0	0.21	0	0	...	1
9	1	0.19	0	0	...	1
10	0	0.03	0	0	...	1

Appendix

L1 Regularization: Lasso Regression

$$RSS(\beta) = \varepsilon^T \varepsilon = \sum_{i=1}^{N} (y_i - f(X_i))^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

$$subject\ to \sum_{j=1}^{p} |\beta_j| \leq threshold$$

Shrink the size of coefficients;
Do variable selection (Guess why?)

L2 Regularization: Ridge Regression

$$RSS(\beta) = \varepsilon^T \varepsilon = \sum_{i=1}^{N} (y_i - f(X_i))^2 = \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

$$subject\ to \sum_{j=1}^{p} \beta_j^2 \leq threshold$$

Shrink the size of coefficients;
No variable selection (Guess why?)

Note the difference between Lasso and Ridge Regression