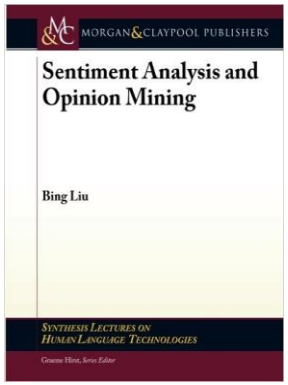
Principles/Social Media Mining

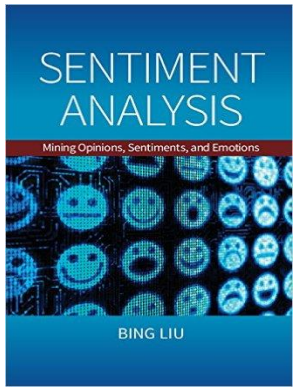
CIS 600

Weeks 11 & 12: Sentiment Analysis, Part 3: Algorithms & Tools

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

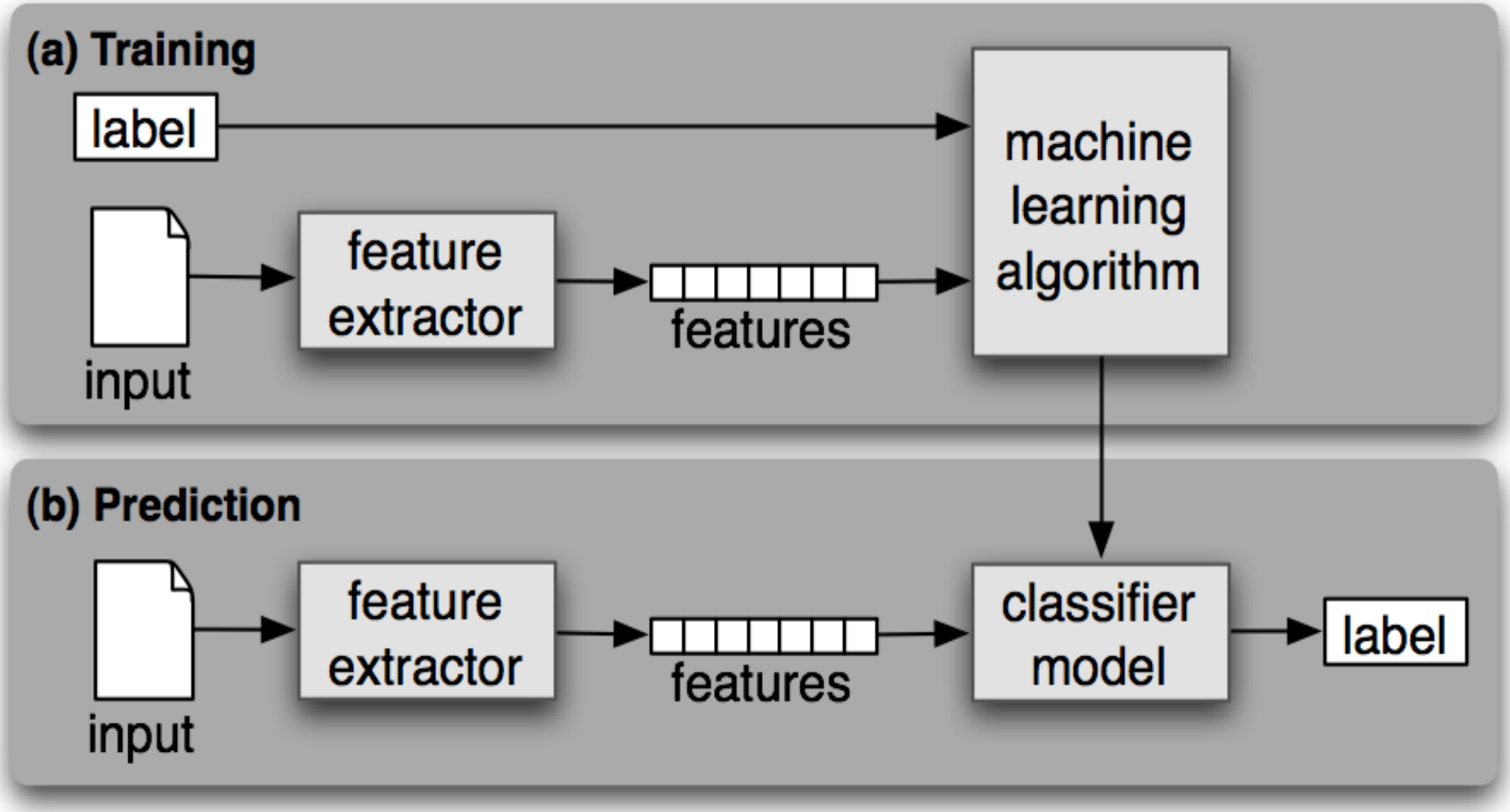
November 5, 10, 2020





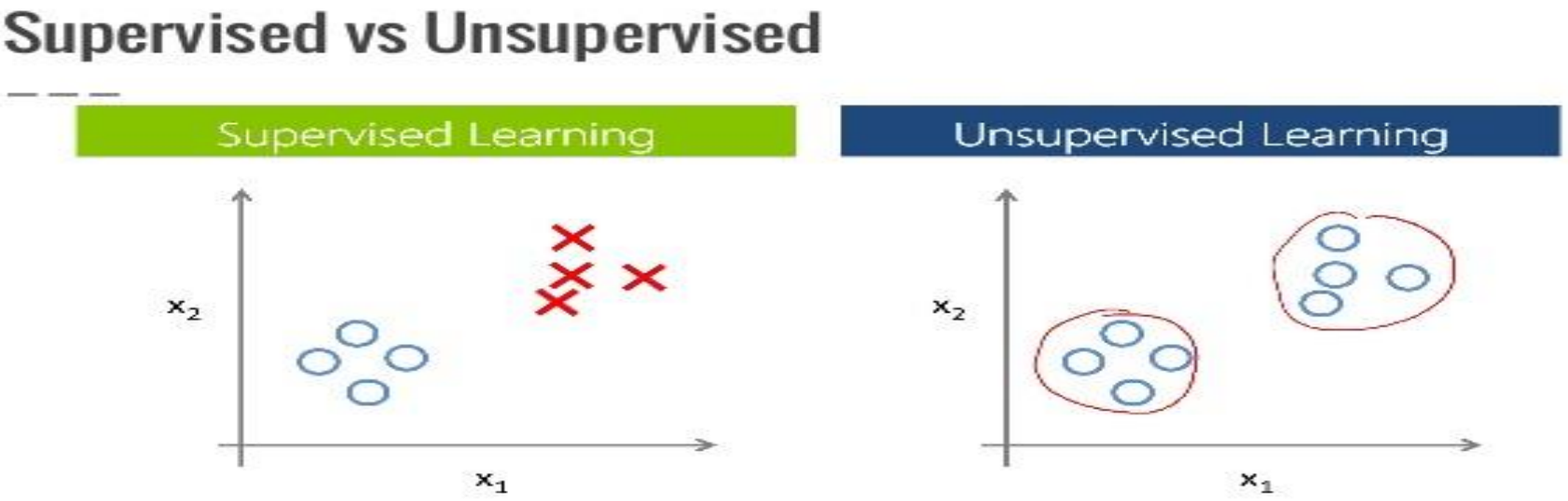
Sentiment Analysis On Twitter

- **Step 2. Classification**. After the data has been prepared, it is time to feed it to the classifier.



Supervised vs. Unsupervised

- **Supervised learning**
  - The training data (observations) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- **Unsupervised learning (clustering)**
  - The class labels of training data is **unknown**
  - Given a set of observations, find the classes or clusters in the data



Semi-supervised Clustering

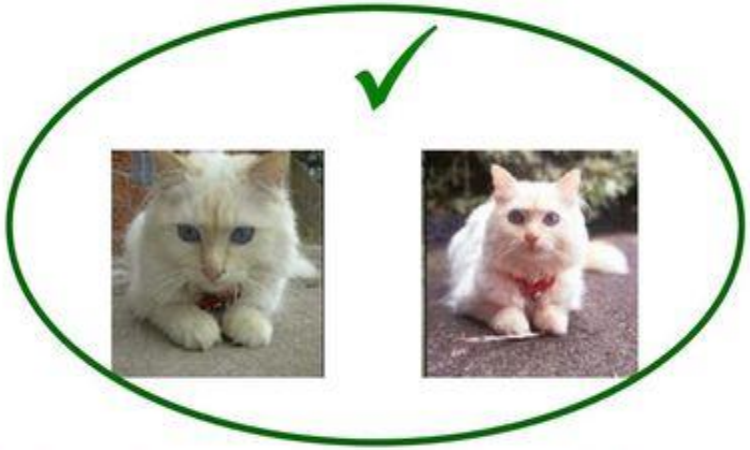
Supervised

Dogs

Cats

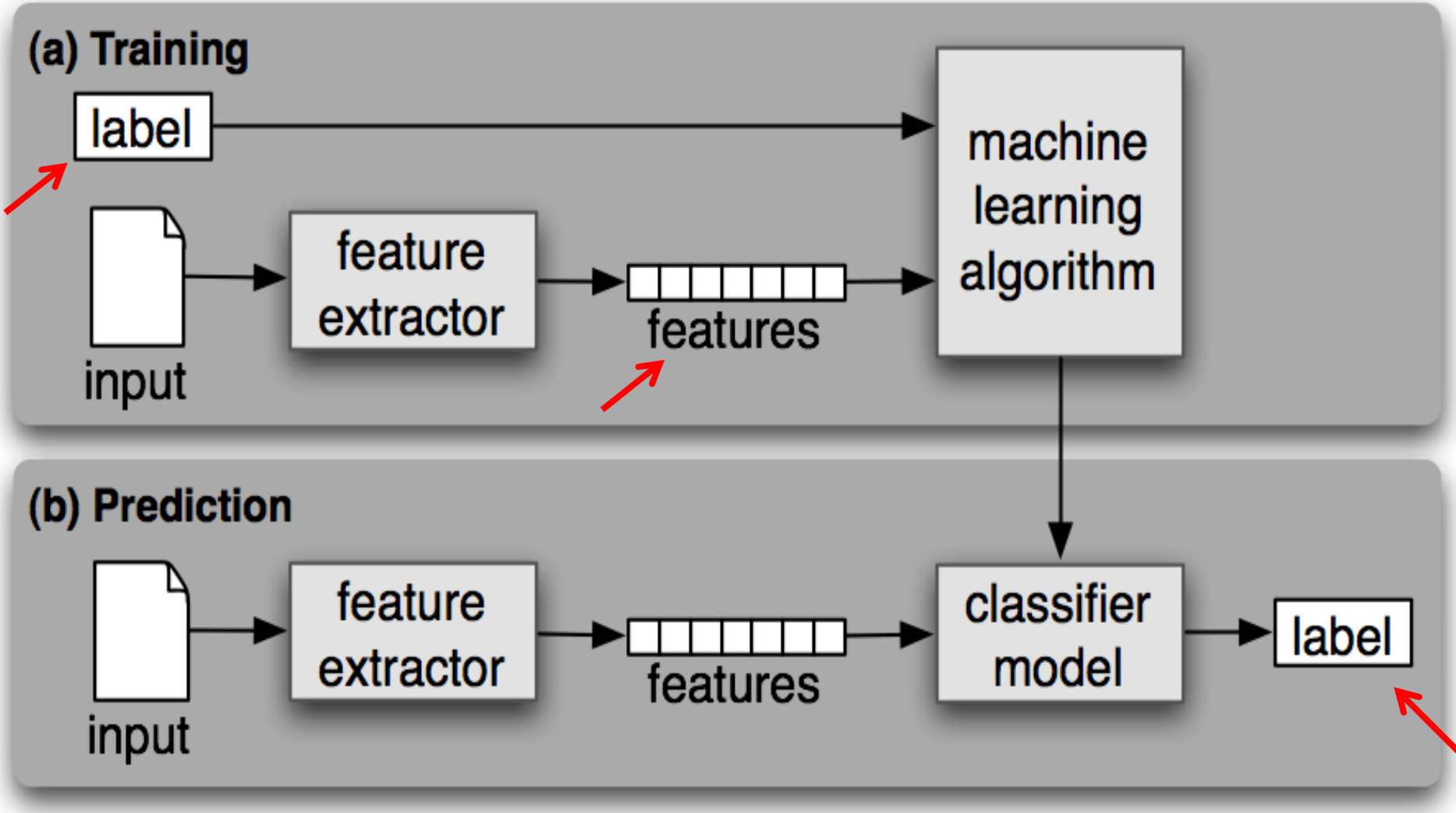
Unsupervised

Semi-supervised



Pairwise constraints improve the clustering performance

Supervised Classification



Labels and Features

- In order to **build** your own classifiers:
  1. You have to select either the **positive/negative** or **positive/negative/neutral** approach to label the training data.
  2. You have to decide on the 'features' you want to use.
     - The commonly used N-Gram model – i.e. split each sentence into chunks of N words/tokens, and use the resulting N-grams to feed the classifier

       "I don't like to drink tea"

     - Uni-grams (1-grams): "I" "don't" "like" "to" "drink" "tea"
     - Bi-grams (2-grams): "I don't" "don't like" "like to" "to drink" "drink tea"
     - Tri-grams (3-grams): "I don't like" "don't like to" "like to drink" "to drink tea"

NGrams: NLTK

- NLTK provides functions for bigrams(list_of_tokens) and trigrams(list_of_tokens), and a general-purpose ngrams(list_of_tokens, n):

```
from nltk import *

t = '#qcpoli enjoyed a hearty laugh today with #plq debate audience for @jflisee
    #notrehome tune was that the intended reaction?'
tt = TweetTokenizer(t)
tokens = tt.tokenize(t)

# N-Grams
for t in bigrams(tokens): print(t)
for t in trigrams(tokens): print(t)
for t in ngrams(tokens, 4): print(t)
…
```

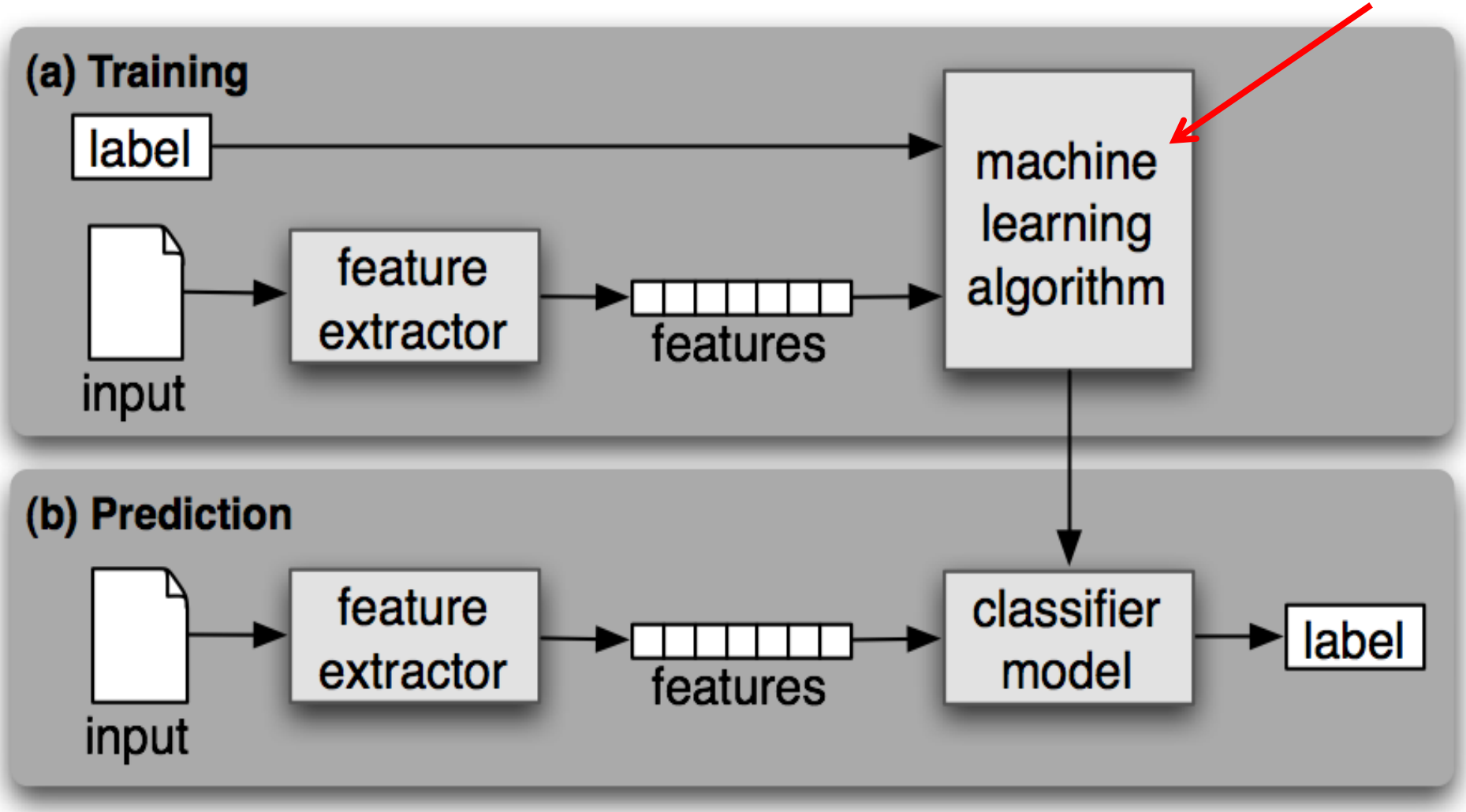
Sentiment Analysis On Twitter

- Once the terms are broken into N-grams, you may attach weights to them based on their sentiment.
  - The weights may come from a dictionary (Sentiment Lexicon)
  - N-grams that display little sentiment may be dropped.
    - In our example, the following uni-grams would probably be dropped:

      "to", "drink" and "tea"
  - Punctuation marks and cases are also common classification features.
    - Tweeters use them to display their **emotions** more clearly to the audience.
    - "I need to talk to you." vs "I NEED TO TALK TO YOU!!!!" .

Sentiment Analysis On Twitter

- **Step 2. Classification**. After the data has been prepared, it is time to feed it to the classifier.



