

Chapter 3.

Clustering

Contents

- Supervised vs. Unsupervised Learning
- Word Vectors
- Hierarchical Clustering
- Drawing the Dendrogram
- Column Clustering

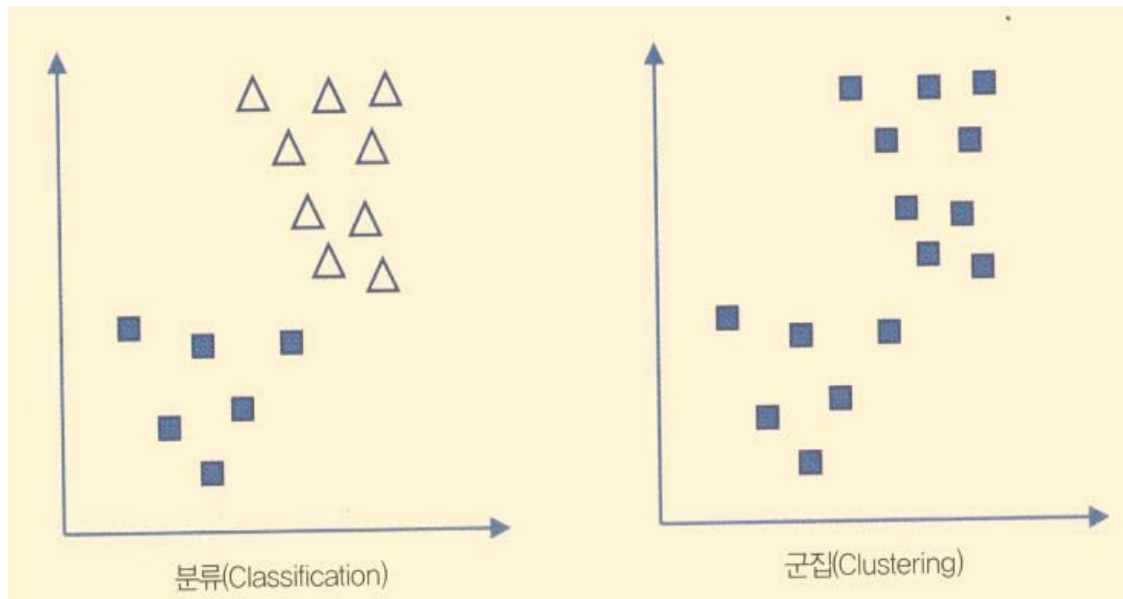
- 대규모 데이터 세트에서 유사항목을 가진 그룹을 자동으로 검출하는 군집기법 소개
- **Supervised Learning (지도학습)**
 - 학습데이터에 레이블이 있는 경우
 - Classification
 - 결과값이 고정
 - KNN, Support Vector Machine, Decision Tree
 - Prediction
 - 결과값이 데이터세트의 범위 내 어떠한 값도 가능
 - Regression
- **Unsupervised Learning (비지도학습)**
 - 학습데이터에 레이블이 없는 경우
 - Clustering
 - 분할 기반 군집 모델: k-means, k-medoids, DBSCAN 등
 - 계층적 군집 모델

Supervised learning

An example training set for four visual categories.



- 분류(Classification)와 군집(Clustering) 비교
 - 분류 모델에서는 레이블이 있으나,
 - 군집 모델에서는 레이블이 없음



2. Word Vectors

- 군집용 데이터 준비
 - 상위 블로거 100명의 집합
 - 블로그 단어 빈도의 일부분

	"china"	"kids"	"music"	"yahoo"
Gothamist	0	3	3	0
GigaOM	6	0	0	2
Quick Online Tips	0	2	2	22

- 피드 내 단어 수 세기
 - 거의 모든 블로그는 RSS feeds를 통해 온라인에서 읽을 수 있음
 - RSS feeds
 - 간단한 XML 문서로, 블로그에 대한 정보 포함
 - Universal Feed Parser
 - 제목, 링크, 게시글을 RSS나 Atom feed를 통해 쉽게 획득케 함
 - <http://www.feedparser.org>
 - RSS와 Atom feed는 제목과 엔트리 목록으로 구성
 - 각 엔트리는 summary나 description로 구성

```
import feedparser
import re

# Returns title and dictionary of word counts for an RSS feed
def getwordcounts(url):
    # Parse the feed
    d=feedparser.parse(url)
    wc={ }

    # Loop over all the entries
    for e in d.entries:
        if 'summary' in e: summary=e.summary
        else: summary=e.description

        # Extract a list of words
        words=getwords(e.title+' '+summary)
        for word in words:
            wc.setdefault(word, 0)
            wc[word]+=1
    return d.feed.title, wc

def getwords(html):
    # Remove all the HTML tags
    txt=re.compile(r'<[^>]+>').sub('', html)

    # Split words by all non-alpha characters
    words=re.compile(r'[^A-Za-z]+').split(txt)

    # Convert to lowercase
    return [word.lower() for word in words if word!='']
```

- 피드 목록

- 직접 만들거나,
- feedlist.txt 사용
 - 99개의 RSS URL 포함

- 피드별로 루프를 돌고 데이터 세트를 생성하는 코드

- generatefeedvector.py

```
apcount={}
wordcounts={}
feedlist=[line for line in file('feedlist.txt')]
for feedurl in feedlist:
    try:
        title,wc=getwordcounts(feedurl)
        wordcounts[title]=wc
        for word,count in wc.items():
            apcount.setdefault(word,0)
            if count>1:
                apcount[word]+=1
    except:
        print 'Failed to parse feed %s' % feedurl

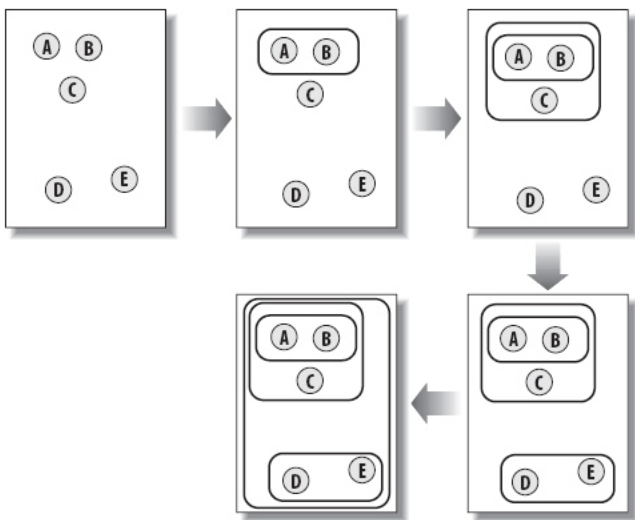
wordlist=[]
for w,bc in apcount.items():
    frac=float(bc)/len(feedlist)
    if frac>0.1 and frac<0.5:
        wordlist.append(w)

out=file('blogdata1.txt','w')
out.write('Blog')
for word in wordlist: out.write('\t%s' % word)
out.write('\n')
for b,wc in wordcounts.items():
    blog = b.encode('utf-8')
    print blog
    out.write(blog)
    for word in wordlist:
        if word in wc: out.write('\t%d' % wc[word])
        else: out.write('\t0')
    out.write('\n')
```

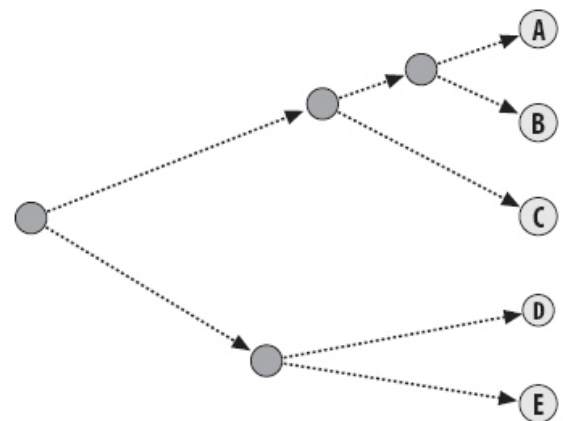
3. 계층적 군집화

- 가장 유사한 두 그룹을 계속 병합하는 방식으로 그룹 계층 생성
- 그룹들은 한 개 항목으로 시작
 - 개별 블로그에 해당
- 모든 그룹 쌍 간의 거리 계산 후 병합하여 새로운 그룹 생성
- 이 과정을 한 개 그룹만 남을 때까지 반복

- 계층적 군집화
 - 항목들의 상대 위치가 유사도라 가정



계층적 군집화 실행 모습



계층적 군집화를 시각화한 계통도 (dendrogram)

- cluster.py

- 블로그 데이터 세트를 군집화해서 블로그 계층도 생성 → 주제별 그룹

- Readfile

- 데이터 파일을 읽는 함수

- Pearson

- 두 개의 숫자목록을 취해 그들간 상관점수 계산
 - 많은 단어가 있으므로 피어슨 계수가 적합
 - 두 항목이 유사할 수록 적은 거리 값을 갖게 하기 위해 마지막 코드에 1.0에서 피어슨 계수 값을 뺀

