


서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

# Classification

# Classification Algorithms

Decision Tree

Logistic Regression

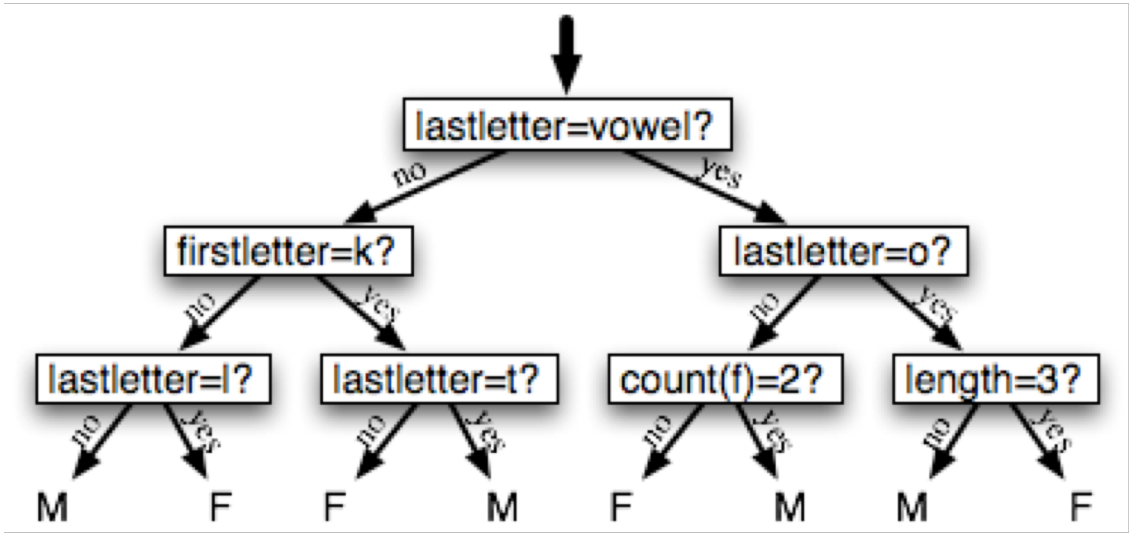
SVM

Random Forest

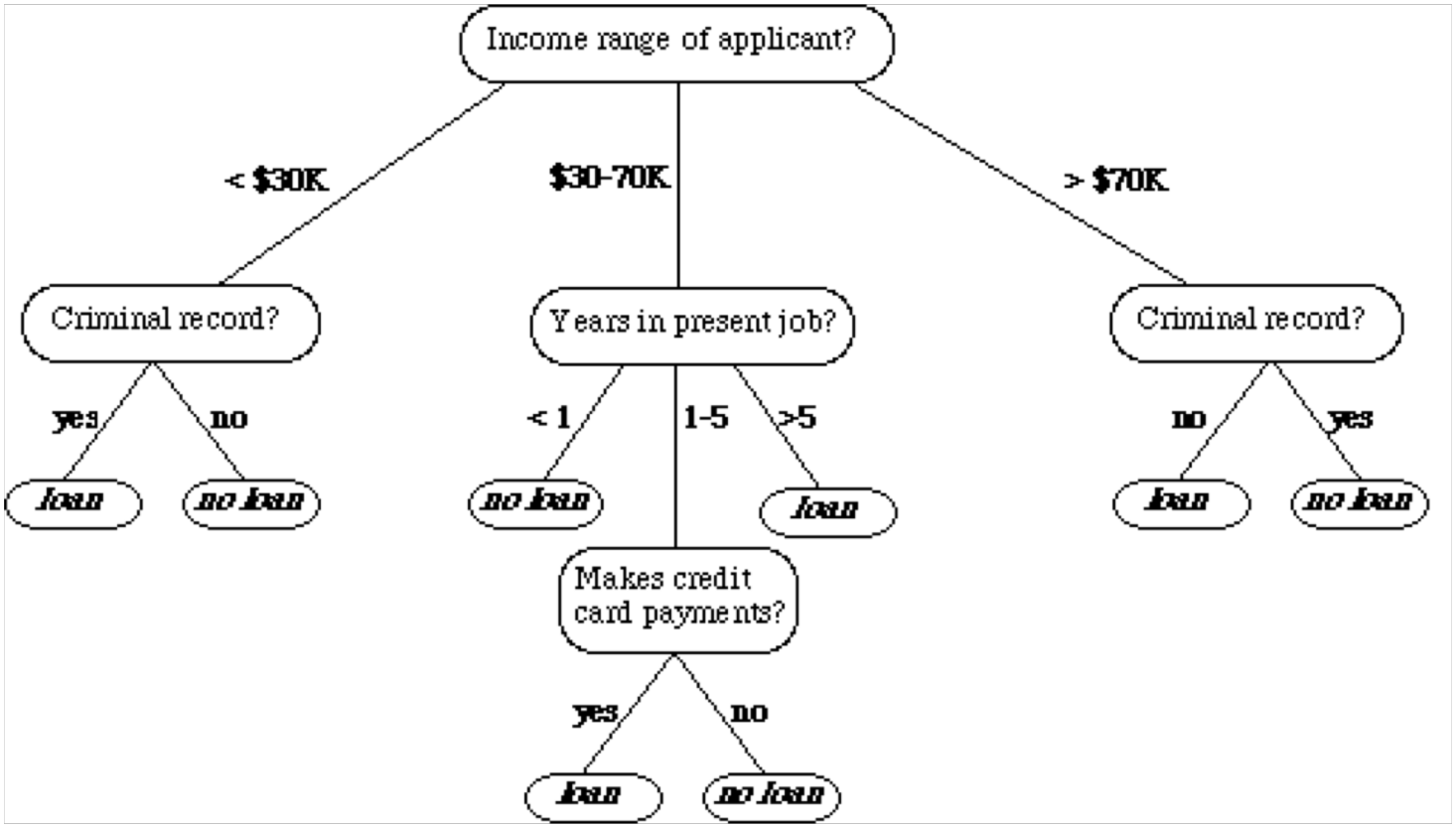
kNN

Ensemble Methods

# Male or Female?



# Decision Tree Classifier




무슨 feature를 먼저 고려해야 하는가?

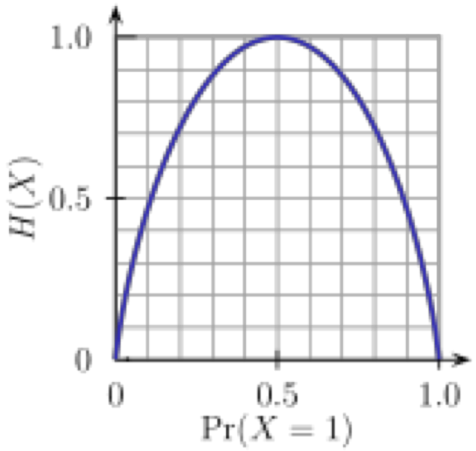
# Entropy – 무질서도

- 물리학에서 유래
  - 열역학 제 2 법칙: 모든 고립된 system에서는 entropy는 계속 증가한다.
  - 시스템은 entropy를 최대화하기 위해 계속 변화한다.
  - 방정리를 안하면 치우지 않는이상 (에너지를 투입하지 않는 이상) 계속 지저분해진다.

- 0과 1사이의 값을 가진다
  - 0: 무질서도가 없다. 모든것이 예측가능하다
    - 0, 0, 0, 0, 0, …., 0
  - 1: 완전 무질서. 예측할 수가 없다. 즉 모든 것이 랜덤이다.
    - 0, 1, 1, 0, 1, 0…

- Entropy를 줄일 수 있는 feature를 찾아서 data set을 자른다!

# Binary Entropy




$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

| | | |
|---|---|---|
| ['male', 'male', 'male', 'male'] | p = 0 | H = 0 |
| ['male', 'female', 'male', 'male'] | p = 1/4 | H = 0.81 |
| ['male', 'female', 'male', 'female'] | p = 1/2 | H = 1 |
| ['male', 'female', 'female', 'female'] | p = 3/4 | H = 0.81 |

# Information Gain – 무슨 feature를 먼저 고르는가?

- 나누기 전과 나누기 후의 Entropy 차이가 큰 것을 먼저고른다

| Film | Country | BigStar | Genre | Success/Failure |
|---|---|---|---|---|
| 1 | USA | Yes | SciFi | Success |
| 2 | USA | No | Comedy | Failure |
| 3 | USA | Yes | Comedy | Success |
| 4 | Europe | No | Comedy | Success |
| 5 | Europe | Yes | SciFi | Failure |
| 6 | Europe | Yes | Romance | Failure |
| 7 | Australia | Yes | Comedy | Failure |
| 8 | Brazil | No | SciFi | Failure |
| 9 | Europe | Yes | Comedy | Success |
| 10 | USA | Yes | Comedy | Success |

http://www.slideshare.net/akashag11111/id3-algorithm

# Is Movie Successful?

- Entropy(Movie Successful)
  - 5번 성공, 5번 실패
  - Entropy(Success) = −(5/10) * log2(5/10) − (5/10) * log2(5/10) = 1
  - 그러므로 현재 상황으로 봐서 영화가 성공하는가에 대한 entropy는 1이다

- Information Gain
  - Information Gain = (나누기 전의 entropy) − (나눈 후의 entropy)
  - 즉 나누어서 “무질서도”가 줄어 들었다 → 더 확실한 판단이 가능하다

# Information Gain for Country

- Information Gain for Country
  - Entropy(USA) = −(3/4) * log2(3/4) − (1/4) * log2(1/4) = 0.811
  - Entropy(Europe) = 1.0 (2 of each)
  - Entropy(Other) = 0.0 (2 Failures)
  - Entropy(Country) = (4/10) * 0.811 + (4/10) * 1.0 + (2/10) * 0.0 = 0.7244
  - InformationGain(Country) = 1.0 − 0.7244 = 0.2756

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |
| 2 | United States | No | Comedy | False |
| 3 | United States | Yes | Comedy | True |
| 4 | United States | Yes | Comedy | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | Europe | No | Comedy | True |
| 2 | Europe | Yes | Science Fiction | False |
| 3 | Europe | Yes | Romance | False |
| 4 | Europe | Yes | Comedy | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | Rest of World | Yes | Comedy | False |
| 2 | Rest of World | No | Science Fiction | False |

# Information Gain for BigStar

- Information Gain for BigStar
  - Entropy(Yes) = −4/7*log2(4/7) − 3/7*log2(3/7) = 0.985
  - Entropy(No) = −1/3*log2(1/3) − 2/3*log2(2/3) = 0.918
  - Entropy(BigStar) = (7/10) *0.985 + (3/10)*0.918 = 0.9649
  - InformationGain(BigStar) = 1 − 0.9649 = 0.0351

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |
| 2 | United States | Yes | Comedy | True |
| 3 | Europe | Yes | Science Fiction | False |
| 4 | Europe | Yes | Romance | False |
| 5 | Rest of World | Yes | Comedy | False |
| 6 | Europe | Yes | Comedy | True |
| 7 | United States | Yes | Comedy | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | No | Comedy | False |
| 2 | Europe | No | Comedy | True |
| 3 | Rest of World | No | Science Fiction | False |

# Information Gain for Genre

- Information Gain for Genre
  - Entropy(SciFi) = − (1/3) Log2(1/3) − (2/3) Log2(2/3) = 0.918
  - Entropy(Comedy) = − (4/6) Log2(4/6) − (2/6) Log2(2/6) = 0.918
  - Entropy(Romance) = − (0/1) Log2(0/1) − (1/1) Log2(1/1) = 0
  - InformationGain(Genre) = 1 − (3/10 *0.918 + 6/10*0.918+ 1/10*0) = 0.1738

