HW2 TA hours

TAs
ntu.mlta@gmail.com

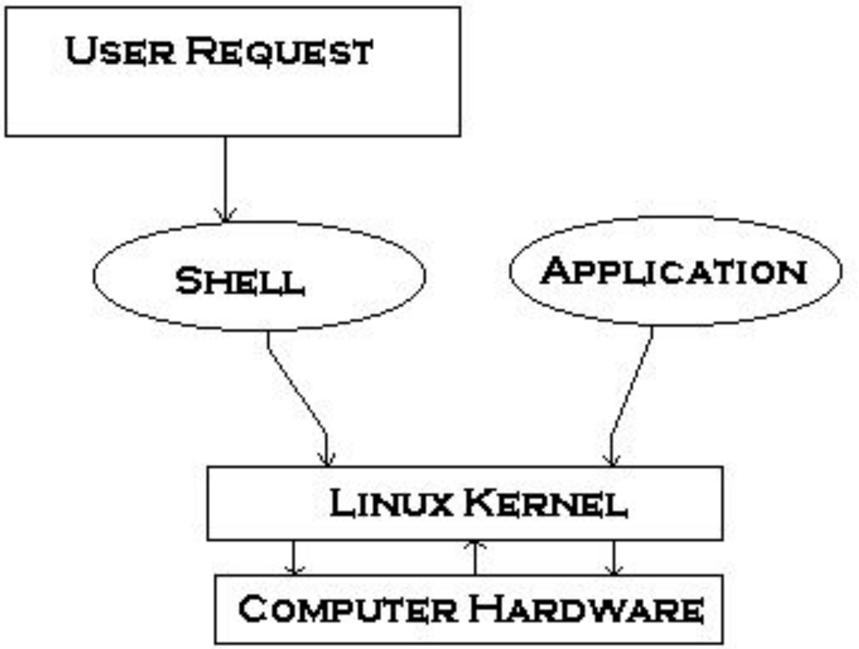
Outline

1. 如何在shell scripts與python檔中接收arguments
2. Logistic Regresion
3. Generative model

Shell Scripts

1. What is Linux Shell?

Shell accepts your instruction or commands (usually in English) and pass it to kernel.



Shell Scripts

2. What is Shell Script?

Shell script defined as:
"*Shell Script is* ***series of command*** *written* ***in plain text file.***

... 把很多指令寫成一個檔案，可以一次做很多事情。

甚至是做判斷式或是迴圈。

Script Tutorial and Example

Shell Script Tutorial:
http://linux.vbird.org/linux_basic/0340bashshell-scripts.php

Example:

```
# feature extraction by yourself
python my_feature_extraction.py $1 $2
python hw2_logistic_train.py
python hw2_logistic_test.py $5 $6
```

Passing Arguments

How to pass arguments to your srctipts or .py files?

1. 在terminal或cmd中的輸入與script中變數對應關係：
/path/to/scriptname /path/to/data /path/to/output
$0 $1 $2

2. 在terminal或cmd中的輸入與.py中變數對應關係：
/path/to/pythonfile /path/to/data /path/to/output
sys.argv[0] sys.argv[1] sys.argv[2]

Passing Arguments

script:

```
echo 'script_arg1 is : '$0''
echo 'script_arg2 is : '$1''
echo 'script_arg3 is : '$2''
echo 'Start to run Python'
python test.py $1 $2
```

python:

```
import sys
a = sys.argv[0]
b = sys.argv[1]
c = sys.argv[2]
print 'python_arg1 is : %s' % a
print 'python_arg2 is : %s' % b
print 'python_arg3 is : %s' % c
```

Passing Arguments

Let's run...

```
acetylSv:~ acetylSv$ ./test.sh path1 path2
script_arg1 is : ./test.sh
script_arg2 is : path1
script_arg3 is : path2
Start to run Python
python_arg1 is : test.py
python_arg2 is : path1
python_arg3 is : path2
```