- Bicluster

- 클러스터의 위치 정보 표현

- Hcluster

- 군집에서 가장 유사한 쌍을 합쳐 한 개의 단일 군집 만듦
 - 새로운 군집용 데이터는 앞의 두 군집들에 대한 데이터의 평균값
 - 이 과정을 단 한 개의 군집만 남을 때까지 반복 수행
 - 시간이 많이 걸림
 - 계산한 상관계수 계산 값을 저장해 두고 재활용

- Printclust

- 군집 트리를 재귀적으로 방문하고 파일시스템 계층과 같이 출력하는 함수
 - 큰 데이터 세트인 경우 보기 불편

```

>>> from clusters import *
>>> blognames, words, data = readfile('blogdata1.txt')
>>> clust = hcluster(data)
>>> printclust(clust, labels=blognames)
-
-
    SpikedHumor - Today's Videos and Pictures
-
    Dave Shea's mezzoblu
-
-
        GoFugYourself
        Cool Hunting
-
        Instapundit.com (v.2)
-
-
            The Full Feed from HuffingtonPost.com
            Wired Top Stories
-
            Eschaton
-
-
                Wonkette: The D.C. Gossip
                The Blotter
-
-
                    Boing Boing
-
                        The Superficial - Because You're Ugly

```

4. 계통도 출력

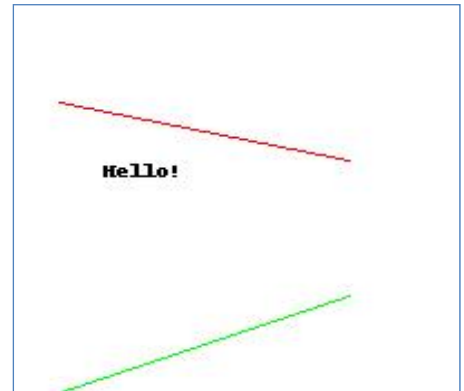
- 계통도
 - 해석이 훨씬 편리
 - 비교적 적은 공간에 많은 정보 담음
 - 그래픽(JPG)으로 저장
 - <http://pythonware.com>에서 파이썬 이미지 라이브러리 다운로드 : p.382
 - 텍스트와 선이 들어간 이미지 쉽게 생성

Python Imaging Library (PIL)

- 다운로드

- <http://www.pythonware.com/products/pil/>

```
>>> from PIL import Image, ImageDraw
>>> img = Image.new('RGB', (200, 200), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.line((20, 50, 150, 80), fill=(255, 0, 0))
>>> draw.line((150, 150, 20, 200), fill=(0, 255, 0))
>>> draw.text((40, 80), 'Hello!', (0, 0, 0))
>>> img.save('test.jpg', 'JPEG')
```



- Cluster.py

- Getheight

- 주어진 군집의 전체 높이 리턴하는 함수
 - 이미지의 전체 높이 결정하기 위해

- Getdepth

- 최상위 루트의 깊이 리턴하는 함수

- Drawdendrogram

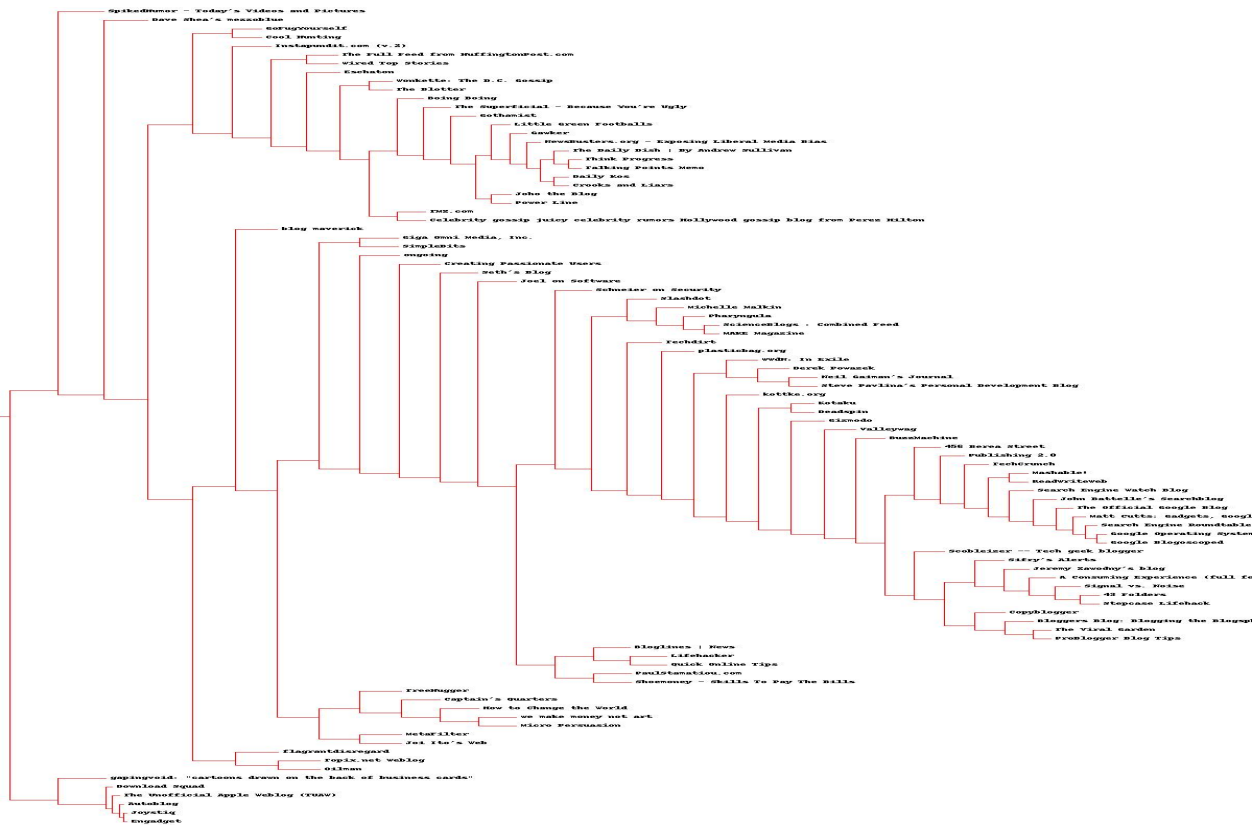
- 최종 군집마다 높이 20픽셀과 고정된 폭을 가진 이미지 생성

- Drawnode

- 자식 노드들의 높이를 계산하여 자신이 어디에 위치하는지 위치 계산
 - 선길이가 짧을수록 두 군집이 좀 더 유사함

- 블로그 군집 계통도

```
>>> drawdendrogram(clust, blognames, jpeg='blogclust.jpg')
```

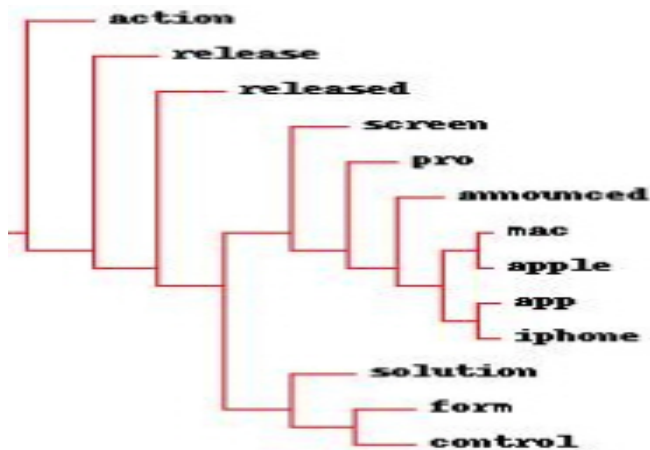


5. 세로줄 군집화 (Column Clustering)

- 블로그 데이터 세트
 - 가로줄 : 블로그, 세로줄 : 단어
- 이전 함수 그대로 사용
 - 세로줄이 가로줄이 되도록 전체 데이터 세트 회전
 - 특정 단어가 각 블로그에서 몇 회 출현했는가

```
def rotatematrix(data):
    newdata=[]
    for i in range(len(data[0])):
        newrow=[data[j][i] for j in range(len(data))]
        newdata.append(newrow)
    return newdata
```

```
>>> blognames, words, data = readfile('blogdata1.txt')
>>> rdata = rotatematrix(data)
>>> wordclust = hcluster(rdata)
>>> drawdendrogram(wordclust, labels=words, jpeg='wordclust.jpg')
```

