Lecture 1

Von-Wun Soo

outline

1. classification and supervised learning
a. k nearness neighbor:
b. Decision tree
c. Neural networks
d. Ensemble learning,
e. Ada boosting
2. Training set, test set, validation set
Performance evaluation metrics, cross validation (Precision/recall, F1 Score, AUC/ROC)

What is machine learning?

Why Learning?

Systems change “something” in order to:

- Survive or live safely/comfortable?
- Acquire information (unknown → known)
- Reduce uncertainty
- Acquire ability (unable → able)
- Avoid/reduce mistake or punishment?
- Obtain/increase reward/payoff?
- Increase efficiency and reduce cost?
- Predict what happen next to prepare or act earlier

Machine learning techniques

- Analytic-based learning (with a world model)
  - Explanation-based learning (EBL)
  - Theory revision/refinement
  - Knowledge acquisition/elicitation/assimilation
- Similarity-based learning (without a world model)
  - Learning from examples: ID3
  - Conceptual formation/clustering: UNIMEM,COWEB
  - Case-based learning
  - Connectionist learning: NN, GA,GP

Classification of machine learning techniques

- Supervised：Decision tree, Error back-propagation neural networks
- Unsupervised: Conceptual Clustering algorithms, k-mean clustering
- Semi-supervised: （half labeling), the unlabeled examples belong to the same classes as the training examples.
- Transfer learning: additional labeled data from different domains are required to transfer learning results from current domain
- Self-taught learning: unlabeled examples may be different classes from training examples.
- Reinforcement learning: Q leaning, Evolutionary learning: GA, GP

Classification of machine learning techniques

- Active Learning: find most effective training instances (ask oracle) to improve performance
- Deep learning: integrate traditional statistical and machine learning into multiple layers neural networks
- Multi-strategies, multi-paradigms, hybrid methods: GA/AQ, Model-based/Instance based, Symbolic/sub-symbolic
- ill-defined machine learning problems:
    - Extended knowledge acquisition
    - scientific discovery
    - tutorial systems

No free lunch theorem

- The no free lunch theorem for machine learning (Wolpert, 1996) states that, averaged over all possible data generating distributions, every classification algorithm has the same error rate when classifying previously unobserved points.
- In other words, in some sense, no machine learning algorithm is universally any better than any other.
- The data distribution matters.

Machine Learning Outline I

- Introduction: learning bias
- Decision tree learning and ensemble learning
- Neutral network learning: Perceptron, Error backpropagation
- Reinforcement learning
- Bayesian learning: NB
- Conceptual clustering: Coweb, UNIMEM, self organizing map, k-means
- Swarm intelligence: GP, Ant colony algorithm, swarm particle swarm optimization

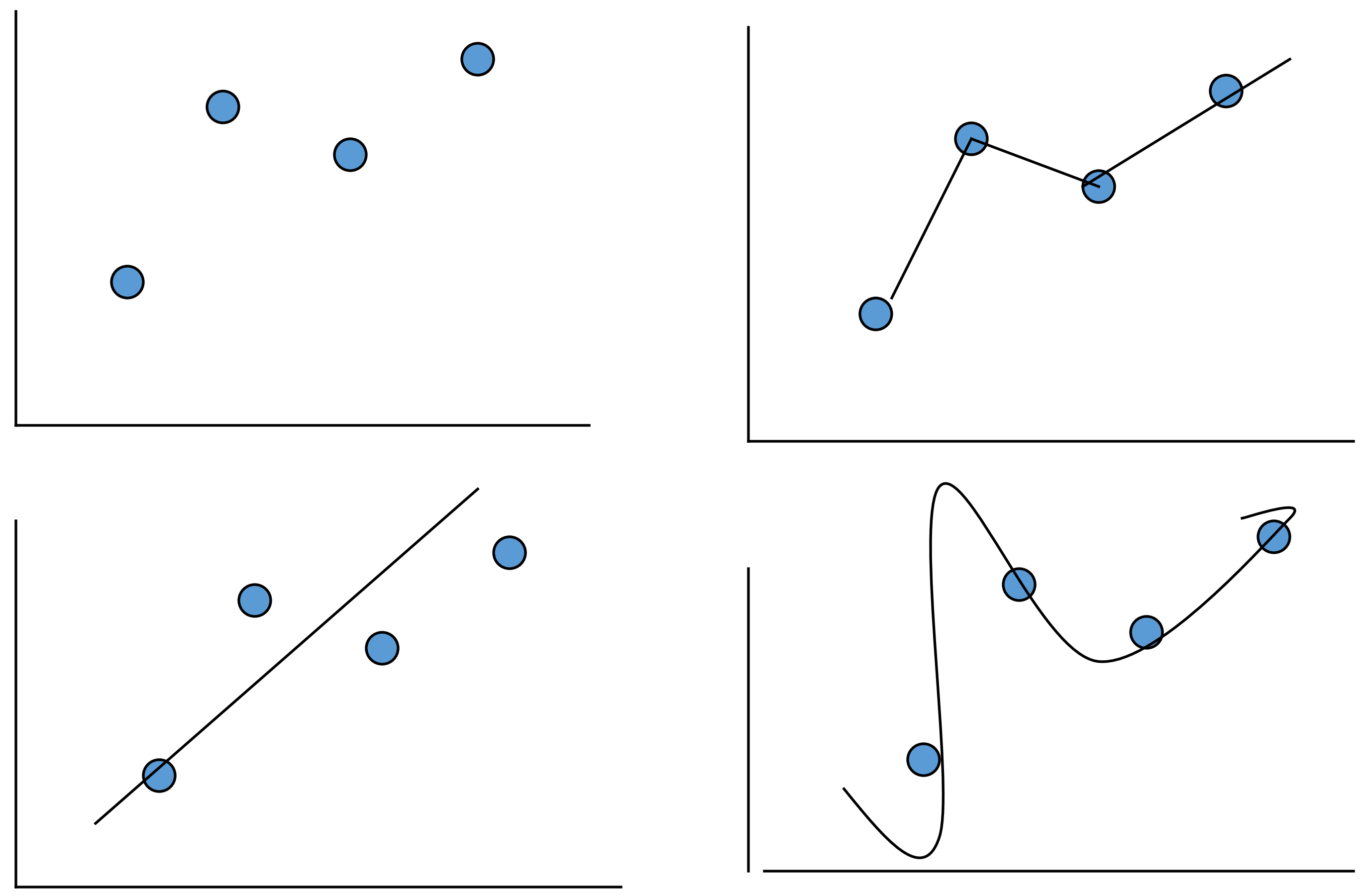
Machine Learning as Search

- Machine learning as a heuristic search
  - Given a set of data (training examples)
  - Find a best hypothesis in the hypothesis space that **best** explain the data
- Learning bias: preference
  - Selection on hypothesis space language: prefer different representation
  - Hypothesis preference: prefer simple hypthesis than complex hypothesis

Machine Learning as an Optimization search

- Given a specification of a dataset and
- a cost function, and
- a model,
- Then an optimization procedure finds the most fit parameters of model to the data according to the cost function.

Learning as finding a function that fits data -- bias



Occam's Razor

- Prefer the simplest hypothesis (simple is beautiful)
- MDL principle: minimum description length principle

  $-\log_2 P(h) - \log_2 P(D|h)$

Adaptive learning agents

- Agents can learn
  - Build world model
  - Predict the behaviors of other agents
  - Estimate the worth of a plan
  - Tune coordination mechanism
  - Adapt local parameters influencing problem solving performance

Application of machine learning to data mining

- Induce rules from databases
- Induce causality from temporal database
- Induce document classification from texts
- Induce user model or profile from browsing sequences [log files]
- Etc.

Classification problems and supervised learning

- Classification problems:
- Find a mapping function f: X ->Y so that $f(x_i)= y_i$ i=1,…,n given n training instance pair $(x_1,y_1),(x_2,y_2),…(x_n,y_n)$ where xi can be an input vector while y_i is the class label.

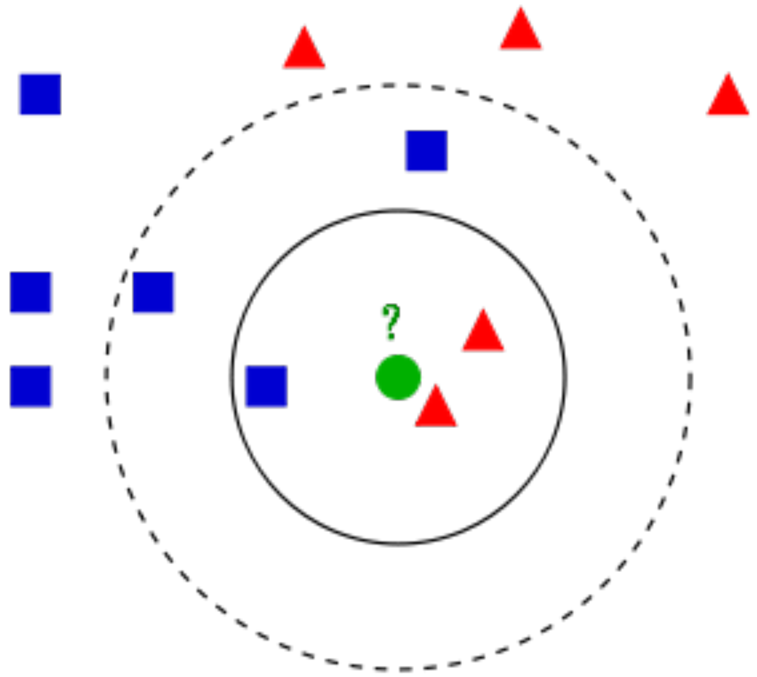
- Supervised learning: For each training instance, we have teachers (experts) to assign correct labels to the training instance.