Naive Bayes Classifiers

- Is a statistical classifier
  - It performs probabilistic prediction, i.e., predicts class membership probabilities
- Is well-founded
  - Based on **Bayes' Theorem**.
- Is simple
  - A simple Bayesian classifier, **naïve Bayes** classifier, has comparable performance with decision tree and neural network classifiers
- Is standard
  - They provide a standard of optimal decision making against which other methods can be measured

Bayes' Theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})} = P(\mathbf{X}|H) \times P(H) / P(\mathbf{X})$$

- Informally, this can be written as

  posterior = likelihood * prior / evidence

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|**X**), *posterior probability*, the probability that the hypothesis holds given the observed data sample **X**
- P(H): *prior probability*, the initial probability
- P(**X**): probability that data sample is observed
- P(**X**|H): *likelihood*, the probability of observing the sample **X**, given that the hypothesis holds

Naïve Bayes Classifiers

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an **n** attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to find the maximum posterior probability, i.e., the maximal P(C_i|**X**)
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(**X**) is constant for all classes, only

$$P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayes Classifiers

- A simplified assumption: attributes are conditionally independent (i.e., no dependence relation between attributes):

$$P(\mathbf{X}\,|\,C_i) = \prod_{k=1}^{n} P(x_k\,|\,C_i) = P(x_1\,|\,C_i) \times P(x_2\,|\,C_i) \times \ldots \times P(x_n\,|\,C_i)$$

Naïve Bayes Classifier: Training

Class:
C1:buys_computer = ‘yes’
C2:buys_computer = ‘no’

Data to be classified:
X = (age <=30,
Income = medium,
Student = yes
Credit_rating = Fair)

age	income	student	credit_rating	_comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Naïve Bayes Classifier

age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$P(C_i)$: P(buys_computer = "yes") = 9/14 = 0.643

P(buys_computer = "no") = 5/14= 0.357

Compute $P(X|C_i)$ for each class:

P(age = "<=30" | buys_computer = "yes") = 2/9 = 0.222

P(age = "<=30" | buys_computer = "no") = 3/5 = 0.6

P(income = "medium" | buys_computer = "yes") = 4/9 = 0.444

P(income = "medium" | buys_computer = "no") = 2/5 = 0.4

P(student = "yes" | buys_computer = "yes) = 6/9 = 0.667

P(student = "yes" | buys_computer = "no") = 1/5 = 0.2

P(credit_rating = "fair" | buys_computer = "yes") = 6/9 = 0.667

P(credit_rating = "fair" | buys_computer = "no") = 2/5 = 0.4

X = (age <= 30 , income = medium, student = yes, credit_rating = fair)

$P(X|C_i)$: P(X|buys_computer = "yes") = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

P(X|buys_computer = "no") = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

$P(X|C_i)*P(C_i)$: P(X|buys_computer = "yes") * P(buys_computer = "yes") = **0.028**

P(X|buys_computer = "no") * P(buys_computer = "no") = 0.007

Therefore, X belongs to class ("buys_computer = yes")

Another Example: Words

w1	w2	w3	w4	Class
1	0	0	1	1
0	0	0	1	0
1	1	0	1	0
1	0	1	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
0	1	0	0	1
0	1	0	1	0
1	1	1	0	0

	Class=1	Class=0
Pr(Class)	0.40	0.60
Pr(w1 | Class)	0.75	0.50
Pr(w2 | Class)	0.25	0.67
Pr(w3 | Class)	0.50	0.33
Pr(w4 | Class)	0.50	0.50

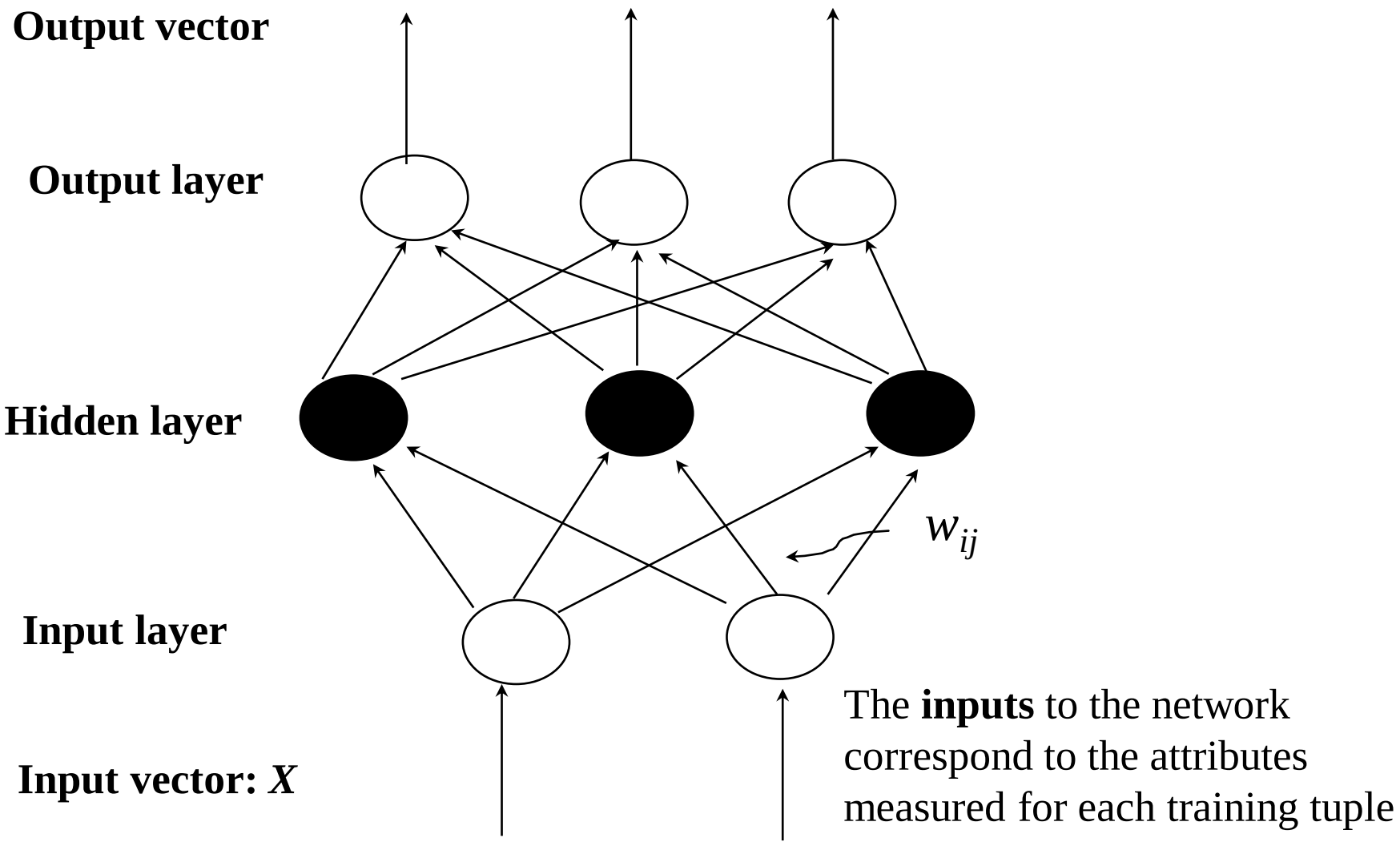
Avoid Zero-Probability

- Naïve Bayesian prediction requires each conditional probability be **non-zero**. Otherwise, the predicted probability will be zero

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

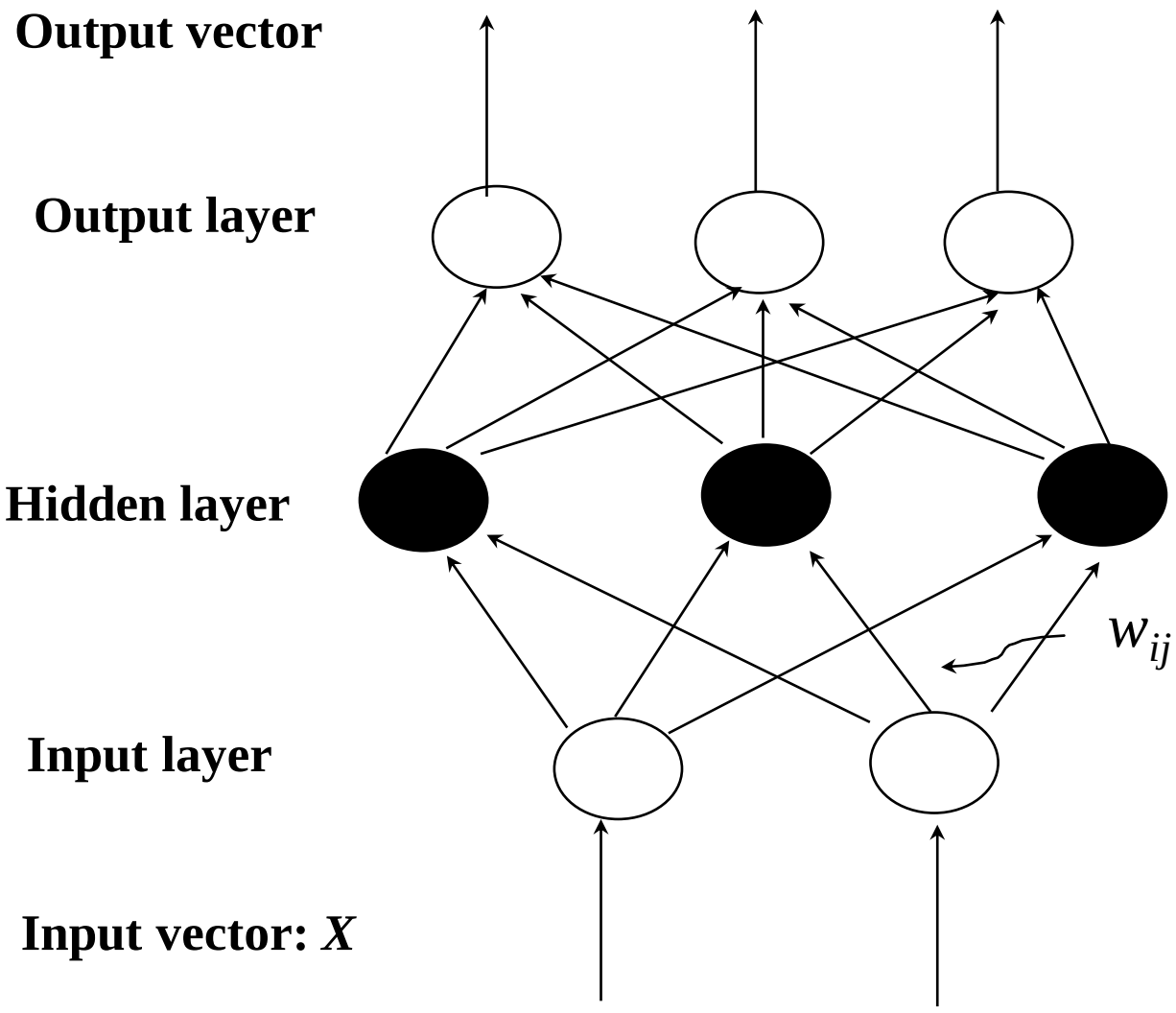
- Example: Suppose a dataset with 1000 tuples, income=low (0), income= medium (990), and income = high (10)
- Use **Laplacian correction** (or Laplacian estimator)
  - *Adding 1 to each case*
    - Prob(income = low) = 1/1003
    - Prob(income = medium) = 991/1003
    - Prob(income = high) = 11/1003
  - The "corrected" probability estimates are close to their "uncorrected" counterparts

A Multi-Layer Feed-Forward Neural Network



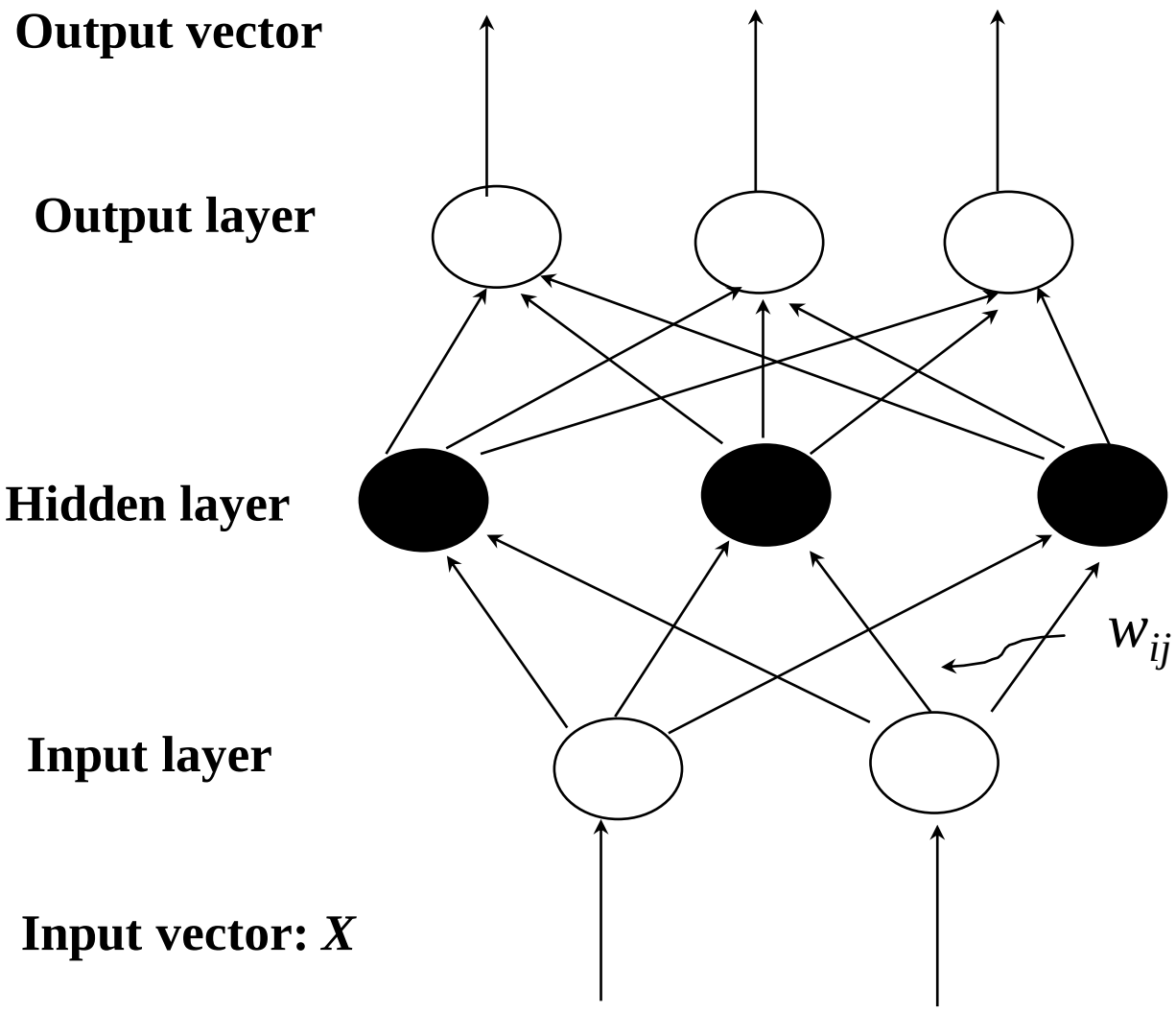
The **inputs** to the network correspond to the attributes measured for each training tuple

A Multi-Layer Feed-Forward Neural Network



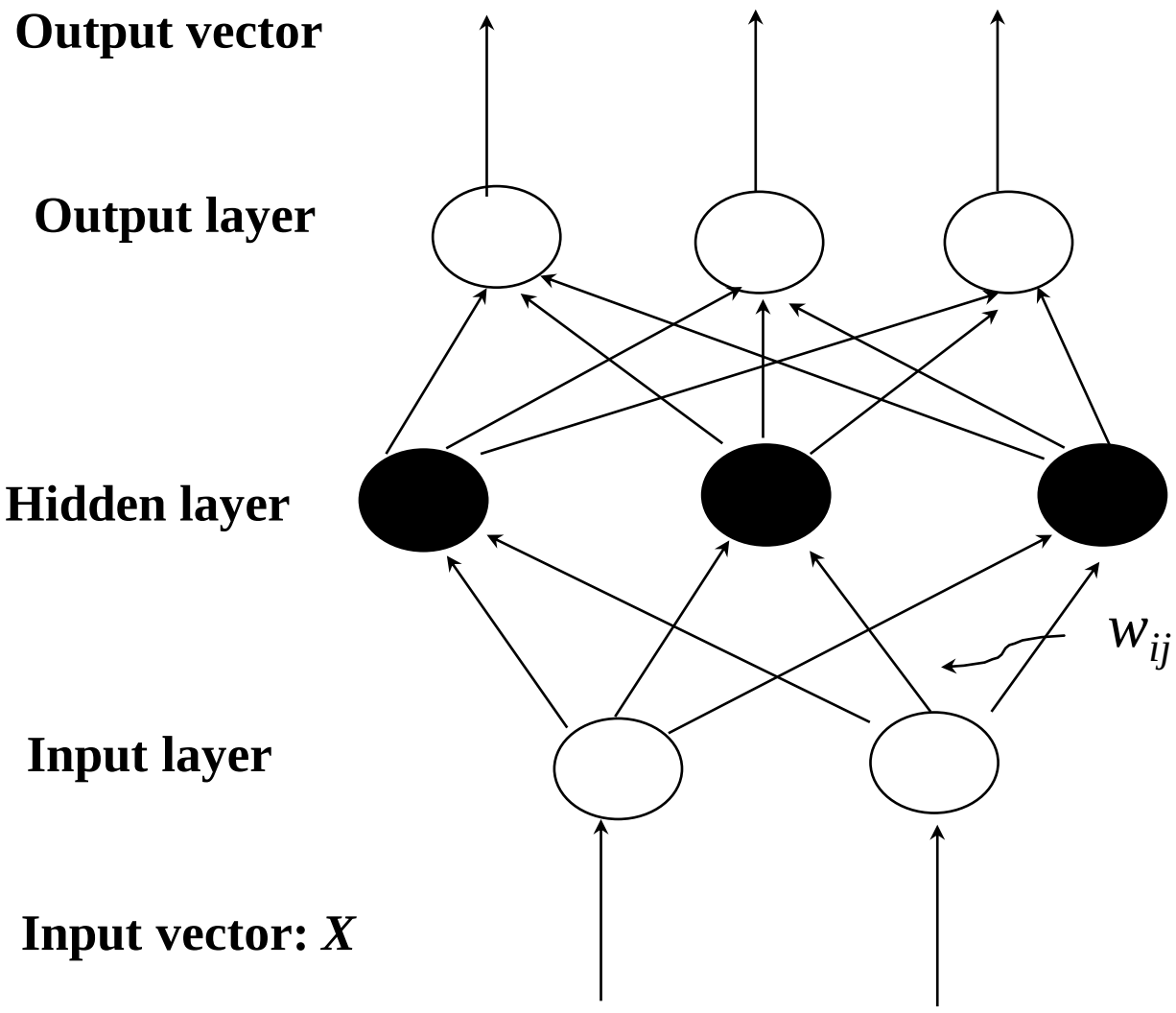
Inputs are fed simultaneously into the units making up the **input layer.**

A Multi-Layer Feed-Forward Neural Network



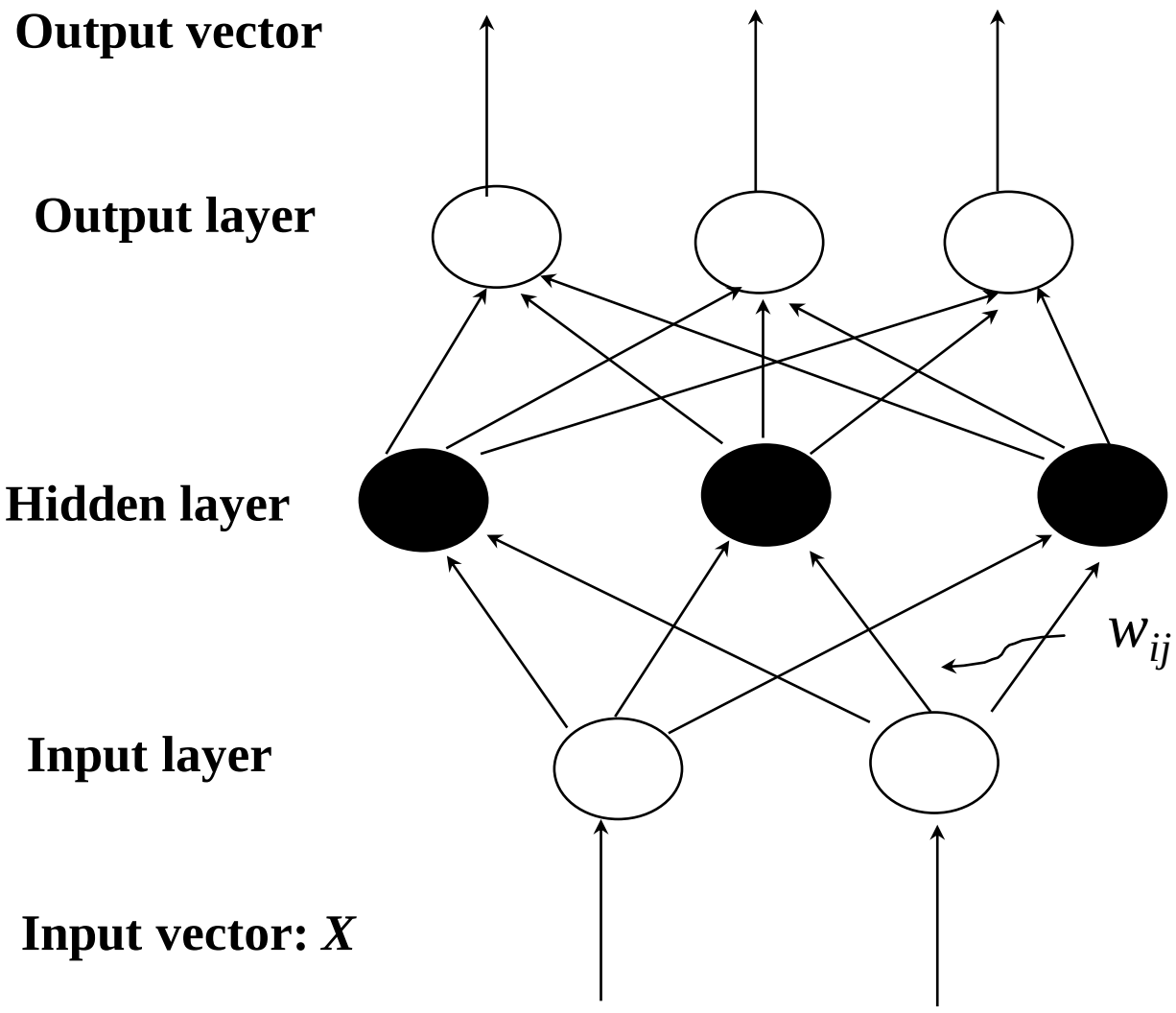
They are then weighted and fed simultaneously to a **hidden layer**