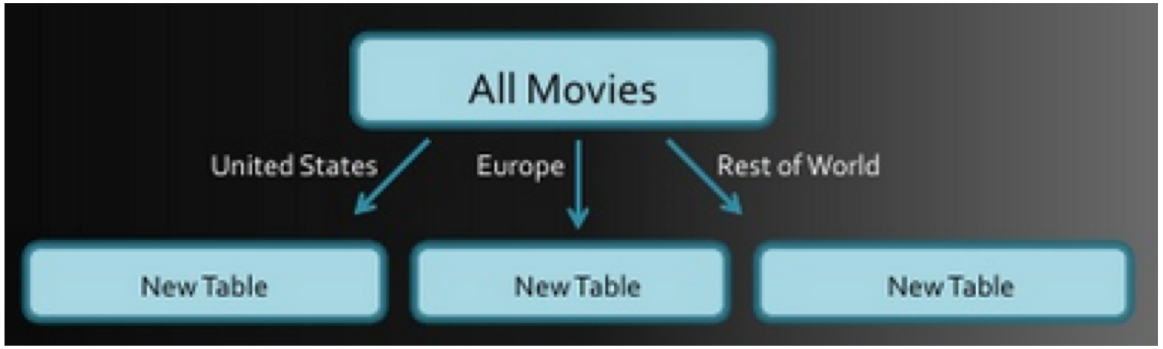
| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |
| 2 | Europe | Yes | Science Fiction | False |
| 3 | Rest of World | No | Science Fiction | False |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | No | Comedy | False |
| 2 | United States | Yes | Comedy | True |
| 3 | Europe | No | Comedy | True |
| 4 | Rest of World | Yes | Comedy | False |
| 5 | Europe | Yes | Comedy | True |
| 6 | United States | Yes | Comedy | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | Europe | Yes | Romance | False |

# What do we need to choose?

- The one with the biggest Information Gain – Country
  - Gain(Country) = 0.276
  - Gain(Star) = 0.0351
  - Gain(Genre) = 0.1738



# New Table – United States

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |
| 2 | United States | No | Comedy | False |
| 3 | United States | Yes | Comedy | True |
| 4 | United States | Yes | Comedy | True |

Entropy(3 Yes, 1 No) = $-(3/4)\,\mathrm{Log}_2(3/4) - (1/4)\,\mathrm{Log}_2(1/4)$

Entropy(Success) = 0.811

# 다음 Feature를 선택 – BigStar or Genre

- BigStar
  - Entropy(Yes) = −(3/3)*log2(3/3) − (0/3)*log2(0/3) = 0
  - Entropy(No) = −(0/1)*log2(0/1) − (1/1)*log2(1/1) = 0
  - InformationGain(BigStar) = 0.811 − 0 = 0.811

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |
| 2 | United States | Yes | Comedy | True |
| 3 | United States | Yes | Comedy | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | No | Comedy | False |

# 다음 Feature를 선택 – BigStar or Genre

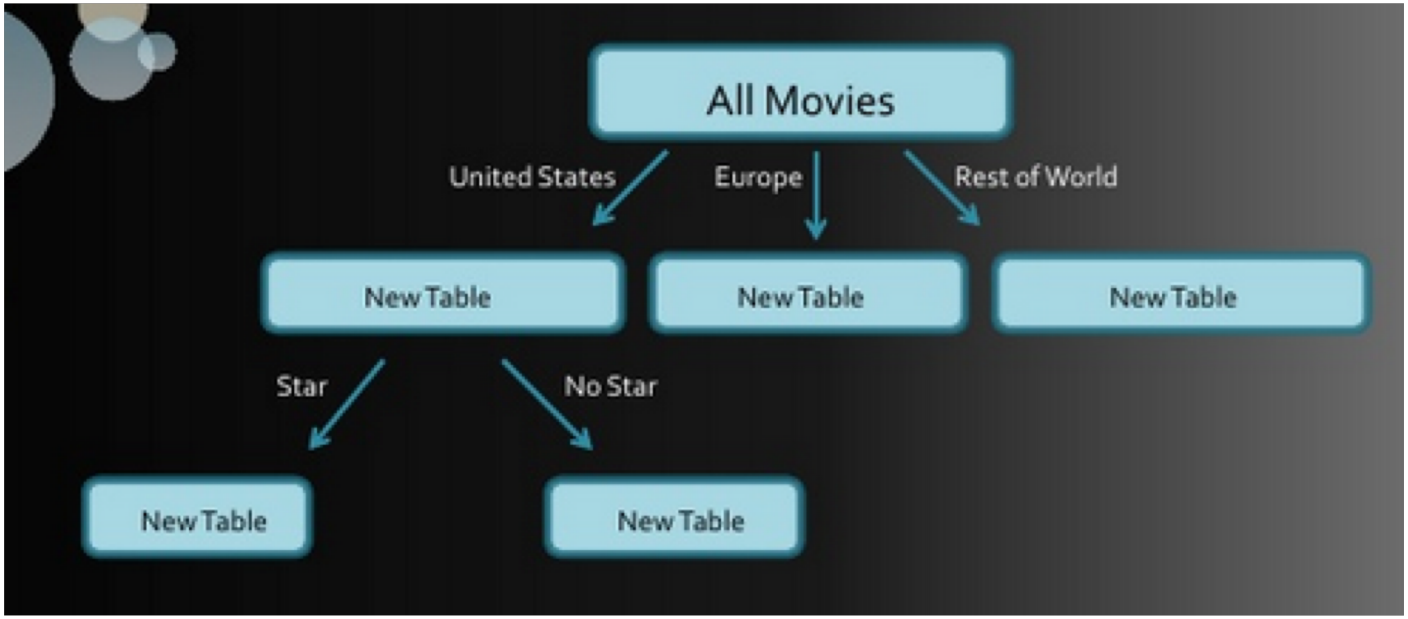
- Genre
  - Entropy(SciFi) = – (1/1)*log2(1/1) – (0/1)*log2(0/1) = 0
  - Entropy(Comedy) = – (2/3)*log2(2/3) – (1/3)*log2(1/3) = 0.918
  - InformationGain(Genre) = 0.811 – (1/4*0 + 3/4*0.918) = 0.1225

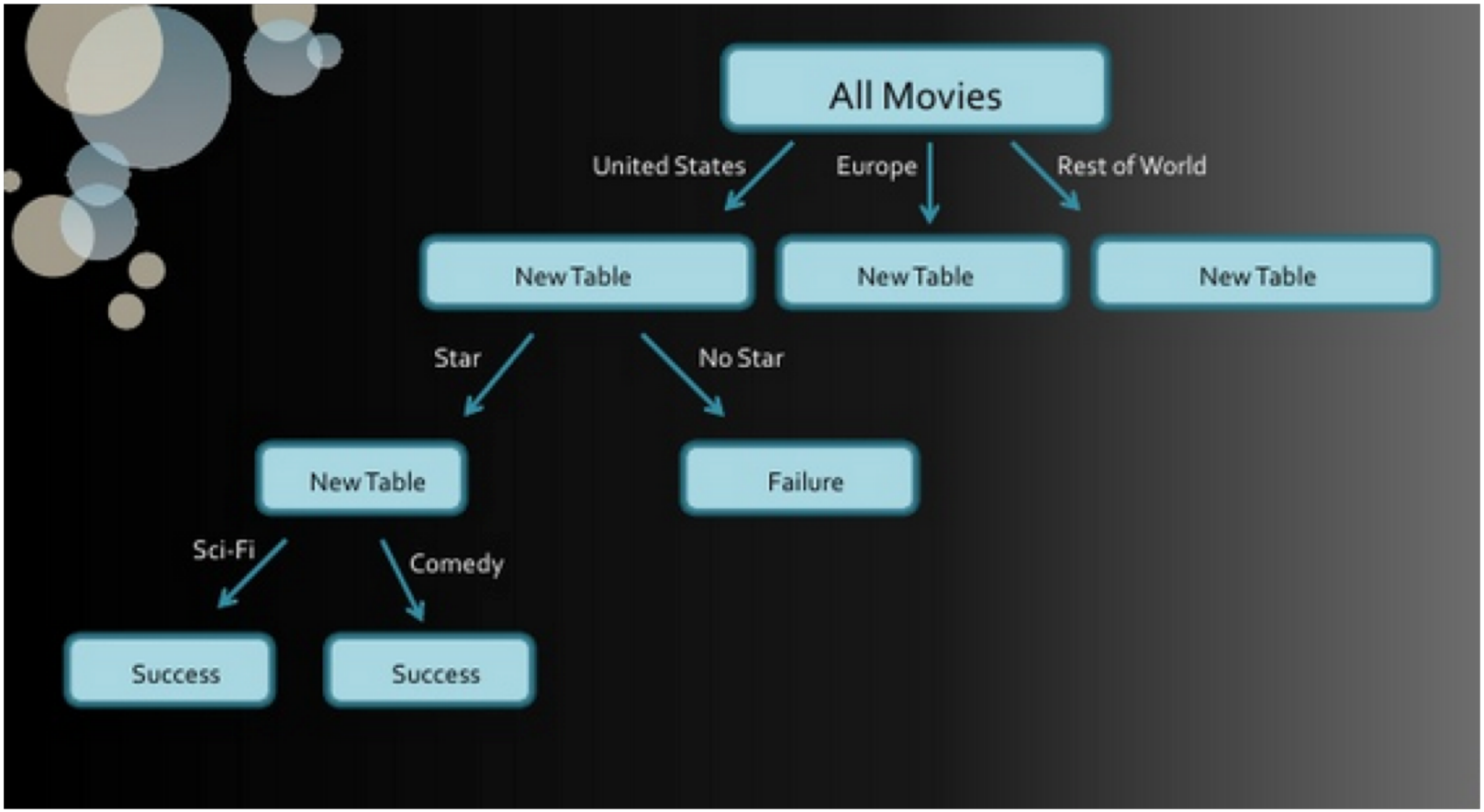
| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | Yes | Science Fiction | True |

| Film | Country of Origin | Big Star | Genre | Success |
|---|---|---|---|---|
| 1 | United States | No | Comedy | False |
| 2 | United States | Yes | Comedy | True |
| 3 | United States | Yes | Comedy | True |

# Second Split

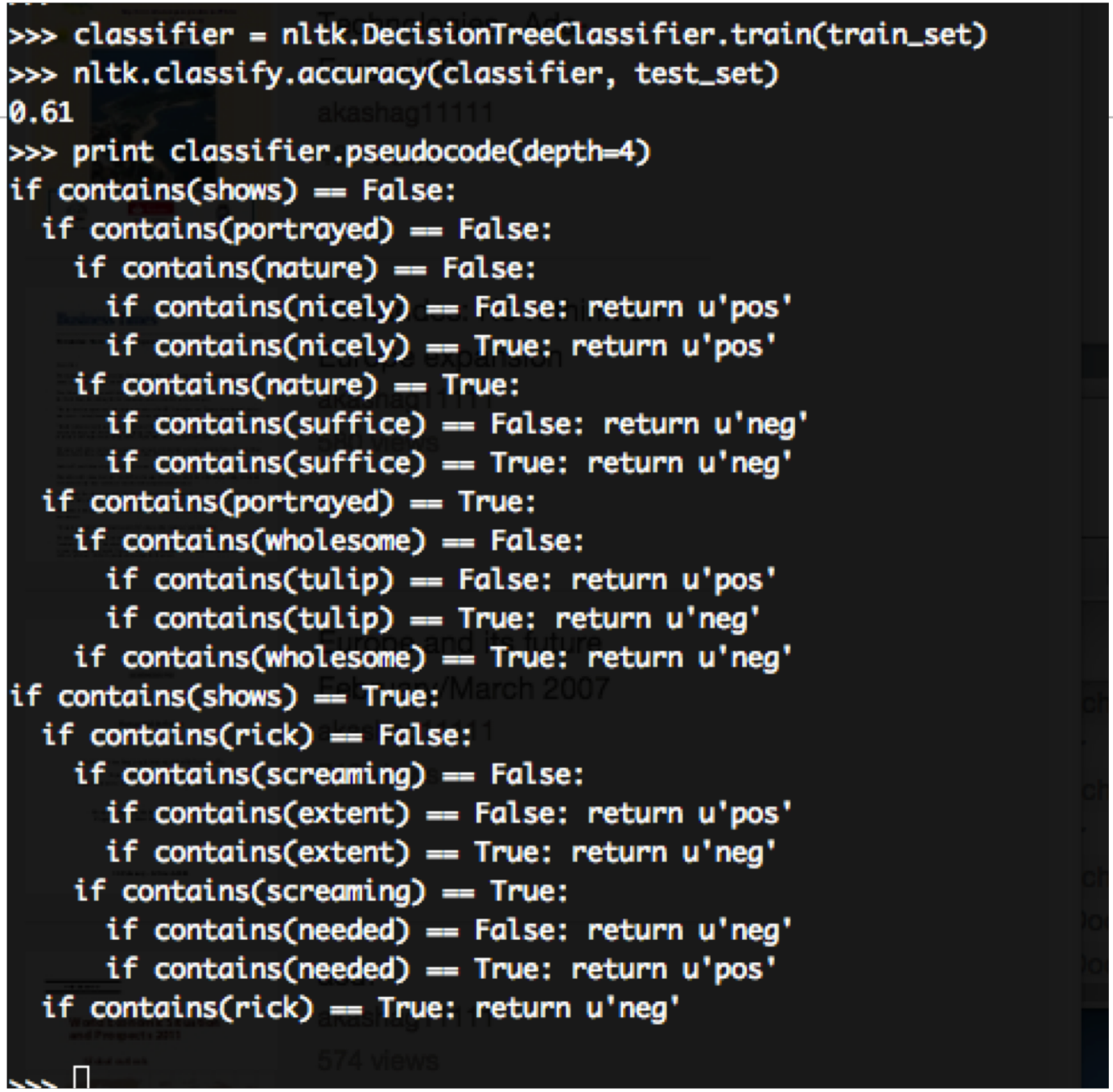
- InformationGain(BigStar) = 0.811
- InformationGain(Genre) = 0.1225