K-nearest neighbor classifier

- Supposed we we have n pairs of training instance $[x_1,y_1],[x_2,y_2]$, …, $[x_n,y_n]$, i=1,…,n
- We have a new instance x’, we wish to predict y’.
- We have a similarity function to compare x_i and x’ , e.g. sim(x_i,x’)
- So we can search k most nearest x_p such that

  x_p= argmin_i {sim(x_i, x’)} and it corresponds to a y_p for each x_p
- We predict y’ as the majority vote of k y_p’s

- Note: if K is odd then we avoid the tie.



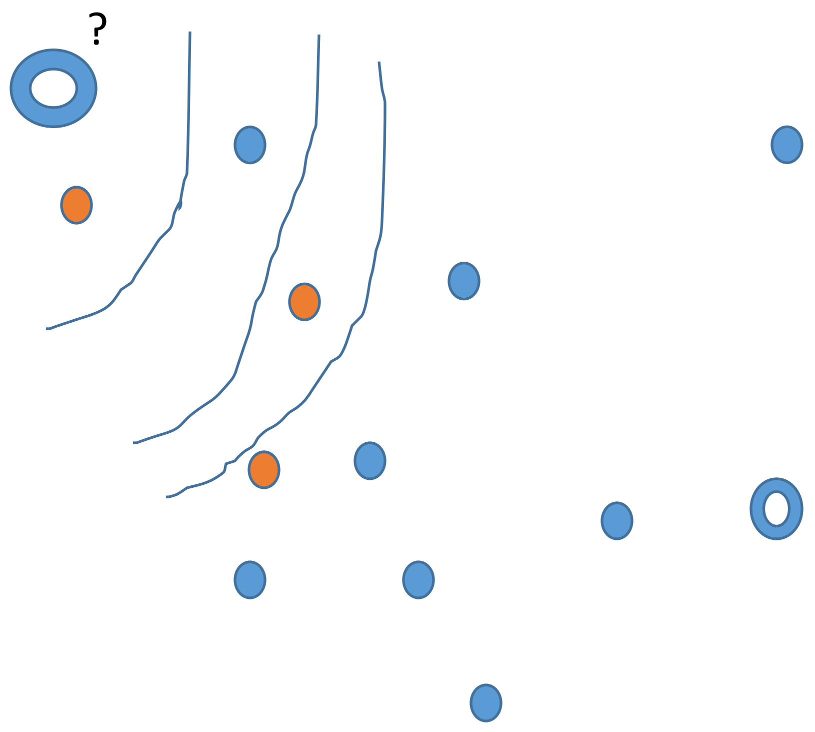
K nearest neighbor classification

K=1 red

K=2 tie

K=3 red

K= 4 red



K-nearest neighbor

- K usually odd namely 1, 3, 5, … to avoid tie.
- Increase k, more smooth

- Has to compute the (similarity) distance between the query example to each previous memorized training examples.

Pro and cons of KNN

Pros:

- Insensitive to outliers — accuracy can be affected from noise or irrelevant features
- No assumptions about data — useful, for example, for nonlinear data
- Simple algorithm — to explain and understand/interpret
- High accuracy (relatively) — it is pretty high but not competitive in comparison to better supervised learning models
- Versatile — useful for classification or regression

Cons:

- Computationally expensive — because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)

Extension or variants of k-NN

1. Weighted nearest neighbor: vote by weights.

- The nearest neighbor has a higher weight than the next nearest neighbor, and overall summing weights equals 1.

2. Condensed nearest neighbor: reduce the data set for *k*-NN classification. Selects the set of prototypes *U* from the training data, such that 1NN with *U* can classify the examples almost as accurately as 1NN does with the whole data set.

Condensed nearest neighbors with prototype selection

CNN model reduction for k-NN classifiers

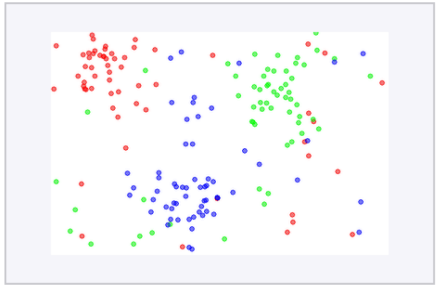


Fig. 1. The dataset.

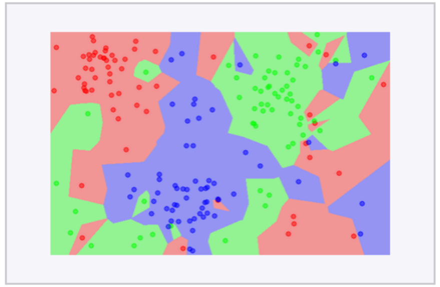


Fig. 2. The 1NN classification map.

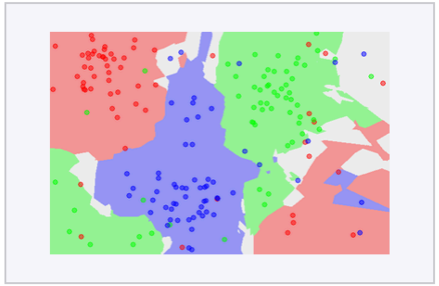


Fig. 3. The 5NN classification map.

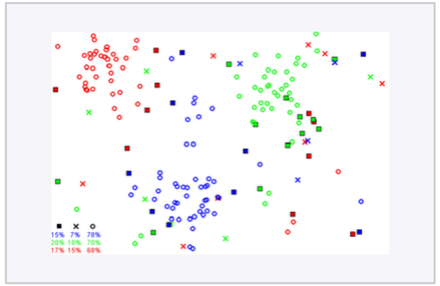


Fig. 4. The CNN reduced dataset.

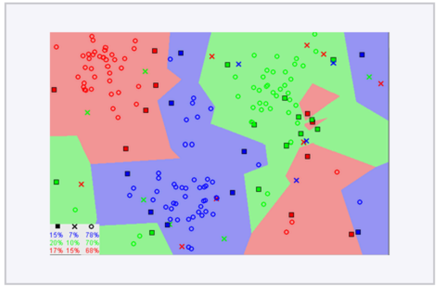


Fig. 5. The 1NN classification map
based on the CNN extracted

ID3--Learning a decision tree

- Finding a hypothesis in terms of a decision tree that classifies the training examples into the correct classes
- But the decision tree hypothesis space can be huge:
    - guiding by the heuristics of information entropy

Information Entropy

Information entropy

- $-p^{+}\log_2 p^{+} - p^{-}\log_2 p^{-}$
- C=

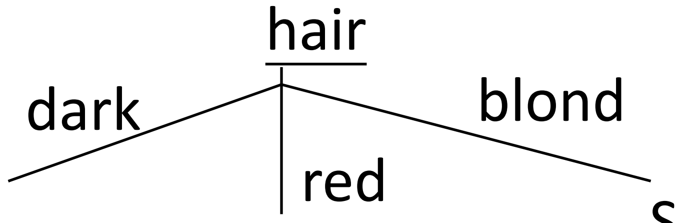
{short, blond, blue: +, short, dark, blue:-, tall, dark, brown, -, tall, blond, brown:-,

tall, dark, blue:-, short, blond, brown:-,

tall, red, blue:+, tall, blond, blue:+}

Select "hair" attribute

- M(C) = $-3/8\log_2 3/8 - 5/8\log_2 5/8$
  = 0.954 bit (original entropy)



short, dark,blue:-
tall,dark,blue:-
tall,dark,brown:-
3/8

tall,red,blue:+
1/8

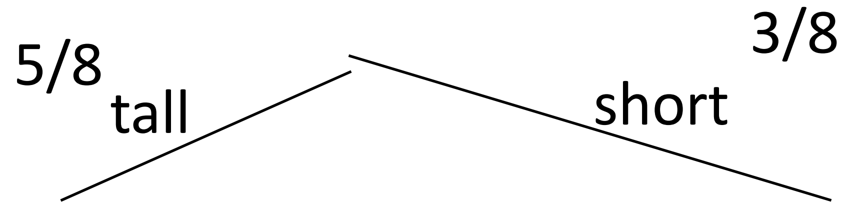
Short,blond,blue:+
Short,blond,brown:-
Tall,blond,blue:+
Short,blond,brown:-