A Multi-Layer Feed-Forward Neural Network



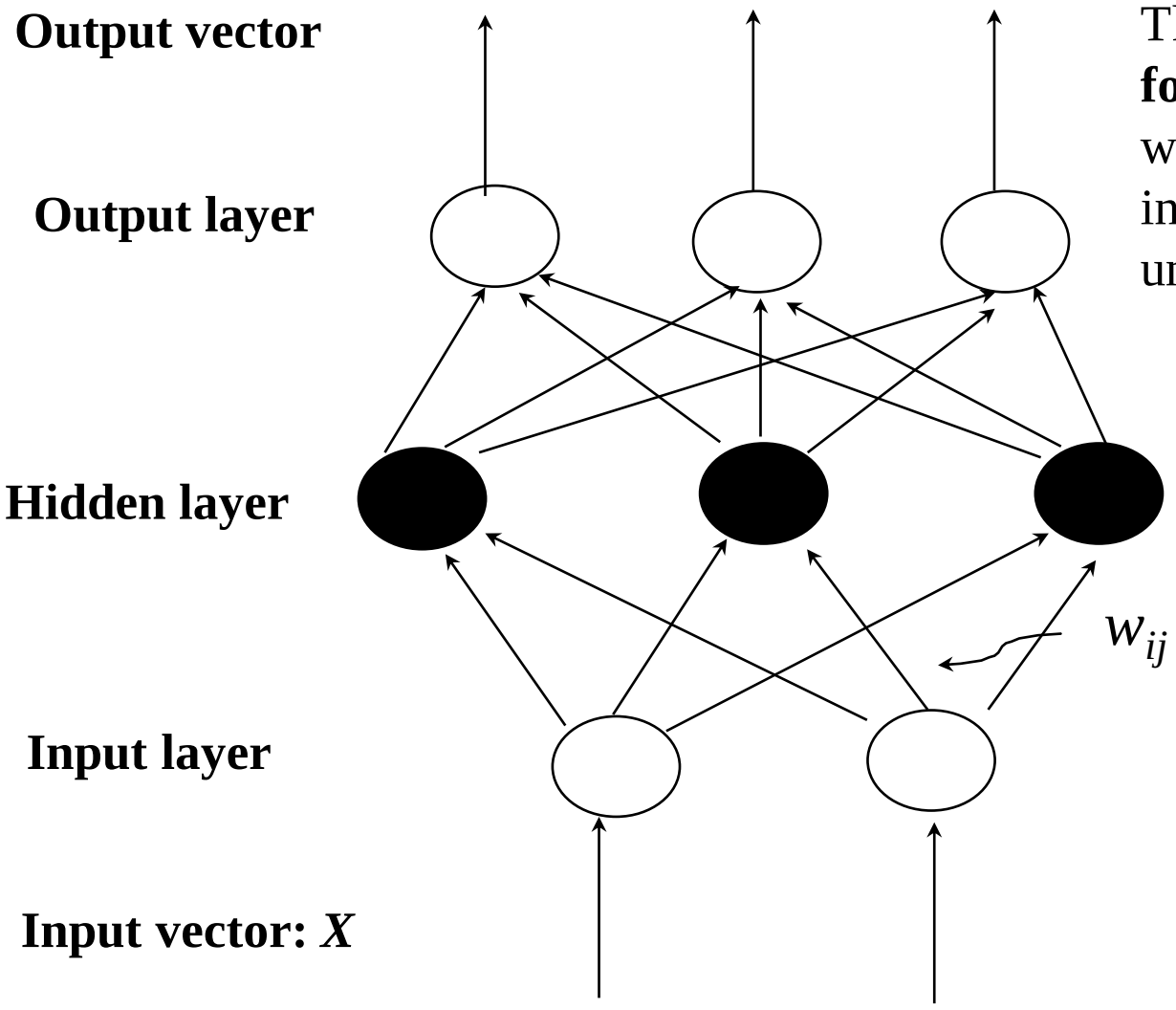
The number of hidden layers is arbitrary, although usually only one

A Multi-Layer Feed-Forward Neural Network



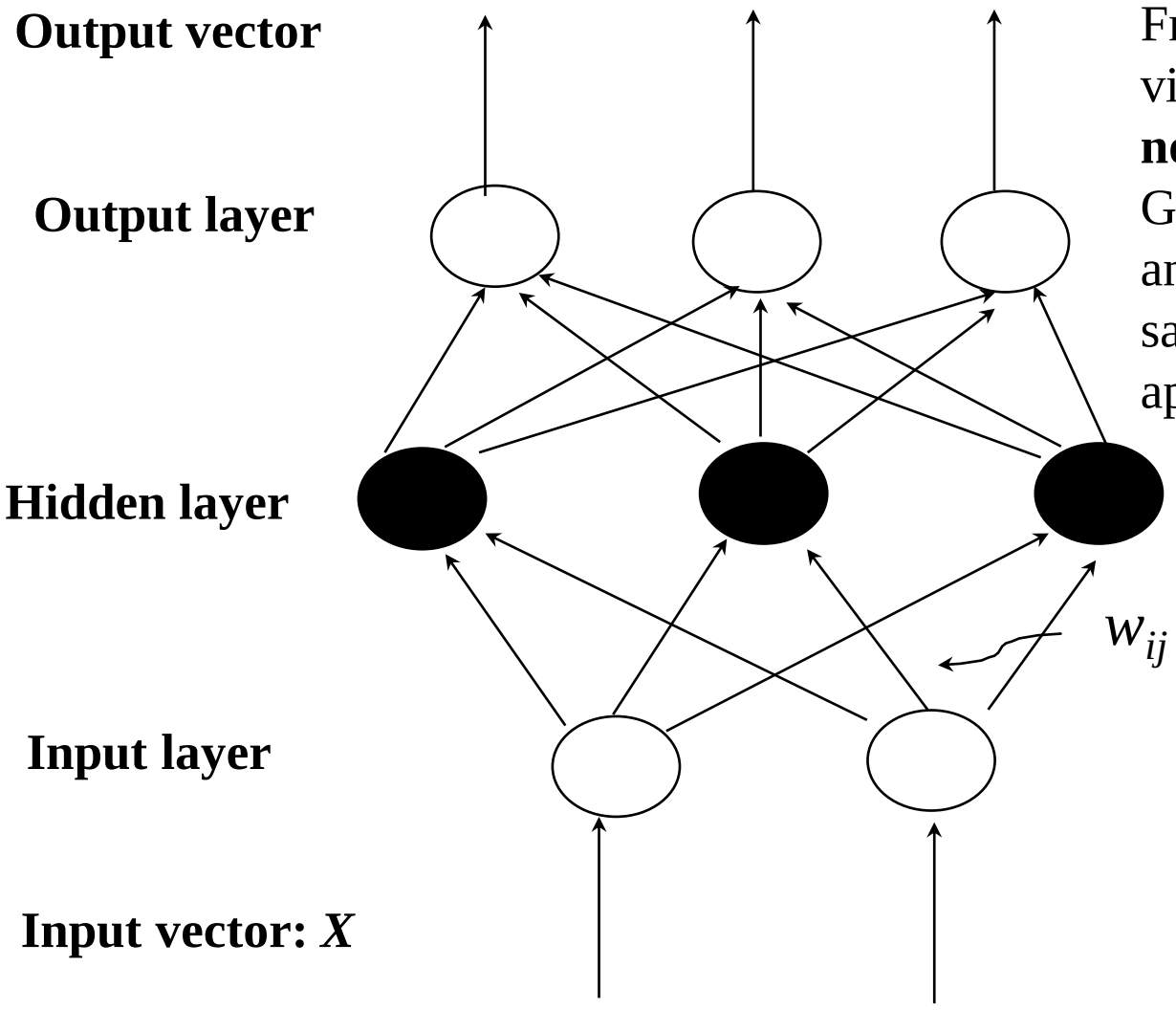
The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction

A Multi-Layer Feed-Forward Neural Network



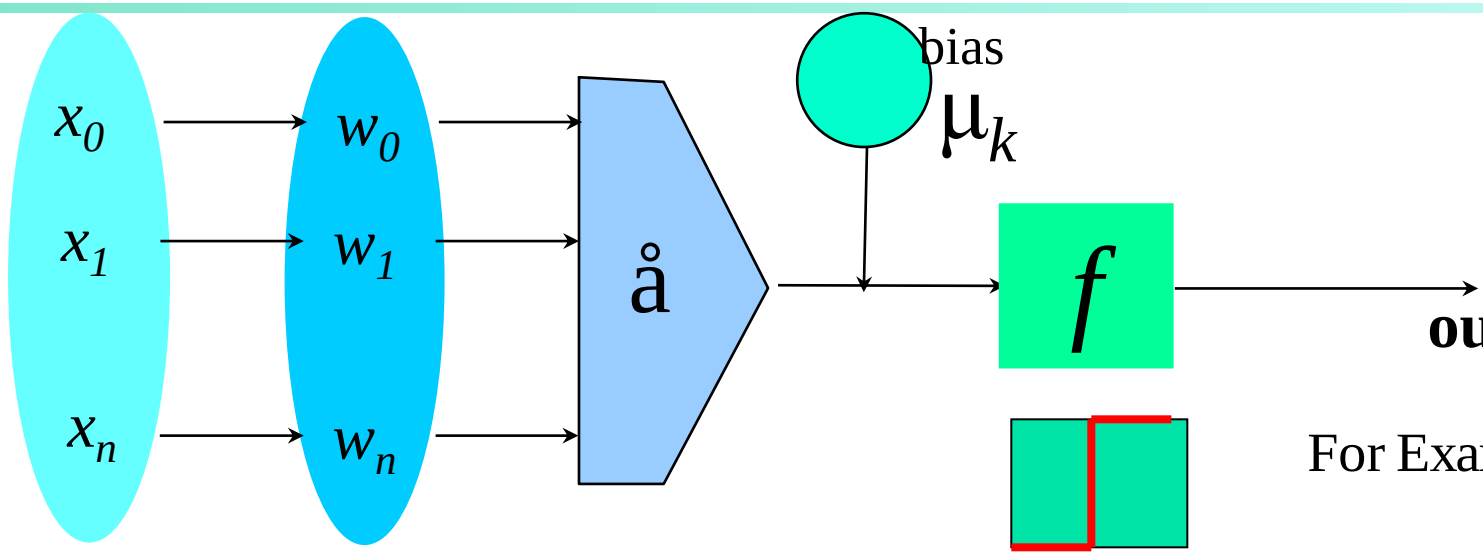
This network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer

A Multi-Layer Feed-Forward Neural Network



From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Neuron: A Hidden/Output Layer Unit



For Example

$$y = \text{sign}\left(\sum_{i=0}^{n} w_i x_i - \mu_k\right)$$

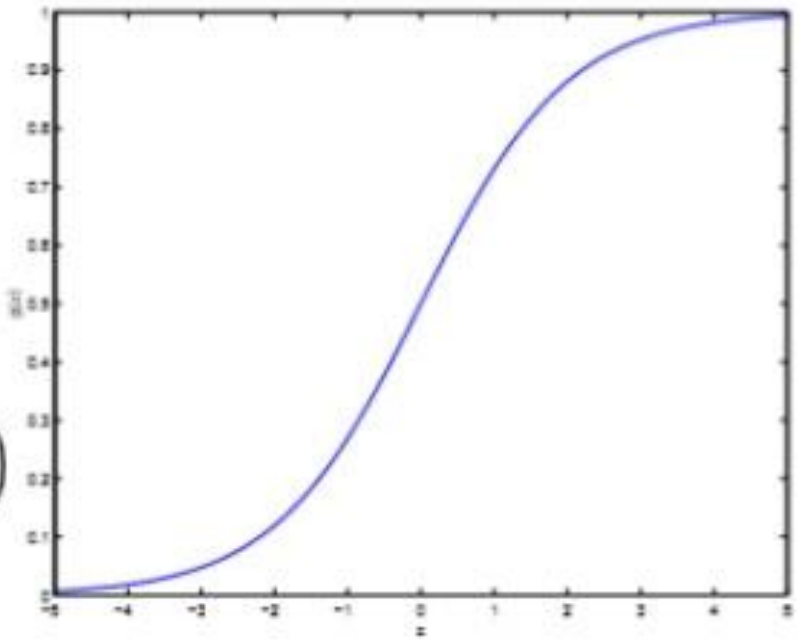
Input vector x **weight vector w** **weighted sum** **Activation function**

- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

Logistic function (Sigmoid)

$$g(z) = \frac{1}{1+e^{-z}}$$

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}(e^{-z}) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$



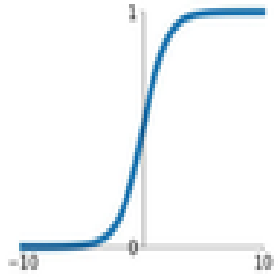
The derivative of logistic function has a nice feature:

$$g'(z) = g(z)(1 - g(z))$$

Activation Functions

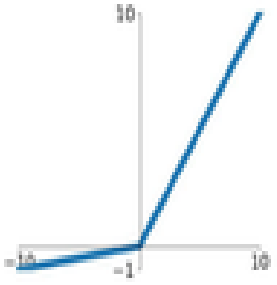
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



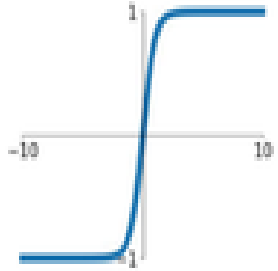
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

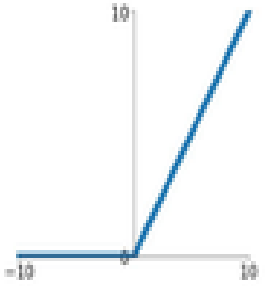


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

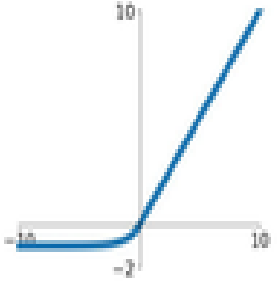
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Defining a Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

Backpropagation Algorithm

Input: Data set D, learning rate l, network **Output:** Trained Neural Network

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3) **for** each training tuple X in D {
(4) // Propagate the inputs forward:
(5) **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7) **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
(10) // Backpropagate the errors:
(11) **for** each unit j in the output layer
(12) $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
(13) **for** each unit j in the hidden layers, from the last to the first hidden layer
(14) $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
(15) **for** each weight w_{ij} in *network* {
(16) $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
(17) $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
(18) **for** each bias θ_j in *network* {
(19) $\Delta \theta_j = (l) Err_j$; // bias increment
(20) $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
(21) } }

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Backpropagation

From Wikipedia, the free encyclopedia

This article is about the computer algorithm. For the biological process, see neural backpropagation.

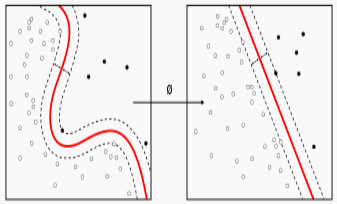
Backpropagation can also refer to the way the result of a playout is propagated up the search tree in Monte Carlo tree search.

In machine learning, specifically deep learning, **backpropagation** (**backprop**,[1] **BP**) is an algorithm widely used in the training of feedforward neural networks for supervised learning; generalizations exist for other artificial neural networks (ANNs), and for functions generally.[2] Backpropagation efficiently computes the gradient of the loss function with respect to the weights of the network for a single input-output example. This makes it feasible to use gradient methods for training multi-layer networks, updating weights to minimize loss; commonly one uses gradient descent or variants such as stochastic gradient descent. The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, iterating backwards one layer at a time from the last layer to avoid redundant calculations of intermediate terms in the chain rule; this is an example of dynamic programming.[3]

The term *backpropagation* strictly refers only to the algorithm for computing the gradient, but it is often used loosely to refer to the entire learning algorithm, also including how the gradient is used, such as by stochastic gradient descent.[4] Backpropagation generalizes the gradient computation in the Delta rule, which is the single-layer version of backpropagation, and is in turn generalized by automatic differentiation, where backpropagation is a special case of reverse accumulation (or "reverse mode").[5] The term *backpropagation* and its general use in neural networks was announced in Rumelhart, Hinton & Williams (1986a), then elaborated and popularized in Rumelhart, Hinton & Williams (1986b), but the technique was independently rediscovered many times, and had many predecessors dating to the 1960s; see § History.[6] A modern overview is given in Goodfellow, Bengio & Courville (2016).[7]

Contents [hide]

Machine learning and data mining



Problems [show]

Supervised learning [show]
(**classification** • **regression**)

Clustering [show]

Dimensionality reduction [show]

Structured prediction [show]

Anomaly detection [show]

Artificial neural networks [show]

Reinforcement learning [show]

Theory [show]

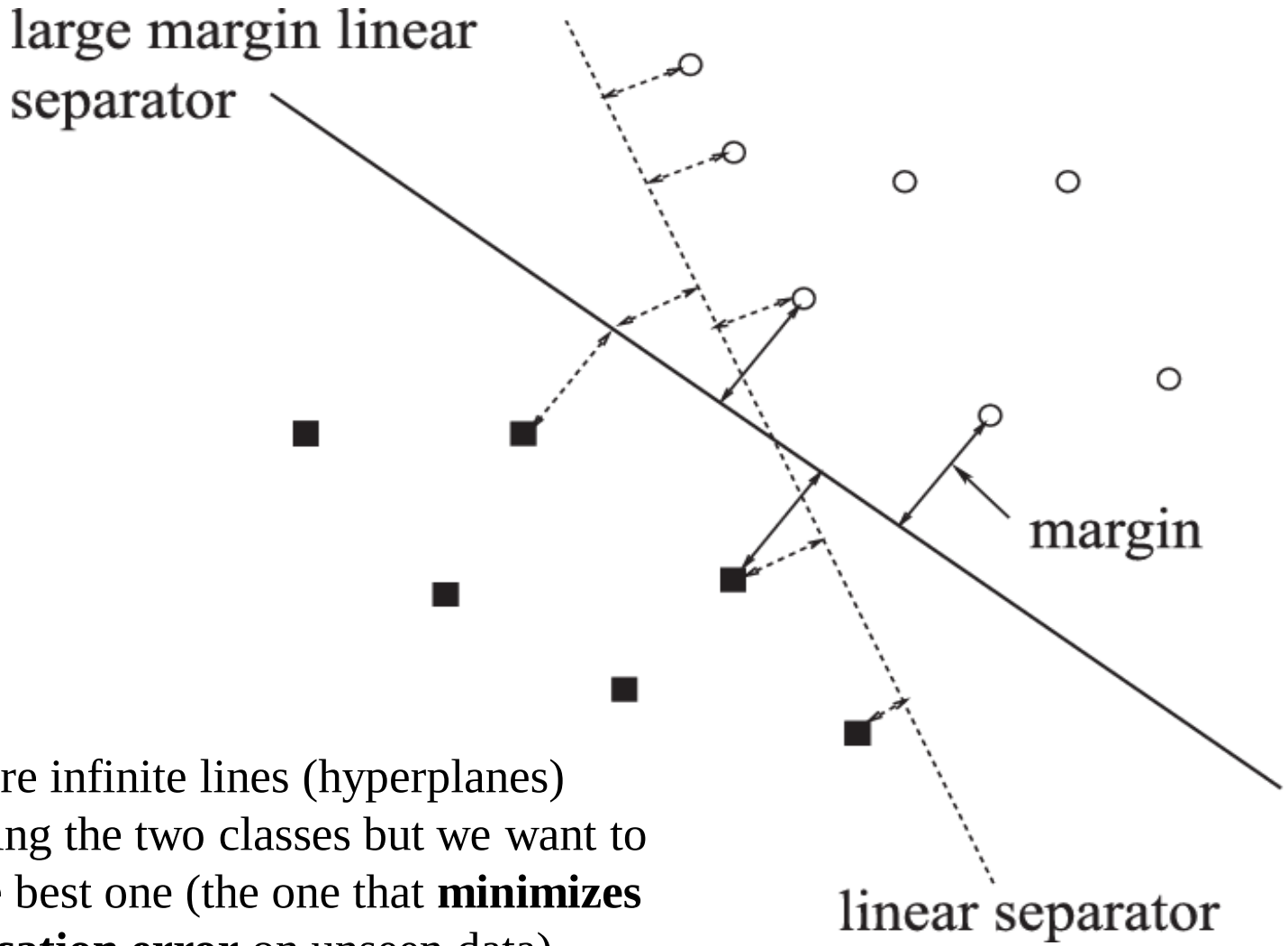
Machine-learning venues [show]

Glossary of artificial intelligence [show]