All Movies
United States
Europe
Rest of World
New Table
New Table
New Table
Star
No Star
New Table
Failure
Sci-Fi
Comedy
Success
Success

# Using Decision Tree Classifier

```
from nltk.corpus import movie_reviews
documents = [(list(movie_reviews.words(fileid)), category)
    for category in movie_reviews.categories()
    for fileid in movie_reviews.fileids(category)]
random.shuffle(documents)

all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
word_features = all_words.keys()[:600]

def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features

featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]

classifier = nltk.DecisionTreeClassifier.train(train_set)
nltk.classify.accuracy(classifier, test_set)
print classifier.pseudocode(depth=4)
```


>>> classifier = nltk.DecisionTreeClassifier.train(train_set)
>>> nltk.classify.accuracy(classifier, test_set)
0.61
>>> print classifier.pseudocode(depth=4)
if contains(shows) == False:
if contains(portrayed) == False:
if contains(nature) == False:
if contains(nicely) == False: return u'pos'
if contains(nicely) == True: return u'pos'
if contains(nature) == True:
if contains(suffice) == False: return u'neg'
if contains(suffice) == True: return u'neg'
if contains(portrayed) == True:
if contains(wholesome) == False:
if contains(tulip) == False: return u'pos'
if contains(tulip) == True: return u'neg'
if contains(wholesome) == True: return u'neg'
if contains(shows) == True:
if contains(rick) == False:
if contains(screaming) == False:
if contains(extent) == False: return u'pos'
if contains(extent) == True: return u'neg'
if contains(screaming) == True:
if contains(needed) == False: return u'neg'
if contains(needed) == True: return u'pos'
if contains(rick) == True: return u'neg'

# Logistic Regression

- We made a distinction earlier between regression (predicting a real value) and classification (predicting a discrete value).

- Logistic regression is designed as a binary classifier (output say {0,1}) but actually outputs the probability that the input instance is in the "1" class.

- A logistic classifier has the form:

$$p(X) = \frac{1}{1 + \exp(-X\beta)}$$

where $X = (X_1, \ldots, X_n)$ is a vector of features.

# Logistic Regression

- Logistic regression is probably the most widely used general-purpose classifier.

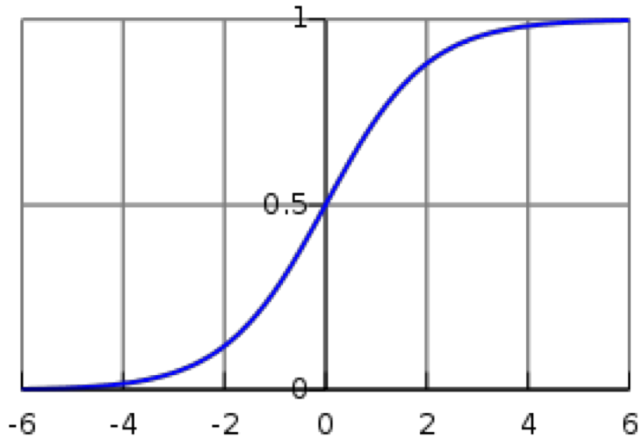
- Its very scalable and can be very fast to train. It's used for
  - Spam filtering
  - News message classification
  - Web site classification
  - Product classification
  - Most classification problems with large, sparse feature sets.

- The only caveat is that it can overfit on very sparse data, so its often used with Regularization

# Logistic Regression

- Logistic regression maps the "regression" value $-X\beta$ in $(-\infty,\infty)$ to the range [0,1] using a "logistic" function:

$$p(X) = \frac{1}{1 + \exp(-X\beta)}$$




- i.e. the logistic function maps any value on the real line to a probability in the range [0,1]

# Example – Donner Party Data

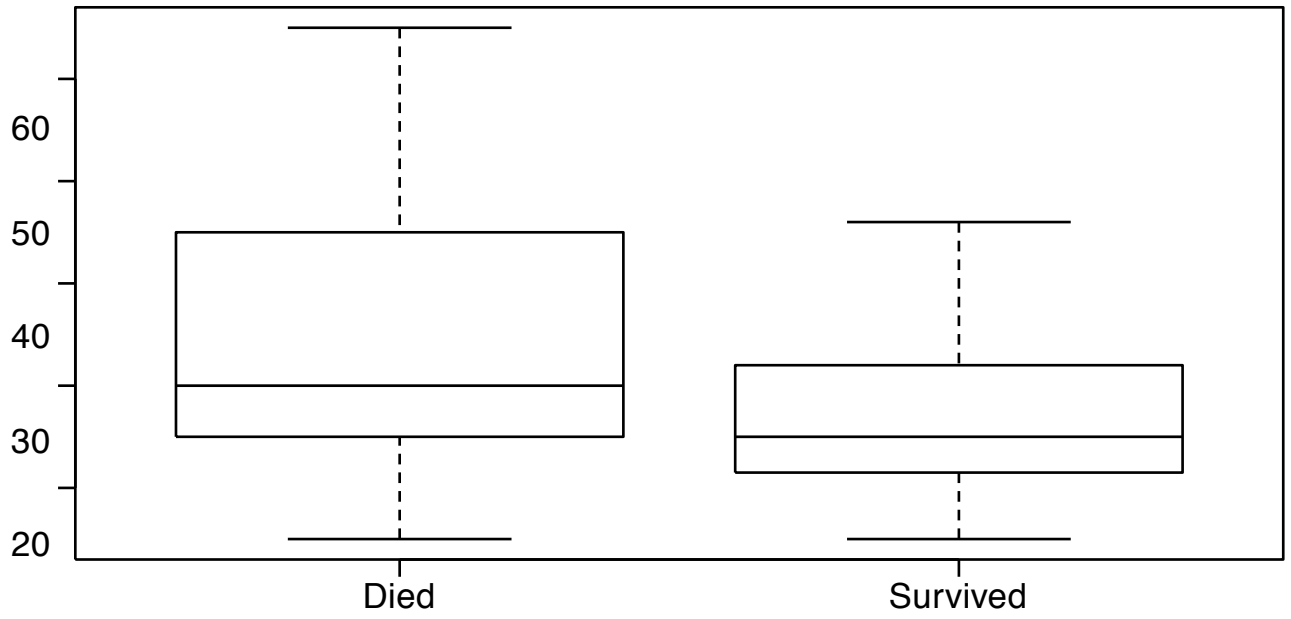
| | Age | Sex | Status |
|---|---|---|---|
| 1 | 23.00 | Male | Died |
| 2 | 40.00 | Female | Survived |
| 3 | 40.00 | Male | Survived |
| 4 | 30.00 | Male | Died |
| 5 | 28.00 | Male | Died |
| ⋮ | ⋮ | ⋮ | ⋮ |
| 43 | 23.00 | Male | Survived |
| 44 | 24.00 | Male | Died |
| 45 | 25.00 | Female | Survived |

# Example – Donner Party Data

Status vs. Gender:

| | Male | Female |
|---|---|---|
| Died | 20 | 5 |
| Survived | 10 | 10 |

Status vs. Age:



# Example – Donner Party – Age and Gender

```
summary(glm(Status ~ Age + Sex, data=donner, family=binomial))

## Call:
## glm(formula = Status ~ Age + Sex, family = binomial, data = donner)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.63312    1.11018   1.471   0.1413
## Age         -0.07820    0.03728  -2.097   0.0359 *
## SexFemale    1.59729    0.75547   2.114   0.0345 *
## ---
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 61.827  on 44  degrees of freedom
## Residual deviance: 51.256  on 42  degrees of freedom
## AIC: 57.256
##
## Number of Fisher Scoring iterations: 4
```

# Generalized linear models

It turns out that this is a very general way of addressing this type of problem in regression, and the resulting models are called generalized linear models (GLMs). Logistic regression is just one example of this type of model.

All generalized linear models have the following three characteristics:

1. A probability distribution describing the outcome variable
2. A linear model
   - $\eta = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$
3. A link function that relates the linear model to the parameter of the outcome distribution
   - $g(p) = \eta$ or $p = g^{-1}(\eta)$

# Logistic Regression

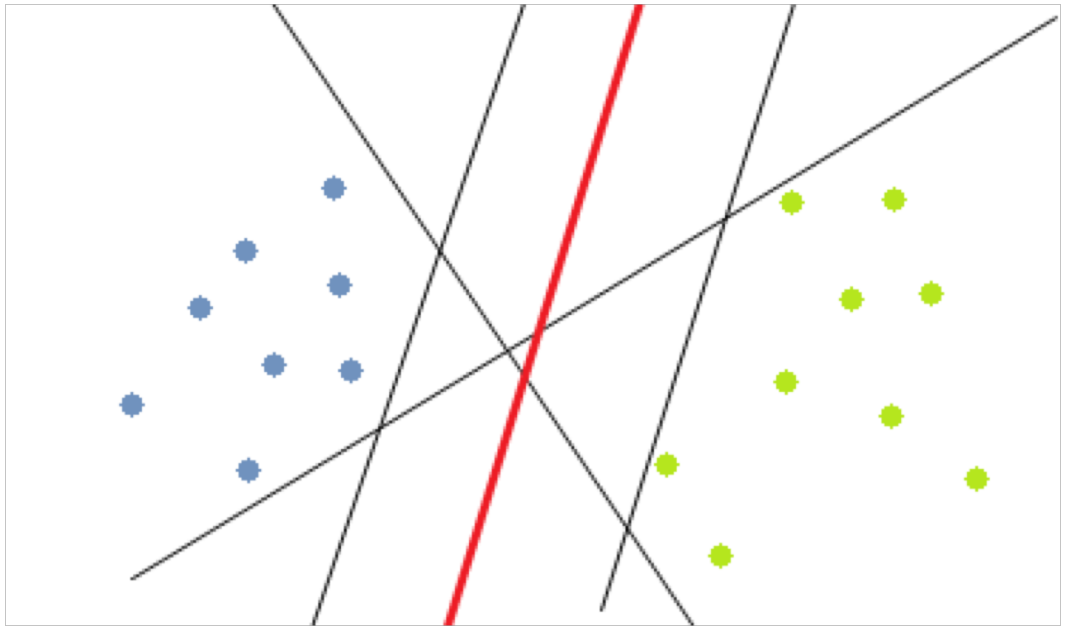
Logistic regression is a GLM used to model a binary categorical variable using numerical and categorical predictors.

We assume a binomial distribution produced the outcome variable and we therefore want to model $p$ the probability of success for a given set of predictors.

To finish specifying the Logistic model we just need to establish a reasonable link function that connects $\eta$ to $p$. There are a variety of options but the most commonly used is the logit function.