4/8

3/8*0+1/*0+4/8*1= 0.5 bit

Select “height” attribute

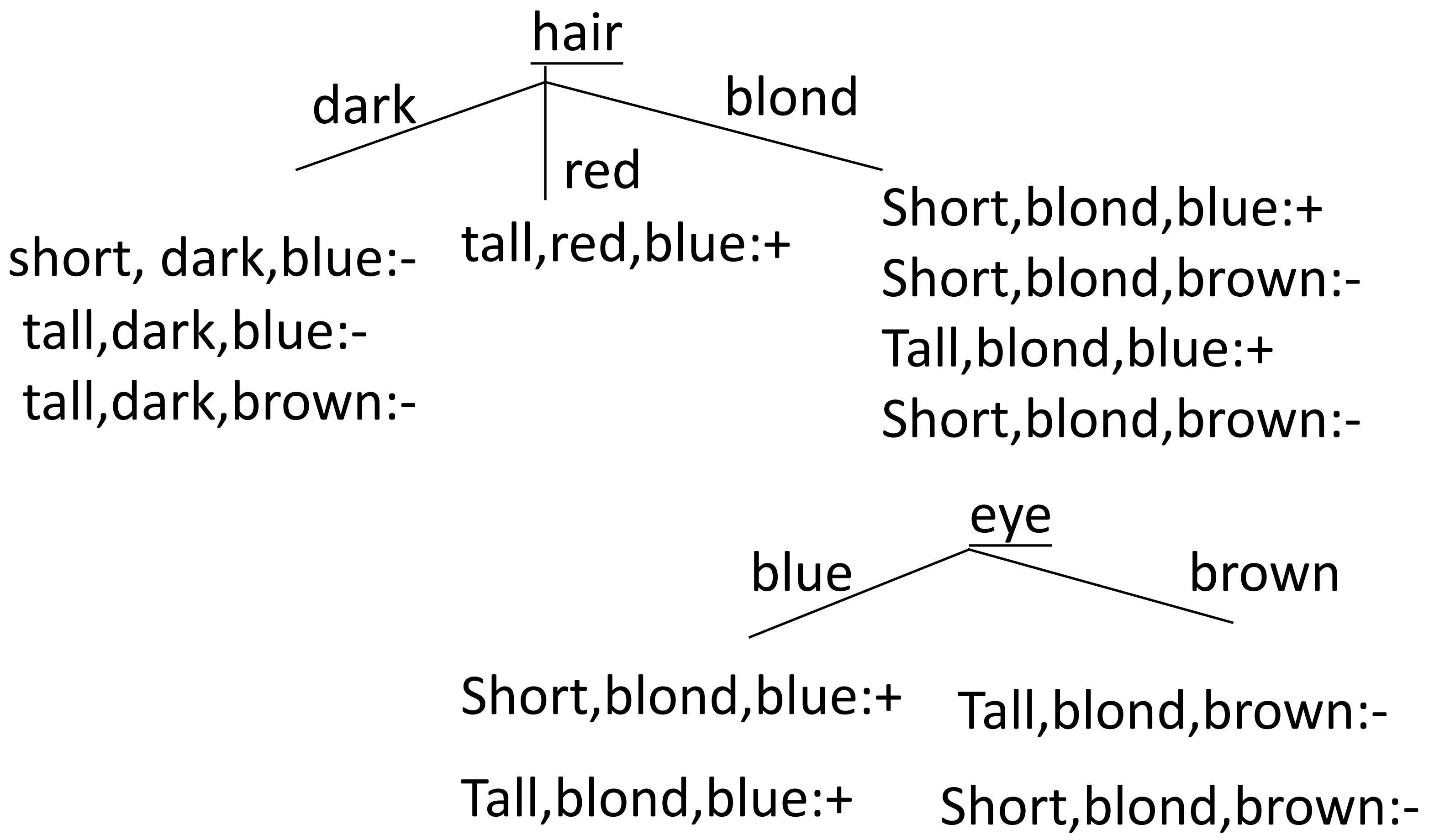
- height



Tall,blond,brown:-	Short,blond,blue:+
Tall,red,blue:+	Short,dark,blue:-
Tall,dark,blue:-	Short,blond,brown:-
Tall,blond,blue:+	12/3log1/3-2/3log2/3=0.918
Tall,dark,brown:-	

-2/5log2/5-3/5log3/5=0.918

B(C,”height”)=5/8*0.917+3/8*0.918 = 0.951


hair
dark
red
blond
short, dark,blue:-
tall,dark,blue:-
tall,dark,brown:-
tall,red,blue:+
Short,blond,blue:+
Short,blond,brown:-
Tall,blond,blue:+
Short,blond,brown:-
eye
blue
brown
Short,blond,blue:+
Tall,blond,blue:+
Tall,blond,brown:-
Short,blond,brown:-

Convert Learned Decision Tree into Acquired Rules

- **If** hair is dark **then** negative
- **If** hair is red **then** positive
- **If** hair is blond and eye is blue **then** positive
- **If** hair is blond and eye is brown **then** negative

Problems of ID3

- Noise problem
- Unknown attribute value Problem
- Continuous value problem
- Prefer multi-value attribute
- Incremental decision tree

Bias of decision tree learning

- Prefer tree hypothesis language
- Prefer shorter tree than longer tree

Noise problem (over-fitting problem)

- Eliminate irrelevant attributes
  - Using chi-square test stochastic independence
- Allow not exact classification
- Reduce error pruning (removing sub-trees)
- Rule Post-Pruning (C4.5)
  - convert into rules and generalize the preconditions

Unknown attribute value problem

- Replacement of unknown value:
  - Bayesian approach
  - Most common value
  - Decision tree
- Bayesian approach: prob(A_i|class = p) = p_i/p

Unknown attribute value problem

- True $p_i = p_i + p_u \cdot ratio_i$

  where

  $$ratio_i = \frac{p_i + n_i}{\Sigma_i\, p_i + n_i}$$

- p_i, n_i : number of positive and negative objects having attribute value Ai
- Similarly, for true n_i

Prefer multi-value attribute (ID3 bias)

- ID3 could tend to select attribute with more values
- Adjust by Gain ratio [Quinlan1986]:
- C: p,n ; A: $A_1,\ldots,A_{v;}$ p_i, n_i for A_i
- IV(A)=

  $- \Sigma_i(p_i+n_i/p+n)\log(p_i+n_i/p+n)$
- Choose A such that gain(A)/IV(A) is maximized

Continuous Value problem

- Discretize the continuous attribute
- $A > c$

Handling attribute with different costs

- $Gain^2(S,A)/cost$
- $(2^{gain(S,A)}-1)/(cost(A)+1)^w$
- Where w [0,1] relative importance of cost versus information gain

Incremental Decision tree — ID5

- ID3 is batch learning
- ID5 is incremental learning algorithm

Multiple decision trees

- Random forest decision:

averaging multiple deep decision trees

Random_forest

- Vote on results from multiple decision trees as final prediction

Ensemble learning

- Combining many classifiers to achieve higher performance
- A kind of meta-algorithm
- Boosting- reduce bias
- Bagging (sampling with replacement)/dagging (sampling w/o replacement) – reduce variance
- stacking

Boosting learning

- Boost a weak classifier whose performance is slightly greater than 0.5 accuracy to higher performance
- Weak learner -> strong learner

The Boosting algorithm --Adaboost

- Given: $(x_1,y_1),\ldots,(x_m,y_m)$ where $x_i \in X$, $y_i \in \{-1,+1\}$.

Initialize:

$D_1(i) = 1/m$ for $i = 1,\ldots,m$.

For $t = 1,\ldots,T$:

Train weak learner using distribution D_t .

Get weak hypothesis $h_t : X \rightarrow \{-1, +1\}$.

Aim: select h_t with low weighted error:

$\varepsilon_t = \Pr_{i\sim Dt} [h_t(x_i) \neq y_i]$.

Choose $\alpha_t = 1/2 \ln\frac{1-\varepsilon_t}{\epsilon_t}$.

Update, for $i = 1,\ldots,m$:

$D_{t+1}(i) = D_t(i) \exp(-\alpha_t y_i h_t(x_i))/ Z_t$

where Zt is a normalization factor (chosen so that D_{t+1} will be a distribution). Output the final hypothesis:

$H(x)=\text{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$.

The Bagging algorithm

Input: Data set D = {(x1, y1), (x2, y2), · · · , (x_m, y_m)};

Base learning algorithm L;

Number of learning rounds T .