Efficiency and Interpretability

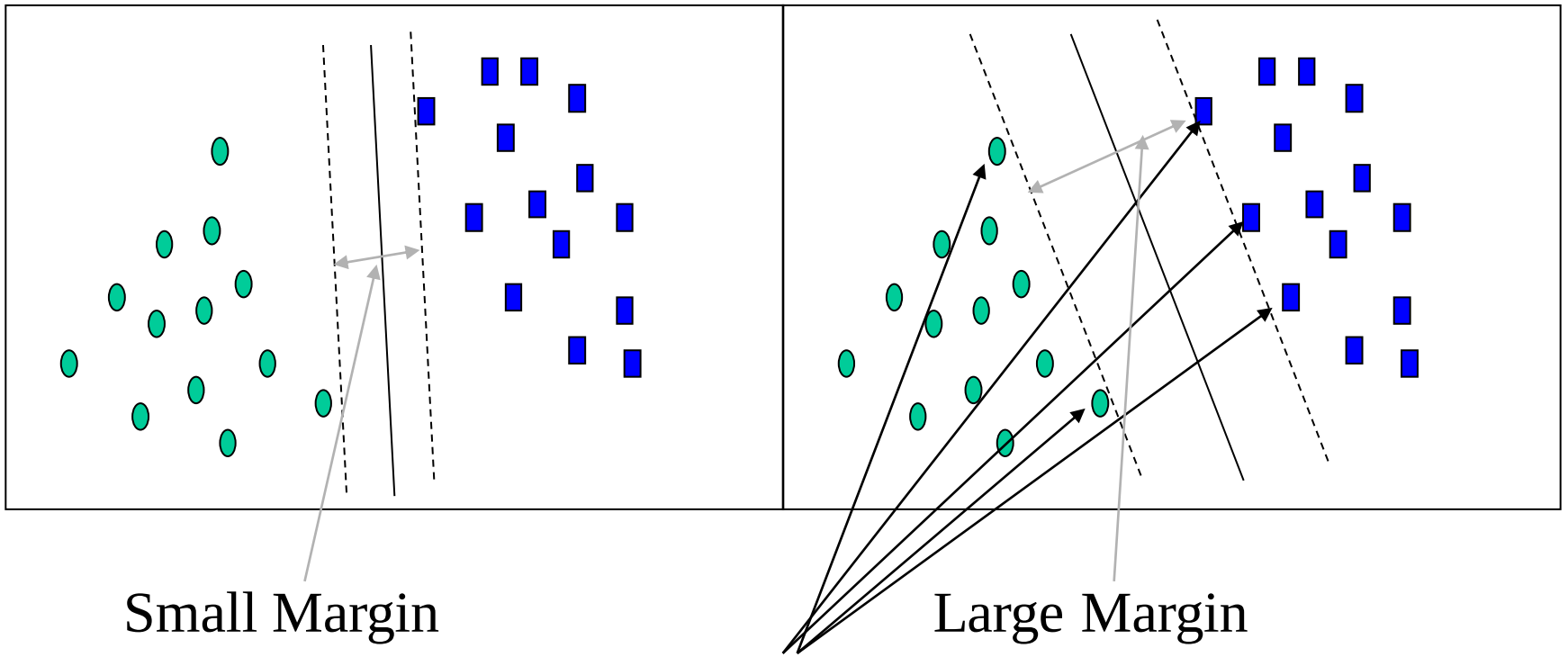
- **Efficiency** of backpropagation: Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in worst case
- For easier comprehension: **Rule extraction** by network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- **Sensitivity analysis**: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

SVM - General Philosophy



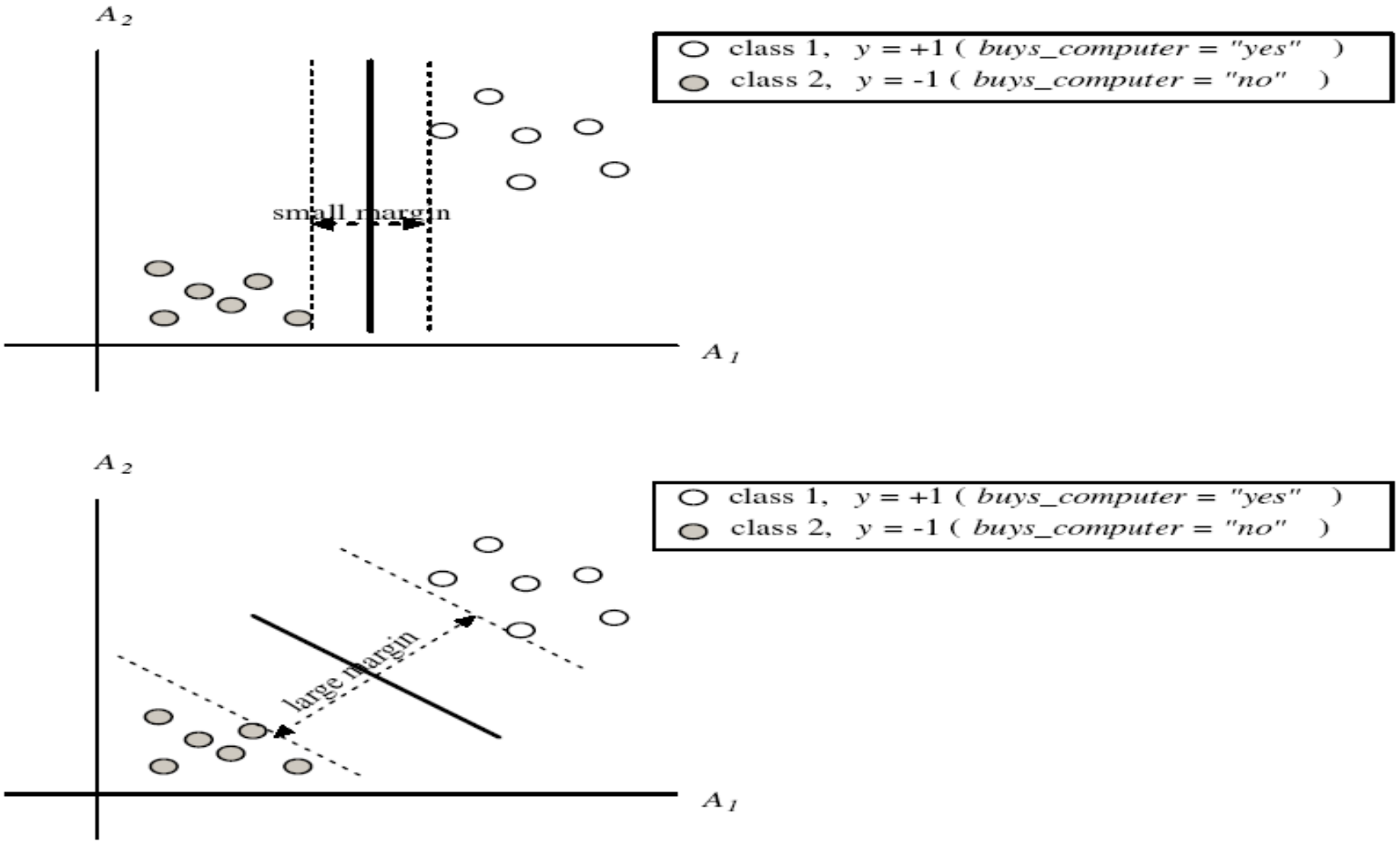
There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that **minimizes classification error** on unseen data)

SVM - General Philosophy



SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane**

SVM - Margins and Support Vectors



SVM - Linearly Separable

A separating hyperplane can be written as

$$\mathbf{w} \bullet \mathbf{X} + b = 0$$

where $\mathbf{w}=\{w_1, w_2, \ldots, w_n\}$ is a **weight vector** and b a scalar (bias)

For 2-D it can be written as

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

SVM - Linearly Separable

1. SUPPORT VECTOR MACHINES FOR CLASSIFICATION

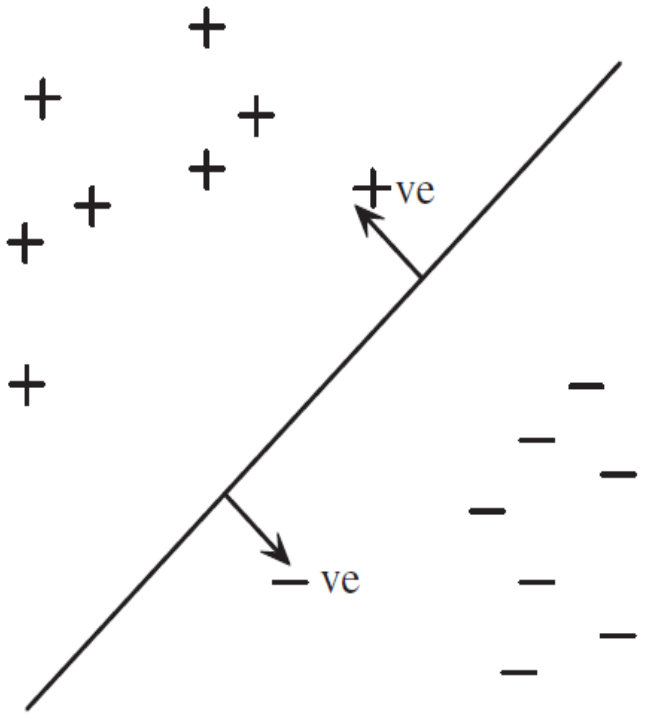


Figure 1.1: The argument inside the decision function of a classifier is $\mathbf{w} \cdot \mathbf{x} + b$. The separating hyperplane corresponding to $\mathbf{w} \cdot \mathbf{x} + b = 0$ is shown as a line in this 2-dimensional plot. This hyperplane separates the two classes of data with points on one side labelled $y_i = +1$ $(\mathbf{w} \cdot \mathbf{x} + b \geq 0)$ and points on the other side labelled $y_i = -1$ $(\mathbf{w} \cdot \mathbf{x} + b < 0)$.

SVM - Linearly Separable

Let's call the following function f the **decision function:**

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

From the decision function we see that the data is correctly classified if

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0$$

since $(\mathbf{w} \cdot \mathbf{x}_i + b)$ should be positive when $y_i = +1$, and it should be negative when $y_i = -1$.

The hyperplanes passing through $\mathbf{w} \cdot \mathbf{x} + b = 1$ and $\mathbf{w} \cdot \mathbf{x} + b = -1$ are called **canonical hyperplanes.**

The region between these canonical hyperplanes is called the **margin band.**

SVM - Linearly Separable

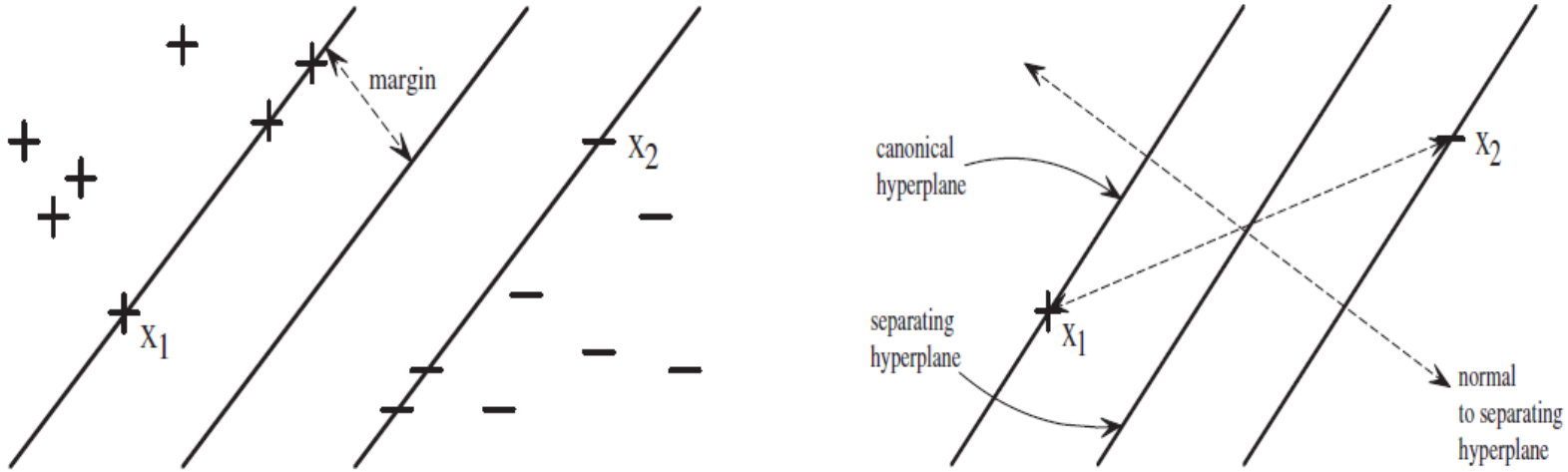


Figure 1.2: *Left*: The perpendicular distance between the separating hyperplane and a hyperplane through the closest points (the support vectors) is called the *margin*, γ. $\mathbf{x}_1$ and $\mathbf{x}_2$ are examples of *support vectors* of opposite sign. The hyperplanes passing through the support vectors are the *canonical hyperplanes*, and the region between the canonical hyperplanes is the *margin band*. *Right*: the projection of the vector $(\mathbf{x}_1 - \mathbf{x}_2)$ onto the normal to the separating hyperplane $(\mathbf{w}/\,||\mathbf{w}||_2)$ is 2γ.

SVM - Linearly Separable

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two points inside the canonical hyperplanes on both sides.

If $\mathbf{w}\cdot\mathbf{x}_1 + b = 1$ & $\mathbf{w}\cdot\mathbf{x}_2 + b = -1 \quad \rightarrow \quad \mathbf{w} \cdot (\mathbf{x}1 - \mathbf{x}2) = 2.$

For the separating hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$, the normal vector is $\mathbf{w}/\|\mathbf{w}\|^2$, where $\|\mathbf{w}\|^2$ is the square root of $\mathbf{w}^T\mathbf{w}$

Thus the distance between the two canonical hyperplanes is equal to the **projection** of $\mathbf{x}_1 - \mathbf{x}_2$ onto the normal vector $\mathbf{w}/\|\mathbf{w}\|^2$:

$$(\mathbf{x}_1 - \mathbf{x}_2) \cdot \mathbf{w}/\|\mathbf{w}\|^2 = 2/\|\mathbf{w}\|^2$$

As half the distance between the two canonical hyperplanes, the margin is therefore $\gamma = 1/\|\mathbf{w}\|^2$

Maximizing the margin is therefore equivalent to minimizing: $\|\mathbf{w}\|^2$ with constraints: $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0 \rightarrow$ **quadratic programming**

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Vector projection

From Wikipedia, the free encyclopedia

The **vector projection** of a vector **a** on (or onto) a nonzero vector **b** (also known as the **vector component** or **vector resolution** of **a** in the direction of **b**) is the orthogonal projection of **a** onto a straight line parallel to **b**. It is a vector parallel to **b**, defined as

$$\mathbf{a}_1 = a_1 \hat{\mathbf{b}}$$

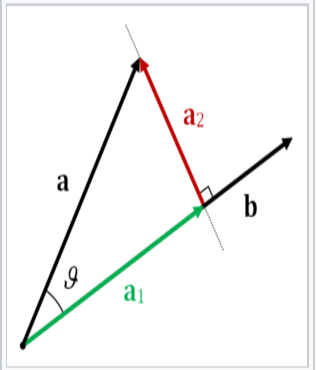
where a_1 is a scalar, called the scalar projection of **a** onto **b**, and $\hat{\mathbf{b}}$ is the unit vector in the direction of **b**. In turn, the scalar projection is defined as

$$a_1 = \|\mathbf{a}\| \cos\theta = \mathbf{a} \cdot \hat{\mathbf{b}} = \mathbf{a} \cdot \frac{\mathbf{b}}{\|\mathbf{b}\|}$$

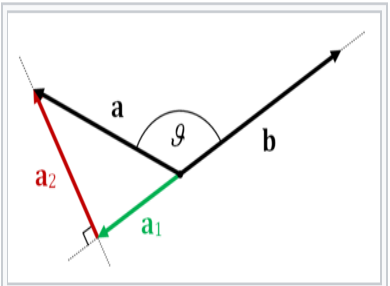
where the operator · denotes a dot product, ‖**a**‖ is the length of **a**, and θ is the angle between **a** and **b**. The scalar projection is equal to the length of the vector projection, with a minus sign if the direction of the projection is opposite to the direction of **b**.

The vector component or vector resolute of **a** perpendicular to **b**, sometimes also called the **vector rejection** of **a** *from* **b**,[1] is the orthogonal projection of **a** onto the plane (or, in general, hyperplane) orthogonal to **b**. Both the projection $\mathbf{a}_1$ and rejection $\mathbf{a}_2$ of a vector **a** are vectors, and their sum is equal to **a**, which implies that the rejection is given by

$$\mathbf{a}_2 = \mathbf{a} - \mathbf{a}_1.$$



Projection of **a** on **b** ($\mathbf{a}_1$), and rejection of **a** from **b** ($\mathbf{a}_2$).

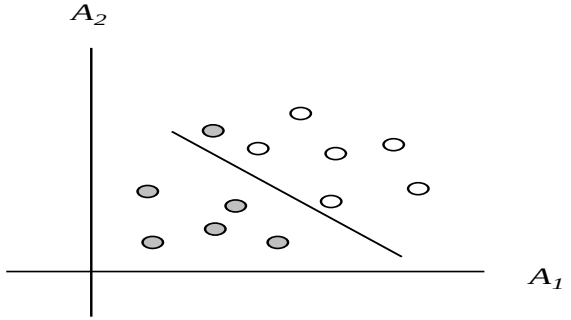


When $90° < \theta \leq 180°$, $\mathbf{a}_1$ has an opposite direction with respect to **b**.

Contents [hide]

When Data Is Linearly Inseparable

Transform the original input data into a higher dimensional space, and search for a linear separating hyperplane in the new space



Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3, \phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new ($\mathbf{Z}$) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned}$$

■

SVM - Linearly Inseparable

- How do we choose the nonlinear mapping (i.e. Φ in the previous slide) to a higher dimensional space?
  - It so happens that in solving the **quadratic optimization** problem of the linear SVM, the training feature vectors appear only in the form of dot products, $\Phi(\boldsymbol{X}_i)\ \Phi(\boldsymbol{X}_j)$, where $\Phi(\boldsymbol{X})$ is simply the nonlinear mapping function applied to transform the training features vectors.
  - So, instead of computing the dot product on the transformed data, it turns out that it is mathematically equivalent to apply a **kernel function**, $K(\boldsymbol{X}_i, \boldsymbol{X}_j)$, to the original input data:

$$K(\boldsymbol{X}_i, \boldsymbol{X}_j) = \Phi(\boldsymbol{X}_i)\ \Phi(\boldsymbol{X}_j)$$

SVM - Linearly Inseparable

- In other words, everywhere that $\Phi(\boldsymbol{X}_i)\ \Phi(\boldsymbol{X}_j)$ appears in the training algorithm, we can replace it with $K(\boldsymbol{X}_i, \boldsymbol{X}_j)$.
  - In this way, all calculations are made in the original input space, which is of potentially much lower dimensionality!
  - We can safely avoid the mapping - it turns out that we don't even have to know what the mapping is!

SVM - Linearly Inseparable

- **What are some of the kernel functions that could be used?**
  - Properties of the kinds of kernel functions that could be used to replace the dot product scenario described above have been studied.
  - Three admissible kernel functions include:
    - **Polynomial** kernel of degree *h*:

      $$K(\boldsymbol{X_i}, \boldsymbol{X_j}) = (\boldsymbol{X_i}\, \boldsymbol{X_j}+1)^h$$

    - **Gaussian radial basis** function kernel:

      $$K(\boldsymbol{X_i}, \boldsymbol{X_j}) = e^{-(\boldsymbol{X_i}, \boldsymbol{X_j})^2/2\sigma^2}$$

    - **Sigmoid kernel**:

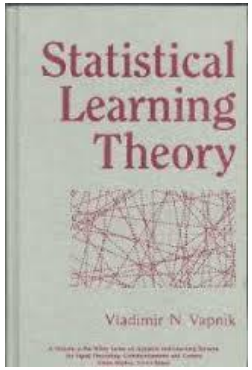
      $$K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \tanh(\beta \boldsymbol{X_i} \cdot \boldsymbol{X_j} + b)$$

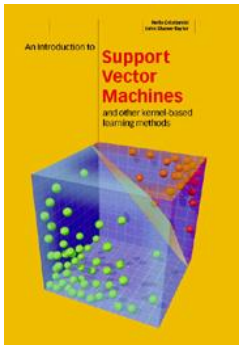
Why Is SVM Effective on High Dimensional Data?