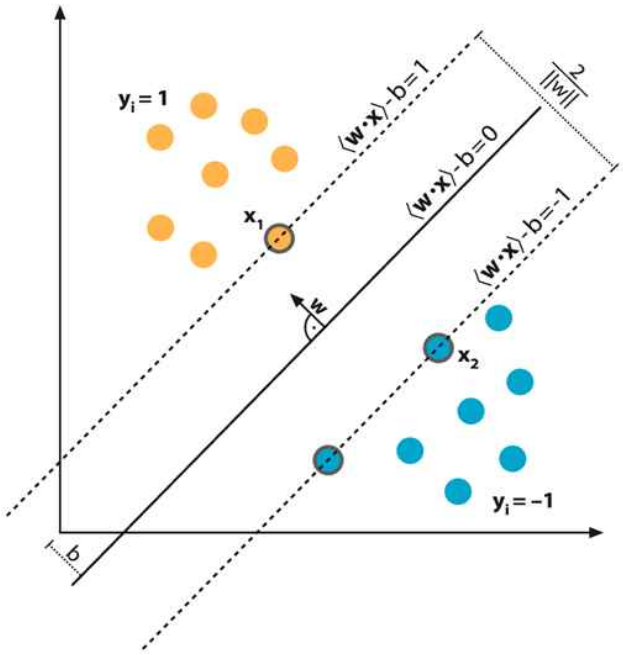
Logit function

$$logit(p) = \log\left(\frac{p}{1-p}\right), \text{ for } 0 \leq p \leq 1$$



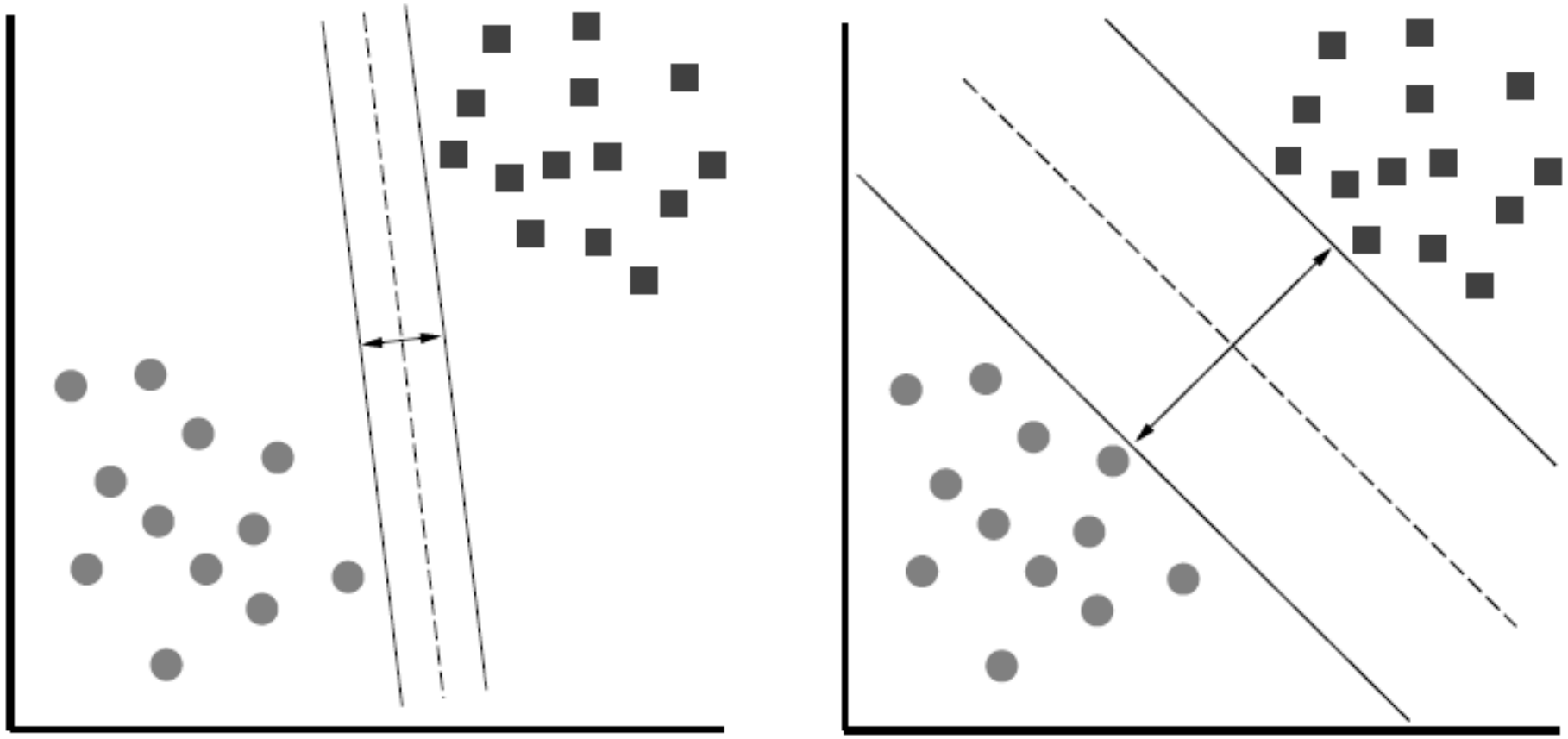
# Support Vector Machines

- A Support Vector Machine (SVM) is a classifier that tries to **maximize the margin** between training data and the classification boundary (the plane defined by $X\beta = 0$)
- 2 / || w || 가 최대가 되도록 → ||w||가 최소가 되도록



# Largest Margin

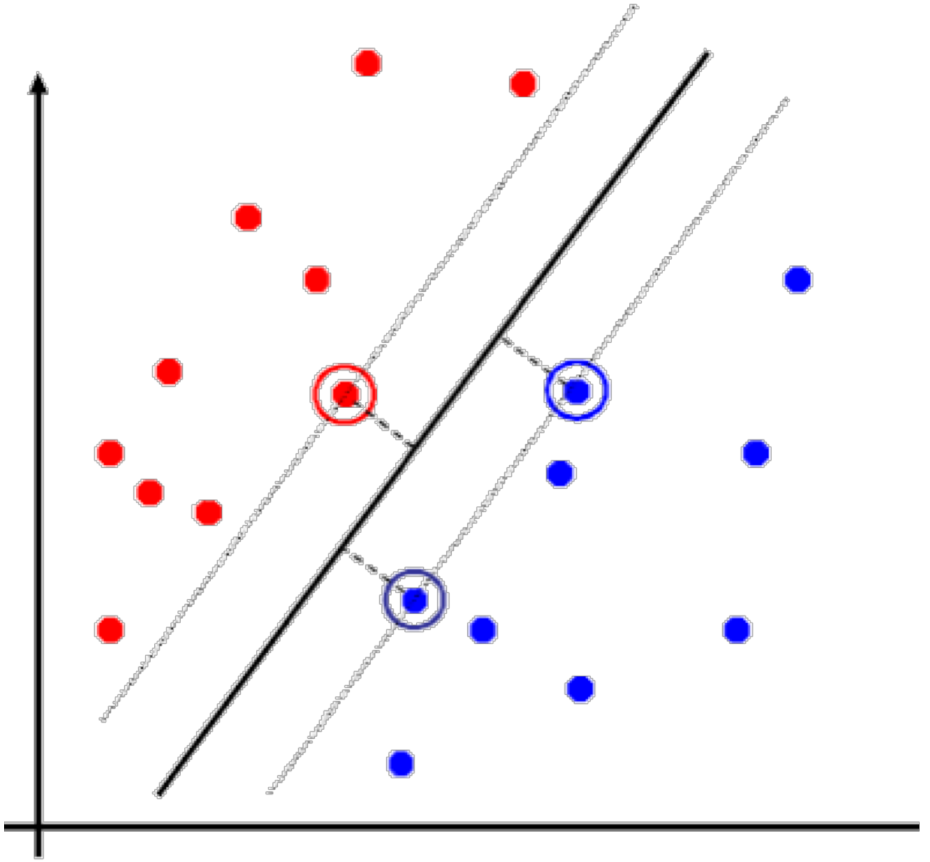
- Margin $\gamma$: Distance of closest example from the decision line



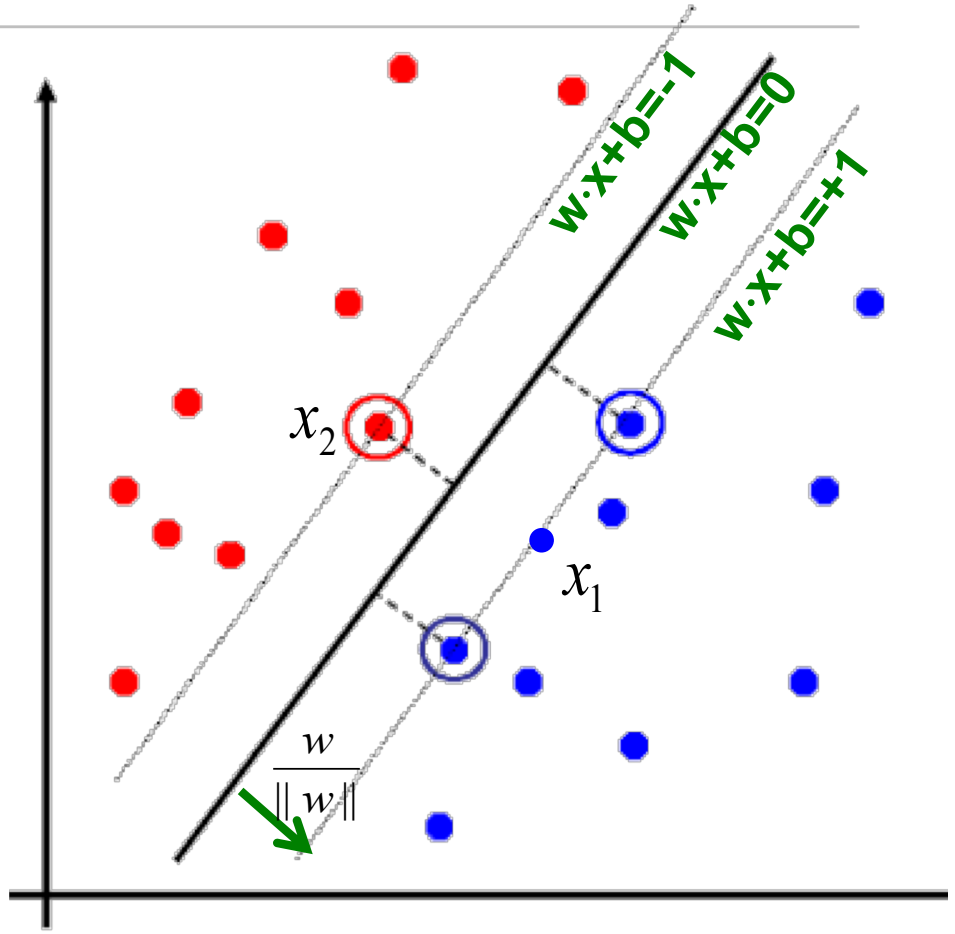
The reason we define margin this way is due to theoretical convenience and existence of generalization error bounds that depend on the value of margin.

# Support Vector Machines

- Separating hyperplane is defined by the support vectors
  - Points on +/− planes from the solution
  - If you knew these points, you could ignore the rest
  - Generally, ***d+1*** support vectors (for ***d*** dim. data)



# Canonical Hyperplane: Problem




- Problem:
  - Let $(\boldsymbol{w}\cdot\boldsymbol{x}+\boldsymbol{b})\boldsymbol{y}=\boldsymbol{\gamma}$

    then $(\boldsymbol{2w}\cdot\boldsymbol{x}+\boldsymbol{2b})\boldsymbol{y}=\boldsymbol{2\gamma}$
    - Scaling $\boldsymbol{w}$ increases margin!
- Solution:
  - Work with normalized $w$:

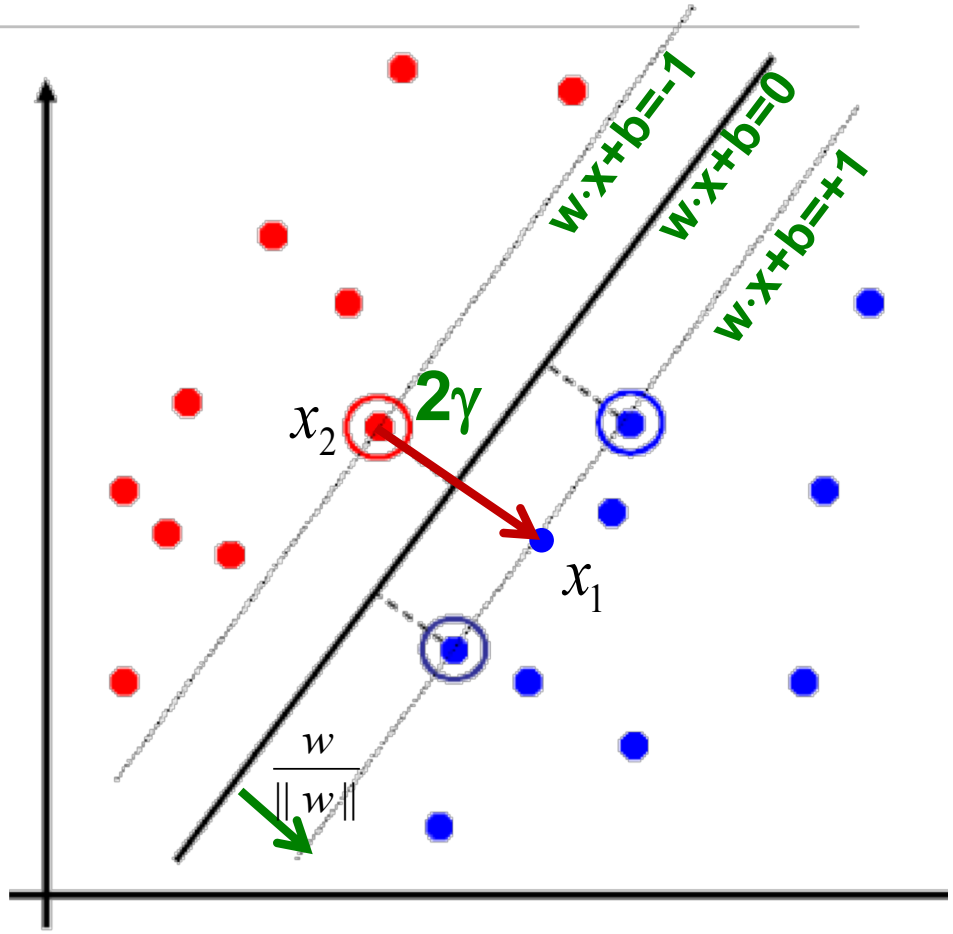
    $$\boldsymbol{\gamma}=\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\cdot\boldsymbol{x}+\boldsymbol{b}\right)\boldsymbol{y}$$
  - Let's also require support vectors $\boldsymbol{x_j}$ to be on the plane defined by: $\boldsymbol{w}\cdot\boldsymbol{x_j}+\boldsymbol{b}=\pm\mathbf{1}$

$$||w|| = \sqrt{\sum_{j=1}^{d}(w^{(j)})^2}$$

# Canonical Hyperplane: Solution

- Want to maximize margin $\gamma$!
- What is the relation between $x_1$ and $x_2$?
  - $\boldsymbol{x_1 = x_2 + 2\gamma \frac{w}{||w||}}$
  - We also know:
    - $\boldsymbol{w \cdot x_1 + b = +1}$
    - $\boldsymbol{w \cdot x_2 + b = -1}$
- So:
  - $\boldsymbol{w \cdot x_1 + b = +1}$
  - $\boldsymbol{w\left(x_2 + 2\gamma \frac{w}{||w||}\right) + b = +1}$
  - $\boldsymbol{w \cdot x_2 + b + 2\gamma \frac{w \cdot w}{||w||} = +1}$

-1




$$\Rightarrow \gamma = \frac{\|w\|}{w \cdot w} = \frac{1}{\|w\|}$$

**Note:**

$w \cdot w = \|w\|^2$

# Non-linearly Separable Data

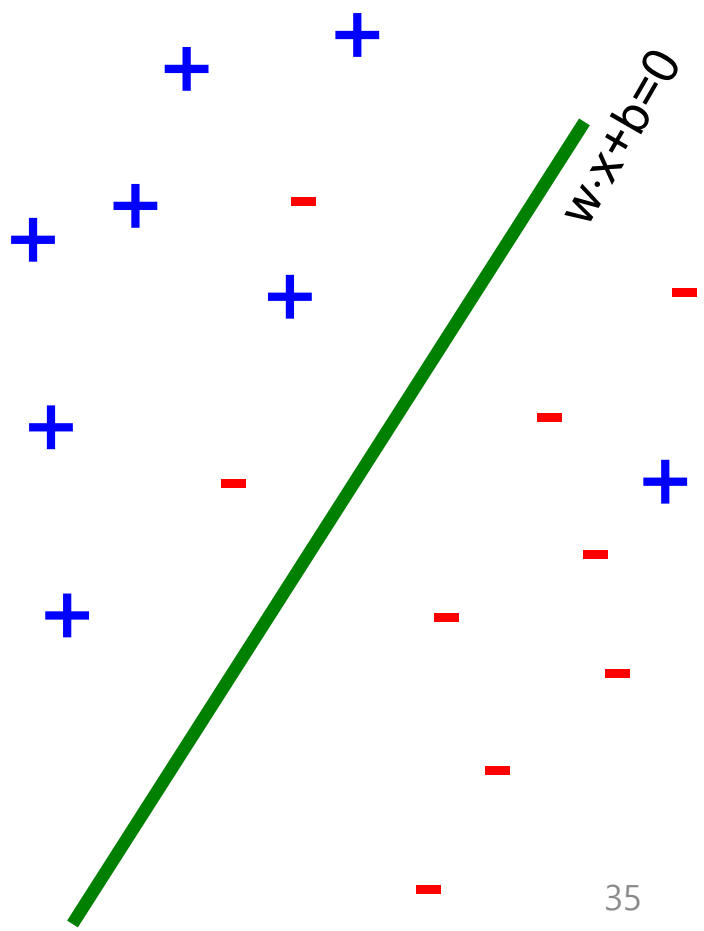
- If data is not separable introduce penalty:

$$\min_w \tfrac{1}{2}\|w\|^2 + \mathrm{C}\cdot(\#\text{number of mistakes})$$

$$s.t.\ \forall i, y_i(w\cdot x_i + b) \geq 1$$

  - Minimize $\|w\|^2$ plus the number of training mistakes
  - Set $C$ using cross validation
  - Soft margin SVM
  - C.f. SVM with **hard constrains**

- How to penalize mistakes?
  - All mistakes are not equally bad!



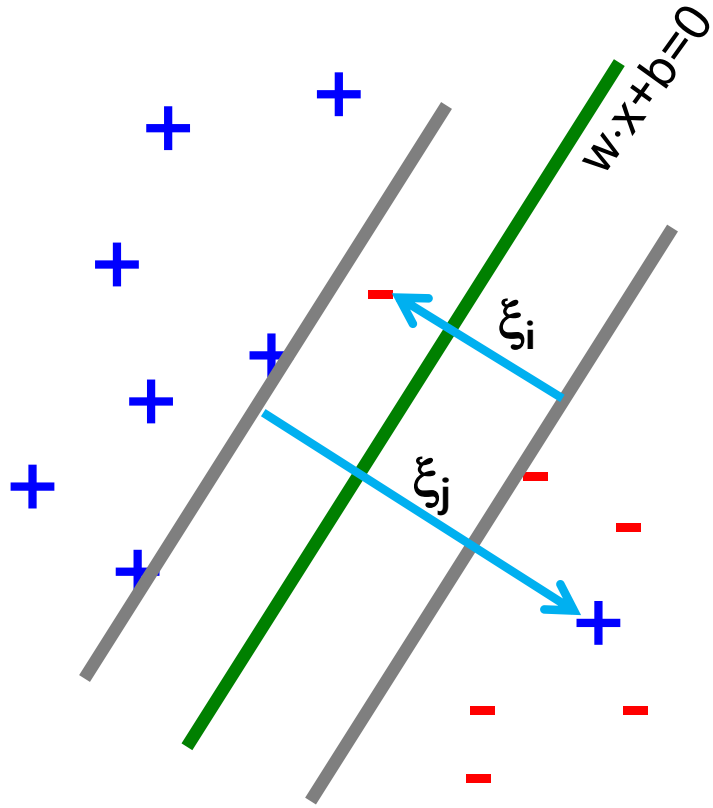
# Support Vector Machines

- Introduce slack variables $\xi_i$ (크사이)

$$\min_{w,b,\xi_i \geq 0} \ \frac{1}{2}\|w\|^2 + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, \ y_i (w \cdot x_i + b) \geq 1 - \xi_i$$

- If point $x_i$ is on the wrong side of the margin then get penalty $\xi_i$



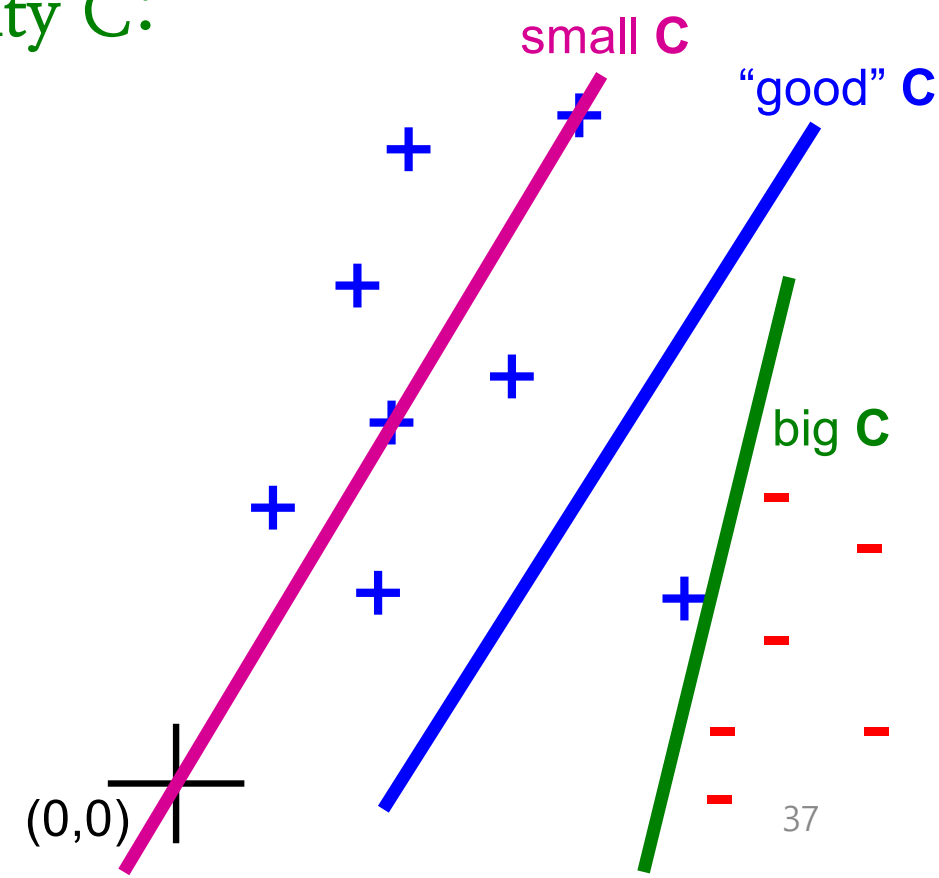

**For each data point:**
If margin ≥ 1, don't care
If margin < 1, pay linear penalty

# Slack Penalty $C$

$$\min_w \tfrac{1}{2}\|w\|^2 + \mathrm{C}\cdot(\#\text{number of mistakes})$$

$$s.t.\forall i, y_i(w\cdot x_i + b) \geq 1$$

- What is the role of slack penalty C:
  - C=∞: Only want to **w, b** that separate the data
  - C=0: Can set $\xi_i$ to anything, then **w=0** (basically ignores the data)



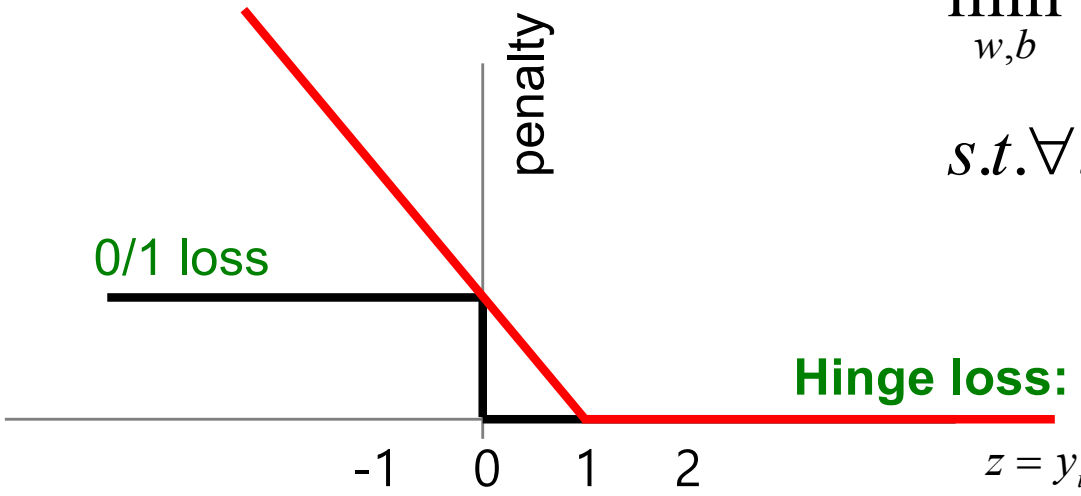
# Support Vector Machines

- SVM in the "natural" form

$$\arg\min_{w,b} \frac{1}{2} \underbrace{w \cdot w}_{\text{Margin}} + C \cdot \sum_{i=1}^{n} \underbrace{\max\{0, 1 - y_i(w \cdot x_i + b)\}}_{\text{Empirical loss L (how well we fit training data)}}$$

Regularization parameter

- SVM uses "Hinge Loss":




$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (w \cdot x_i + b) \geq 1 - \xi_i$$

# Support Vector Machines: How to compute the margin?

# SVM: How to estimate $w$?

$$\min_{w,b} \ \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

- Want to estimate ***w*** and ***b***!
  - **Standard way:** Use a solver!
    - **Solver:** software for finding solutions to "common" optimization problems
- Use a quadratic solver:
  - Minimize quadratic function
  - Subject to linear constraints
- **Problem:** Solvers are inefficient for big data!