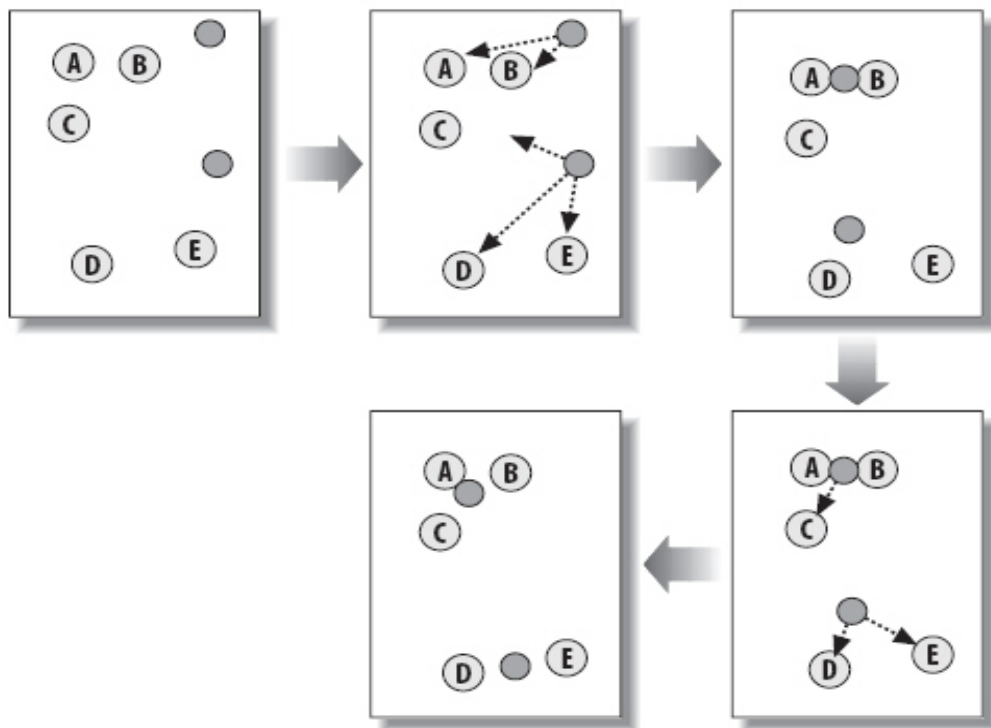


- 변수에 비해 항목들이 훨씬 더 많은 경우
 - 의미없는 군집들이 증가
 - 블로그보다 단어가 더 많음
 - 단어 군집화보다 블로그 군집화에서 더 의미있는 패턴 볼 수 있음

6. K-평균 군집화

- 계층적 군집화 기법
 - 결과를 트리 형태로 표현
 - 데이터를 뚜렷한 그룹으로 나누지 못함
 - 많은 계산 필요 : 큰 데이터 세트에서는 느림
- K-means Clustering
 - 사전에 생성할 군집의 개수(k) 지정
 - 무작위로 선정된 k개의 중심점 선정
 - 이 점에서 가장 근접한 항목들을 할당
 - 할당 후 할당된 노드들의 평균위치로 중심점 이동
 - 할당이 더 이상 없을 때까지 재할당 수행

- 2개의 군집을 가진 K평균 군집화 과정



- Cluster.py

- Kcluster

- 각 변수의 허용범위 내에서 군집들을 무작위로 생성
 - 리턴 결과는 매번 달라짐
 - 매 반복마다 가로줄을 중심점 중의 하나에 할당
 - 중심점 데이터를 모든 할당점들의 평균값으로 갱신
 - 이전과 할당이 동일하면 이 과정 종료
 - K개의 목록 리턴
 - 계층적 군집화에 비해 빠르게 동작

- Kclust에 각 군집의 ID 목록이 담김
- K 값을 변경하여 실행해 볼 것

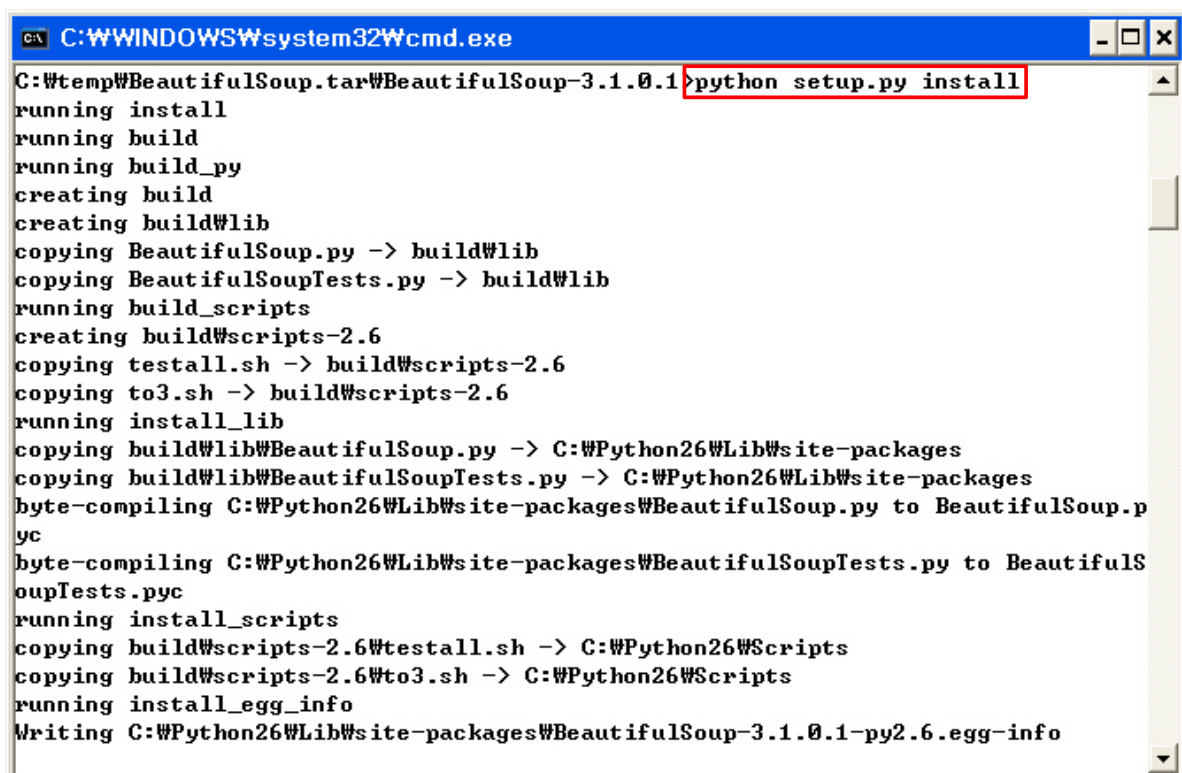
```
>>> kclust = kcluster(data, k=10)
Iteration 0
Iteration 1
Iteration 2
Iteration 3
>>> [blognames[r] for r in kclust[0]]
['Eschaton', 'Wonkette: The D.C. Gossip', 'Joho the Blog', 'Power Line', '
Michelle Malkin', 'Gothamist', 'Boing Boing', 'Crooks and Liars', 'Think P
rogress', 'Little Green Footballs', 'Instapundit.com (v.2)', 'NewsBusters.
org - Exposing Liberal Media Bias', 'The Daily Dish | By Andrew Sullivan',
'Gawker', 'Talking Points Memo']
>>> [blognames[r] for r in kclust[1]]
['Mashable!', 'we make money not art', 'ReadWriteWeb', 'Stepcase Lifehack'
, 'Giga Omni Media, Inc.', 'ScienceBlogs : Combined Feed', 'Lifehacker', '
SimpleBits', 'Micro Persuasion', 'MAKE Magazine', 'blog maverick', 'Techdi
rt', '456 Berea Street', 'Pharyngula', 'MetaFilter', 'ongoing', 'Oilman',
'Joi Ito's Web', 'plasticbag.org', 'WwdN: In Exile']
```

7. 선호도 군집

- 소셜 네트워크 사이트
 - 사람들이 자발적으로 공헌한 대량의 데이터가 발생
 - Zebo 사이트 : <http://www.zebo.com>
 - 가진 물건 목록이나 가지고 싶은 물건 목록 구축
 - 선호도 정보로 활용 가능
- 데이터 얻기와 준비
 - Zebo 사이트에서 페이지 다운로드 받아 파싱 후 데이 터 세트 추출하거나,
 - 다운로드
 - <http://kiwitobes.com/clusters/zebo.txt>