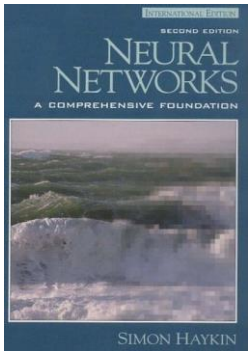
- The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples —they lie closest to the decision boundary
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, **an SVM with a small number of support vectors can have good generalization**, even when the dimensionality of the data is high

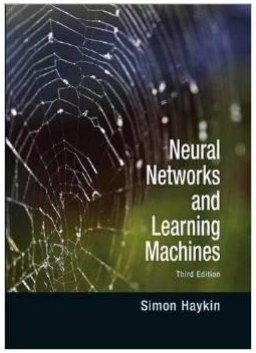
SVM - Introductory Literature

- **Statistical Learning Theory** by Vapnik, 1998: extremely hard to understand, containing many errors too.
- C. J. C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition*. Knowledge Discovery and Data Mining, 2(2), 1998: Better than the Vapnik's book, but still written too hard for introduction, and the examples are so not-intuitive
- **An Introduction to Support Vector Machines** by N. Cristianini and J. Shawe-Taylor: also written too hard for introduction.
- The Neural Networks book by Simon Haykin
  - Contains one nice chapter of SVM introduction (Chapter 6)









MORGAN & CLAYPOOL PUBLISHERS

Search Results: 415 matches found
Search Query: **All**: support vector machines

Results page 1 of 21
Order results: by relevancy by date Display snippets: no yes

Previous page | Next page

Add to favorites | View abstracts | Download to citation manager
To select/unselect all items click here

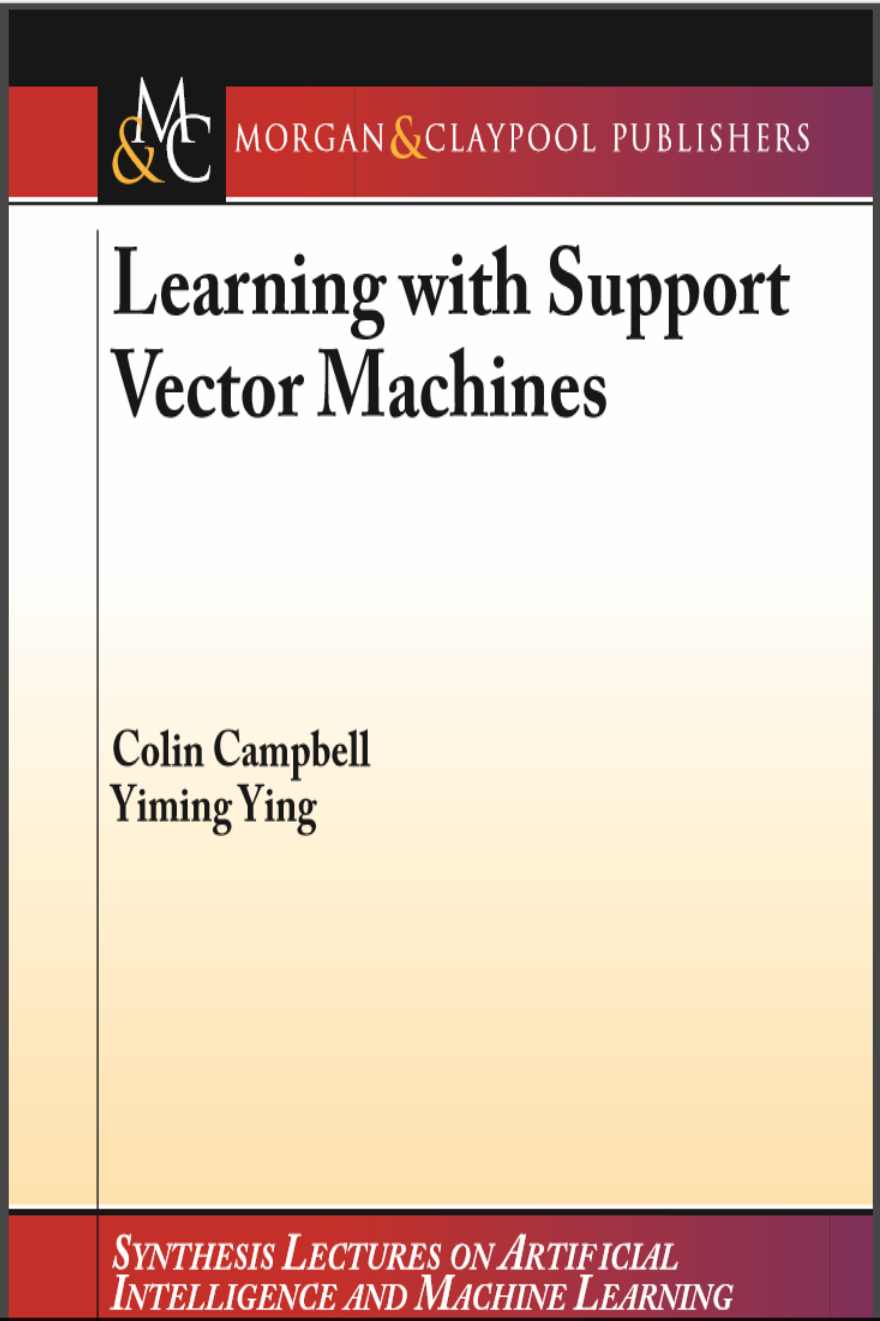
1. **Learning with Support Vector Machines**
Colin Campbell, Yiming Ying
Synthesis Lectures on Artificial Intelligence and Machine Learning Feb 2011, Vol. 5, No. 1, Pages 1-95
Abstract | PDF (672 KB) | PDF Plus (672 KB) | Add to Favorites | Related

2. **Support Vector Machines for Antenna Array Processing and Electromagnetics**
Manel Martínez-Ramón, Christos Christodoulou
Synthesis Lectures on Computational Electromagnetics Jan 2006, Vol. 1, No. 1, Pages 1-120
Abstract | PDF (5296 KB) | PDF Plus (3078 KB) | Add to Favorites | Related

Refine Search
- Modify Your Search
- New Simple Search
- New Advanced Search

Search within results
Narrow your search - search within these results for:
SEARCH

Save Search



SVM Related Links

- Representative implementations
  - **LIBSVM**: an efficient implementation of SVM, multi-class classifications, including also various interfaces with **java**, **python**, etc.
    - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - **SVM-light**: simpler but performance is not as good as LIBSVM; supports only binary classification and only in **C**
    - http://svmlight.joachims.org/
  - **SVM-torch**: another recent implementation also written in C (Note: *SVM-torch* is now part of the new *Torch* machine learning library: http://torch.ch/)

New York, NY, USA: ACM. pp. 408–415. CiteSeerX 10.1.1.149.5594. doi:10.1145/1390156.1390208. ISBN 978-1-60558-205-4.

19. ^ Rosasco, Lorenzo; De Vito, Ernesto; Caponnetto, Andrea; Piana, Michele; Verri, Alessandro (2004-05-01). "Are Loss Functions All the Same?". *Neural Computation*. **16** (5): 1063–1076. CiteSeerX 10.1.1.109.6786. doi:10.1162/089976604773135104. ISSN 0899-7667. PMID 15070510.
20. ^ Meyer, David; Leisch, Friedrich; Hornik, Kurt (September 2003). "The support vector machine under test". *Neurocomputing*. **55** (1–2): 169–186. doi:10.1016/S0925-2312(03)00431-4.
37. ^ Shalev-Shwartz, Shai; Singer, Yoram; Srebro, Nathan (2007). *Pegasos: Primal Estimated sub-GrAdient SOlver for SVM* (PDF). ICML. Archived (PDF) from the original on 2013-12-15.
38. ^ Fan, Rong-En; Chang, Kai-Wei; Hsieh, Cho-Jui; Wang, Xiang-Rui; Lin, Chih-Jen (2008). "LIBLINEAR: A library for large linear classification" (PDF). *Journal of Machine Learning Research*. **9**: 1871–1874.
39. ^ Allen Zhu, Zeyuan; Chen, Weizhu; Wang, Gang; Zhu, Chenguang; Chen, Zheng (2009). *P-packSVM: Parallel Primal grAdient desCent Kernel SVM* (PDF). ICDM. Archived (PDF) from the original on 2014-04-07.

Further reading [edit]

- Bennett, Kristin P.; Campbell, Colin (2000). "Support Vector Machines: Hype or Hallelujah?" (PDF). *SIGKDD Explorations*. **2** (2): 1–13. doi:10.1145/380995.380999.
- Cristianini, Nello; Shawe-Taylor, John (2000). *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press. ISBN 0-521-78019-5.
- Fradkin, Dmitriy; Muchnik, Ilya (2006). "Support Vector Machines for Classification" (PDF). In Abello, J.; Carmode, G. (eds.). *Discrete Methods in Epidemiology*. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. **70**. pp. 13–20.
- Ivanciuc, Ovidiu (2007). "Applications of Support Vector Machines in Chemistry" (PDF). *Reviews in Computational Chemistry*. **23**: 291–400.
- James, Gareth; Witten, Daniela; Hastie, Trevor; Tibshirani, Robert (2013). "Support Vector Machines" (PDF). *An Introduction to Statistical Learning : with Applications in R*. New York: Springer. pp. 337–372. ISBN 978-1-4614-7137-0.
- Schölkopf, Bernhard; Smola, Alexander J. (2002). *Learning with Kernels*. Cambridge, MA: MIT Press. ISBN 0-262-19475-9.
- Steinwart, Ingo; Christmann, Andreas (2008). *Support Vector Machines*. New York: Springer. ISBN 978-0-387-77241-7.
- Theodoridis, Sergios; Koutroumbas, Konstantinos (2009). *Pattern Recognition* (4th ed.). Academic Press. ISBN 978-1-59749-272-0.

External links [edit]

- libsvm, LIBSVM is a popular library of SVM learners
- liblinear is a library for large linear classification including some SVMs
- SVM light is a collection of software tools for learning and classification using SVM
- SVMJS live demo is a GUI demo for JavaScript implementation of SVMs

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

LIBSVM

From Wikipedia, the free encyclopedia

LIBSVM and **LIBLINEAR** are two popular open source machine learning libraries, both developed at the National Taiwan University and both written in C++ though with a C API. LIBSVM implements the Sequential minimal optimization (SMO) algorithm for kernelized support vector machines (SVMs), supporting classification and regression.[1] LIBLINEAR implements linear SVMs and logistic regression models trained using a coordinate descent algorithm.[2]

The SVM learning code from both libraries is often reused in other open source machine learning toolkits, including GATE, KNIME, Orange[3] and scikit-learn. Bindings and ports exist for programming languages such as Java, MATLAB, R, and Python.

Both libraries are free software released under the 3-clause BSD license.[4][5]

LIBSVM

Developer(s)	Chih-Chung Chang and Chih-Jen Lin
Stable release	3.23 / June 15, 2018; 16 months ago
Repository	github.com/cjlin1/libsvm
Written in	Java, C++
Operating system	Cross-platform
Type	Machine Learning
License	BSD
Website	www.csie.ntu.edu.tw/~cjlin/libsvm

References [edit]

1. ^ Chang, Chih-Chung; Lin, Chih-Jen (2011). "LIBSVM: A library for support vector machines". *ACM Transactions on Intelligent Systems and Technology*. **2** (3).
2. ^ R.-E. Fan; K.-W. Chang; C.-J. Hsieh; X.-R. Wang; C.-J. Lin (2008). "LIBLINEAR: A Library for Large Linear Classification". *Journal of Machine Learning Research*. **9**: 1871–1874.
3. ^ Janez Demšar; Tomaž Curk; Aleš Erjavec; Črt Gorup; Tomaž Hočevar; Mitar Milutinovič; Martin Možina; Matija Polajnar; Marko Toplak; Anže Starič; Miha Stajdohar; Lan Umek; Lan Žagar; Jure Žbontar; Marinka Žitnik; Blaž Zupan (2013). "Orange: data mining toolbox in Python" (PDF). *Journal of Machine Learning Research*. **14** (1): 2349–2353.
4. ^ "COPYRIGHT". *LIBSVM*. National Taiwan University.
5. ^ "COPYRIGHT". *LIBLINEAR*. National Taiwan University.

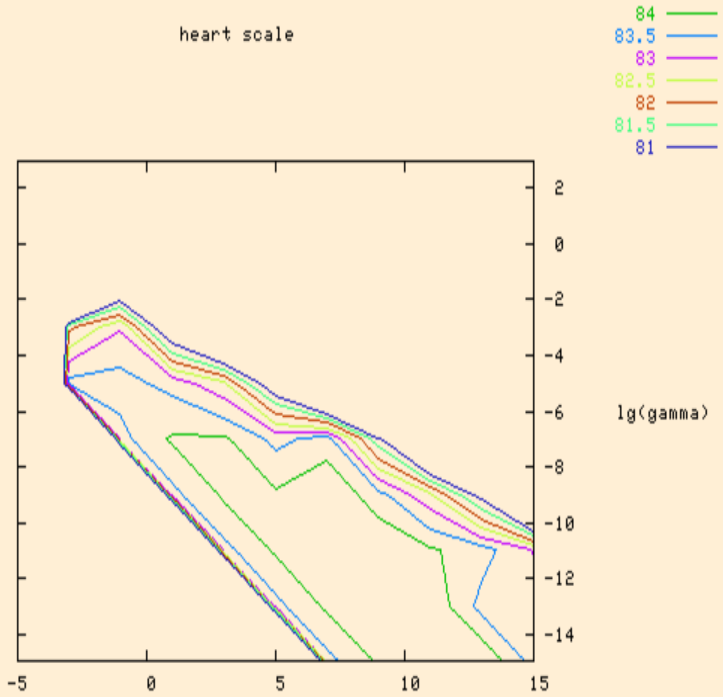
External links [edit]

- LIBSVM homepage

LIBSVM -- A Library for Support Vector Machines

Chih-Chung Chang and Chih-Jen Lin

NEW Version 3.24 released on September 11, 2019. It conducts some minor fixes.
NEW LIBSVM tools provides **many extensions** of LIBSVM. Please check it if you need some functions not supported in LIBSVM.
NEW We now have a nice page LIBSVM data sets providing problems in LIBSVM format.
NEW A practical guide to SVM classification is available now! (mainly written for beginners)
We now have an easy script (easy.py) for users who know NOTHING about SVM. It makes everything automatic--from data scaling to parameter selection.
The parameter selection tool grid.py generates the following contour of cross-validation accuracy. To use this tool, you also need to install python and gnuplot.



LibSVM

Table of Contents

Description

Wrapper class for the libsvm library by Chih-Chung Chang and Chih-Jen Lin. The original wrapper, named WLSVM, was developed by Yasser EL-Manzalawy. The current version is complete rewrite of the wrapper, using Reflection in order to avoid compilation errors, in case the `libsvm.jar` is not in the CLASSPATH.

Important note:
From Weka >= 3.7.2 installation and use of libsvm in Weka has been simplified by the creation of a LibSVM package that can be installed using either the graphical or command line package manager.

Reference (Weka <= 3.6.8)

- libsvm
- WLSVM

Package

`weka.classifiers.functions`

Download

The wrapper class is part of Weka since version 3.5.2. But *libsvm*, as a third-party-tool needs to be downloaded separately (see libsvm's Reference). It is recommended to upgrade to a post-3.5.3 version (or Subversion) for bug-fixes and extensions (contains now the `distributionForInstance` method).

CLASSPATH

Add the `libsvm.jar` from the libsvm distribution to your CLASSPATH to make it available.

Note: Do NOT start Weka then with `java -jar weka.jar`. The **-jar** option *overwrites* the CLASSPATH, not augments it (a very common trap to fall into). Instead use something like this on Linux:

Explorer: building "classifiers"

- Classifiers in WEKA are models for predicting nominal or numeric quantities
- Implemented learning schemes include:
  - Decision trees and lists, instance-based classifiers, support vector machines, multi-layer perceptrons, logistic regression, Bayes' nets, …
- "Meta"-classifiers include:
  - Bagging, boosting, stacking, error-correcting output codes, locally weighted learning, …

nltk.classify.weka module

Classifiers that make use of the external 'Weka' package.

class `nltk.classify.weka.ARFF_Formatter`(*labels*, *features*) [source]

Bases: `object`

Converts featuresets and labeled featuresets to ARFF-formatted strings, appropriate for input into Weka.

Features and classes can be specified manually in the constructor, or may be determined from data using `from_train`.

`data_section`(*tokens*, *labeled=None*) [source]

Returns the ARFF data section for the given data.

Parameters

- **tokens** – a list of featuresets (dicts) or labelled featuresets which are tuples (featureset, label).
- **labeled** – Indicates whether the given tokens are labeled or not. If None, then the tokens will be assumed to be labeled if the first token's value is a tuple or list.

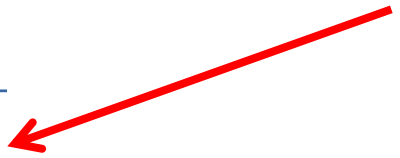
`format`(*tokens*) [source]

Returns a string representation of ARFF output for the given data.

static `from_train`(*tokens*) [source]

Constructs an ARFF_Formatter instance with class labels and feature types determined from the given data. Handles boolean, numeric and string (note: not nominal) types.

`header_section`() [source]



nltk.classify.svm module

nltk.classify.svm was deprecated. For classification based on support vector machines SVMs use nltk.classify.scikitlearn (or scikit-learn directly).

class nltk.classify.svm.**SvmClassifier**(**args*, ***kwargs*) [source]

Bases: object

nltk.classify.tadm module

nltk.classify.tadm.**call_tadm**(*args*) [source]

Call the tadm binary with the given arguments.

nltk.classify.tadm.**config_tadm**(*bin=None*) [source]

nltk.classify.tadm.**encoding_demo**() [source]

nltk.classify.tadm.**names_demo**() [source]

nltk.classify.tadm.**parse_tadm_weights**(*paramfile*) [source]

Given the stdout output generated by tadm when training a model, return a numpy array containing the corresponding weight vector.

nltk.classify.tadm.**write_tadm_file**(*train_toks*, *encoding*, *stream*) [source]

Generate an input file for tadm based on the given corpus of classified tokens.

Parameters:

- **train_toks** (*list(tuple(dict, str))*) – Training data, represented as a list

nltk.classify.scikitlearn module

scikit-learn (http://scikit-learn.org) is a machine learning library for Python. It supports many classification algorithms, including SVMs, Naive Bayes, logistic regression (MaxEnt) and decision trees.

This package implements a wrapper around scikit-learn classifiers. To use this wrapper, construct a scikit-learn estimator object, then use that to construct a SklearnClassifier. E.g., to wrap a linear SVM with default settings:

```
>>> from sklearn.svm import LinearSVC
>>> from nltk.classify.scikitlearn import SklearnClassifier
>>> classif = SklearnClassifier(LinearSVC())
```

A scikit-learn classifier may include preprocessing steps when it's wrapped in a Pipeline object. The following constructs and wraps a Naive Bayes text classifier with tf-idf weighting and chi-square feature selection to get the best 1000 features:

```
>>> from sklearn.feature_extraction.text import TfidfTransformer
>>> from sklearn.feature_selection import SelectKBest, chi2
>>> from sklearn.naive_bayes import MultinomialNB
>>> from sklearn.pipeline import Pipeline
>>> pipeline = Pipeline([('tfidf', TfidfTransformer()),
...                      ('chi2', SelectKBest(chi2, k=1000)),
...                      ('nb', MultinomialNB())])
>>> classif = SklearnClassifier(pipeline)
```

class nltk.classify.scikitlearn.**SklearnClassifier**(*estimator*, *dtype=<class 'float'>*, *sparse=True*)

[source]

Bases: nltk.classify.api.ClassifierI

Previous 1.3. Kernel r... | Next 1.5. Stochast... | Up 1. Supervised...

1.4. Support Vector Machines

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

The support vector machines in scikit-learn support both dense (`numpy.ndarray` and convertible to that by `numpy.asarray`) and sparse (any `scipy.sparse`) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data. For optimal performance, use C-ordered `numpy.ndarray` (dense) or `scipy.sparse.csr_matrix` (sparse) with `dtype=float64`.