# SVM: How to estimate $w$?

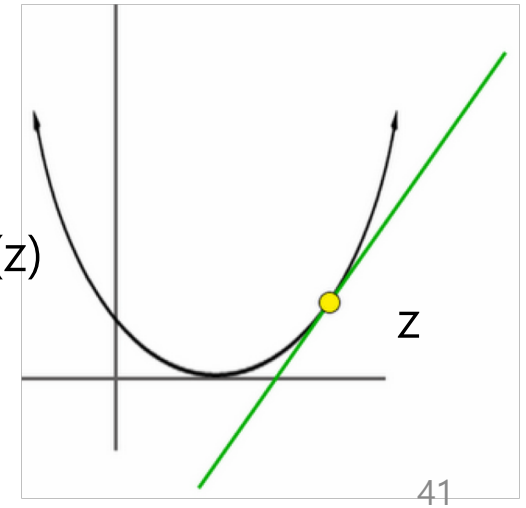
- Want to estimate w, b!
- Alternative approach:
  - Want to minimize $f(w,b)$:

$$\min_{w,b} \ \frac{1}{2} w \cdot w + C \sum_{i=1}^{n} \xi_i$$

$$s.t. \forall i, y_i \cdot (x_i \cdot w + b) \geq 1 - \xi_i$$

$$f(w,b) = \frac{1}{2} w \cdot w + C \cdot \sum_{i=1}^{n} \max\left\{0, 1 - y_i \left(\sum_{j=1}^{d} w^{(j)} x_i^{(j)} + b\right)\right\}$$

- Side note:
  - How to minimize convex functions $g(z)$?
  - Use gradient descent: $\min_z g(z)$
  - Iterate: $z_{t+1} \leftarrow z_t - \eta \nabla g(z_t)$



# SVM: How to estimate *w*?

- Want to minimize *f(w,b)*:

$$f(w,b) = \frac{1}{2}\sum_{j=1}^{d}\left(w^{(j)}\right)^2 + C\sum_{i=1}^{n}\max\left\{0,1-y_i\left(\sum_{j=1}^{d}w^{(j)}x_i^{(j)}+b\right)\right\}$$

Empirical loss $L(x_i\, y_i)$

- Compute the gradient ∇(j) w.r.t. $w^{(j)}$

$$\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i,y_i)}{\partial w^{(j)}}$$

$$\frac{\partial L(x_i,y_i)}{\partial w^{(j)}} = 0 \qquad \text{if } y_i(\mathrm{w}\cdot x_i + b) \geq 1$$

$$= -y_i x_i^{(j)} \qquad \text{else}$$

# SVM: How to estimate $w$?

- Gradient descent:

**Iterate until convergence:**

- **For j = 1 … d**
  - **Evaluate:**
  - **Update:**

$$\nabla f^{(j)} = \frac{\partial f(w,b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

$\mathbf{w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}}$

- Problem:
  - Computing $\nabla f^{(j)}$ takes O(n) time!
    - n … size of the training dataset

η…learning rate parameter

**C**… regularization parameter

# SVM: How to estimate $w$?

**We just had:**

$$\nabla f^{(j)} = w^{(j)} + C\sum_{i=1}^{n} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

- Stochastic Gradient Descent
  - Instead of evaluating gradient over all examples evaluate it for each individual training example

$$\nabla f^{(j)}(x_i) = w^{(j)} + C \cdot \frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

Notice: no summation over ***i*** anymore

- Stochastic gradient descent:

**Iterate until convergence:**

- **For i = 1 … n** 모든 데이터에 대해서
  - **For j = 1 … d** 각 dimension 마다
    - **Compute:** $\nabla f^{(j)}(x_i)$
    - **Update:** $w^{(j)} \leftarrow w^{(j)} - \eta \nabla f^{(j)}(x_i)$

# Support Vector Machines: Example

# Example: Text categorization

- Example by Leon Bottou:
  - Reuters RCV1 document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - $n$ = 781,000 training examples (documents)
  - 23,000 test examples
  - $d$ = 50,000 features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

# Example: Text categorization

- Questions:
  - (1) Is SGD successful at minimizing *f(w,b)*?
  - (2) How quickly does SGD find the min of *f(w,b)*?
  - (3) What is the error on a test set?

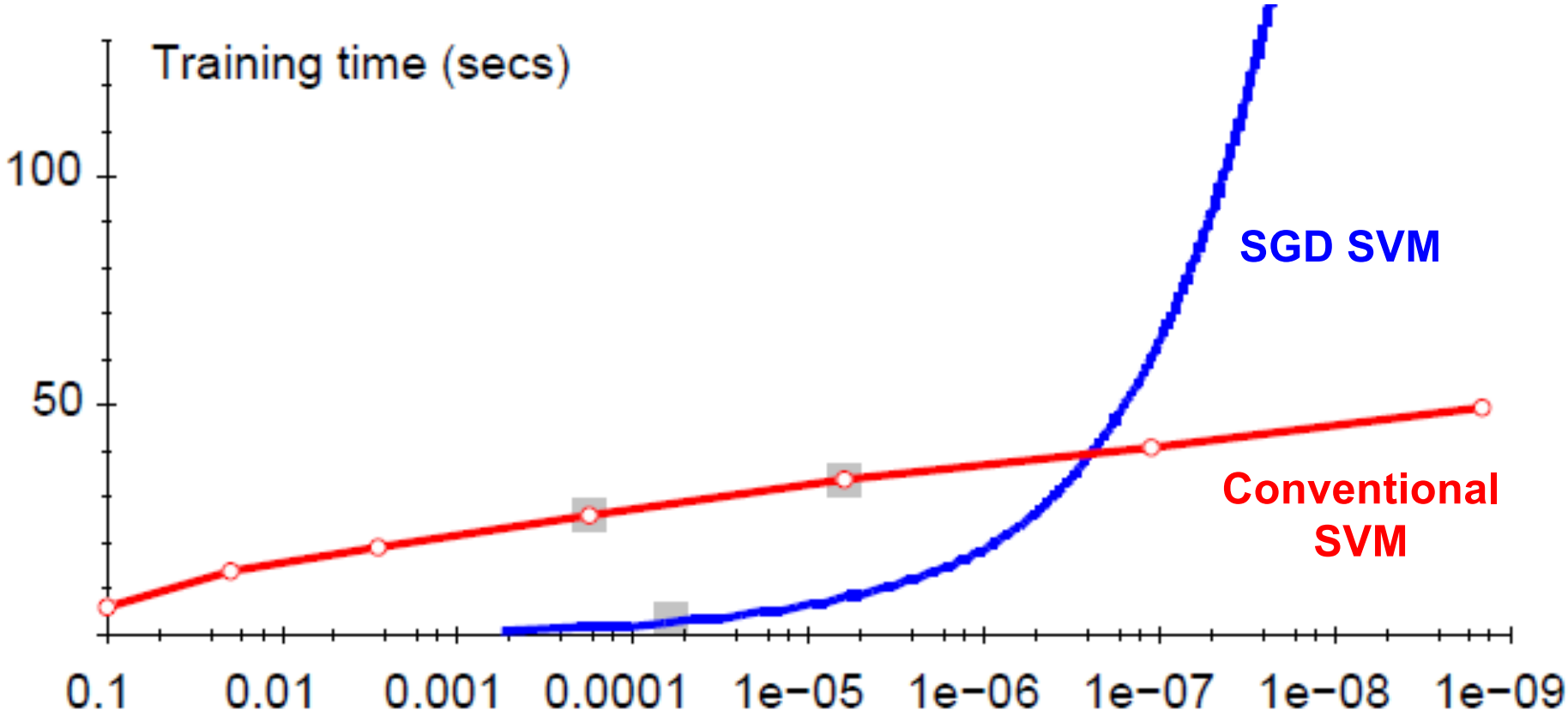
| | ***Training time*** | ***Value of f(w,b)*** | ***Test error*** |
|---|---|---|---|
| Standard SVM | 23,642 secs | *0.2275* | *6.02%* |
| "Fast SVM" | 66 secs | 0.2278 | 6.03% |
| **SGD SVM** | 1.4 secs | *0.2275* | *6.02%* |

**(1)** SGD-SVM is successful at minimizing the value of ***f(w,b)***

**(2)** SGD-SVM is super fast

**(3)** SGD-SVM test set error is comparable

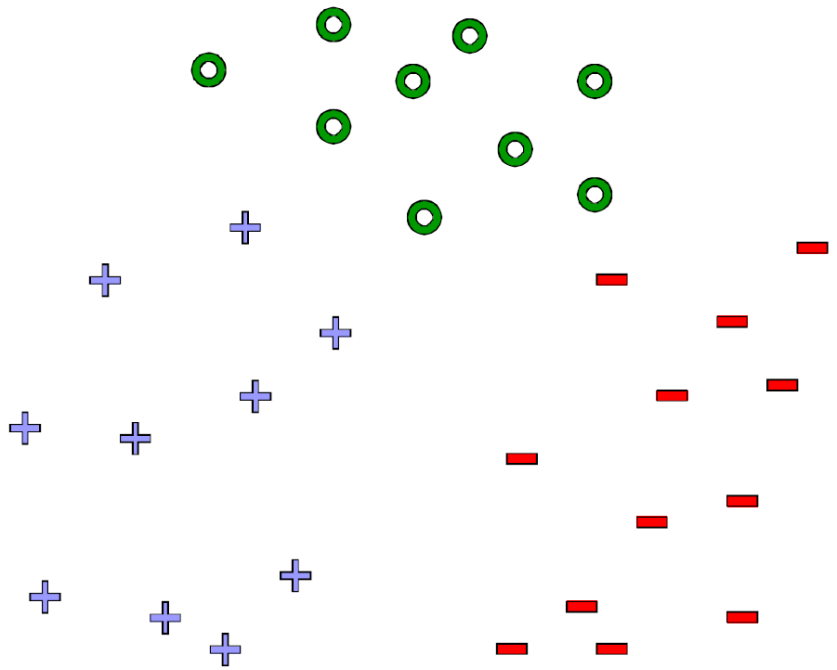
# Optimization "Accuracy"




**Optimization quality: | $f(w,b) - f(w^{opt},b^{opt})$ |**

For optimizing *f(w,b) within reasonable* quality
*SGD-SVM* is super fast

# What about multiple classes?



- Idea 1:

  One against all

  Learn 3 classifiers
  - + vs. {o, −}
  - − vs. {o, +}
  - o vs. {+, −}

  Obtain:

  $w_+ b_+$, $w_- b_-$, $w_o b_o$

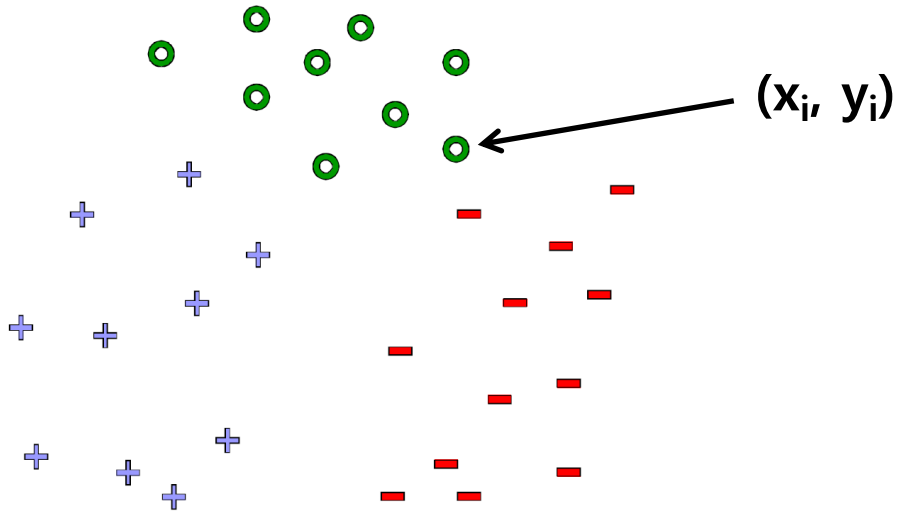
- How to classify?
- Return class *c*

  $\mathbf{arg\ max_c\ w_c x + b_c}$

# Learn 1 classifier: Multiclass SVM

- Idea 2: Learn 3 sets of weights simultaneously!
  - For each class $c$ estimate $w_c$, $b_c$
  - Want the correct class to have highest margin:

$$\mathbf{w_{y_i} x_i + b_{y_i} \geq 1 + w_c x_i + b_c \quad \forall c \neq y_i \ , \forall i}$$