- Beautiful Soup

- 웹 페이지를 파싱하고 구조적 표현을 생성하는 라이브러리
- 유형, ID, 속성으로 페이지 요소에 접근 가능
 - 내용을 문자열로 얻음
- 다운로드
 - <http://crummy.com/software/BeautifulSoup>



```
C:\WINDOWS\system32\cmd.exe
C:\temp\BeautifulSoup.tar\BeautifulSoup-3.1.0.1>python setup.py install
running install
running build
running build_py
creating build
creating build\lib
copying BeautifulSoup.py -> build\lib
copying BeautifulSoupTests.py -> build\lib
running build_scripts
creating build\scripts-2.6
copying testall.sh -> build\scripts-2.6
copying to3.sh -> build\scripts-2.6
running install_lib
copying build\lib\BeautifulSoup.py -> C:\Python26\Lib\site-packages
copying build\lib\BeautifulSoupTests.py -> C:\Python26\Lib\site-packages
byte-compiling C:\Python26\Lib\site-packages\BeautifulSoup.py to BeautifulSoup.pyc
byte-compiling C:\Python26\Lib\site-packages\BeautifulSoupTests.py to BeautifulSoupTests.pyc
running install_scripts
copying build\scripts-2.6\testall.sh -> C:\Python26\Scripts
copying build\scripts-2.6\to3.sh -> C:\Python26\Scripts
running install_egg_info
Writing C:\Python26\Lib\site-packages\BeautifulSoup-3.1.0.1-py2.6.egg-info
```

– 수프(soup)

- BeautifulSoup가 웹 페이지를 표현하는 방식
- 예) 'a'와 같은 태그 유형으로 수프 호출
 - 해당 유형을 가진 객체 목록 리턴

```
>>> import urllib2
>>> from bs4 import BeautifulSoup
>>> c = urllib2.urlopen('http://www.daegu.ac.kr/')
>>> soup = BeautifulSoup(c.read())
>>> frames = soup('frame')
>>> len(frames)
2
>>> frames[1]
<frame frameborder="NO" name="mainFrame" scrolling="YES" src="http://www.daegu.ac.kr/web/index/index/index.asp" title="메인콘텐츠">
<!--frame name="mainFrame" src="/index_sugang.htm" frameborder="NO" scrolling="YES"-->
</frame>
>>> frames[1]['src']
u'http://www.daegu.ac.kr/web/index/index/index.asp'
```

• 거리 지표 결정

– 피어슨 상관 지표

- 값들이 실제 단어 출현 횟수인 블로그 데이터 세트에서 잘 동작

– 타니모토 계수(Tanimoto coefficient)

- 두 집합 간의 유사도를 측정하는 지표
- 두 항목을 원하는 사람간의 중첩도를 측정하는 지표로 활용

$$T = \frac{N_c}{(N_a + N_b - N_c)}$$

N_a 는 A에 있는 항목 수, N_b 는 B에 있는 항목 수, 그리고 N_c 는 교집합 C에 있는 항목 수

Example

```
A = { car, train, aircraft, ship }  
B = { car, motorcycle, train }
```

$$T = \frac{N_c}{(N_a + N_b - N_c)}$$

N_a 는 A에 있는 항목 수, N_b 는 B에 있는 항목 수, 그리고 N_c 는 교집합 C에 있는 항목 수

교집합 C는 { car, train }

타니모토 계수는 $2/(4+3-2) = 2/5 = 0.4$

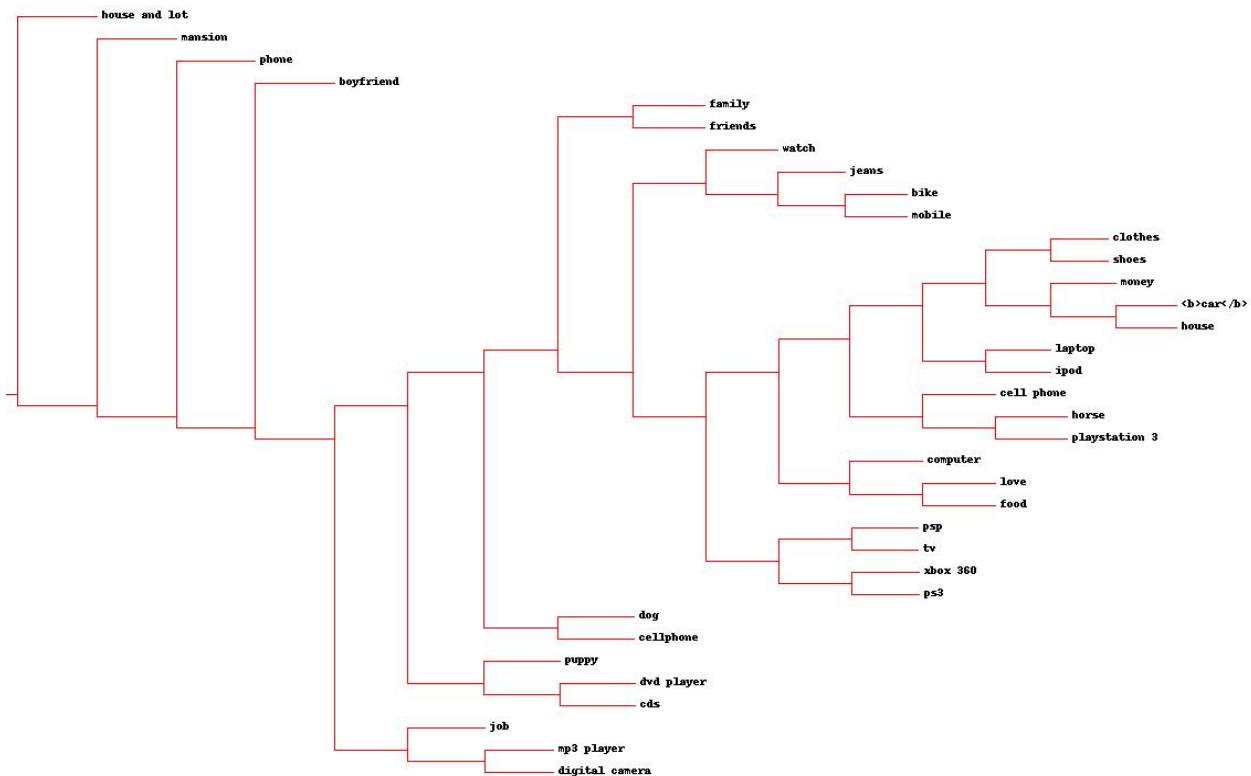
```
def tanamoto(v1,v2):  
    c1,c2,shr=0,0,0  
  
    for i in range(len(v1)):  
        if v1[i]!=0: c1+=1 # in v1  
        if v2[i]!=0: c2+=1 # in v2  
        if v1[i]!=0 and v2[i]!=0: shr+=1 # in both  
  
    return 1.0-(float(shr)/(c1+c2-shr))
```

- 결과 군집화
– 동일한 함수 사용

```
>>> wants, people, data = readfile('zebo.txt')  
>>> wants  
['bike', 'clothes', 'dvd player', 'phone', 'cell phone', 'dog', 'xbox 360', 'boyfriend', 'watch', 'laptop', 'love', '<b>car</b>', 'shoes', 'jeans', 'money', 'ps3', 'psp', 'puppy', 'house and lot', 'tv', 'family', 'food', 'house', 'horse', 'mobile', 'cds', 'playstation 3', 'mp3 player', 'ipod', 'digital camera', 'mansion', 'cellphone', 'computer', 'job', 'friends']  
>>> people  
['U0', 'U1', 'U2', 'U3', 'U4', 'U5', 'U6', 'U7', 'U8', 'U9', 'U10', 'U11', 'U12', 'U13', 'U14', 'U15', 'U16', 'U17', 'U18', 'U19', 'U20', 'U21', 'U22', 'U23']
```


- 원하는 소유물의 군집

```
>>> clust = hcluster(data, distance=tanamoto)
>>> drawdendrogram(clust, wants)
```



8. 2차원으로 데이터 보기

• 다차원 비례 축소법

(multidimensional scaling)

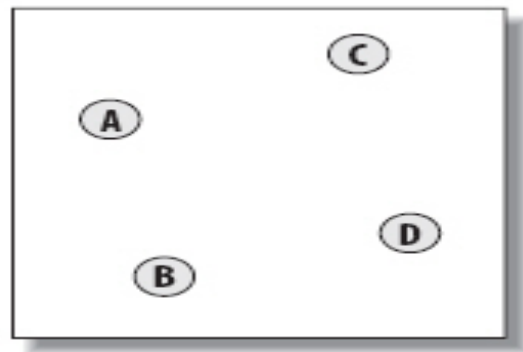
- 다차원 데이터 세트에 대한 2차원 표현을 찾는 데 사용
- 모든 항목 쌍의 차이값을 구하고 이 값과 항목 간 거리가 일치하도록 도표 만들기

- 블로그 데이터 세트의 경우 피어슨 상관 지표를 사용하여 항목간 비교 수행

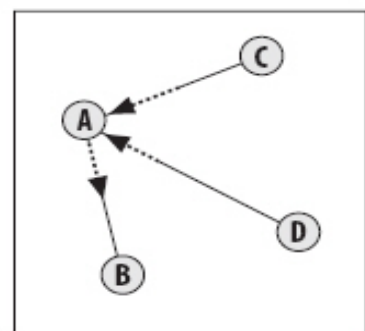
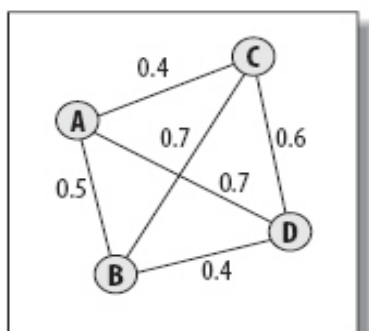
Table 3-2. Sample distance matrix