1.4.1. Classification

SVM – sklearn

```
from sklearn import svm
from sklearn import datasets

classifier = svm.SVC()  # or classifier = svm.LinearSVC()

iris = datasets.load_iris()
X, y = iris.data, iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred, y_test))
print(accuracy_score(y_pred, y_test, normalize=False))
```

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

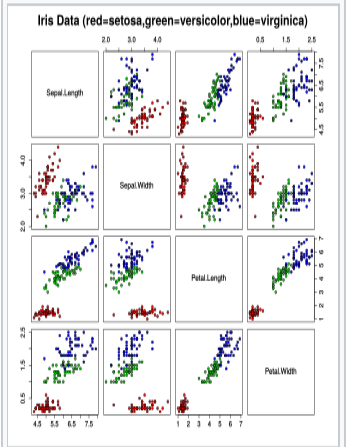
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Iris flower data set

From Wikipedia, the free encyclopedia

The ***Iris* flower data set** or **Fisher's *Iris* data set** is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper *The use of multiple measurements in taxonomic problems* as an example of linear discriminant analysis.[1] It is sometimes called **Anderson's *Iris* data set** because Edgar Anderson collected the data to quantify the morphologic variation of *Iris* flowers of three related species.[2] Two of the three species were collected in the Gaspé Peninsula "all from the same pasture, and picked on the same day and measured at the same time by the same person with the same apparatus".[3]

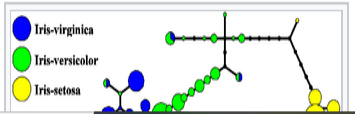


Scatterplot of the data set

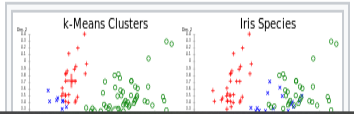
The data set consists of 50 samples from each of three species of *Iris* (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

Contents [hide]

Use of the data set [edit]



Based on Fisher's linear discriminant model, this data set became a typical test case for many statistical classification techniques in machine learning such as support vector machines.[5]



The iris data set is widely used as a beginner's dataset for machine learning purposes. Here are some lines of python code that define how this works.

```
from sklearn.datasets import load_iris
iris=load_iris()
iris
```

This code gives:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
```

Several versions of the dataset have been published.[8]

See also [edit]

- Classic data sets

References [edit]

1. ^ R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". *Annals of Eugenics*. **7** (2): 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x. hdl:2440/15227.
2. ^ Edgar Anderson (1936). "The species problem in *Iris*". *Annals of the Missouri Botanical Garden*. **23** (3): 457–509. doi:10.2307/2394164. JSTOR 2394164.
3. ^ Edgar Anderson (1935). "The irises of the Gaspé Peninsula". *Bulletin of the American Iris Society*. **59**: 2–5.
4. ^ [a] [b] A. N. Gorban, A. Zinovyev. Principal manifolds and graphs in practice: from molecular biology to dynamical systems, International Journal of Neural Systems, Vol. 20, No. 3 (2010) 219–232.
5. ^ "UCI Machine Learning Repository: Iris Data Set". *archive.ics.uci.edu*. Retrieved 2017-12-01.
6. ^ Ines Färber, Stephan Günnemann, Hans-Peter Kriegel, Peer Kröger, Emmanuel Müller, Erich Schubert, Thomas Seidl, Arthur Zimek

Iris setosa

Iris versicolor

Iris virginica

Attribute-Relation File Format (ARFF)

November 1st, 2008

This documentation is superceded by the Wiki article on the ARFF format.

April 1st, 2002

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software. This document descibes the version of ARFF used with Weka versions 3.2 to 3.3; this is an extension of the ARFF format as described in the data mining book written by Ian H. Witten and Eibe Frank (the new additions are string attributes, date attributes, and sparse instances).

This explanation was cobbled together by Gordon Paynter (gordon.paynter at ucr.edu) from the Weka 2.1 ARFF description, email from Len Trigg (lenbok at myrealbox.com) and Eibe Frank (eibe at cs.waikato.ac.nz), and some datasets. It has been edited by Richard Kirkby (rkirkby at cs.waikato.ac.nz). Contact Len if you're interested in seeing the ARFF 3 proposal.

Overview

ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.

The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

All pages
All tags
All files
Packages
FAQ
Not So FAQ
Troubleshooting
Mailing List

ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.

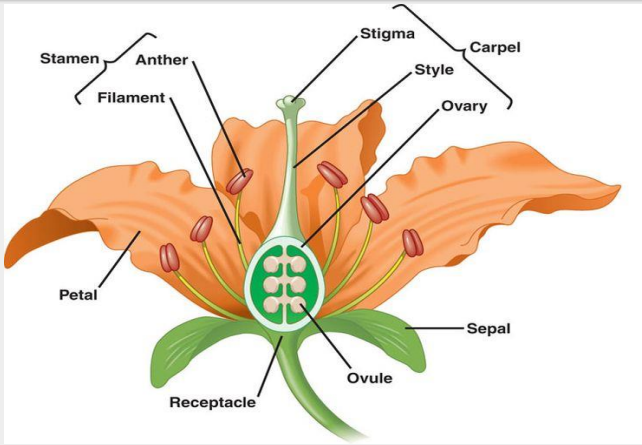
The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on the standard IRIS dataset looks like this:

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The **Data** of the ARFF file looks like the following:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```



MLP – sklearn

```
from sklearn.neural_network import MLPClassifier
from sklearn import datasets

mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)

iris = datasets.load_iris()
X, y = iris.data, iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
mlp.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_pred, y_test))
print(accuracy_score(y_pred, y_test, normalize=False))
```

classification_report & confusion_matrix

```
from sklearn.neural_network import MLPClassifier
from sklearn import datasets

mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)

iris = datasets.load_iris()
X, y = iris.data, iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Another Example: Words (Revisited)

w1	w2	w3	w4	Class
1	0	0	1	1
0	0	0	1	0
1	1	0	1	0
1	0	1	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	1
0	1	0	0	1
0	1	0	1	0
1	1	1	0	0

	Class=1	Class=0
Pr(Class)	0.40	0.60
Pr(w1 | Class)	0.75	0.50
Pr(w2 | Class)	0.25	0.67
Pr(w3 | Class)	0.50	0.33
Pr(w4 | Class)	0.50	0.50

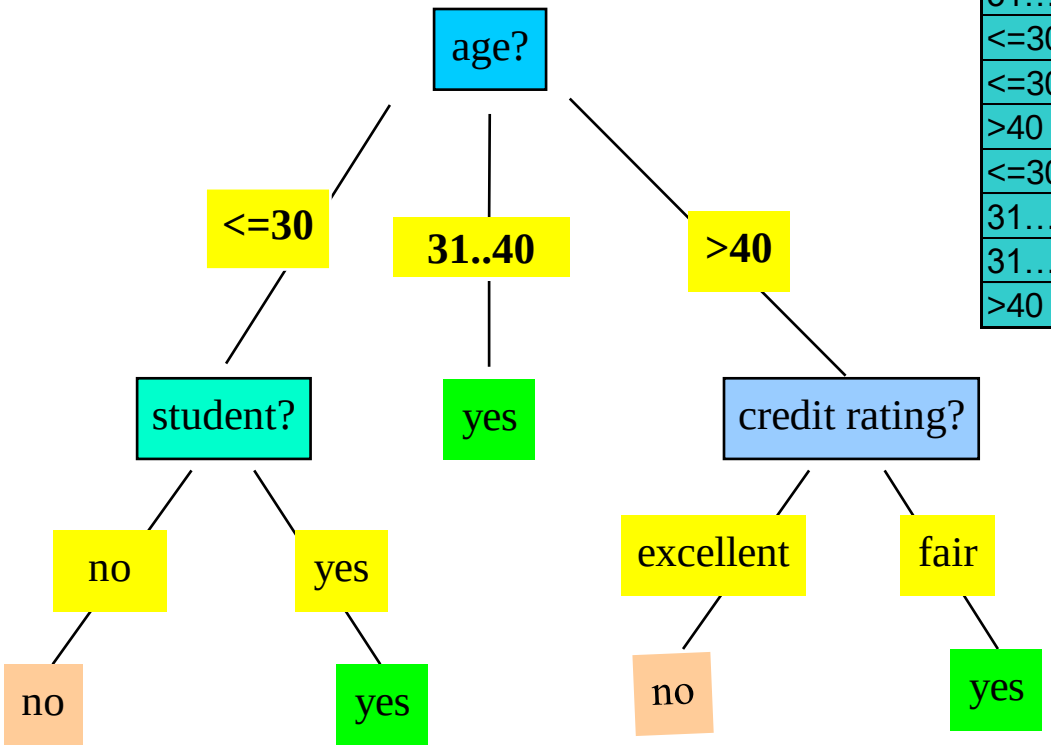
SVM vs. Neural Network

- **SVM**
  - Deterministic algorithm
  - Nice generalization properties
  - Hard to learn – learned in batch mode using quadratic programming techniques
  - Using kernels can learn very complex functions
- **Neural Network**
  - Nondeterministic algorithm
  - Generalizes well but doesn't have strong mathematical foundation
  - Can easily be learned in incremental fashion
  - To learn complex functions—use multilayer perceptron (nontrivial)

Decision Tree Induction

- Training data set: Buys_computer
- This data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Attributes are selected on the basis of a heuristic or statistical measure (e.g., **information gain, next slide**)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Information Gain

- Select the attribute with the highest information gain
- Let p_i be the probability that a tuple in D (the training set) belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D: (m is number of classes)

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D is:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

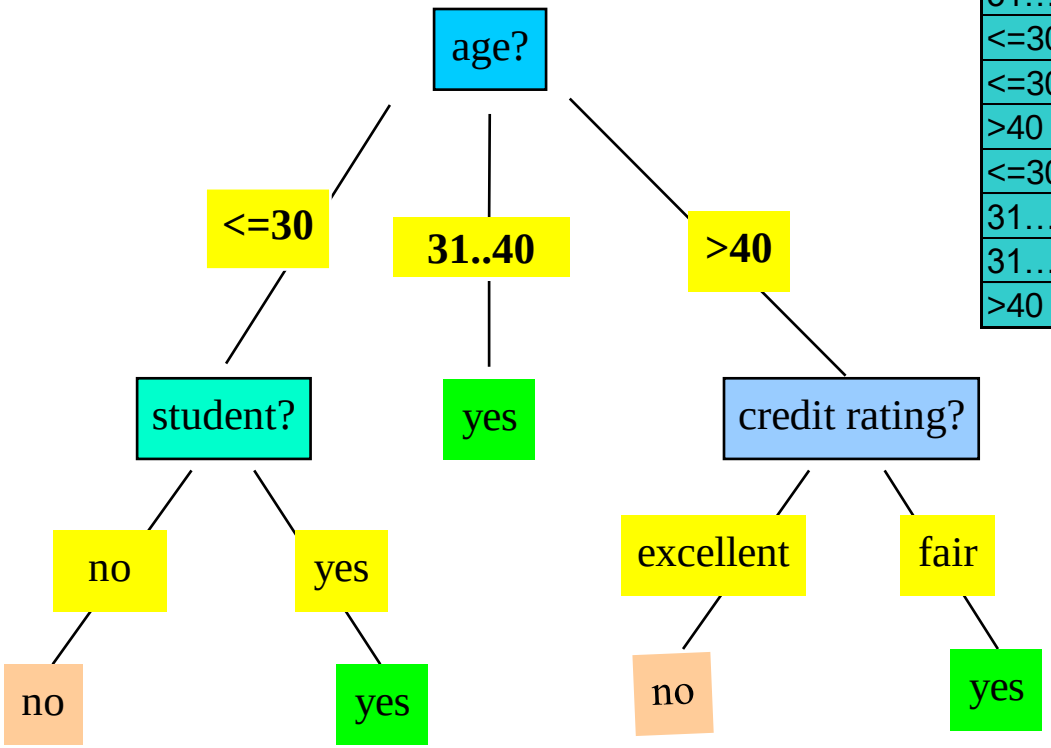
Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Decision Tree Induction

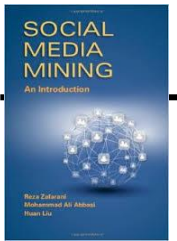
- Training data set: Buys_computer
- This data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Nearest Neighbor Classifier

- k-nearest neighbor or k-NN,
  - Utilizes the neighbors of an instance to perform classification.
- It uses the k nearest instances, called **neighbors**, to perform classification.
- The instance being classified is assigned the label (class attribute value) that the majority of its k neighbors are assigned
- When $k = 1$, the closest neighbor's label is used as the predicted label for the instance being classified
- To determine the neighbors of an instance, we need to measure its distance to all other instances based on some distance metric
  - Often **Euclidean distance** is employed

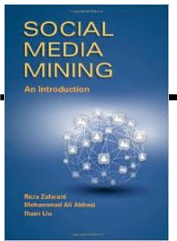


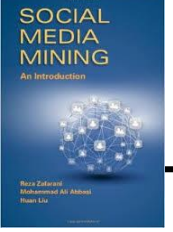
Nearest Neighbor Classifier

Algorithm 5.1 k-Nearest Neighbor Classifier

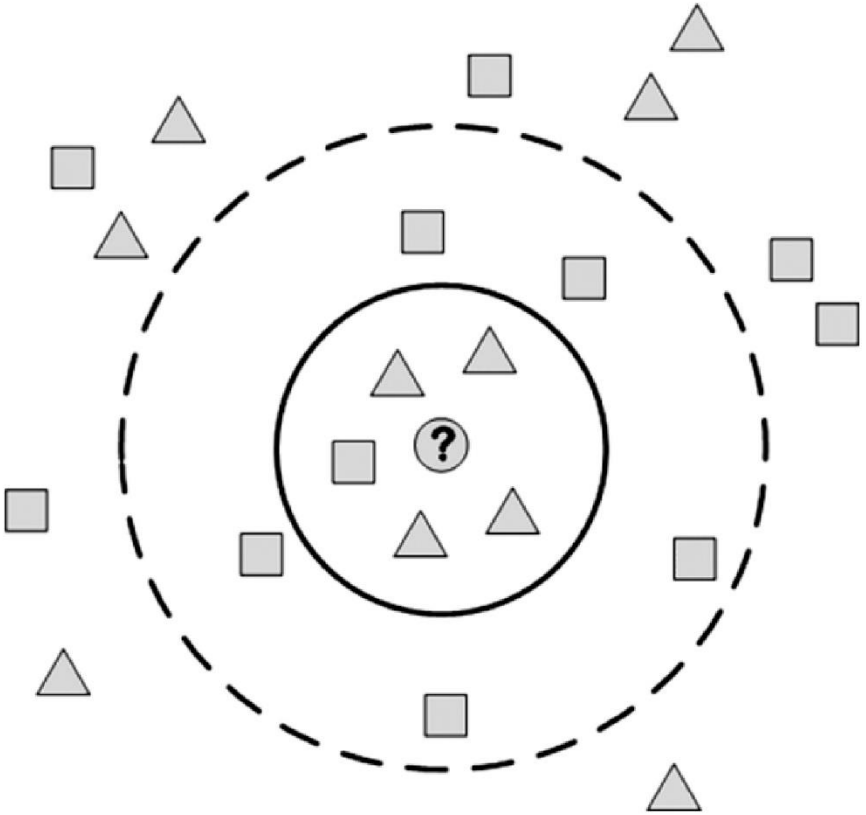
Require: Instance i, A Dataset of Real-Value Attributes, k (number of neighbors), distance measure d

1: **return** Class label for instance i
2: Compute k nearest neighbors of instance i based on distance mea- sure d.
3: l = the majority class label among neighbors of instance i. If more than one majority label, select one randomly.
4: Classify instance i as class l

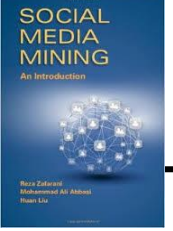




Nearest Neighbor Classifier



- When $k = 5$, the predicted label is: Δ
- When $k = 9$, the predicted label is: $\square$



Nearest Neighbor Classifier

No.	Outlook (O)	Temperature (T)	Humidity (H)	Play Golf (PG)
1	sunny	hot	high	N
2	sunny	mild	high	N
3	overcast	hot	high	Y
4	rain	mild	high	Y
5	sunny	cool	normal	Y
6	rain	cool	normal	N
7	overcast	cool	normal	Y
8	sunny	mild	high	?

Similarity between row 8 and other data instances;
(Similarity = 1 if attributes have the same value, otherwise similarity = 0)

Data instance	Outlook	Temperature	Humidity	Similarity	Label	K	Prediction
2	1	1	1	3	N	1	**N**
1	1	0	1	2	N	2	**N**
4	0	1	1	2	Y	3	**N**
3	0	0	1	1	Y	4	**?**
5	1	0	0	1	Y	5	**Y**
6	0	0	0	0	N	6	**?**
7	0	0	0	0	Y	7	**Y**

Regression

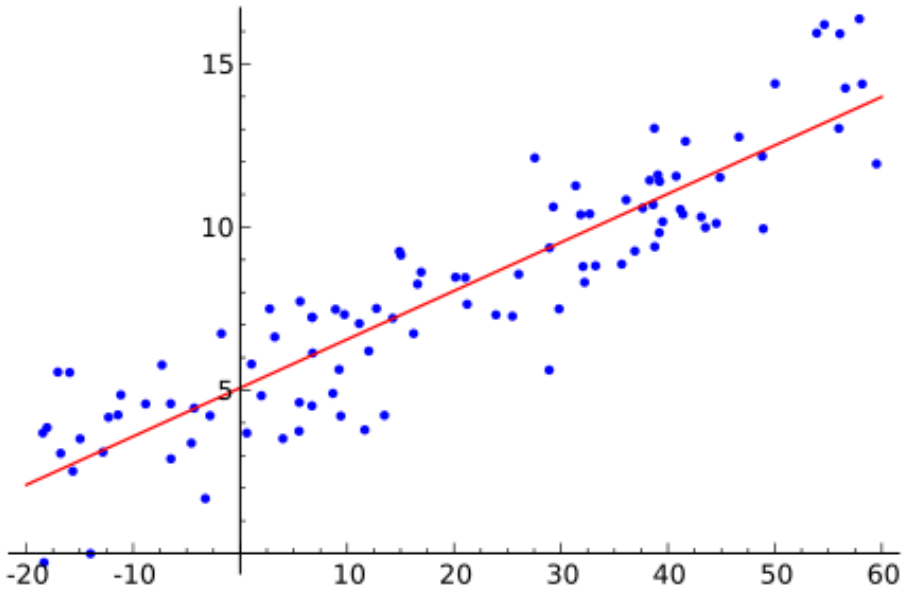
In regression,

- Class values are real numbers
  - In classification, class values are categorical

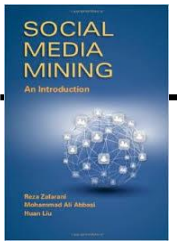
$$y \approx f(X)$$

Class attribute
(dependent variable)
$y \in R$

Features
(regressors)
$X = (x_1, x_2, \ldots, x_m)$



Goal: find the relation between y and vector $X = (x_1, x_2, \ldots, x_m)$



Linear Regression

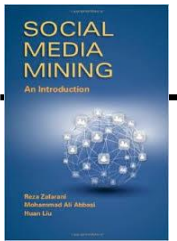
- **Linear regression**: we assume the relation between the class attribute Y and feature set $\boldsymbol{X}$ is linear

$$Y = XW + \epsilon$$

- W represents the vector of regression coefficients

- Regression can be solved by estimating W and ε using the provided dataset and the labels Y
  - "**Least squares**" is a popular method to solve regression
  - The goal is to minimize

$$\epsilon^2 = ||\epsilon^2|| = ||Y - XW||^2$$



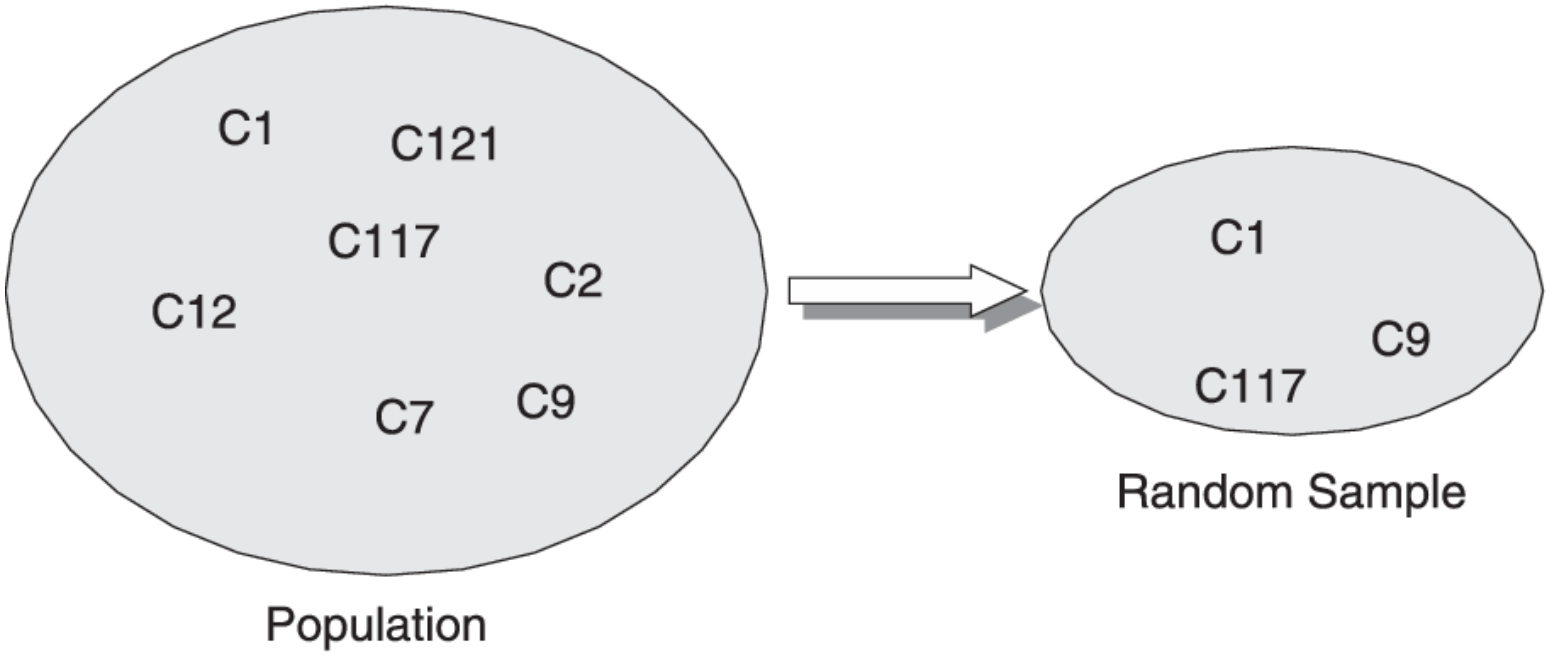
Evaluation

Performance Evaluation

- The learning methods provide potential solutions, but they do not guarantee that these are good solutions.
- To get the best results, we must ensure that
  - The wisest choices are made in applying the methods
  - Estimates are found for the future performance of proposed solutions
- Let's take a closer look at the immediate task of evaluating a solution and estimating its future performance.

Performance Evaluation

- The standard statistical model assumes that a sample is randomly drawn from some general population:

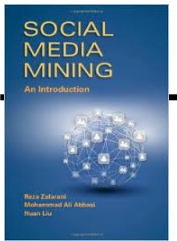


Performance Evaluation

- To evaluate the performance of a solution, we train on one sample and test on another sample.
  - Typically, our data might be randomly divided into two parts: one for training and one for testing.
  - Are new documents from the same population?
    - Very often they are not, but over relatively short time horizons, we assume that new documents are similar to old documents or that events will unfold in a similar way.