Process:

```
for t = 1,··· ,T:
        $D_t$ = Bootstrap(D); % Generate a bootstrap sample from D  {bagging or dagging sampling}
        $h_t = L(D_t)$        % Train a base learner $h_t$ from the bootstrap sample
end.
```

Output: H(x) = argmax $_{y \in Y} \sum_{t=1}^{T} 1(y = ht(x))$ % the value of 1(a) is 1 if a is true and 0 otherwise

The Stacking Framework

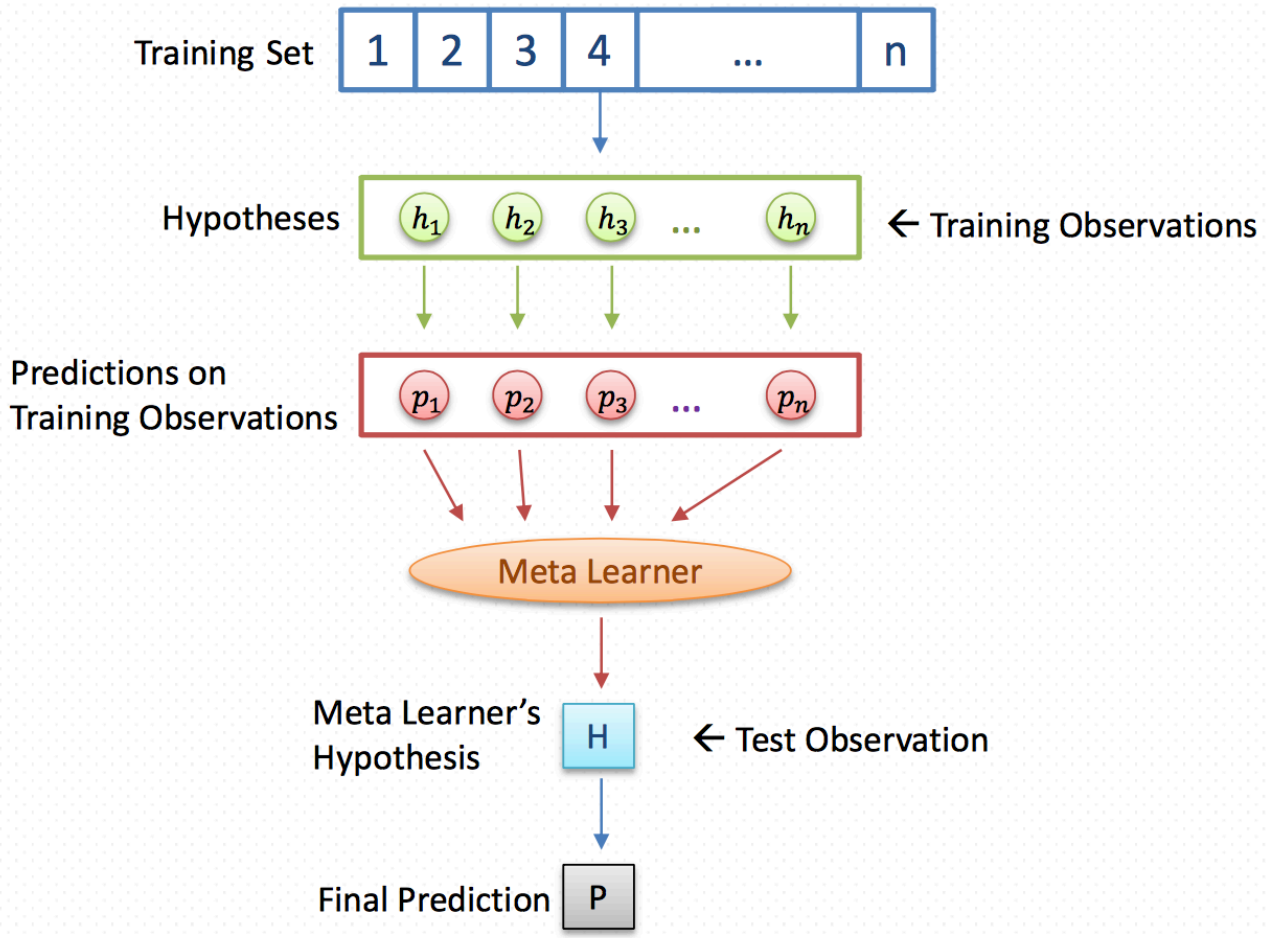
- Stacking is concerned with combining multiple classifiers generated by different learning algorithms L_1, ... , L_N on a single dataset S, which is composed by a feature vector s_i = (x_i , y_i).
- The stacking process can be broken into two phases:

1. Generate a set of base-level classifiers C_1 , ... , C_N

- Where C_i =L_i(S)

2. Train a meta-level classifier to combine the outputs of the base-level classifiers

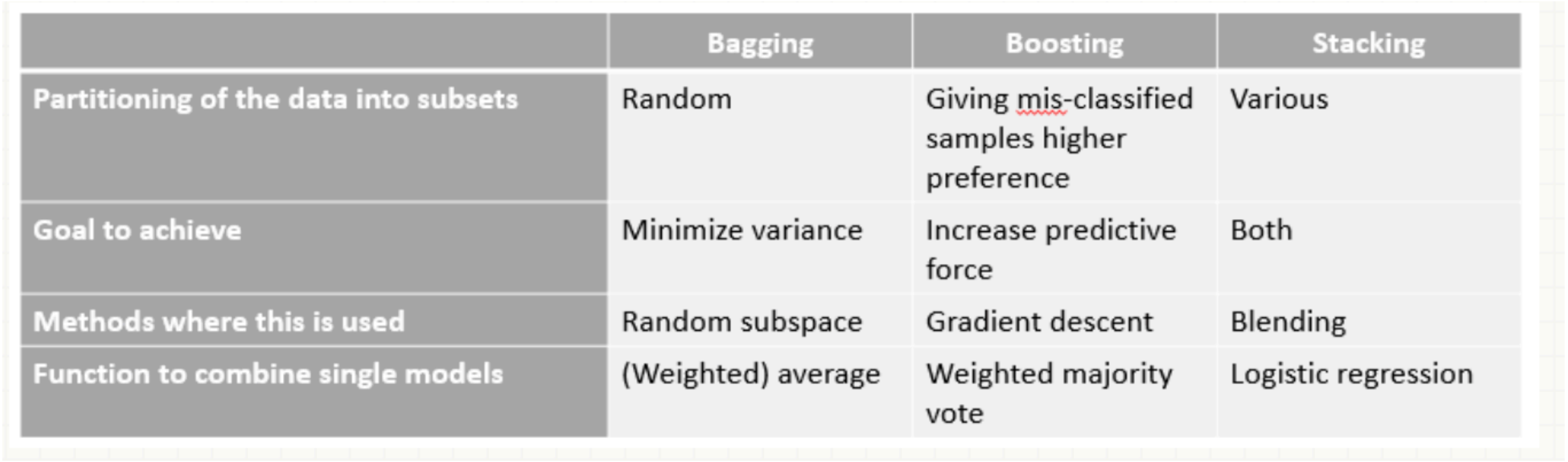
The Stacking Framework



The Stacking algorithm

```
Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)\}$;
First-level learning algorithms $L_1, \cdots, L_T$;
Second-level learning algorithm $L$.
Process:
for $t = 1, \cdots, T$:
    $h_t = L_t(D)$      % Train a first-level individual learner $h_t$ by applying the first-level
end;                    % learning algorithm $L_t$ to the original data set $D$
$D' = \emptyset$;       % Generate a new data set D' with m data
for $i = 1, \cdots, m$:
    for $t = 1, \cdots, T$:
    $z_{it} = h_t(x_i)$     % Use $h_t$ to classify the training example $x_i$
    end;
    $D' = D' \cup \{((z_{i1}, z_{i2}, \cdots, z_{iT}), y_i)\}$   % with respect to $x_i$
end;
  $h' = L(D')$.     % Train the second-level learner $h'$ by applying the second-level
                    % learning algorithm $L$ to the new data set $D'$
Output: $H(x) = h'(h_1(x), \cdots, h_T(x))$
```

Comparison

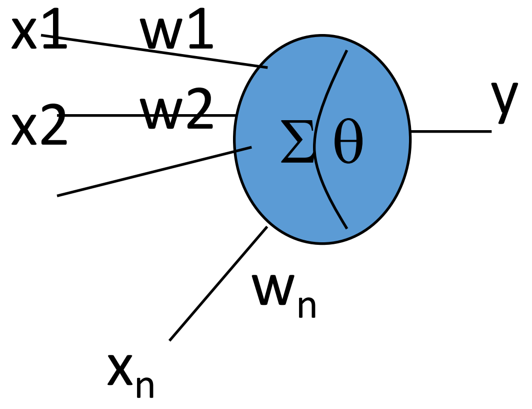


	Bagging	Boosting	Stacking
Partitioning of the data into subsets	Random	Giving mis-classified samples higher preference	Various
Goal to achieve	Minimize variance	Increase predictive force	Both
Methods where this is used	Random subspace	Gradient descent	Blending
Function to combine single models	(Weighted) average	Weighted majority vote	Logistic regression

Neural Networks

- Perceptron learning
- Feedforward Error backpropagation NN learning
- Self-organized feature map (unsupervised)