	A	B	C	D
A	0.0	0.2	0.8	0.7
B	0.2	0.0	0.9	0.8
C	0.8	0.9	0.0	0.1
D	0.7	0.8	0.1	0.0

1. 모든 항목을 2차원 도표 안에 임의로 위치



2. 모든 항목 간 현재 거리를 실제 거리(제공차의 합)를 사용하여 계산
3. 모든 항목 쌍에 대해 목표 거리를 현재 거리와 비교하고 오류 값 계산
4. 오류 값을 줄이는 방향으로 항목 위치 조정
 - 모든 항목은 밀고 당기는 다른 모든 항목들의 조합에 의해 이동
5. 항목을 움직여도 오류 값이 줄어들지 않을 때 까지 이 과정 반복



- Cluster.py

- Scaledown

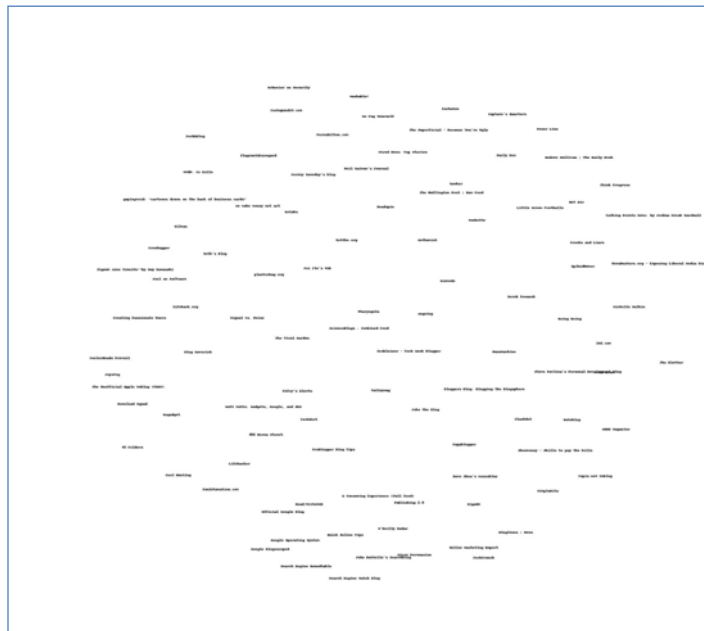
- 데이터 벡터를 입력으로 받아 두 개의 세로줄 리턴
 - 세로줄 : 2차원 도표 안에서 항목들의 X, Y 좌표 값

- Draw2d

- 위의 결과를 PIL을 이용해서 모든 항목들의 라벨들을 새로운 좌표축에 출력

```
>>> blognames, words, data = readfile('blogdata.txt')
>>> coords = scaledown(data)
4544.95881636
3548.40985742
3467.64057261
3422.73085624

>>> draw2d(coords, blognames, jpeg='blogs2d.jpg')
```



9. 군집 가능한 다른 것들

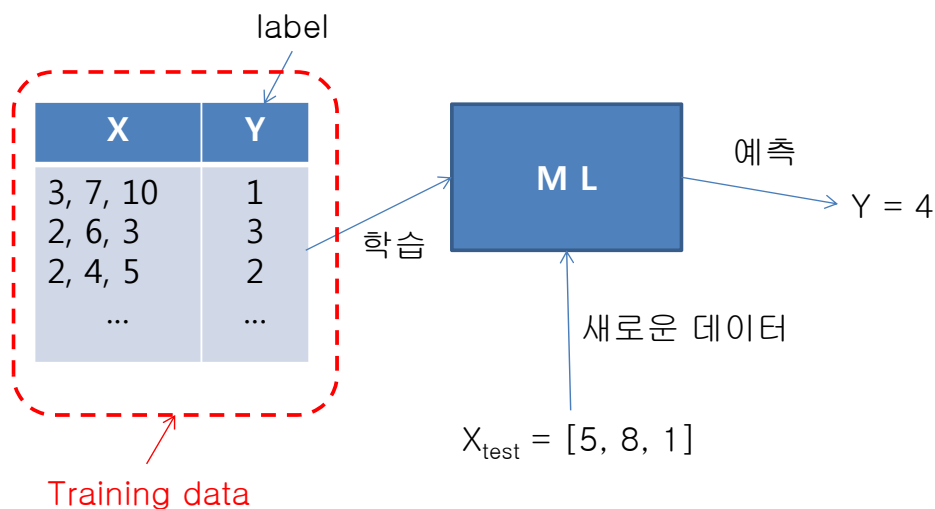
- 생각해 보기

Classification:
Linear Regression

References

- Hackeling, Gavin (2014). **Mastering Machine Learning with scikit-learn**. Packt Publishing.
- HKUST Prof. Kim's Lectures
 - <http://hunkim.github.io/ml/>
- Andrew Ng's ML class
 - <https://class.coursera.org/ml-003/lecture>
 - <http://www.holehouse.org/mlclass/> (note)
- Convolutional Neural Networks for Visual Recognition.
 - <http://cs231n.github.io/>
- Tensorflow
 - <https://www.tensorflow.org>
 - <https://github.com/aymericdamien/TensorFlow-Examples>

Training data set



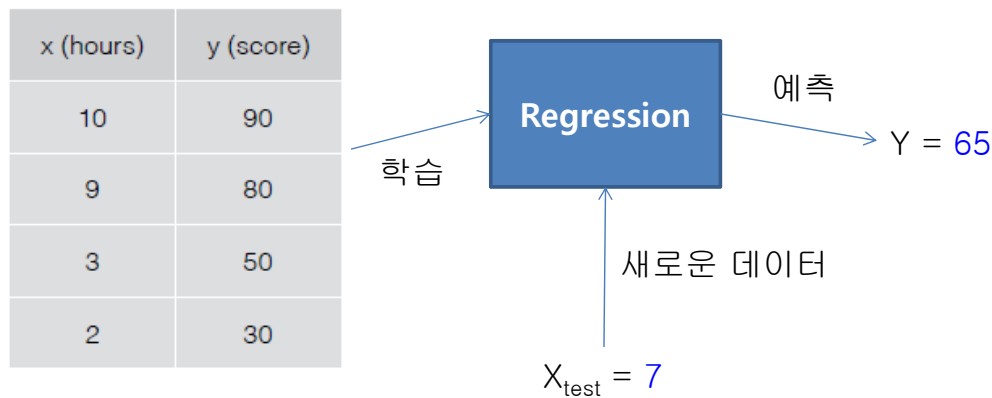
Training data and test data

- **Training set**
 - Composed of past observations of explanatory variables and their corresponding response variables
- **Test set**
 - Similar collection of observations that is used to evaluate the performance of the model

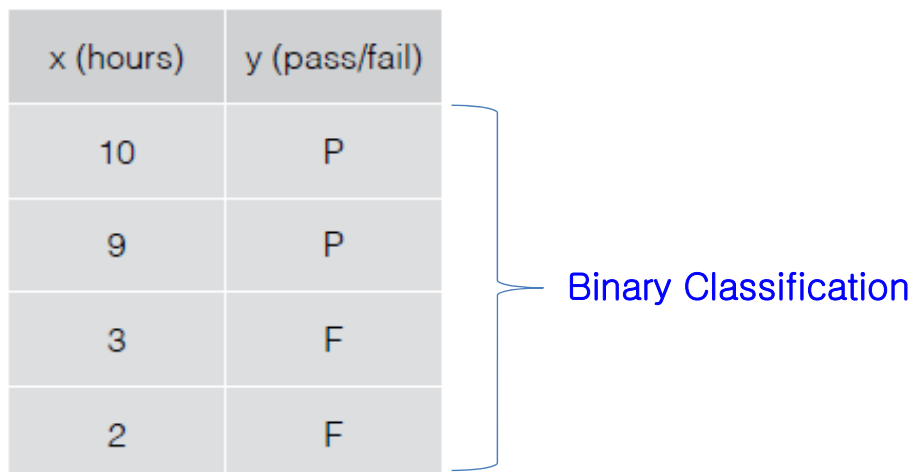
Types of supervised learning

- Predicting final exam score based on time spent
 - regression
- Pass/non-pass based on time spent
 - binary classification
- Letter grade (A, B, C, D and F) based on time spent
 - multi-label classification