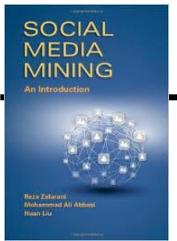
Evaluating Supervised Learning

- Training/Testing Framework:
  - A training dataset (i.e., the labels are known) is used to train a model
  - the model is evaluated on a test dataset.
- The correct labels of the test dataset are unknown,
  - In practice, the training set is divided into two parts,
    - One used for training and
    - The other used for testing.
- When testing, the labels from this test set are removed.
  - After these labels are predicted using the model, the predicted labels are compared with the masked labels (**ground truth**).



Evaluating Supervised Learning

- Dividing the training set into train/test sets
  - **Leave-one-out training**
    - Divide the training set into k equally sized partitions
      - Often called **folds**
    - Use all folds but one to train and the one left out for testing
  - ***k*-fold cross validation training**
    - Divide the training set into k equally sized sets
    - Run the algorithm k times
    - In round i, we use all folds but fold i for training and fold i for testing.
    - The average performance of the algorithm over k rounds measures the performance of the algorithm.



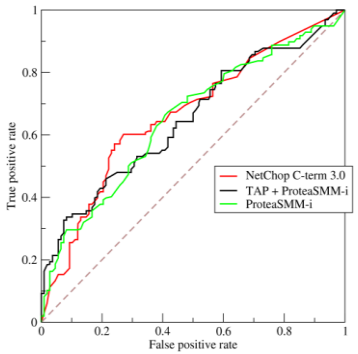
Evaluating Supervised Learning

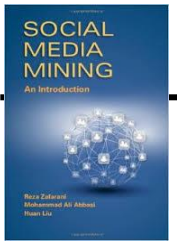
- As the class labels are discrete, we can measure the accuracy by dividing number of correctly predicted labels (C) by the total number of instances (N)

$$\text{accuracy} = \frac{C}{N}$$

$$\text{error rate} = 1 \text{ - accuracy}$$

- More sophisticated approaches of evaluation
  - **F-Measure**
  - ROC/AUC…





Performance Evaluation

- Performance can be estimated in terms of several measures. The standard measure for classification is the error rate:

  Error rate ***(erate)*** = number of errors/number of documents

- Its standard error is:

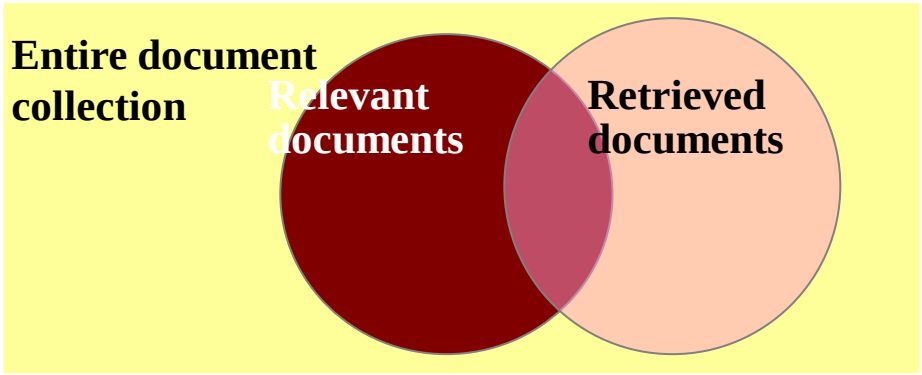
  Standard Error***(SE)*** = $\sqrt{erate * (1 - erate)}$/number of documents

  - The error rate is binomially distributed and is approximately normal. Two standard errors are often used to approximate **95%** confidence bounds.

Performance Evaluation

- Although error-rates and associated standard errors are useful for estimating the performance of predictors in general, for most text applications, such as text categorization, a more detailed analysis of the errors is desirable.
  - For information retrieval applications, there is usually a large number of negative data. A classifier can achieve a very high accuracy (i.e., a very low error rate) by simply saying that all data are negative.
  - It is thus useful to measure the classification performance by ignoring correctly predicted negative data (true negatives) and then examining the sorts of errors made by the classifier.
  - Three ratios have achieved particular prominence: precision, recall, and ***F*-measure**.

Precision and Recall



	retrieved	not retrieved
relevant	retrieved & relevant	Not retrieved & relevant
irrelevant	retrieved & irrelevant	not retrieved but irrelevant

$$recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

To consolidate Precision & Recall into 1 measure: **F = 2PR/(P+R)**

F-score / F-measure [edit]

Main article: F-score

The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

$$F = \frac{2 \cdot \mathrm{precision} \cdot \mathrm{recall}}{(\mathrm{precision} + \mathrm{recall})}$$

This is also known as the F_1 measure, because recall and precision are evenly weighted.

The general formula for non-negative real β is:

$$F_\beta = \frac{(1 + \beta^2) \cdot (\mathrm{precision} \cdot \mathrm{recall})}{(\beta^2 \cdot \mathrm{precision} + \mathrm{recall})}$$

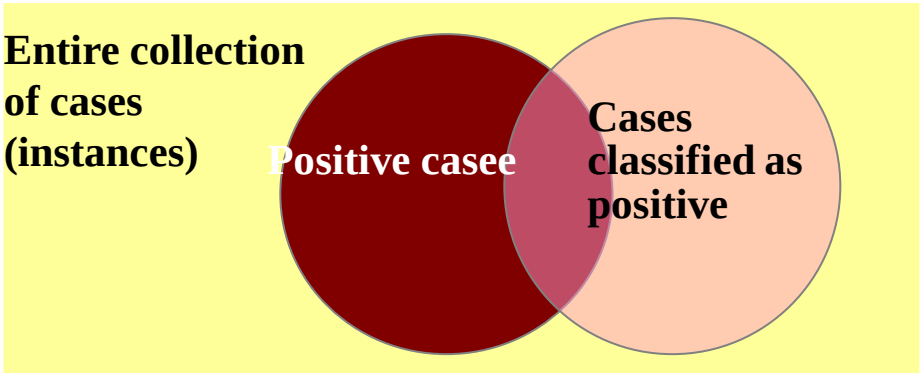
Two other commonly used F measures are the F_2 measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

The F-measure was derived by van Rijsbergen (1979) so that F_β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision". It is based on van Rijsbergen's effectiveness measure $E = 1 - \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}}$. Their relationship is:

$F_\beta = 1 - E$ where $\alpha = \frac{1}{1 + \beta^2}$

F-measure can be a better single metric when compared to precision and recall; both precision and recall give different information that can complement each other when combined. If one of them excels more than the other, F-measure will reflect it.[*citation needed*]

Precision and Recall: Classification



Positive	True Positive	False Negative
Negative	False Positive	True Negative
	Classified as Positive	Classified as Negative

$$recall = \frac{True\ positive\ cases}{Total\ number\ of\ \ positive\ cases}$$

$$precision = \frac{True\ positive\ cases}{Total\ number\ of\ \ cases\ classified\ \ as\ \ positvie}$$

Performance Evaluation

- Because document collections are typically large, **high precision** is often more valued.
  - In order to achieve high precision, the computer/algorithm may fail to catch all positives (this is measured by recall).
  - If a program identifies spam e-mail with high precision and low recall, it may often leave spam in your Inbox (low recall), but when it puts a spam document in the trash, it is usually correct (high precision).
- Is it possible to adjust the precision and recall of a classifier?
  - Since precision and recall measure different kinds of errors, if the overall error rate remains the same, increasing the precision (reducing one kind of error) lowers the recall (increases the other kind of error). → **precision–recall tradeoff**

Precision–Recall Tradeoff

- For probabilistic scoring, the threshold for a class can be altered from 0.5 to some other value.
  - Lower thresholds would boost recall, while higher values would boost precision.
- For linear models, the threshold can be changed from zero to a different value.
  - Lower values would help recall, and higher values would boost precision.

Precision–Recall Tradeoff

- For *k* nearest-neighbor methods, the threshold can be varied from a simple majority to some other value.
  - If five nearest neighbors being used classify a document within a topic, instead of requiring that three of the five nearest neighbors belong to the topic, one may change this threshold to a different value.
  - A value less than 3 would boost recall, whereas values greater than 3 would boost precision.

Sentiment Analysis

- As mentioned before, since sentiment analysis is a **text categorization** problem, any existing supervised learning method can be applied, e.g., naïve Bayes, maximum entropy, decision tress, and support vector machines (SVM).

Sentiment Analysis

- Pang et al., 2002 (**Thumbs up**? Sentiment Classification using Machine Learning Techniques) was the first paper to take this approach to classify movie reviews into two classes, positive and negative.
- It was shown that using unigrams (a bag of words) as features in classification performed quite well with either naïve Bayes or SVM, although the authors also tried a number of other feature options.

Pang et al., 2002, Thumbs Up?

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	**78.7**	N/A	72.8
(2)	unigrams	"	pres.	81.0	80.4	**82.9**
(3)	unigrams+bigrams	32330	pres.	80.6	80.8	**82.7**
(4)	bigrams	16165	pres.	77.3	**77.4**	77.1
(5)	unigrams+POS	16695	pres.	81.5	80.4	**81.9**
(6)	adjectives	2633	pres.	77.0	**77.7**	75.1
(7)	top 2633 unigrams	2633	pres.	80.3	81.0	**81.4**
(8)	unigrams+position	22430	pres.	81.0	80.1	**81.6**

Sentiment Analysis with NLTK

- Using built-in corpora, we can build classifiers that will automatically tag new documents with appropriate category labels (pos or neg in this case).
  - First, we construct a list of documents, labeled with the appropriate categories.
  - For this example, we've chosen the Movie Reviews Corpus, which categorizes each review as positive or negative.

```
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)
```

`random.`**`choice`**(*seq*)

Return a random element from the non-empty sequence *seq*. If *seq* is empty, raises `IndexError`.

`random.`**`shuffle`**(*x*[, *random*])

Shuffle the sequence *x* in place. The optional argument *random* is a 0-argument function returning a random float in [0.0, 1.0); by default, this is the function `random()`.

Note that for even rather small `len(x)`, the total number of permutations of *x* is larger than the period of most random number generators; this implies that most permutations of a long sequence can never be generated.

`random.`**`sample`**(*population*, *k*)

Return a *k* length list of unique elements chosen from the population sequence. Used for random sampling without replacement.

New in version 2.3.

Returns a new list containing elements from the population while leaving the original population unchanged. The resulting list is in selection order so that all sub-slices will also be valid random samples. This allows raffle winners (the sample) to be partitioned into grand prize and second place winners (the subslices).

Members of the population need not be *hashable* or unique. If the population contains repeats, then each occurrence is a possible selection in the sample.

Sentiment Analysis with NLTK

- Next, we define a feature extractor for documents, so the classifier will know which aspects of the data it should pay attention to:

```
# I did more proceessing in my version od the code (in Blackboard)
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

# most frequent 2000 words/tokens
# word_features = all_words.keys()[:2000] # wrong, older version NLTK 2.0
# word_features = list(all_words)[:2000]  # also wrong, in the 2nd Edition of the book
word_features = [w for (w, c) in all_words.most_common(2000)]

def document_features(document): # input document is a list of words/tokens
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

Table 3.1:

Functions Defined for NLTK's Frequency Distributions

Example	Description
`fdist = FreqDist(samples)`	create a frequency distribution containing the given samples
`fdist[sample] += 1`	increment the count for this sample
`fdist['monstrous']`	count of the number of times a given sample occurred
`fdist.freq('monstrous')`	frequency of a given sample
`fdist.N()`	total number of samples
`fdist.most_common(n)`	the `n` most common samples and their frequencies
`for sample in fdist:`	iterate over the samples
`fdist.max()`	sample with the greatest count
`fdist.tabulate()`	tabulate the frequency distribution
`fdist.plot()`	graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	cumulative plot of the frequency distribution
`fdist1 |= fdist2`	update `fdist1` with counts from `fdist2`
`fdist1 < fdist2`	test if samples in `fdist1` occur less frequently than in `fdist2`

Table 3.1:

Functions Defined for NLTK's Frequency Distributions

Example	Description
`fdist = FreqDist(samples)`	create a frequency distribution containing the given samples
`fdist[sample] += 1`	increment the count for this sample
`fdist['monstrous']`	count of the number of times a given sample occurred
`fdist.freq('monstrous')`	frequency of a given sample
`fdist.N()`	total number of samples
`fdist.most_common(n)`	the `n` most common samples and their frequencies
`for sample in fdist:`	iterate over the samples
`fdist.max()`	sample with the greatest count
`fdist.tabulate()`	tabulate the frequency distribution
`fdist.plot()`	graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	cumulative plot of the frequency distribution
`fdist1 |= fdist2`	update `fdist1` with counts from `fdist2`
`fdist1 < fdist2`	test if samples in `fdist1` occur less frequently than in `fdist2`

Table 1-2. Functions defined for NLTK's frequency distributions

Example	Description
`fdist = FreqDist(samples)`	Create a frequency distribution containing the given samples
`fdist.inc(sample)`	Increment the count for this sample
`fdist['monstrous']`	Count of the number of times a given sample occurred
`fdist.freq('monstrous')`	Frequency of a given sample
`fdist.N()`	Total number of samples
`fdist.keys()`	The samples sorted in order of decreasing frequency
`for sample in fdist:`	Iterate over the samples, in order of decreasing frequency
`fdist.max()`	Sample with the greatest count
`fdist.tabulate()`	Tabulate the frequency distribution
`fdist.plot()`	Graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	Cumulative plot of the frequency distribution
`fdist1 < fdist2`	Test if samples in `fdist1` occur less frequently than in `fdist2`

Sentiment Analysis with NLTK

- Next, we define a feature extractor for documents, so the classifier will know which aspects of the data it should pay attention to:

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())

# most frequent 2000 words/tokens
word_features = [w for (w, c) in all_words.most_common(2000)]

def document_features(document): # input document is a list of words/tokens
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

```
>>> print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
{'contains(waste)': False, 'contains(lot)': False, ...}
```

Sentiment Analysis with NLTK

- Now that we've defined our feature extractor, we can use it to train a classifier to label new movie reviews:

```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

- To check how reliable the resulting classifier is, we compute its accuracy on the test set .

```
>>> print nltk.classify.accuracy(classifier, test_set)
0.81        ← this value fluctuates
```

Sentiment Analysis with NLTK

- We can use **show_most_informative_features()** to find out which features the classifier found to be most informative .

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
contains(outstanding) = True    pos : neg = 11.1 : 1.0
contains(seagal) = True         neg : pos = 7.7 : 1.0
contains(wonderfully) = True    pos : neg = 6.8 : 1.0
contains(damon) = True          pos : neg = 5.9 : 1.0
contains(wasted) = True         neg : pos = 5.8 : 1.0
```

- Apparently in this corpus, a review that mentions *Seagal* is almost 8 times more likely to be negative than positive, while a review that mentions *Damon* is about 6 times more likely to be positive.

SA on Tweets Using NLTK

The Training Data:

Positive tweets:

I love this car.

This view is amazing.

I feel great this morning.

I am so excited about the concert.

He is my best friend.

Negative tweets:

I do not like this car.

This view is horrible.

I feel tired this morning.

I am not looking forward to the concert.

He is my enemy.

SA on Tweets Using NLTK

The Training Data:

```
pos_tweets = [('I love this car.', 'positive'),
              ('This view is amazing.', 'positive'),
              ('I feel great this morning.', 'positive'),
              ('I am so excited about the concert.', 'positive'),
              ('He is my best friend.', 'positive')]

neg_tweets = [('I do not like this car.', 'negative'),
              ('This view is horrible.', 'negative'),
              ('I feel tired this morning.', 'negative'),
              ('I am not looking forward to the concert.', 'negative'),
              ('He is my enemy.', 'negative')]
```

SA on Tweets Using NLTK

- We take both of those lists and create a single list of tuples each containing two elements.
  - First element is an array containing the words and second element is the type of sentiment.
  - We get rid of the words smaller than 2 characters and we use lowercase for everything.

```
tweets = []

for (words, sentiment) in pos_tweets + neg_tweets:
    words_filtered = [e.lower() for e in words.split() if len(e) >= 3]
    tweets.append((words_filtered, sentiment))
```

SA on Tweets Using NLTK

- The resulting list of tweets now looks like this:

```
tweets = [(['love', 'this', 'car'], 'positive'),
          (['this', 'view', 'amazing'], 'positive'),
          (['feel', 'great', 'this', 'morning'], 'positive'),
          (['excited', 'about', 'the', 'concert'], 'positive'),
          (['best', 'friend'], 'positive'),
          (['not', 'like', 'this', 'car'], 'negative'),
          (['this', 'view', 'horrible'], 'negative'),
          (['feel', 'tired', 'this', 'morning'], 'negative'),
          (['not', 'looking', 'forward', 'the', 'concert'], 'negative'),
          (['enemy'], 'negative')]
```

SA on Tweets Using NLTK

- Next, the list of **word features** needs to be extracted from the tweets.
  - It is a list with every distinct words ordered by frequency of appearance.
  - We use the following function to get the list plus the two helper functions: (on next slide)

SA on Tweets Using NLTK

```
def get_words_in_tweets(tweets):
    all_words = []
    for (words, sentiment) in tweets:
        all_words.extend(words)
    return all_words

def get_word_features(wordlist):
    wordlist = nltk.FreqDist(wordlist)
    word_features = [w for (w, c) in wordlist.most_common(2000)]
    return word_features

word_features = get_word_features(get_words_in_tweets(tweets))
```

SA on Tweets Using NLTK

❖ We end up with the following list of word features:

```
word_features = [
    'this',
    'car',
    'concert',
    'feel',
    'morning',
    'not',
    'the',
    'view',
    'about',
    'amazing',
    ...
]
```

SA on Tweets Using NLTK

- Next, we can use the same function, **document_features(document),** from the NLP With Python book that we saw earlier in the previous example for the movie review dataset, to extract features from each tweet:

```
def extract_features(document): # I renamed it
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features
```

SA on Tweets Using NLTK

For the first positive tweet, (['love', 'this', 'car'], 'positive'), we obtain the following dictionary which indicates that the document contains the words: ‘love’, ‘this’ and ‘car’:

{'contains(not)': False,
 'contains(view)': False,
 'contains(best)': False,
 'contains(excited)': False,
 'contains(morning)': False,
 'contains(about)': False,
 'contains(horrible)': False,
 'contains(like)': False,
 'contains(this)': True,
 'contains(friend)': False,
 'contains(concert)': False,

SA on Tweets Using NLTK

'contains(feel)': False,

'contains(love)': True,

'contains(looking)': False,

'contains(tired)': False,

'contains(forward)': False,

'contains(car)': True,

'contains(the)': False,

'contains(amazing)': False,

'contains(enemy)': False,

'contains(great)': False}

SA on Tweets Using NLTK

- We can then use this feature extractor, **extract_features()**, to create the training set as before:

```
training_set = [(extract_features(d), c) for (d,c) in tweets]
```

- There also a built-in function, **apply_features(),** that you can use to achieve the same goal:

```
training_set = classify.apply_features(extract_features, tweets)
```

SA on Tweets Using NLTK

- The variable ‘training_set’ contains the labeled feature sets.
  - It is a list of tuples which each tuple containing the **feature dictionary** and the **sentiment string** for each tweet. The sentiment string is also called ‘label’.

[({'contains(not)': False,

...

'contains(this)': True,

...

'contains(love)': True,

...

'contains(car)': True,

...

'contains(great)': False}, **'positive'**),

SA on Tweets Using NLTK

({'contains(not)': False,
'contains(view)': True,
...
'contains(this)': True,
...
'contains(amazing)': True,
...
'contains(enemy)': False,
'contains(great)': False}, '**positive**'),
(…)
…
(…)]

SA on Tweets Using NLTK

- Now that we have our training set, we can train our classifier as before.

```
classifier = NaiveBayesClassifier.train(training_set)
```

- We can display the most informative features for our classifier using the method show_most_informative_features.

```
classifier.show_most_informative_features(100) # default 10
```

SA on Tweets Using NLTK

The Test Data:

I feel happy this morning. positive.
Larry is my friend. positive.
I do not like that man. negative.
My house is not great. negative.
Your song is annoying. negative.

```
test_tweets = ['I feel happy this morning',
               'Larry is my friend',
               'I do not like that man',
               'My house is not great',
               'Your song is annoying']
```

SA on Tweets Using NLTK

- Now that we have our classifier trained, we can try to classify a tweet and see what the sentiment type output is.