PATH

- 絕對路徑 : 相對於**根目錄**的路徑。
  - Ex: /Users/MLTA/Desktop/hw2
- 相對路徑 : 相對於**目前資料夾**的路徑。
  - Ex: ./hw2/logistic.py、../hw1/linear_regression.py

★ 助教會在git clone整個 ML2017/hw2資料夾

★ 並在 ML2017/hw2資料夾執行程式(Ex : hw2_best.sh)

★ 若要 **讀/存model**請用**相對路徑**

★ Data的Path助教會用絕對路徑下在hw2_best.sh的argument裡。

Logistic Regression

Implementation:

1. feature normalization
2. logistic regression using gradient descent with batch

Code Example:

http://codepad.org/RvIY9BU2

If we let batch_size=25...

Logistic Regression

```
X.shape=(25, 106)
w.shape=(106,)
b.sahpe=(1,)
z.shape=(25,)
y.shape=(25,)
Y.shape=(25,)
```

Logistic Regression

Step 1: $f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$

```
z = np.dot(X, np.transpose(w)) + b
y = sigmoid(z)
```

Step 2: $\boxed{\hat{y}^n\text{: 1 for class 1, 0 for class 2}}$

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

```
cross_entropy = -(np.dot(Y, np.log(y)) + np.dot((1 - Y), np.log(1 - y)))
```

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n lnf(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Logistic Regression

Step 3: Find the best function

$$\frac{-lnL(w,b)}{\partial w_i} = \sum_n -\left[\hat{y}^n \underbrace{\boxed{\frac{ln f_{w,b}(x^n)}{\partial w_i}}}_{} + (1-\hat{y}^n)\boxed{\frac{ln\left(1-f_{w,b}(x^n)\right)}{\partial w_i}}\right]$$

$$\left(1-f_{w,b}(x^n)\right)x_i^n \qquad -f_{w,b}(x^n)x_i^n$$

$$= \sum_n -\left[\hat{y}^n\left(1-f_{w,b}(x^n)\right)x_i^n - (1-\hat{y}^n)f_{w,b}(x^n)x_i^n\right]$$

```
w_grad = np.sum(-1 * X * (Y - y).reshape((batch_size,1)), axis=0)
b_grad = np.sum(-1 * (Y - y))
```

$$= \sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right)x_i^n$$

```
w = w - l_rate * w_grad
b = b - l_rate * b_grad
```

Larger difference, larger update

$$w_i \leftarrow w_i - \eta\sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right)x_i^n$$

Gaussian Distribution-Training(mean)

$$\mu^* = \frac{1}{79}\sum_{n=1}^{79} x^n$$

average

```
#calclulate mu1 and mu2
mu1 = np.zeros((dim,))
mu2 = np.zeros((dim,))
for i in range(train_data_size):
    if Y_train[i] == 1:
        mu1 += X_train[i]
        cnt1 += 1
    else:
        mu2 += X_train[i]
        cnt2 += 1
mu1 /= cnt1
mu2 /= cnt2
```

Gaussian Distribution-Training(sigma)

$$\Sigma^* = \frac{1}{79}\sum_{n=1}^{79}(x^n - \mu^*)(x^n - \mu^*)^T$$

```
#calculate sigma1 and sigma2
sigma1 = np.zeros((dim,dim))
sigma2 = np.zeros((dim,dim))
for i in range(train_data_size):
    if Y_train[i] == 1:
        sigma1 += np.dot(np.transpose([X_train[i] - mu1]), [(X_train[i] - mu1)])
    else:
        sigma2 += np.dot(np.transpose([X_train[i] - mu2]), [(X_train[i] - mu2)])
sigma1 /= cnt1
sigma2 /= cnt2
```

Gaussian Distribution-Training(sigma)

Shared sigma: $$\Sigma = \frac{79}{140}\Sigma^1 + \frac{61}{140}\Sigma^2$$

```
shared_sigma = (float(cnt1) / train_data_size) * sigma1 \
             + (float(cnt2) / train_data_size) * sigma2
```