Predicting final exam score based on time spent



Pass/non-pass based on time spent

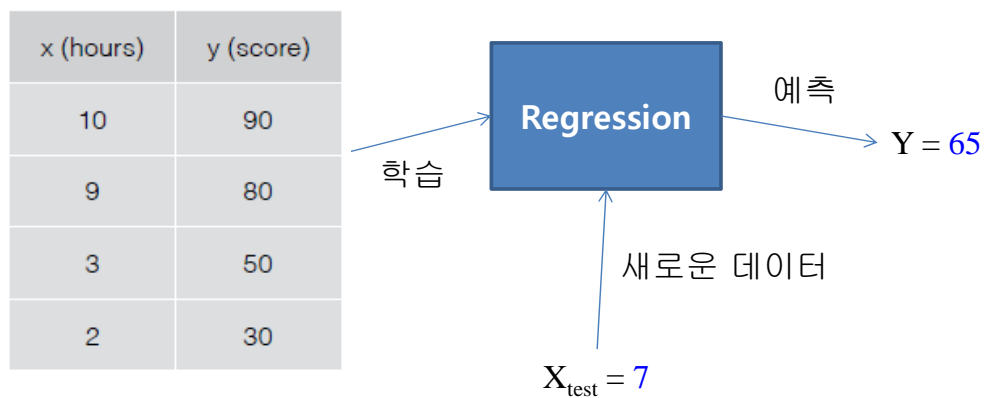


Letter grade (A, B, ...) based on time spent

x (hours)	y (grade)
10	A
9	B
3	D
2	F

Multi-label Classification

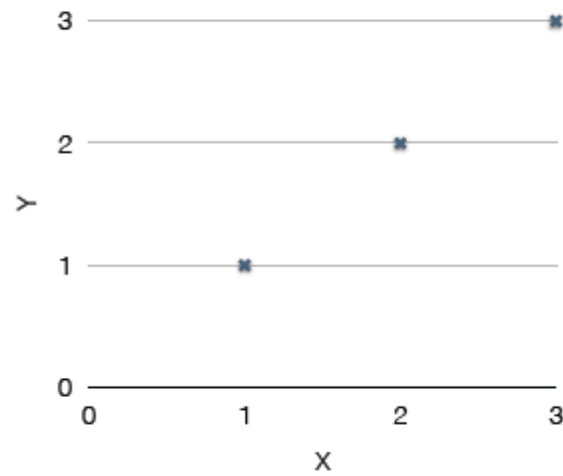
Predicting exam score: Regression



Regression

x	Y
1	1
2	2
3	3

data

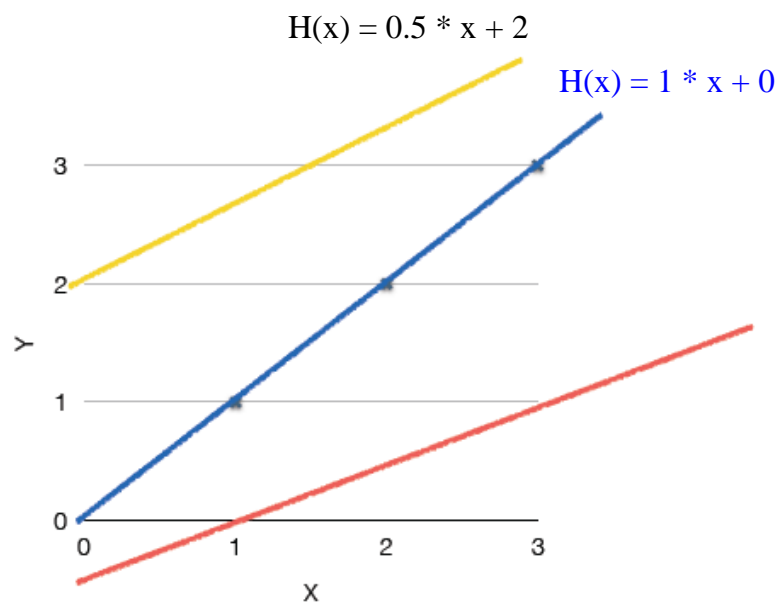


presentation

(Linear) Hypothesis

$$\underline{H(x)} = Wx + b$$

가설



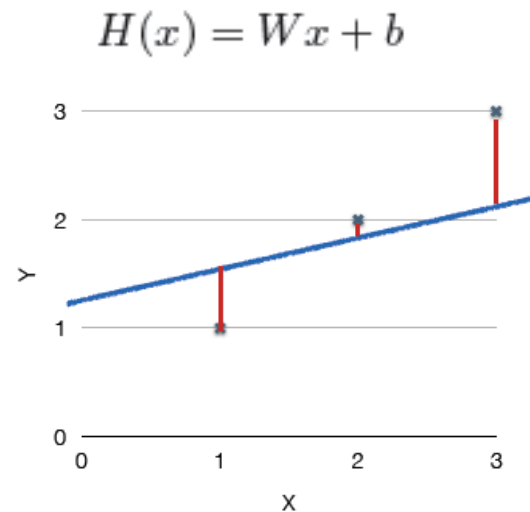
Which hypothesis is better?

- Cost function (= Loss function)

$$\frac{H(x) - y}{\text{예측값} \quad \text{실제값}} \rightarrow \times$$

$$(H(x) - y)^2 \rightarrow \bigcirc$$

양수와 음수를 통일시키고,
차이를 증폭시키는 역할

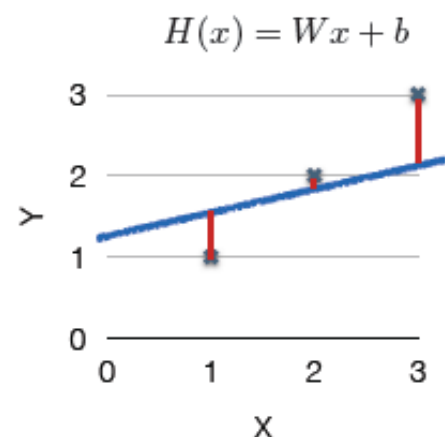


Cost function

$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

m: 학습데이터의 개수



Cost function

$$cost = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$\underset{W, b}{\text{minimize}} \text{cost}(W, b)$$

Cost 함수의 값을 최소화하는 W, b를 구하는 것이 학습의 목표

An introduction to scikit-learn

- **Scikit-learn**

- Open source machine learning libraries for Python

- NumPy

- support efficient operations on large arrays and multidimensional matrices

- Matplotlib

- provides visualization tools

- SciPy

- provides modules for scientific computing

- Wraps some popular implementations of machine learning algorithms
 - such as LIBSVM and LIBLINEAR
- Licensed under the permissive BSD license
 - scikit-learn can be used in commercial applications without restrictions

Installing scikit-learn

- **Python**
 - <https://www.python.org/downloads/>
- **Scikit-learn**
 - <http://scikit-learn.org/stable/install.html>
 - 설치
 - Python2.7용(32bit) 5개 실행파일 설치 후,
 - pip install six
 - pip install python-dateutil
 - pip install pyparsing

```
C:\> 관리자: C:\windows\system32\cmd.exe - python

C:\Users\Administrator> pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.17.1-cp27-cp27m-win32.whl (3.1MB)
    100% |#####| 3.1MB 303kB/s
Installing collected packages: scikit-learn
Successfully installed scikit-learn-0.17.1

C:\Users\Administrator> python
Enthought Canopy Python 2.7.11 | 32-bit | (default, Jun 11 2016, 11:34:14) [MSC
v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sklearn
>>> sklearn.__version__
'0.17.1'
>>> _
```

Installing pandas and matplotlib

- **Pandas**

- Open source library that provides data structures and analysis tools for Python
- <http://pandas.pydata.org/getpandas.html>

- **Matplotlib**

- library used to easily create plots, histograms, and other charts with Python
- <http://matplotlib.org/downloads.html>

Lab with scikit-learn

- **Simple linear regression**

Ex) Write a program with scikit-learn

- predict the price of a pizza given its size
- Training data

Training instance	X		Y
	Diameter (in inches)		Price (in dollars)
1	6		7
2	8		9
3	10		13
4	14		17.5
5	18		18

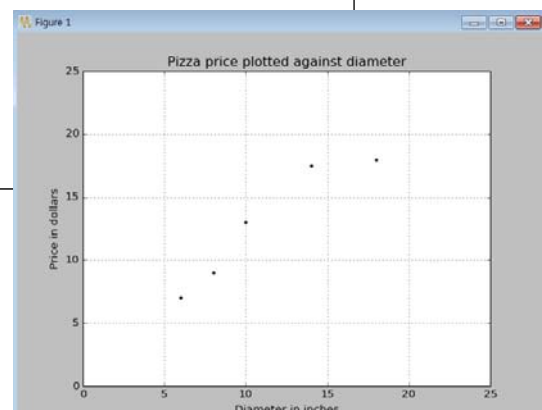
- visualize our training data by plotting it on a graph using matplotlib:

sklearn1.py

```
import matplotlib.pyplot as plt

X = [[6], [8], [10], [14], [18]]
Y = [[7], [9], [13], [17.5], [18]]

plt.figure()
plt.title('Pizza price plotted against diameter')
plt.xlabel('Diameter in inches')
plt.ylabel('Price in dollars')
plt.plot(X, Y, 'k.')
plt.axis([0, 25, 0, 25])
plt.grid(True)
plt.show()
```