```
for t in test_tweets:
    print "{0} : {1}".format(t, classifier.classify(extract_features(t.split())))
```

Output:

I feel happy this morning : positive

Larry is my friend : positive

I do not like that man : negative

My house is not great : negative

Your song is annoying : positive

NLTK Tweet Corpus: A Reminder

```
# 3 Files: negative_tweets.json (5K), positive_tweets.json (5K),
# & tweets.20150430-223406.json (20K)

from nltk.corpus import twitter_samples

print twitter_samples.fileids()

strings = twitter_samples.strings('positive_tweets.json')

for string in strings[:10]:
    print string
```

Sentiment Analysis On Twitter

- Or, you could use off-the-shelf classifiers
  - If you are less concerned with the training/classification process, and more interested in the application.
  - Examples:
    - sentiment140.com (API)
    - NLTK/Vader
    - OpinionFinder
    - LIWC ($89.99)
    - Stanford Sentiment Analyzer
    - LingPipe Sentiment Analyzer
    - TextBlob (https://textblob.readthedocs.io/en/dev/)

Sentiment Analysis

In [17]:
```
# pip install nltk
import nltk
nltk.download('vader_lexicon')

import numpy as np
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\Chris\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [18]:
```
twitter_stream = twitter.TwitterStream(auth=auth)
iterator = twitter_stream.statuses.sample()
```

In [19]:
```
tweets = []
for tweet in iterator:
    try:
        if tweet['lang'] == 'en':
            tweets.append(tweet)
    except:
        pass
    if len(tweets) == 100:
        break
```

In [20]:
```
analyzer = SentimentIntensityAnalyzer()
```

In [21]:
```
analyzer.polarity_scores('Hello')
```

Vader

```
>>> from nltk.sentiment.vader import SentimentIntensityAnalyzer
>>> sentences = ["VADER is smart, handsome, and funny.", # positive sentence example
...    "VADER is smart, handsome, and funny!", # punctuation emphasis handled correctly (sentiment intensity adjusted)
...    "VADER is very smart, handsome, and funny.",  # booster words handled correctly (sentiment intensity adjusted)
...    "VADER is VERY SMART, handsome, and FUNNY.",  # emphasis for ALLCAPS handled
...    "VADER is VERY SMART, handsome, and FUNNY!!!",# combination of signals - VADER appropriately adjusts intensity
...    "VADER is VERY SMART, really handsome, and INCREDIBLY FUNNY!!!",# booster words & punctuation make this close to ceiling for score
...    "The book was good.",         # positive sentence
...    "The book was kind of good.", # qualified positive sentence is handled correctly (intensity adjusted)
...    "The plot was good, but the characters are uncompelling and the dialog is not great.", # mixed negation sentence
...    "A really bad, horrible book.",       # negative sentence with booster words
...    "At least it isn't a horrible book.", # negated negative sentence with contraction
...    ":) and :D",     # emoticons handled
...    "",              # an empty string is correctly handled
...    "Today sux",     #  negative slang handled
...    "Today sux!",    #  negative slang with punctuation emphasis handled
...    "Today SUX!",    #  negative slang with capitalization emphasis
...    "Today kinda sux! But I'll get by, lol" # mixed sentiment example with slang and constrastive conjunction "but"
... ]
>>> paragraph = "It was one of the worst movies I've seen, despite good reviews. \
... Unbelievably bad acting!! Poor direction. VERY poor production. \
... The movie was bad. Very bad movie. VERY bad movie. VERY BAD movie. VERY BAD movie!"

>>> from nltk import tokenize
>>> lines_list = tokenize.sent_tokenize(paragraph)
>>> sentences.extend(lines_list)

>>> tricky_sentences = [
...    "Most automated sentiment analysis tools are shit.",
...    "VADER sentiment analysis is the shit.",
...    "Sentiment analysis has never been good.",
...    "Sentiment analysis with VADER has never been this good.",
...    "Warren Beatty has never been so entertaining.",
...    "I won't say that the movie is astounding and I wouldn't claim that \
...    the movie is too banal either.",
...    "I like to hate Michael Bay films, but I couldn't fault this one",
...    "It's one thing to watch an Uwe Boll film, but another thing entirely \
...    to pay for it",
...    "The movie was too good",
```

OpinionFinder

- OpinionFinder looks for subjective statements, and analyze the words in them for positive or negative sentiment. It has 4 components:
  - The first component is a Naive Bayes classifier that distinguishes between subjective and objective sentences using a variety of lexical and contextual features (Wiebe and Riloff, 2005; Riloff and Wiebe, 2003).
  - The second component identifies speech events (e.g., “said,” “according to”) and direct subjective expressions (e.g., “fears,” “is happy”).
  - The third component is a source identifier that identifies the sources of speech events and direct subjective expressions (Choi et al., 2005). The source of a speech event is the speaker; the source of a subjective expression is the experiencer.
  - The final component uses two classifiers, one for identifying sentiment expressions, one for identifying the positive or negative sentiments in the sentiment expressions (Wilson et al., 2005).

http://mpqa.cs.pitt.edu/opinionfinder/ OpinionFinder | MPQA

http://mpqa.cs.pitt.edu/opinionfinder/

Multi-Perspective Question Answering

Main	Corpora	Lexicons	Annotation	OpinionFinder
MPQA Home	News, debates, etc.	Subj. clues, etc.	GATE, MPQA, gfbf	Subjectivity detector

OpinionFinder

Version 1.x

Version 2.x

OpinionFinder System

OpinionFinder is a system that processes documents and automatically identifies subjective sentences as well as various aspects of subjectivity within sentences, including agents who are sources of opinion, direct subjective expressions and speech events, and sentiment expressions. OpinionFinder was developed by researchers at the University of Pittsburgh, Cornell University, and the University of Utah. In addition to OpinionFinder, we are also releasing the automatic annotations produced by running OpinionFinder on a subset of the Penn Treebank. To go to the OpinionFinder download page click here.

contact: mpqa.project@gmail.com

[nlp] [cs] [pitt]

LIWC

- Linguistic Inquiry and Word Count (LIWC) is a text analysis software designed

  "To asses emotional, cognitive, and structural components of text samples using a psychometrically validated internal dictionary." (LIWC.net)

- The software looks for signs of emotions (positive or negative).

- It also keeps a number of statistics on the text samples.

http://liwc.wpengine.com/

DISCOVER LIWC2015

LIWC2015 is the gold standard in computerized text analysis. Learn how the words we use in everyday language reveal our thoughts, feelings, personality, and motivations. Based on years of scientific research, LIWC2015 is more accurate, easier to use, and provides a broader range of social and psychological insights compared to earlier LIWC versions. Check it out.

BUY NOW

WHAT'S NEW

Many new language dimensions. More flexible input and output options. More professional feel. Designed for the beginner and professional. Lease it for a month or buy it.

COMPARE VERSIONS ›

LEARN MORE

LIWC2015 is a powerful research and learning tool based on solid science. To learn how to run LIWC on your desktop, download the the Operator's Manual or a paper on the science behind its development. To compare the new LIWC2015 categories with those from earlier LIWC versions, visit Compare Dictionaries. To get a better appreciation

LIWC LICENSE

The LIWC license is for academic and university purposes only and entitles you to install and activate the software under two usernames. Discounts are available for multi-user versions. All non-academic/non-university use of LIWC requires a commercial license. Commercial licenses are available through Receptiviti, Inc.

http://nlp.stanford.edu/sentiment/index.html

Sentiment Analysis

Deeply Moving: Deep Learning for Sentiment Analysis

This website provides a live demo for predicting the sentiment of movie reviews. Most sentiment prediction systems work just by looking at words in isolation, giving positive points for positive words and negative points for negative words and then summing up these points. That way, the order of words is ignored and important information is lost. In constrast, our new deep learning model actually builds up a representation of whole sentences based on the sentence structure. It computes the sentiment based on how words compose the meaning of longer phrases. This way, the model is not as easily fooled as previous models. For example, our model learned that funny and witty are positive but the following sentence is still negative overall:

This movie was actually neither that funny, nor super witty.

The underlying technology of this demo is based on a new type of Recursive Neural Network that builds on top of grammatical structures. You can also browse the Stanford Sentiment Treebank, the dataset on which this model was trained. The model and dataset are described in an upcoming EMNLP paper. Of course, no model is perfect. You can help the model learn even more by labeling sentences we think would help the model or those you try in the live demo.

Paper Title and Abstract

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank
Semantic word spaces have been very useful but cannot express the meaning of longer phrases in a principled way. Further progress towards understanding compositionality in tasks such as sentiment detection requires richer supervised training and evaluation resources and more powerful models of composition. To remedy this, we introduce a

Paper: Download pdf

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng and Christopher Potts

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)

Dataset Downloads:

Main zip file with readme (6mb)
Dataset raw counts (5mb)
Train,Dev,Test Splits in PTB Tree Format

Code: Download Page

Press: Stanford Press Release

Dataset visualization and web design by Jason Chuang. Live demo by Jean Wu, Richard Socher, Rukmani Ravisundaram and Tayyab Tariq.

Sentiment Analysis

Code for Deeply Moving: Deep Learning for Sentiment Analysis

The original code was written in Matlab. Due to the strong interest in this work we decided to re-write the entire algorithm in Java for easier and more scalable use without license restrictions.

The current model is integrated into Stanford CoreNLP as of version 3.3.0 and is available here. This includes the model and the source code, as well as the parser and sentence splitter needed to use the sentiment tool.

Stanford CoreNLP home page

You can run this code with our trained model on text files with the following command:

```
java -cp "*" -mx5g edu.stanford.nlp.sentiment.SentimentPipeline -file foo.txt
java -cp "*" -mx5g edu.stanford.nlp.sentiment.SentimentPipeline -stdin
```

An evaluation tool is included with the distribution:

```
java edu.stanford.nlp.sentiment.Evaluate edu/stanford/nlp/models/sentiment/sentiment.ser.gz test.txt
```

Models can be retrained using the following command using the PTB format dataset:

```
java -mx8g edu.stanford.nlp.sentiment.SentimentTraining -numHid 25 -trainPath train.txt -devPath dev.txt -train -model model.ser.gz
```

Paper: Download pdf

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng and Christopher Potts

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)

Dataset Downloads:

Main zip file with readme (6mb)
Dataset raw counts (5mb)
Train,Dev,Test Splits in PTB Tree Format

Code: Download from the CoreNLP home page

Dataset visualization and web design by Jason Chuang. Live demo by Jean Wu, Richard Socher, Rukmani Ravisundaram and Tayyab Tariq. Java code package by

Sentiment Tutorial

alias-i

What is Sentiment Analysis?

Sentiment analysis involves classifying opinions in text into categories like "positive" or "negative" often with an implicit category of "neutral". A classic sentiment application would be tracking what bloggers are saying about a brand like Toyota. Sentiment analysis is also called opinion mining or voice of the customer. There are lots of startups in this area and conferences.

This tutorial covers assigning sentiment to movie reviews using language models. There are many other approaches to sentiment. One we use fairly often is sentence based sentiment with a logistic regression classifier. Contact us if you need more information. For movie reviews we focus on two types of classification problem:

- Subjective (opinion) vs. Objective (fact) sentences
- Positive (favorable) vs. Negative (unfavorable) movie reviews

How is it Done?

The high-level idea is to use LingPipe's language classification framework to do two classification tasks: separating subjective from objective sentences, and separating positive from negative movie reviews. In the third section, we show how to build a hierarchical classifier by composing these models.

Who's Idea was This?

This tutorial essentially reimplements the basic classifiers and then the hierarchical classification technique described in Bo Pang and Lillian Lee's 2004 ACL paper "A sentimental education."

Downloading Training Corpora

Luckily for us, Lillian Lee and Bo Pang have provided annotated slices of movie review data for polarity (both boolean and scalar), and subjectivity. These three datasets are described at:

- Movie Review Data Home Page

We will be using the subjectivity and boolean polarity data:

Pang and Lee's Data

Data Set	Data	Read Me	Description
Polarity v2.0	Data (3.1MB)	README	1000 positive, 1000 negative full text movie reviews. Drawn from IMDB's archive of rec.arts.movies.reviews. Heuristic scripts used to extract first review score from text.
Subjectivity v1.0	Data (508KB)	README	5000 "objective", 5000 "subjective" sentences. Objective from Internet Movie Database (IMDB) plot summaries, subjective from Rotten Tomatoes customer review "snippets"

Assignment #3 of My Text Mining Class

1. Use the UMich Sentiment Classification Data Set (See next slide), which I have already uploaded to Blackboard. (Under **Information**)
2. Train your own Sentiment Analyzer using their train data. You can use any machine learning algorithm in nltk.classify (for Python programmers) or Weka (for Java programmers), and/or a sentiment lexicon (e.g. SentiWordNet, also uploaded to Blackboard)
3. Run the test data through your Sentiment Analyzer, and submit a zipped folder that contains your code, with comments, and the output, which should be a **.txt** file with **33052** lines. In each line, there should be exactly one integer, 0 (negative) or 1 (positive), according to your classification results.
4. I'll randomly select a subset to evaluate your performance, in terms of Accuracy **(1.0 – wrongly classified tweets/total number of tweets**)

and give you an appropriate score.

https://inclass.kaggle.com/c/si650winter11

Completed • Knowledge • 28 teams

UMICH SI650 - Sentiment Classification

Mon 28 Mar 2011 – Fri 15 Apr 2011 (4 years ago)

Dashboard

Home
Data
Make a submission

Information
Description
Rules

Leaderboard
Public
Private

Private Leaderboard

1. William Wilcox
2. Breakfast Pants
3. Rodger Devine

Competition Details » Get the Data » Make a submission

This is an in-class contest hosted by University of Michigan SI650 (Information Retrieval)

This is a text classification task - sentiment classification. Every document (a line in the data file) is a sentence extracted from social media (blogs). Your goal is to classify the sentiment of each sentence into "positive" or "negative".

The training data contains 7086 sentences, already labeled with 1 (positive sentiment) or 0 (negative sentiment). The test data contains 33052 sentences that are unlabeled. The submission should be a .txt file with 33052 lines. In each line, there should be exactly one integer, 0 or 1, according to your classification results.

Sentiment Analysis in Tv

alt.qcri.org/semeval2017/task4

http://alt.qcri.org/semeval2017/task4/

Home | Important Dates | Data and Tools | Results | Papers

SemEval-2017 Task 4

Sentiment Analysis in Twitter

Sentiment Analysis in Twitter

Summary

This will be a rerun of SemEval-2016 Task 4 with several changes:

- new subtasks: another language and user information
- new evaluation measures
- new training datasets
- new test datasets

I. Introduction

The recent rise of social media has greatly democratized content creation. Facebook, Twitter, Skype, Whatsapp and LiveJournal are now commonly used to share thoughts and opinions about anything in the surrounding world. This proliferation of social media content has created new opportunities to study public opinion, with Twitter being especially popular for research due to its scale, representativeness, variety of topics discussed, as well as ease of public access to content.

Contact Info

- Sara Rosenthal, IBM Research
- Noura Farra, Columbia University
- Preslav Nakov, Qatar Computing Research Institute, HBKU

email: semevaltweet@googlegroups.com

Other Info

Announcements

- **Results, and gold labels are released**
- Arabic and English TEST INPUT v1.0 for phase 2 (subtasks B, D) released
- Arabic and English TEST INPUT v3.0 for phase 1 (subtasks A, C, E) released

SemEval-2018

International Workshop on Semantic Evaluation

Sponsored by SIGLEX

Tasks

We are pleased to announce the following tasks in SemEval-2018.

Affect and Creative Language in Tweets

- Task 1: Affect in Tweets
- Task 2: Multilingual Emoji Prediction
- Task 3: Irony Detection in English Tweets

Coreference

- Task 4: Character Identification on Multiparty Dialogues
- Task 5: Counting Events and Participants within Highly Ambiguous Data covering a very long tail

Information Extraction

- Task 6: Parsing Time Normalizations
- Task 7: Semantic Relation Extraction and Classification in Scientific Papers
- Task 8: Semantic Extraction from CybersecUrity REports using Natural Language Processing

Contact Info

Organizers

- Marianna Apidianaki, LIMSI, CNRS, University Paris-Saclay & University of Pennsylvania
- Saif M. Mohammad, National Research Council Canada
- Jonathan May, ISI, University of Southern California
- Ekaterina Shutova, University of Cambridge
- Steven Bethard, University of Alabama at Birmingham
- Marine Carpuat, University of Maryland

Email

semeval-organizers@googlegroups.com

Note that this is the mailing list for SemEval organizers. For questions on a particular task, post them at the *task*

http://alt.qcri.org/semeval2019/index.php?id=tasks

Home | Tasks | CodaLab | Papers | Frequently Asked Questions | Workshop | Sponsors

SemEval-2019

International Workshop on Semantic Evaluation

Sponsored by SIGLEX and Microsoft

Tasks

We are pleased to announce the following tasks in SemEval-2019.

Frame semantics and semantic parsing

- Task 1: Cross-lingual Semantic Parsing with UCCA [mailing list] [email organizers]
- Task 2: Unsupervised Lexical Semantic Frame Induction [mailing list] [email organizers]

Opinion, emotion and abusive language detection

- Task 3: EmoContext: Contextual Emotion Detection in Text [discussion group] [email organizers]
- Task 4: Hyperpartisan News Detection [mailing list] [email organizers]
- Task 5: HatEval: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter [mailing list] [email organizers]
- Task 6: OffensEval: Identifying and Categorizing Offensive Language in Social Media [mailing list] [email organizers]

Fact vs fiction

Contact Info

Organizers

- Jonathan May, ISI, University of Southern California
- Ekaterina Shutova, University of Amsterdam
- Aurelie Herbelot, University of Trento
- Xiaodan Zhu, Queen's University
- Marianna Apidianaki, LIMSI, CNRS, Université Paris-Saclay & University of Pennsylvania
- Saif M. Mohammad, National Research Council Canada

Email

semeval-organizers@googlegroups.com

Note that this is the mailing list for SemEval organizers. For questions on a particular task, post them at the *task*

SemEval-2020

International Workshop on Semantic Evaluation

Sponsored by SIGLEX

Tasks

We are pleased to announce the following tasks in SemEval-2020.

Lexical semantics

- Task 1: Unsupervised Lexical Semantic Change Detection [mailing list] [email organizers]
- Task 2: Predicting Multilingual and Cross-Lingual (Graded) Lexical Entailment [mailing list] [email organizers]
- Task 3: Graded Word Similarity in Context (GWSC) [discussion forum] [mailing list] [email organizers]

Common Sense Knowledge and Reasoning, Knowledge Extraction

- Task 4: Commonsense Validation and Explanation [mailing list] [email organizers]
- Task 5: Modelling Causal Reasoning in Language: Detecting Counterfactuals [mailing list] [email organizers]
- Task 6: DeftEval: Extracting Definitions from Free Text in Textbooks [mailing list] [email organizers]

Humour, Emphasis, and Sentiment

Contact Info

Organizers

- Aurelie Herbelot, University of Trento
- Xiaodan Zhu, Queen's University
- Nathan Schneider, Georgetown University
- Alexis Palmer, University of North Texas
- Jonathan May, ISI, University of Southern California
- Ekaterina Shutova, University of Amsterdam

Email

- **Specific tasks:** See red links on the tasks page
- **General SemEval organization:** semeval-organizers@googlegroups.com

Other Info

Announcements

- 2020/03/31: Paper submission

Tasks

We are pleased to announce the following tasks in SemEval-2020.

Lexical semantics

- Task 1: Unsupervised Lexical Semantic Change Detection [mailing list] [email organizers]
- Task 2: Predicting Multilingual and Cross-Lingual (Graded) Lexical Entailment [mailing list] [email organizers]
- Task 3: Graded Word Similarity in Context (GWSC) [discussion forum] [mailing list] [email organizers]

Common Sense Knowledge and Reasoning, Knowledge Extraction

- Task 4: Commonsense Validation and Explanation [mailing list] [email organizers]
- Task 5: Modelling Causal Reasoning in Language: Detecting Counterfactuals [mailing list] [email organizers]
- Task 6: DeftEval: Extracting Definitions from Free Text in Textbooks [mailing list] [email organizers]

Humour, Emphasis, and Sentiment

- Task 7: Assessing Humor in Edited News Headlines [mailing list][email organizers]
- Task 8: Memotion Analysis [mailing list] [email organizers]
- Task 9: Sentiment Analysis for Code-Mixed Social Media Text [mailing list] [email organizers]
- Task 10: Emphasis Selection for Written Text in Visual Media [mailing list] [email organizers]

Societal Applications of NLP

- Task 11: Detection of Propaganda Techniques in News Articles [mailing list] [email organizers]
- Task 12: OffensEval 2: Multilingual Offensive Language Identification in Social Media [mailing list] [email organizers]

Contact Info

Organizers

- Aurelie Herbelot, University of Trento
- Xiaodan Zhu, Queen's University
- Nathan Schneider, Georgetown University
- Alexis Palmer, University of North Texas
- Jonathan May, ISI, University of Southern California
- Ekaterina Shutova, University of Amsterdam

Email

- **Specific tasks:** See red links on the tasks page
- **General SemEval organization:** semeval-organizers@googlegroups.com

Other Info

Announcements

- 2020/03/31: Paper submission deadlines extended (May 1, May 8)
- 2020/03/31: COLING 2020 announces new dates in December 2020
- 2020/03/31: New awards for SemEval 2020: Best Task, Best Paper!
- 2020/03/31: New submission guidelines (new style files, new length restrictions) for SemEval 2020 papers.
- 2020/03/19: SemEval-2021 task proposal submission deadline extended