Gaussian Distribution-predict

$$\Sigma_1 = \Sigma_2 = \Sigma$$

$$z = \underbrace{(\mu^1 - \mu^2)^T \Sigma^{-1}}_{\boldsymbol{w}^T} x \underbrace{- \frac{1}{2}(\mu^1)^T \Sigma^{-1} \mu^1 + \frac{1}{2}(\mu^2)^T \Sigma^{-1} \mu^2 + ln \frac{N_1}{N_2}}_{b}$$

$$P(C_1|x) = \sigma(w \cdot x + b)$$

How about directly find ***w*** and b?

```
def predict(X_test, mu1, mu2, shared_sigma, N1, N2):
    sigma_inverse = np.linalg.inv(shared_sigma)
    w = np.dot( (mu1-mu2), sigma_inverse)
    x = X_test.T
    b = (-0.5) * np.dot(np.dot([mu1], sigma_inverse), mu1) \
        + (0.5) * np.dot(np.dot([mu2], sigma_inverse), mu2) + np.log(float(N1)/N2)
    a = np.dot(w, x) + b
    y = sigmoid(a)
    return y
```

Gaussian Distribution-sigmoid

```
def sigmoid(z):
    res = 1 / (1.0 + np.exp(-z))
    return np.clip(res, 0.00000000000001, 0.99999999999999)
```

使用np.clip() 避免數值太小或太大而overflow

Generative model -Naive Bayes Classifier

1. Calculate the probability in each category
2. P(C1 | X) = 連乘P(C1 | X in each category)
3. log(P(C1 | X)) = sigma(log(P(C1 | X_i)))
4. 比較 log(P(C1 | X)) 和 log(P(C2 | X_i))

$$\begin{bmatrix} \text{白人} \\ \text{黑人} \\ \text{黃人} \end{bmatrix}$$

P(C1 ｜ 白人) = N(白人, C1) / N(白人)

P(C2 ｜ 白人) = N(白人, C2) / N(白人)

P(? ｜ C1) = N(C1) / N(total)

code

http://codepad.org/wlrT0lw4

HW2 TA hours

TAs
ntu.mlta@gmail.com

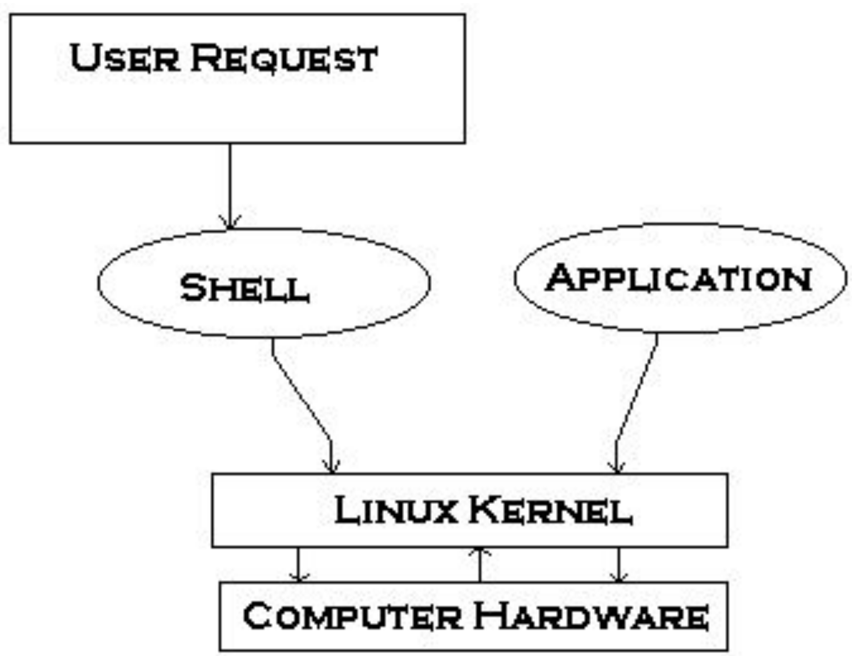
Outline

1. 如何在shell scripts與python檔中接收arguments
2. Logistic Regresion
3. Generative model

Shell Scripts

1. What is Linux Shell?

Shell accepts your instruction or commands (usually in English) and pass it to kernel.