# Multiclass SVM

- Optimization problem:

$$\min_{w,b} \ \frac{1}{2}\sum_{c}\|w_c\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$w_{y_i}\cdot x_i + b_{y_i} \geq w_c \cdot x_i + b_c + 1 - \xi_i \qquad \forall c \neq y_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

  - To obtain parameters $w_c$, $b_c$ (for each class $c$)
    we can use similar techniques as for 2 class SVM

- SVM is widely perceived a very powerful learning algorithm

# SVM Example in R

- To install e1071 package in R, type

```
install.packages('e1071', dependencies = TRUE)
```

- To start to use the package, type

```
library(e1071)
```

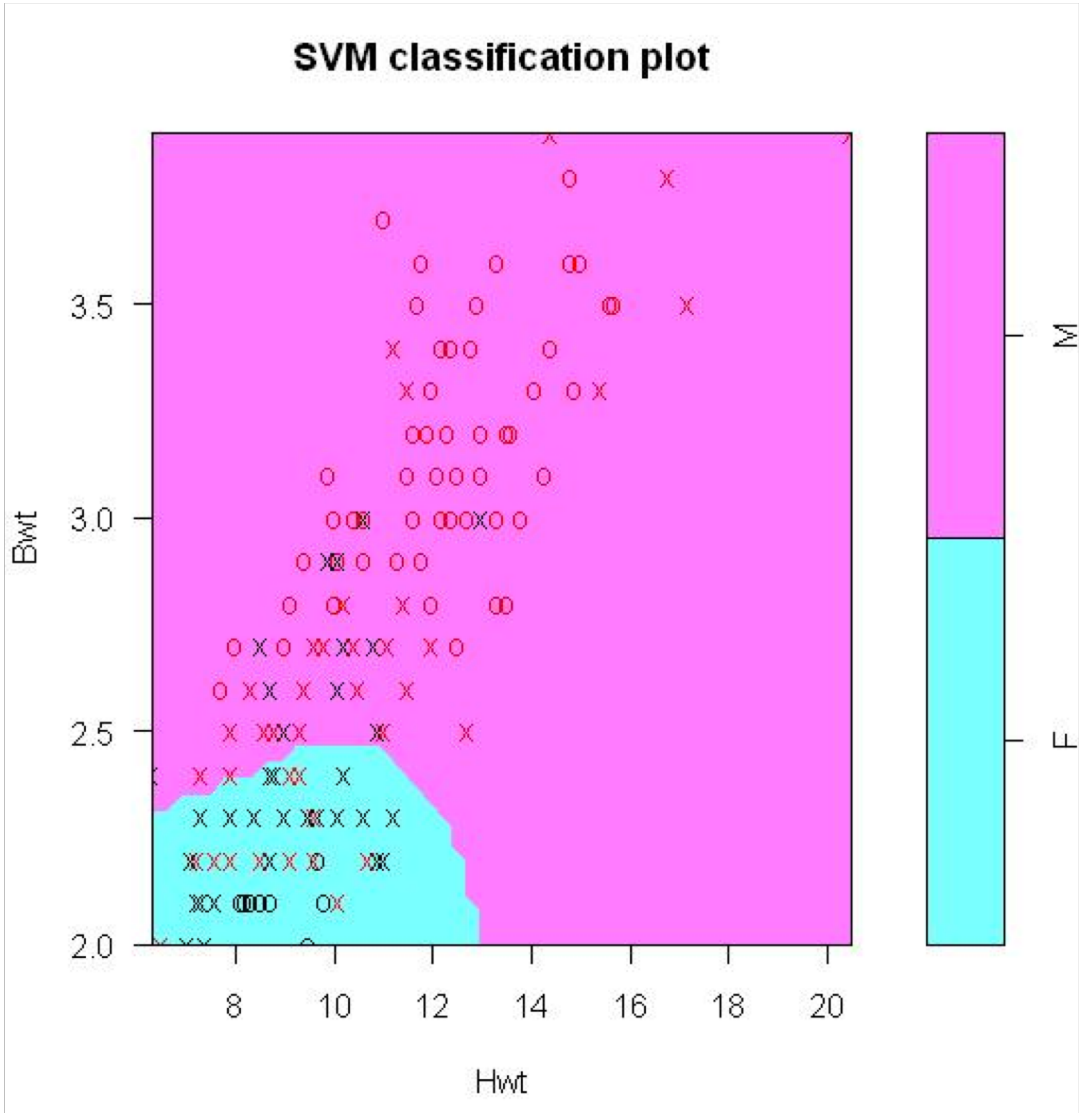
```
library(MASS)
data(cats)
model <- svm(Sex~., data = cats)
```

```
print(model)
summary(model)
```

```
Call:
svm(formula = Sex ~ ., data = cats)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  84

 ( 39 45 )


Number of Classes:  2

Levels:
 F M
```

# SVM Result

- `Plot(model, cats)`



# SVM Example

- Divide the cats dataset into train set and test set

```
index <- 1:nrow(cats)
testindex <- sample(index, trunc(length(index)/3))
testset <- cats[testindex,]
trainset <- cats[-testindex,]
```

- Run the model and predict classes

```
model <- svm(Sex~., data = trainset)
prediction <- predict(model, testset[,-1])  ## -1 은 처음이 데이터가 아니라서
```

- Confusion Matrix

```
tab <- table(pred = prediction, true = testset[,1])
      True
pred  F   M
F     10  8
M     6   24
```

# Ensemble Methods

- Are like Crowdsourced machine learning algorithms:
  - Take a collection of simple or *weak* learners
  - Combine their results to make a single, better learner

- Bagging: train learners in parallel on different samples of the data, then combine by voting (discrete output) or by averaging (continuous output).
- Stacking: combine model outputs using a second-stage learner like linear regression.
- Boosting: train learners on the filtered output of other learners.

# Random Forests

- Grow K trees on datasets sampled from the original dataset with replacement (bootstrap samples), p = number of features.
- Draw K bootstrap samples of size N
- Grow each Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
    - Typically m might be e.g. sqrt(p) but can be smaller.
- Aggregate the predictions of the trees (most popular vote) to produce the final class.

# Random Forests

- Principles: we want to take a vote between different learners so we don't want the models to be too similar. These two criteria ensure diversity in the individual trees:
- Draw K bootstrap samples of size N:
  - Each tree is trained on different data.
- Grow a Decision Tree, by selecting a random set of m out of p features at each node, and choosing the best feature to split on.
  - Corresponding nodes in different trees (usually) cant use the same feature to split.

# Random Forests

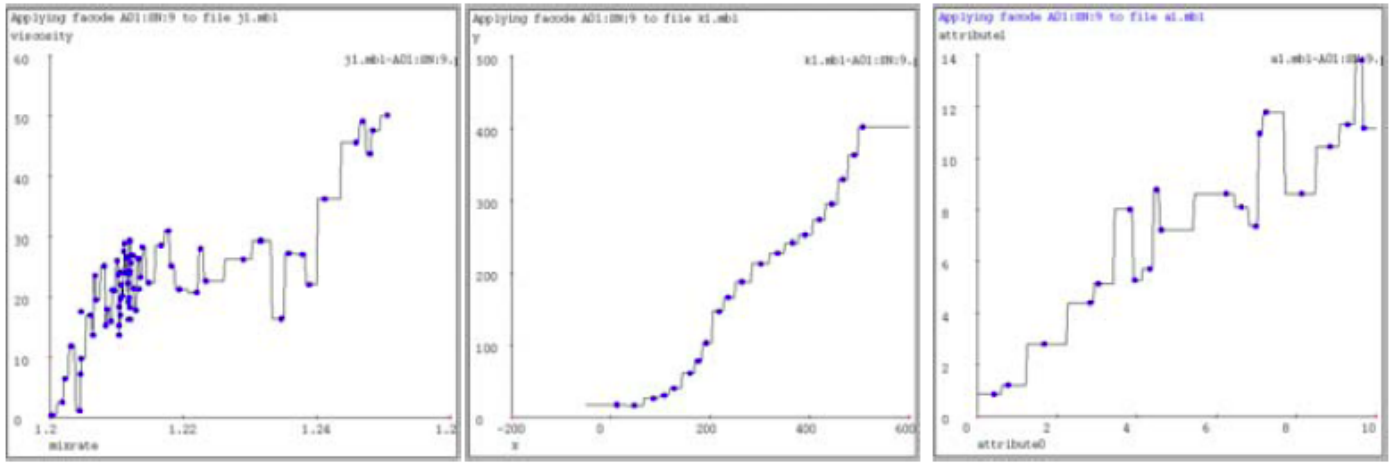
- Very popular in practice, probably the most popular classifier for dens e data (<= a few thousand features)
- Easy to implement (train a lot of trees). Good match for MapReduce.
- Parallelizes easily (but not necessarily efficiently).
- Not quite state-of-the-art accuracy – DNNs generally do better, and sometimes gradient boosted trees.
- Needs many passes over the data – at least the max depth of the trees.
- Easy to overfit – hard to balance accuracy/fit tradeoff.

# Instance Based Learning

- Instance based learning
- Example: Nearest neighbor
  - Keep the whole training dataset: {(**x**, **y**)}
  - A query example (vector) $q$ comes
  - Find closest example(s) $\mathbf{x}^*$
  - Predict $\mathbf{y}^*$
- Works both for regression and classification
  - **Collaborative filtering** is an example of k−NN classifier
    - Find $k$ most similar people to user **x** that have rated movie **y**
    - Predict rating $\mathbf{y}_x$ of **x** as an average of $\mathbf{y}_k$

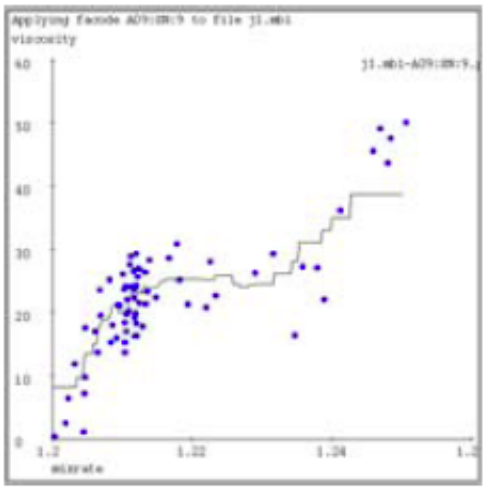
# 1-Nearest Neighbor

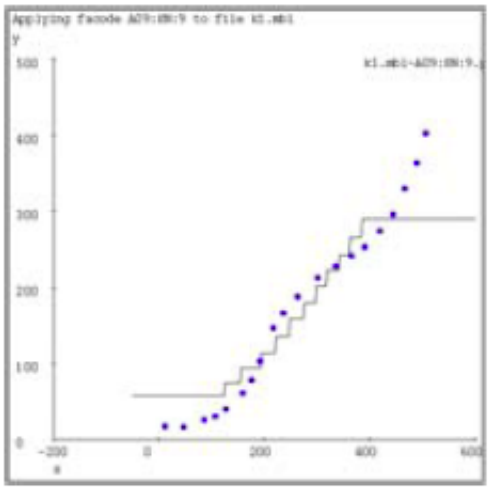
- To make Nearest Neighbor work we need 4 things:
  - Distance metric:
    - Euclidean
  - How many neighbors to look at?
    - One
  - Weighting function (optional):
    - Unused
  - How to fit with the local points?
    - Just predict the same output as the nearest neighbor