- pizza-price predictor program using linear regression

sklearn2.py

```
from sklearn.linear_model import LinearRegression

# training data
X = [[6], [8], [10], [14], [18]]
Y = [[7], [9], [13], [17.5], [18]]

# create and fit the model
model = LinearRegression()
model.fit(X, Y)

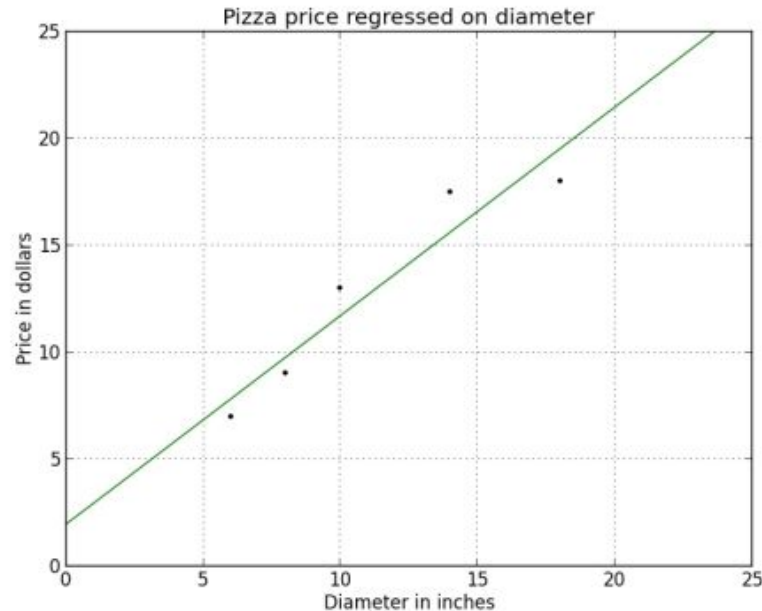
# predict
print '12" pizza should cost: $%.2f' % model.predict([12])
```

12" pizza should cost: \$13.68

- **sklearn.linear_model.LinearRegression** class is an **estimator**
 - Estimators predict a value based on the observed data
- In scikit-learn, all estimators implement the **fit()** and **predict()** methods
 - **fit()** method is used to learn the parameters of a model
 - **predict()** method is used to predict the value of a response variable for an explanatory variable using the learned parameters

- **fit()** method of LinearRegression learns the parameters of the following model

$$y = \alpha + \beta x$$



How to minimize cost

- Hypothesis and Cost

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

cost를 최소화하는 W, b 찾기

- Simplified hypothesis

$$H(x) = Wx \quad \leftarrow \text{문제를 간략화 하기 위해 } b \text{를 생략}$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

What $cost(W)$ looks like?

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- $W=1, cost(W)=?$

$$((1 * 1 - 1)^2 + (1 * 2 - 2)^2 + (1 * 3 - 3)^2) / 3 = 0$$

What $cost(W)$ looks like?

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- $W=1, cost(W)=0$

$$\frac{1}{3}((1 * 1 - 1)^2 + (1 * 2 - 2)^2 + (1 * 3 - 3)^2)$$

- $W=0, cost(W)=4.67$

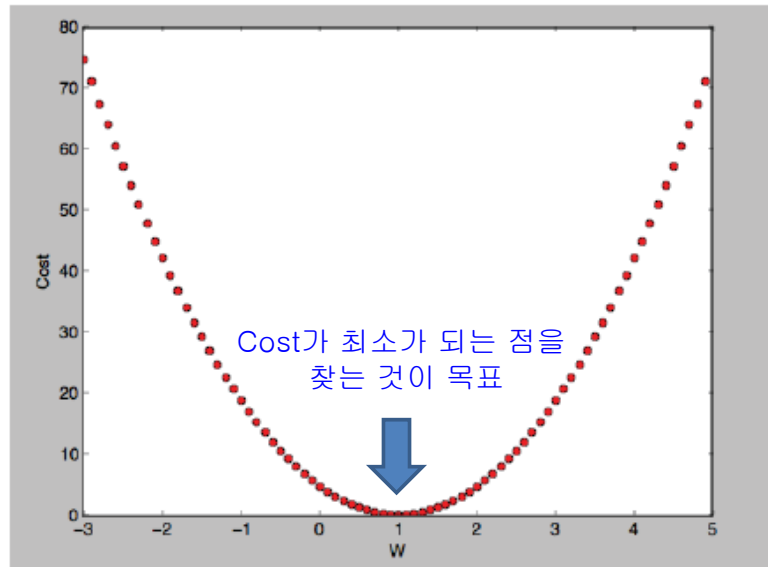
$$\frac{1}{3}((0 * 1 - 1)^2 + (0 * 2 - 2)^2 + (0 * 3 - 3)^2)$$

- $W=2, cost(W)=?$

How to minimize cost?

- $W=1, \text{cost}(W)=0$
- $W=0, \text{cost}(W)=4.67$
- $W=2, \text{cost}(W)=4.67$
- ...

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

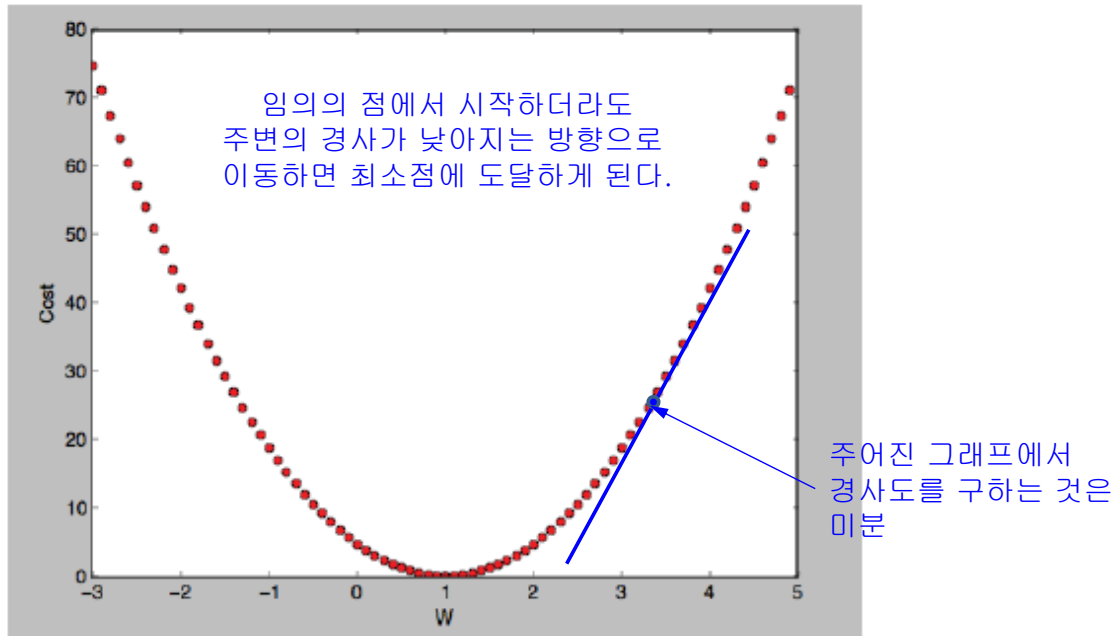


Gradient descent algorithm

- 경사를 따라 내려가는 알고리즘
- Minimize cost function
- Gradient descent is used many minimization problems
- For a given cost function, ***cost (W, b)***, it will find W, b to minimize cost
- It can be applied to more general function:
 - $\text{cost}(w1, w2, \dots)$

How it works?

- How would you find the lowest point?



How it works?