Shell Scripts

2. What is Shell Script?

Shell script defined as:
"*Shell Script is* ***series of command*** *written* ***in plain text file.***

... 把很多指令寫成一個檔案，可以一次做很多事情。

甚至是做判斷式或是迴圈。

Script Tutorial and Example

Shell Script Tutorial:
http://linux.vbird.org/linux_basic/0340bashshell-scripts.php

Example:

```
# feature extraction by yourself
python my_feature_extraction.py $1 $2
python hw2_logistic_train.py
python hw2_logistic_test.py $5 $6
```

Passing Arguments

How to pass arguments to your srctipts or .py files?

1. 在terminal或cmd中的輸入與script中變數對應關係：
/path/to/scriptname /path/to/data /path/to/output
$0 $1 $2

2. 在terminal或cmd中的輸入與.py中變數對應關係：
/path/to/pythonfile /path/to/data /path/to/output
sys.argv[0] sys.argv[1] sys.argv[2]

Passing Arguments

script:

```
echo 'script_arg1 is : '$0''
echo 'script_arg2 is : '$1''
echo 'script_arg3 is : '$2''
echo 'Start to run Python'
python test.py $1 $2
```

python:

```
import sys
a = sys.argv[0]
b = sys.argv[1]
c = sys.argv[2]
print 'python_arg1 is : %s' % a
print 'python_arg2 is : %s' % b
print 'python_arg3 is : %s' % c
```

Passing Arguments

Let’s run…

```
acetylSv:~ acetylSv$ ./test.sh path1 path2
script_arg1 is : ./test.sh
script_arg2 is : path1
script_arg3 is : path2
Start to run Python
python_arg1 is : test.py
python_arg2 is : path1
python_arg3 is : path2
```

PATH

- 絕對路徑 : 相對於**根目錄**的路徑。
  - Ex: /Users/MLTA/Desktop/hw2
- 相對路徑 : 相對於**目前資料夾**的路徑。
  - Ex: ./hw2/logistic.py、../hw1/linear_regression.py

★ 助教會在git clone整個 ML2017/hw2資料夾

★ 並在 ML2017/hw2資料夾執行程式(Ex : hw2_best.sh)

★ 若要 **讀/存model**請用**相對路徑**

★ Data的Path助教會用絕對路徑下在hw2_best.sh的argument裡。

Logistic Regression

Implementation:

1. feature normalization
2. logistic regression using gradient descent with batch

Code Example:

http://codepad.org/RvIY9BU2

Logistic Regression

If we let batch_size=25...

```
1 X.shape=(25, 106)
2 w.shape=(106,)
3 b.sahpe=(1,)
4 z.shape=(25,)
5 y.shape=(25,)
6 Y.shape=(25,)
```

Logistic Regression

Step 1: $f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$

```
z = np.dot(X, np.transpose(w)) + b
y = sigmoid(z)
```

Step 2: $\boxed{\hat{y}^n\text{: 1 for class 1, 0 for class 2}}$

$$L(f) = \sum_n C(f(x^n), \hat{y}^n)$$

```
cross_entropy = -(np.dot(Y, np.log(y)) + np.dot((1 - Y), np.log(1 - y)))
```

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n ln f(x^n) + (1 - \hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

Logistic Regression

Step 3: Find the best function

$$\left(1-f_{w,b}(x^n)\right)x_i^n \qquad -f_{w,b}(x^n)x_i^n$$

$$\frac{-lnL(w,b)}{\partial w_i} = \sum_n -\left[\hat{y}^n \boxed{\frac{lnf_{w,b}(x^n)}{\partial w_i}} + (1-\hat{y}^n)\boxed{\frac{ln\left(1-f_{w,b}(x^n)\right)}{\partial w_i}}\right]$$

$$= \sum_n -\left[\hat{y}^n\left(1-f_{w,b}(x^n)\right)x_i^n - (1-\hat{y}^n)f_{w,b}(x^n)x_i^n\right]$$

```
w_grad = np.sum(-1 * X * (Y - y).reshape((batch_size,1)), axis=0)
b_grad = np.sum(-1 * (Y - y))
```

$$= \sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right)x_i^n$$

```
w = w - l_rate * w_grad
b = b - l_rate * b_grad
```