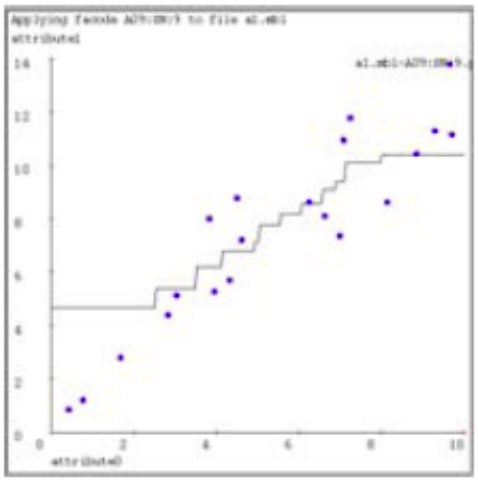


# $k$–Nearest Neighbor

- Distance metric:
  - Euclidean
- How many neighbors to look at?
  - $k$
- Weighting function (optional):
  - Unused
- How to fit with the local points?
  - Just predict the average output among $k$ nearest neighbors

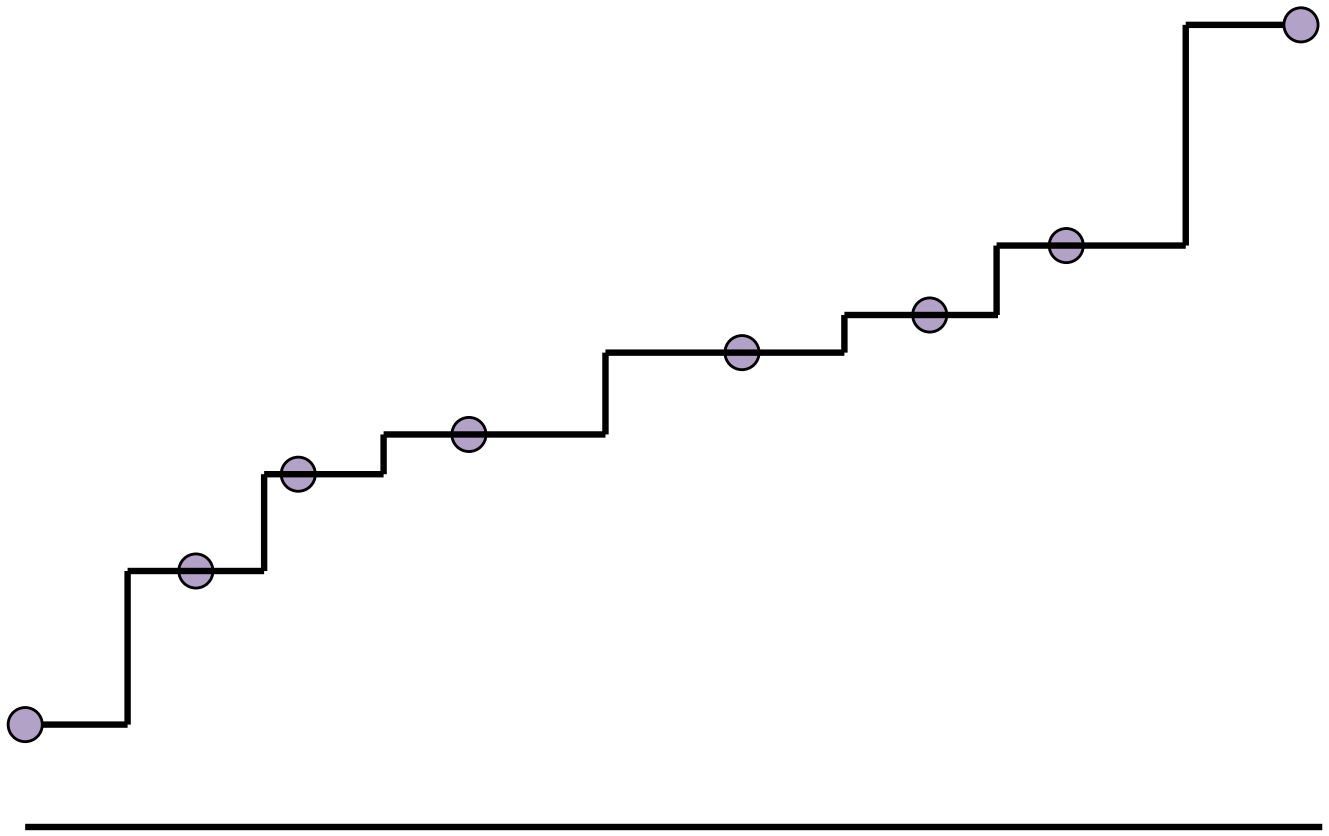




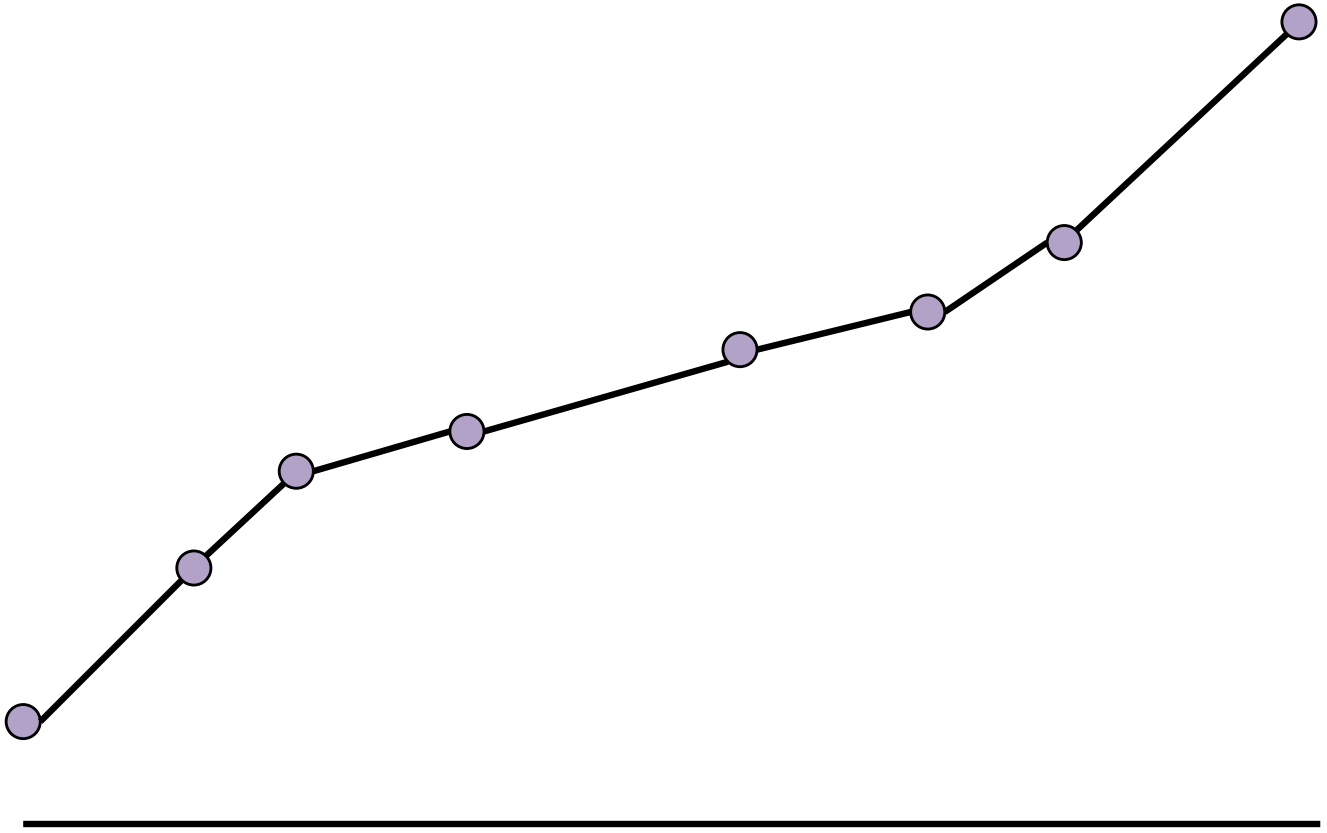


**k=9**

# Example: Nearest Point

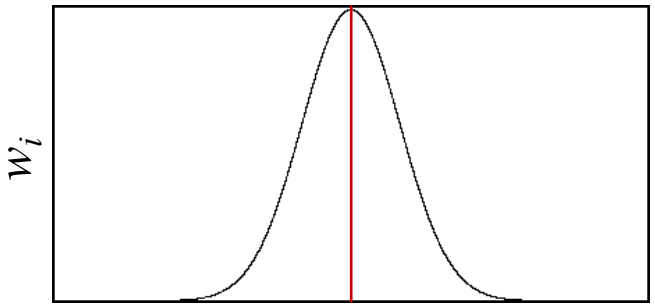


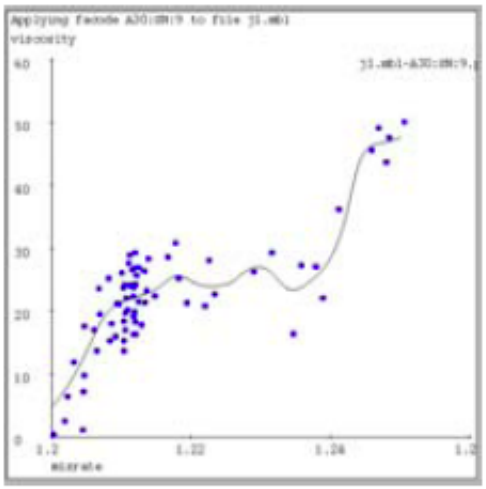
# Example: Weighted Average

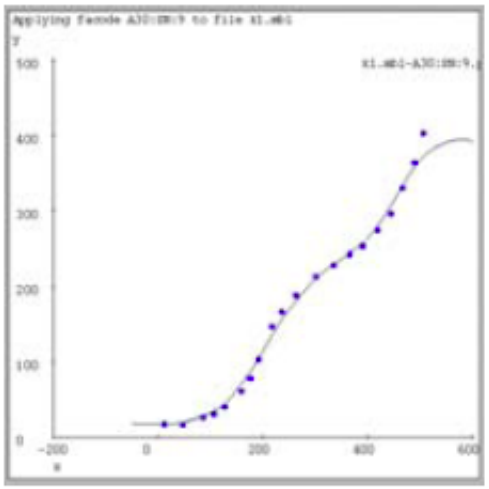


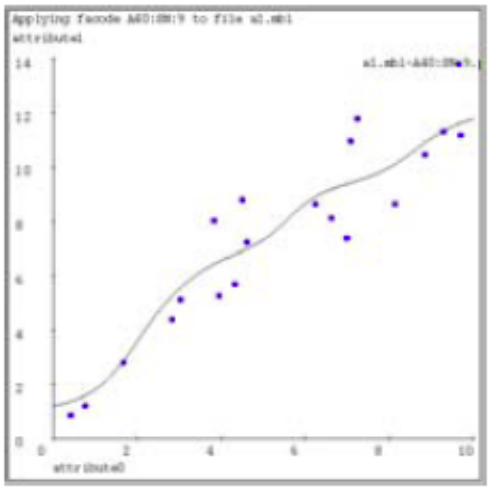
# Kernel Regression

- Distance metric:
  - Euclidean
- How many neighbors to look at?
  - All of them (!)
- Weighting function:
  - $w_i = \exp(-\frac{d(x_i,q)^2}{K_w})$
    - Nearby points to query q are weighted more strongly. $K_w$…kernel width.
- How to fit with the local points?
  - Predict weighted average: $\frac{\sum_i w_i y_i}{\sum_i w_i}$



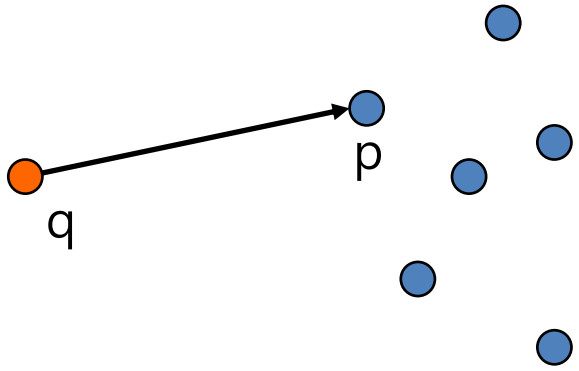







# How to find nearest neighbors?

- Given: a set $P$ of $n$ points in $R^d$
- Goal: Given a query point $q$
  - NN: Find the *nearest neighbor* $p$ of $q$ in $P$
  - Range search: Find one/all points in $P$ within distance $r$ from $q$



# Algorithms for NN

- Main memory:
  - Linear scan
  - Tree based:
    - Quadtree
    - kd-tree
  - Hashing:
    - Locality-Sensitive Hashing
- Secondary storage:
  - R-trees