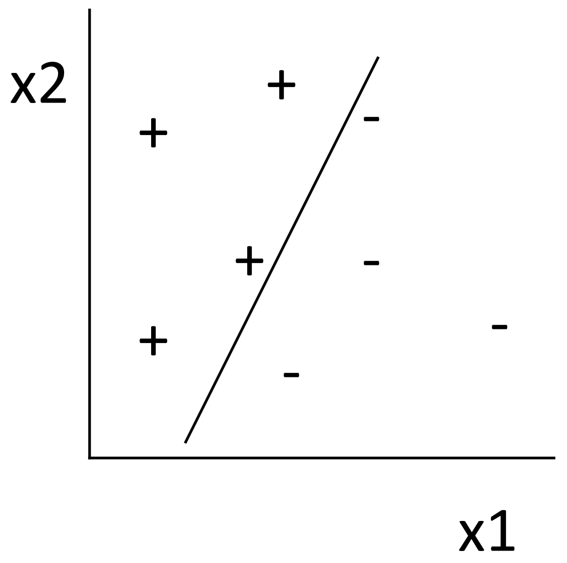
Perceptron



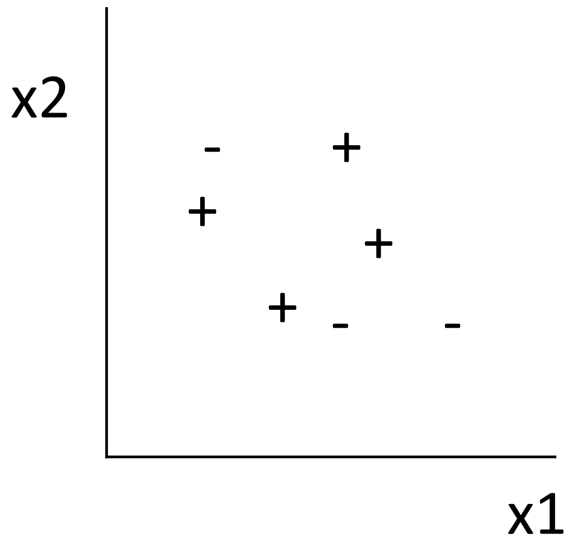
If $\Sigma_{i=1,n} w_i x_i > \theta$ then $y = 1$ otherwise $y = -1$

Perceptron learning

- Learning linear separable functions only



linear separable



Not linear separable

Cover Theorem

- *The probability that classes are linearly separable increases when the features are nonlinearly mapped to a higher dimensional feature space.*
- (Cover, 1965)

Perceptron learning

- Weight updating formula:

$w_{i\text{-new}} = w_{i\text{-old}} + \Delta\Delta w_i$

$\Delta \ \Delta w_i = \eta\alpha(t - y)\, x_i$

where $\eta\alpha$ is learning rate

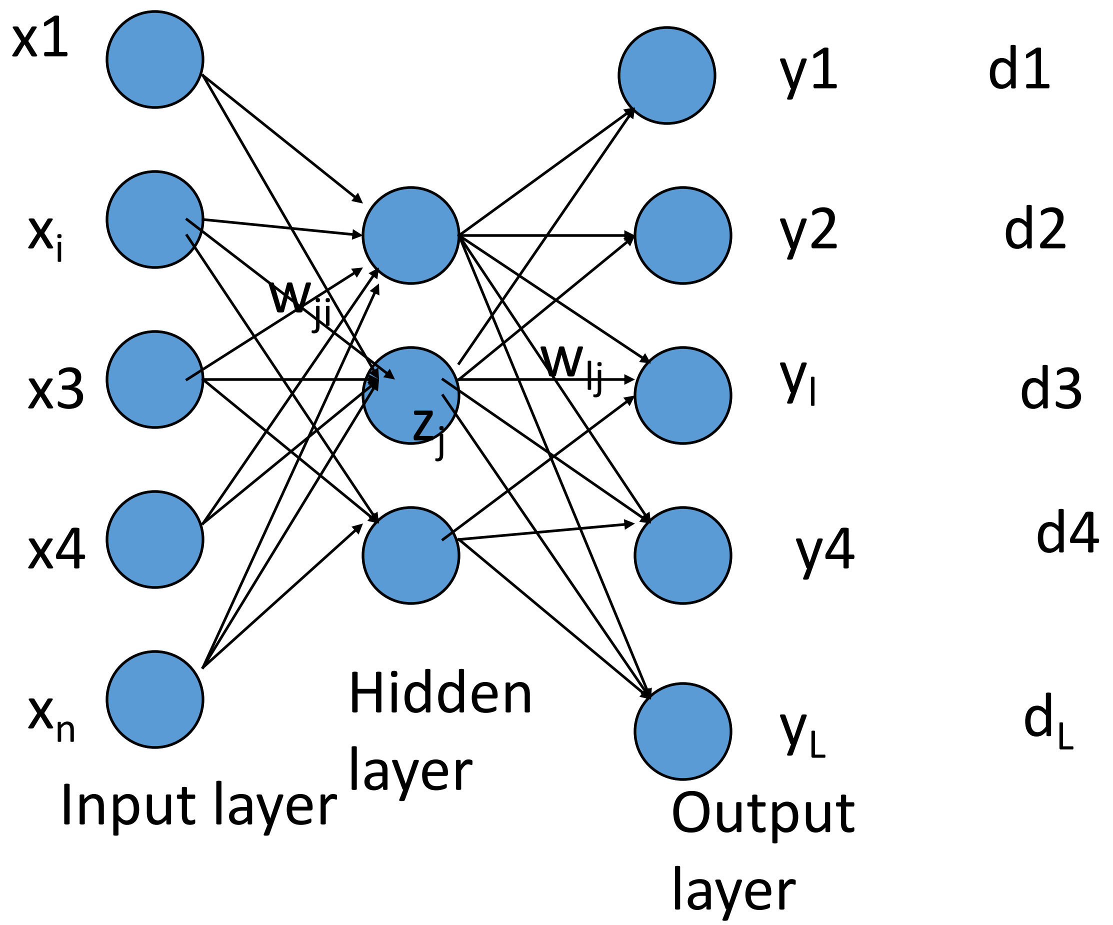
t is the target value

y is predicted value

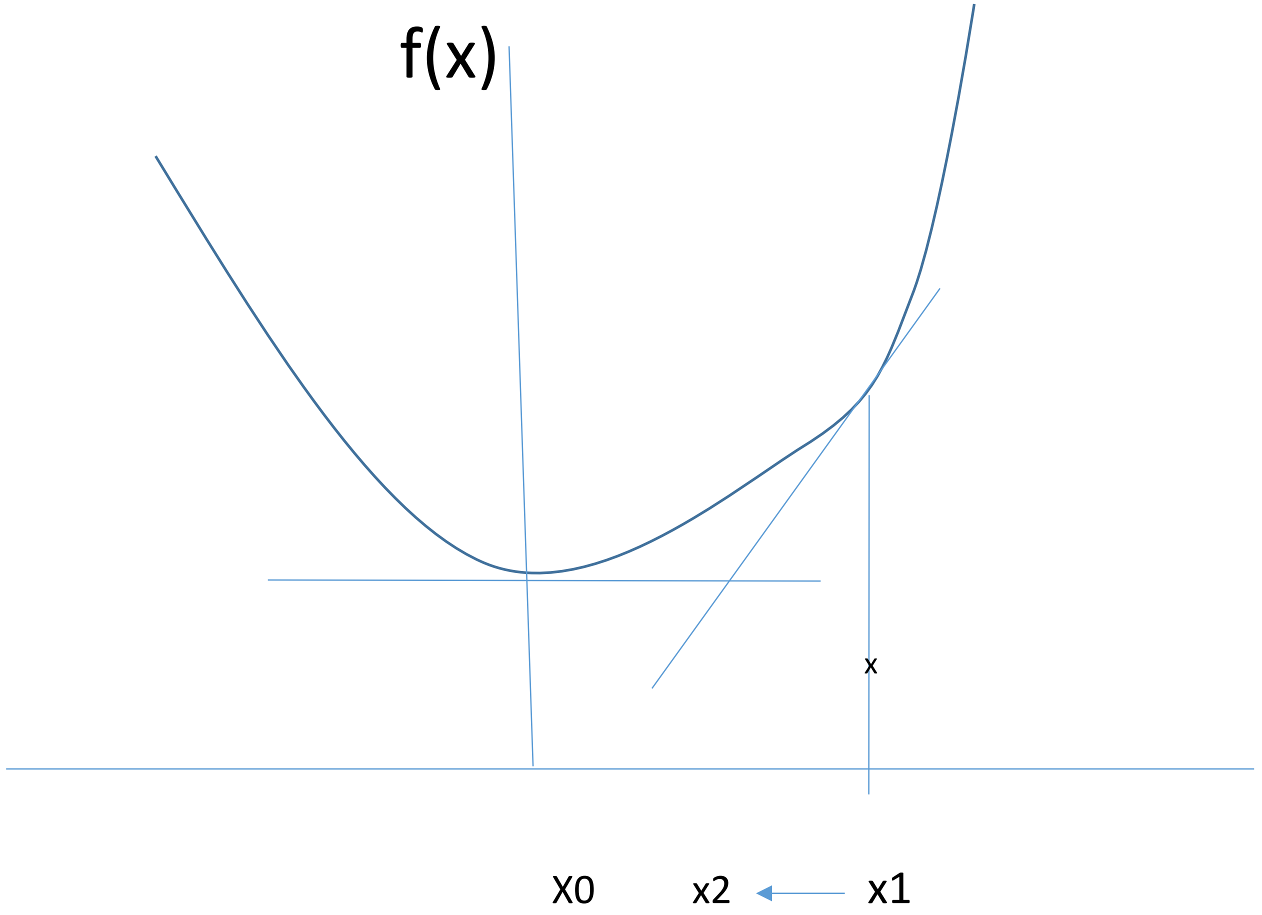
Convergence of perceptron

- If the problem is linear separable, the perceptron can be proved to converge.