- Start with initial guesses
 - Start at 0.0 (or any other value)
 - Keeping changing W and b a little bit to try and reduce $\text{cost}(W, b)$
- Each time you change the parameters, you select the gradient which reduces $\text{cost}(W, b)$ the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
 - Where you start can determine which minimum you end up

Formal definition

$$\text{cost}(W) = \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$



$$\text{cost}(W) = \frac{1}{\underline{2m}} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

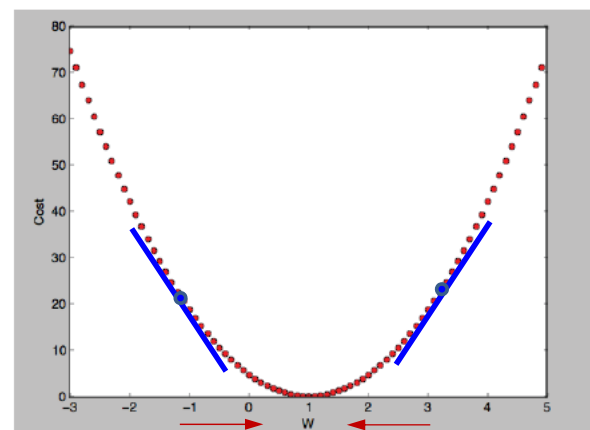
- 미분을 쉽게 적용하기 위해 cost함수를 2로 나눔
- $1/m$ 을 최소화하는 것이나 $1/2m$ 을 최소화하는 것이나 동일하기 때문

$$\text{cost}(W) = \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{\partial}{\partial W} \text{cost}(W)$$

Learning rate, constant
즉, W의 이동속도 또는 변화속도

cost(W)함수를 미분한 결과



미분한 값이
음수인 경우

미분한 값이
양수인 경우

- $\text{cost}(W)$ 함수를 미분하는 절차

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^m (W x^{(i)} - y^{(i)})^2$$

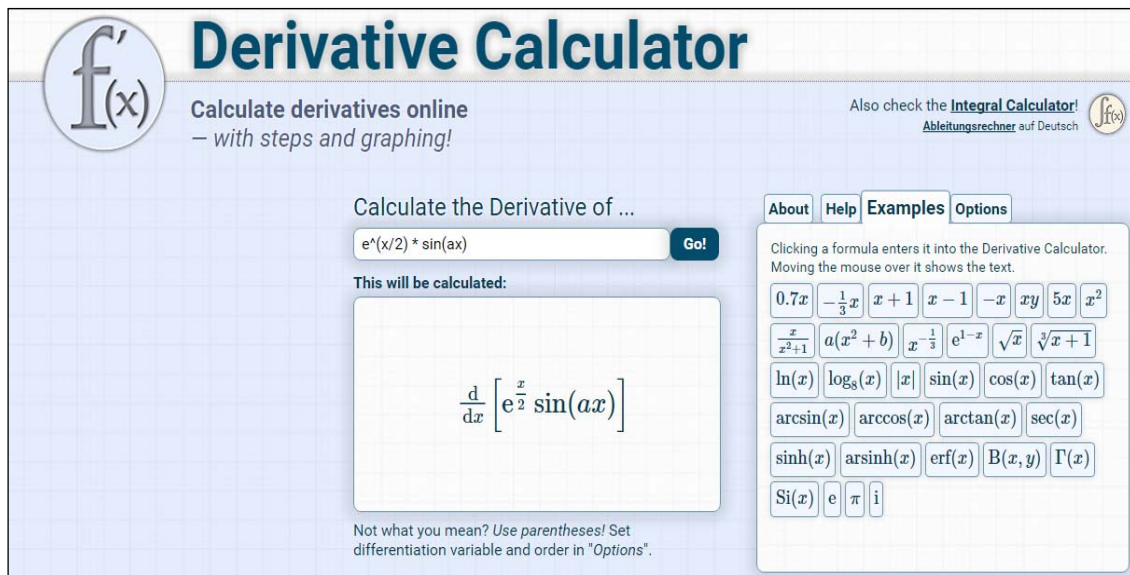
$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (W x^{(i)} - y^{(i)}) x^{(i)}$$

- W 학습 알고리즘 → Gradient descent algorithm

Derivative Calculator

- <http://www.derivative-calculator.net/>



Derivative Calculator
Calculate derivatives online
— with steps and graphing!

Also check the [Integral Calculator!](#)
Ableitungsrechner auf Deutsch

Calculate the Derivative of ...
e^{x/2} * sin(ax) **Go!**

This will be calculated:

$$\frac{d}{dx} \left[e^{\frac{x}{2}} \sin(ax) \right]$$

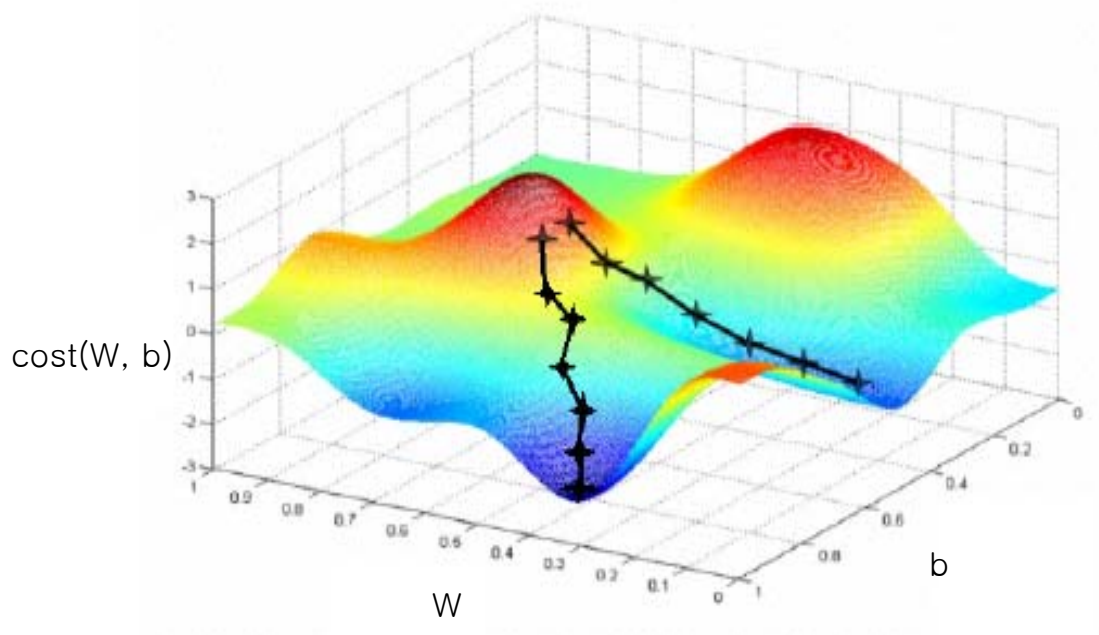
Not what you mean? Use parentheses! Set differentiation variable and order in "Options".

About Help Examples Options

Clicking a formula enters it into the Derivative Calculator.
Moving the mouse over it shows the text.

0.7x $-\frac{1}{3}x$ $x+1$ $x-1$ $-x$ xy $5x$ x^2
 $\frac{x}{x^2+1}$ $a(x^2+b)$ $x^{-\frac{1}{2}}$ e^{1-x} \sqrt{x} $\sqrt[3]{x+1}$
 $\ln(x)$ $\log_8(x)$ $|x|$ $\sin(x)$ $\cos(x)$ $\tan(x)$
 $\arcsin(x)$ $\arccos(x)$ $\arctan(x)$ $\sec(x)$
 $\sinh(x)$ $\operatorname{arsinh}(x)$ $\operatorname{erf}(x)$ $B(x,y)$ $\Gamma(x)$
 $\operatorname{Si}(x)$ e π i

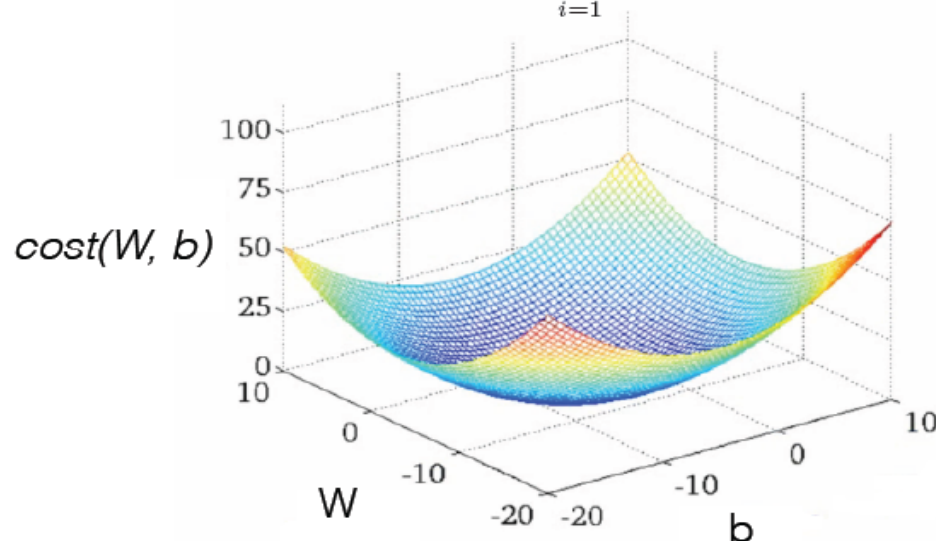
Linear regression이 제대로 동작하지 않는 경우



From: <http://www.holehouse.org/mlclass/>,

Convex function

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$



- Linear regression이 제대로 적용되려면
cost 함수가 convex function이 되는지를 확인해야 함

From: <http://www.holehouse.org/mlclass/>