Larger difference, larger update

$$w_i \leftarrow w_i - \eta\sum_n -\left(\hat{y}^n - f_{w,b}(x^n)\right)x_i^n$$

Gaussian Distribution-Training(mean)

$$\mu^* = \frac{1}{79}\sum_{n=1}^{79} x^n$$

average

```
#calclulate mu1 and mu2
mu1 = np.zeros((dim,))
mu2 = np.zeros((dim,))
for i in range(train_data_size):
    if Y_train[i] == 1:
        mu1 += X_train[i]
        cnt1 += 1
    else:
        mu2 += X_train[i]
        cnt2 += 1
mu1 /= cnt1
mu2 /= cnt2
```

Gaussian Distribution-Training(sigma)

$$\Sigma^* = \frac{1}{79}\sum_{n=1}^{79}(x^n - \mu^*)(x^n - \mu^*)^T$$

```
#calculate sigma1 and sigma2
sigma1 = np.zeros((dim,dim))
sigma2 = np.zeros((dim,dim))
for i in range(train_data_size):
    if Y_train[i] == 1:
        sigma1 += np.dot(np.transpose([X_train[i] - mu1]), [(X_train[i] - mu1)])
    else:
        sigma2 += np.dot(np.transpose([X_train[i] - mu2]), [(X_train[i] - mu2)])
sigma1 /= cnt1
sigma2 /= cnt2
```

Gaussian Distribution-Training(sigma)

Shared sigma: $$\Sigma = \frac{79}{140}\Sigma^1 + \frac{61}{140}\Sigma^2$$

```python
shared_sigma = (float(cnt1) / train_data_size) * sigma1 \
             + (float(cnt2) / train_data_size) * sigma2
```

Gaussian Distribution-predict

$$\Sigma_1 = \Sigma_2 = \Sigma$$

$$z = \underbrace{(\mu^1 - \mu^2)^T \Sigma^{-1}}_{\boldsymbol{w}^T} x \underbrace{- \frac{1}{2}(\mu^1)^T \Sigma^{-1} \mu^1 + \frac{1}{2}(\mu^2)^T \Sigma^{-1} \mu^2 + ln \frac{N_1}{N_2}}_{\mathrm{b}}$$

$$P(C_1|x) = \sigma(w \cdot x + b)$$

How about directly find ***w*** and b?

```
def predict(X_test, mu1, mu2, shared_sigma, N1, N2):
    sigma_inverse = np.linalg.inv(shared_sigma)
    w = np.dot( (mu1-mu2), sigma_inverse)
    x = X_test.T
    b = (-0.5) * np.dot(np.dot([mu1], sigma_inverse), mu1) \
        + (0.5) * np.dot(np.dot([mu2], sigma_inverse), mu2) + np.log(float(N1)/N2)
    a = np.dot(w, x) + b
    y = sigmoid(a)
    return y
```

Gaussian Distribution-sigmoid

```python
def sigmoid(z):
    res = 1 / (1.0 + np.exp(-z))
    return np.clip(res, 0.00000000000001, 0.99999999999999)
```

使用np.clip() 避免數值太小或太大而overflow

Generative model -Naive Bayes Classifier

1. Calculate the probability in each category
2. P(C1 | X) = 連乘P(C1 | X in each category)
3. log(P(C1 | X)) = sigma(log(P(C1 | X_i)))
4. 比較 log(P(C1 | X)) 和 log(P(C2 | X_i))

白人

黑人

黃人

P(C1 | 白人) = N(白人, C1) / N(白人)

P(C2 | 白人) = N(白人, C2) / N(白人)

P(? | C1) = N(C1) / N(total)

code

http://codepad.org/wlrT0lw4