Adaptive multilayer feedforward neural networks



Gradient decent to find the optimality of error function



$$\Delta(x) = x2 - x1 = -\eta f'(x1) = -\eta \nabla f(x1)$$

Gradient Descent and the Delta Rule

- $o(\vec{x}) = \vec{w}\,\vec{x}$ % output
- $E(\vec{w}) = 1/2 \sum_{d\in \; \in \; D}(td-od)2\Sigma$% sum square error

$\vec{w} = \vec{w} + \Delta\vec{w}$ % update weight

where $\overrightarrow{\Delta w_i} = -\eta\eta\nabla\, E(\overrightarrow{w_i})$ % gradient descent for node i

$\nabla E(\vec{w}) = \equiv[\partial E/ \partial \overrightarrow{w_0}, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$

$\partial\, \partial E/ \partial w_i = \sum_{d_\in \in D}(td-od)(-xid)\ \Sigma\ \Delta$

$\Delta w_i = \eta \sum_{d_\in \in D}(td-od)(xid)$

Error backpropagation

- Error function: E(w) = $1/2\sum_{l=1,L}$ (dl−yl)2 Σ
- $y_l = f_o(net_l) = f_o(\sum_{l=1,L} w_{lj}z_j\Sigma)$ %activation function

  $z_j = f_h(net_j) = f_h(\sum_{i=1,n} w_{ji}x_i\Sigma)$ %activation function
- delta learning rule:

  $\Delta w_{lj} =- \eta\partial E/ \partial W_{lj} = \eta\ (d_l-y_l)f'_o(net_l)z_j$

  $\Delta\ w_{ji}=-\eta\partial E/ \partial W_{ji}$ (by chain rule)

Chain rule of differentiation

$\Delta\Delta w_{ji} = -\eta\ \partial E/\ \partial W_{ji}$

by chain rule of differentiation:

$\partial E/\ \partial W_{ji} = (\partial E/\ \partial z_j)\ (\partial z_j/\ \partial net_j)\ (\partial net_j/\ \partial W_{ji})$

$\partial\ \partial E/\ \partial z_j = -\sum_{l=1,L} (dl - yl) f'_o(netl) wlj\ \Sigma$

$\partial\ \partial z_j/\ \partial net_j = f_h'(net_j)$

$\partial\ \partial\ net_j/\ \partial W_{ji} = x_i$

$\Delta\Delta w_{ji} = \eta\ [\sum_{l=1,L} \Sigma(dl - yl) f'_o(netl) wlj]\ f_h'(net_j)\ x_i$

$= \rho\ \eta\ f_h'(net_j)(d_j - z_j)x_i$

Error backpropagation learning algorithm (incremental version)

1) Initialize all weights w_{lj}, w_{ji}
2) For each example x calculate the output y by feedforward thru the network
3) Calculate the weight change of hidden-to-output: Δw_{lj}
4) Calculate the weight change of input-to-hidden: Δw_{ji}

Cont'd (incremental learning)

5) update the weight change

$w^{new} = w' + \Delta w$ for all i, l, j

6) Test convergence otherwise go step2

Error backpropagation (batch version)

E(w) = 1/2 $\sum_{k=1,m} \sum_{l=1,L}$ (dl−yl)2 ΣΣ

m is total number of input x's

Incremental is more desired for two reasons:

1) less storage required

2) makes search stochastic (each input is randomly selected)

Finnoff (1993,1994) showed that for very small learning rates, incremental backpropagation approaches batch backpropagation and produces essential the same result

Mini-batch and stochastic gradient

- Sampling a subset from the set of training examples several times
- Compute the gradient based on the sampling results… (A deep learning approach)

Problems of backprop

- Slow convergence
- Might trap to local optimal

Variations of Backprop

- Weight initialization:
  - Backprop is sensitive to initial weight conditions
  - Start with a small and random weighted sum net

Variations of Backprop

- Learning rate:
  - Large learning rate converge initially very fast but will eventually oscillate and thus not reach minimum
  - Small learning rate convergence tend to be very slow
  - Many heuristics of adapting learning rate are proposed

Variations of Backprop

- [Sutton86] learning rate αρshould increase or decrease based on the sign change of

  ∂ $\partial E / \partial W_j$

- [Daken and Moody 91] proposed schedule

  ρ $\alpha(t) = \alpha_o/[1+(t/T)]$

Variations of Backprop

- Momentum (accelerate convergence)

$$w_i(t) = -\rho \partial E / \partial W_i(t) + \alpha\lambda\, w_i(t-1)$$

- Quickprop [Falman 89] - adaptive momentum rates

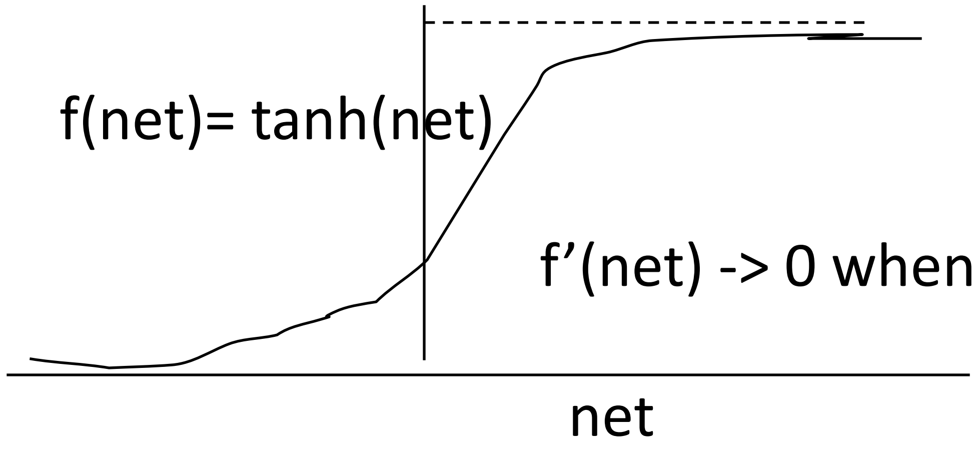
$$\alpha\lambda(t) = \frac{\partial\, \partial E / \partial W_i(t)}{\partial\, \partial E / \partial W_i(t-1) - \partial E / \partial W_i(t)}$$

Variations of Backprop

- If current gradient is smaller than the previous gradient, the sign is positive +> accelerate
- If current gradient is larger than the previous gradient, the sign is negative => deccelerate
- Special handling o the situation where gradients are the same

Variation of Backprop-Activation function

- "flat spot problem" if an unit receives a weighted signal net with a large magnitude, this unit outputs a value close to one of the saturation levels of its activation function
- A "vanishing of gradient" problem.



Variation of backprop

- To overcome this problem, many activation functions are proposed

  $$f_{\alpha}(net) = \alpha \, net + (1- \alpha) \, f(net)$$

- f_{α} (net) forms a homotopy between a linear and a sigmoid function
- Other non-sigmoid activation function may be used as long as they are differentiable.
- The performance varies according to different conditions on data set.

Recurrent neural networks

- Recurrent_neural_network
- **Elman networks vs Jordan networks**
  - **Feedback from hidden vs from output**
- **architecture_picutres**
- **Input layer – hidden layer –output layer**

$$\begin{tikzcd}\text{Input layer} \arrow[r, dash, "–"] & \text{hidden layer} \arrow[loop below] \arrow[r, dash, "–"] & \text{output layer}\end{tikzcd}$$

- **Input layer – hidden layer – output layer**

$$\begin{tikzcd}\text{Input layer} \arrow[r, dash, "–"] & \text{hidden layer} \arrow[r, dash, "–"] & \text{output layer} \arrow[l, bend left]\end{tikzcd}$$

Deep learning neural networks

- Hierarchical: multiple layers
- Recurrent: feedback loop between nodes between layers

- More sophisticated architectures, control gates and connections

Performance on Machine learning -- Ground truth and base line

- Prediction performance must compare with ground truth and base line.
- Obtain ground truth of data can be sometimes difficult.
- Base line performance can be assumed as random guess （no information known) or it can be also assumed to some naïve approach (dummy estimator)
- Or if the training data is imperfect (due to the error rate), what is the best performance bound.
- Even human expert performance to be compared cannot be 100% accurate we are looking into how close the machines can perform as close to human experts.

Cross validation training method and evaluation of performance

- Divide the training set into n subsets.
- Select n-1 subsets as training set and the last one as test set to get performance.
- This can be carried out in n times separately.
- Average the performance (precision; recall; ...).

Training set, validation set, test set

- Decompose your data into three sets. (for example, 8-1-1)
- Training with training set to get the optimal performance (minimize error)
- Use validation set to determine if the performance doe not improve
- If validation set get worse for some iteration, then stop training to avoid overfit.
- Use test set to evaluate the performance of prediction

Data Augmentation

- To augment data size by altering the data so that the labels won’t change.
- For example, in image classification learning, popular augmentations are grayscales, horizontal flips, vertical flips, random crops, color jitters, translations, rotations, and much more.
- By applying just a couple of these transformations to your training data, you can easily double or triple the number of training examples.

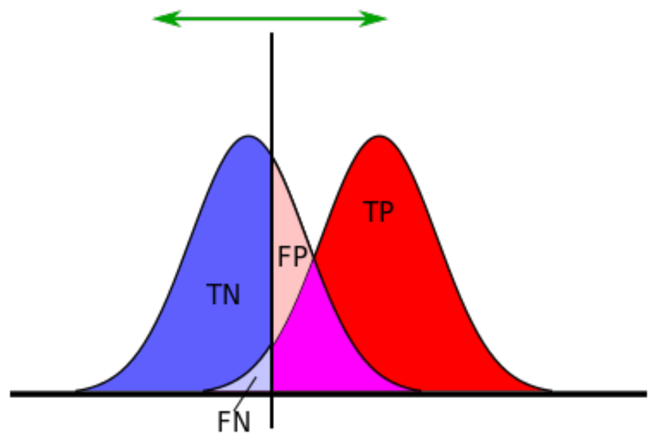
Classification evaluation Metrics

- Evaluation metrics can be some loss, score, and utility functions.
- Classification metrics: Accuracy, precision, recall,
- ROC, AUC curves
- Dummy estimator

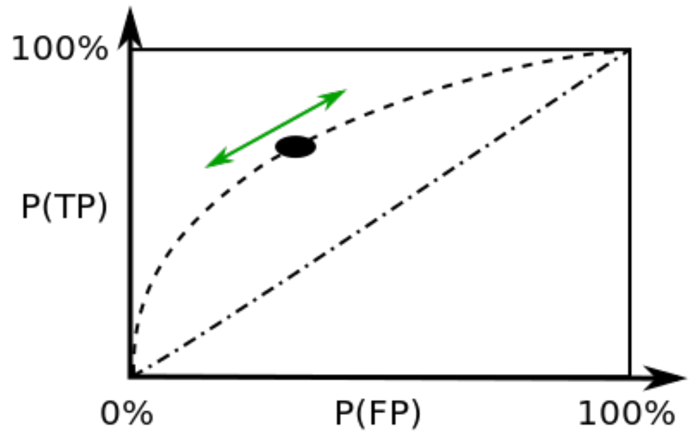
Dummy estimator – base line

- Stratified: generates random predictions by respecting the training set class distribution.
- most_frequent: always predicts the most frequent label in the training set.
- Prior: always predicts the class that maximizes the class prior (like most_frequent`) and ``predict_proba returns the class prior.
- Uniform: generates predictions uniformly at random.
- Constant always predicts a constant label that is provided by the user.

Classification performance depends on the threshold of the classifier



TP	FP
FN	TN



Confusion matrix

```
>>> y_true = [0, 0, 0, 1, 1, 1, 1, 1]
>>> y_pred = [0, 1, 0, 1, 0, 1, 0, 1]
>>> tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
>>> tn, fp, fn, tp (2, 1, 2, 3)
```

	Y_pred = 1	**Y_pred = 0**
y_true = 1	3 tp	2 fn
y_true = 0	1 fp	2 tn

Precision recall and F-score

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

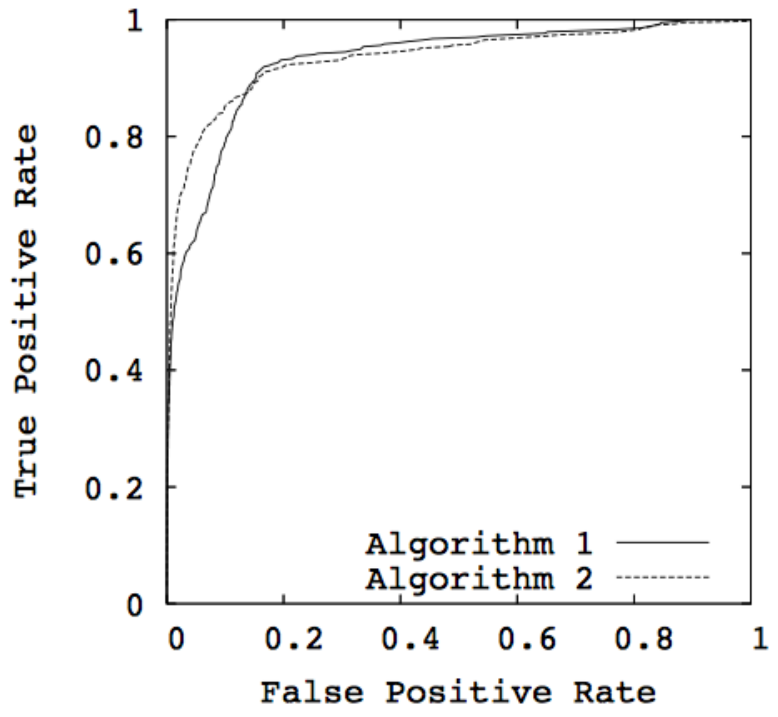
$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}.$$

F1 score

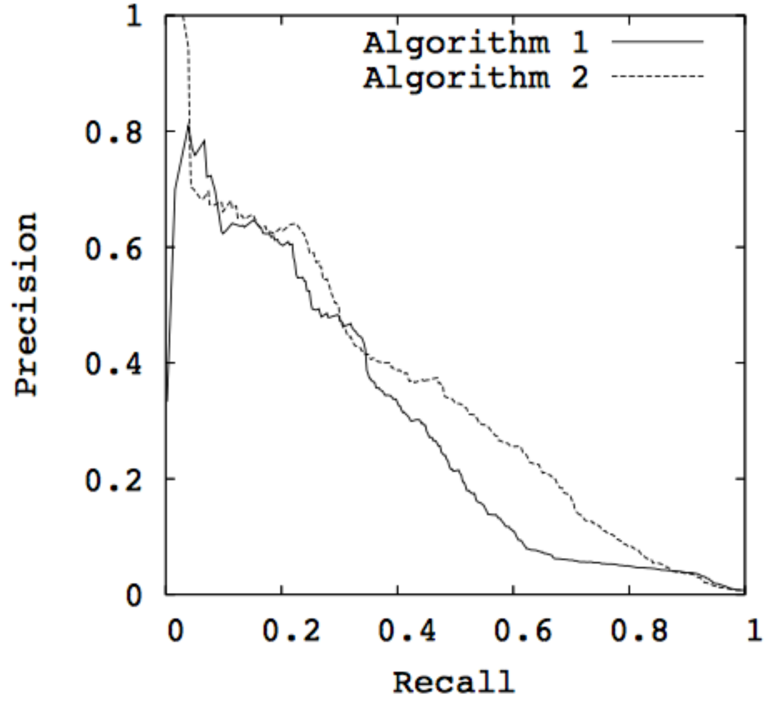
- $\beta = 1$
- $F_\beta = F_1$ is commonly used

- F_1= 2 precision*recall/(recall+ precision)

Precision-recall vs ROC



(a) Comparison in ROC space



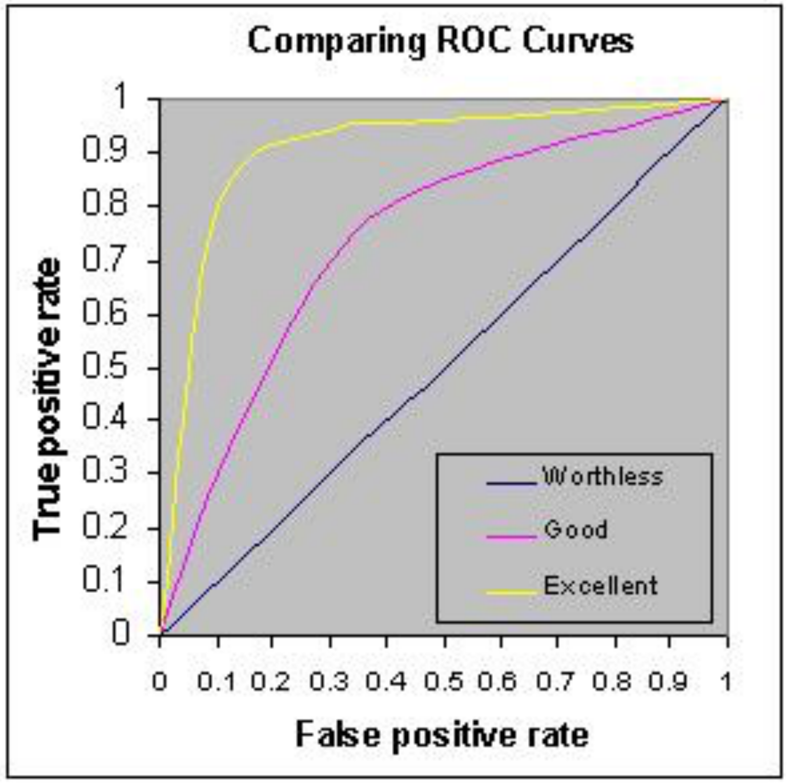
(b) Comparison in PR space

Figure 1. The difference between comparing algorithms in ROC vs PR

AUC: Area under the ROC curve

A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system:

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

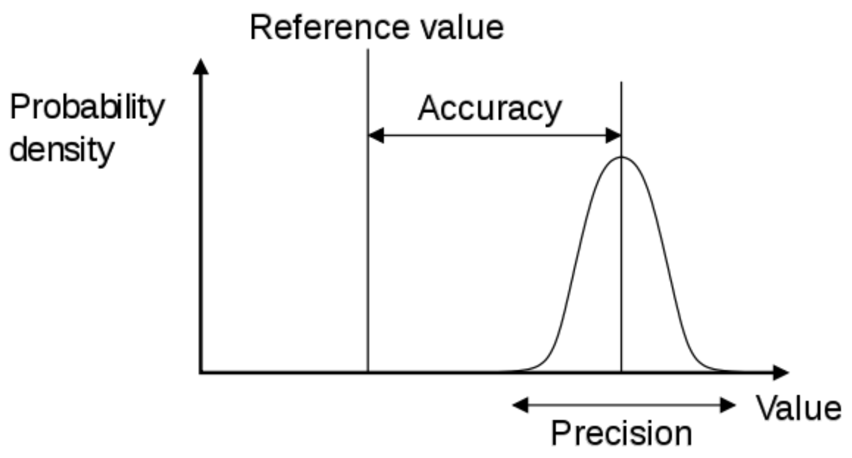


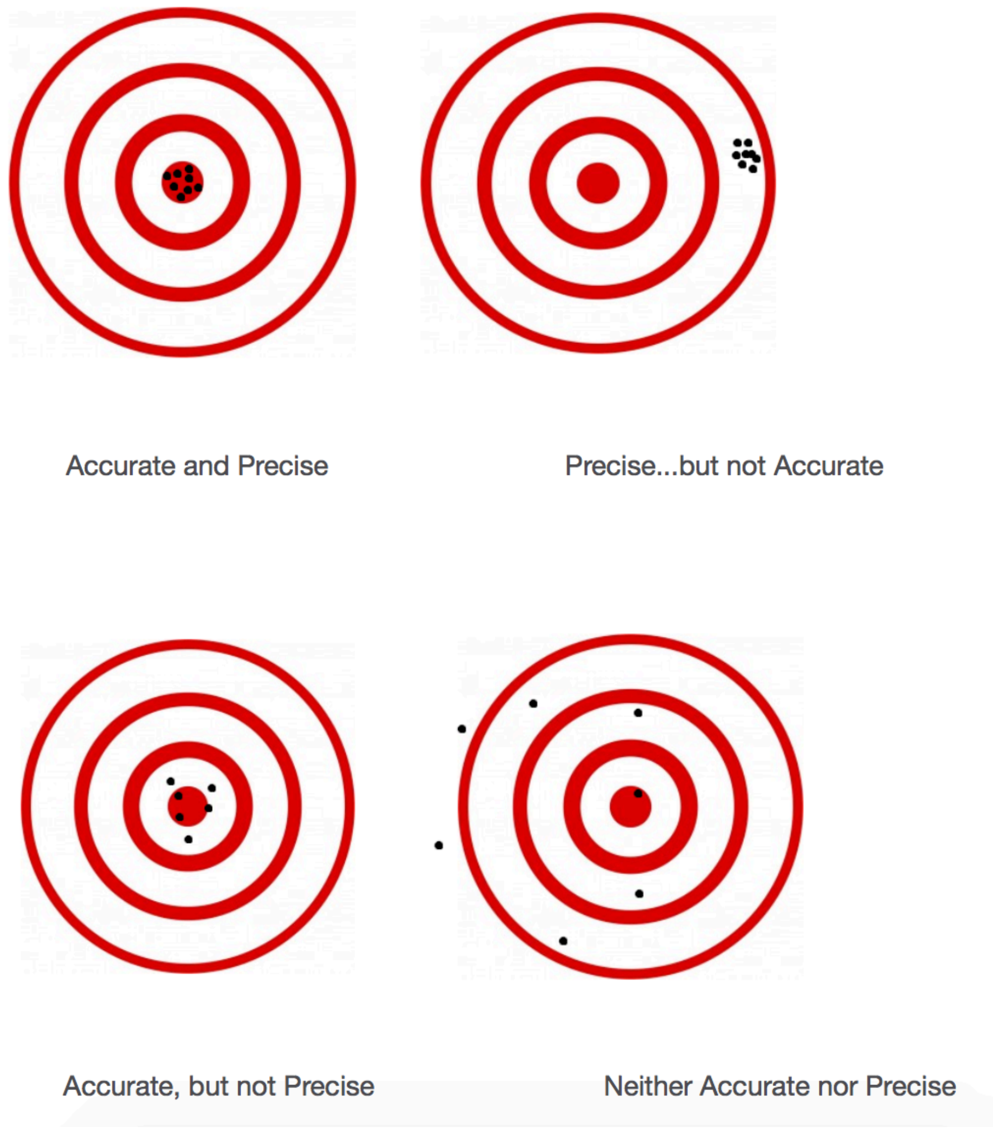
ROC

		True condition				
	Total population	Condition positive	Condition negative	Prevalence= Σ Condition positive/Σ Total population	Accuracy (ACC) = Σ True positive + Σ True negative/Σ Total population	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value(PPV), Precision = Σ True positive/Σ Predicted condition positive	False discovery rate (FDR) = Σ False positive/Σ Predicted condition positive	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = Σ False negative/Σ Predicted condition negative	Negative predictive value (NPV) = Σ True negative/Σ Predicted condition negative	
		True positive rate(TPR), Recall, Sensitivity, probability of detection = Σ True positive/Σ Condition positive	False positive rate(FPR), Fall-out, probability of false alarm = Σ False positive/Σ Condition negative	Positive likelihood ratio (LR+)= TPR/FPR	Diagnostic odds ratio(DOR) = LR+/LR−	F_1 score = 2/1/Recall + 1/Precision
		False negative rate(FNR), Miss rate = Σ False negative/Σ Condition positive	True negative rate(TNR), Specificity (SPC) = Σ True negative/Σ Condition negative	Negative likelihood ratio (LR−)= FNR/TNR		

Sensitivity, accuracy, precision

- Sensitivity = recall
- Precision =\= accuracy
- Precision = tp/(tp+fp)
- Accuracy = (tp+ tn)/(tp+fp+tn+fn)




Accurate and Precise
Precise...but not Accurate
Accurate, but not Precise
Neither Accurate nor Precise

Matthews correlation coefficient

- Apply well especially when the data is imbalance.
- The MCC can be calculated directly from the confusion matrix using the formula:

$$\text{MCC} = \frac{(TP \times TN - FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

- MCC is between -1 and 1 and us suitable for imbalanced data set. (-1 the worst; 1 the best)

An exercise example– too good to be true!!

- Suppose you have a imbalance data: 100 elements, 95 of which are positive elements, and only 5 are negative elements.
- So your dummy algorithm predicts all examples as positive: then you get performance: TP = 95, FP = 5; TN = 0, FN = 0.
- accuracy = (tp+ tn)/(tp+fp+tn+fn) = 95/100= 95%, and
- Precision = tp/(tp + fp) = 95/100 = 95%
- Recall = tp/(tp + fn) = 1
- F1 score = 2 *precision*recall/(precision + recall) = 2*0.95/1.95= 97.44%.
- MCC= $\frac{(TP \times TN - FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
- undefined in this case since denominator has a zero term TN+FN=0 (TN = 0, FN = 0).

Another example

- Using the same data set, the classifier predicts the confusion matrix as: TP = 90, FP = 5; TN = 1, FN = 4.
- accuracy = (tp+ tn)/(tp+fp+tn+fn) = 91/100= 91%,
- Precision = tp/(tp + fp) = 90/95= 0.947
- Recall= and tp/(tp + fn) =90/94= 0.957
- F1 score = 0.952
- If we check the MCC = $\frac{(TP \times TN - FP \times FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$ = 0.14

Micro vs macro metrics

- **Micro-average Method**

  In Micro-average method, you sum up the individual true positives, false positives, and false negatives of the system for different sets and the apply them to get the statistics.

- For example, for a set of data, the system's

  True positive (TP1) = 12
  False positive (FP1) =9
  False negative (FN1) =3

  Then precision (P1) and recall (R1) will be 57.14 and 80

  and for a different set of data, the system's

  True positive (TP2)= 50
  False positive (FP2)=23
  False negative (FN2)=9

  Then precision (P2) and recall (R2) will be 68.49 and 84.75

  Now, the average precision and recall of the system using the Micro-average method is

  Micro-average of precision = (TP1+TP2)/(TP1+TP2+FP1+FP2) = (12+50)/(12+50+9+23) = 65.96
  Micro-average of recall = (TP1+TP2)/(TP1+TP2+FN1+FN2) = (12+50)/(12+50+3+9) = 83.78

  The Micro-average F-Score will be simply the harmonic mean of these two figures.

Macro-average Method

The method is straightforward. Just take the average of the precision and recall of the system on different sets. For example, the macro-average precision and recall of the system for the given example is

Macro-average precision = (P1+P2)/2 = (57.14+68.49)/2 = 62.82
Macro-average recall = (R1+R2)/2 = (80+84.75)/2 = 82.25

The Macro-average F-Score will be simply the harmonic mean of these two figures.

Micro and macro

- Macro-average method can be used when you want to know how the system performs overall across the sets of data. You should not come up with any specific decision with this average.

  On the other hand, micro-average can be a useful measure when your dataset varies in size.

Error functions:

http://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error

- Mean absolute error

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \left| y_i - \hat{y}_i \right| .$$

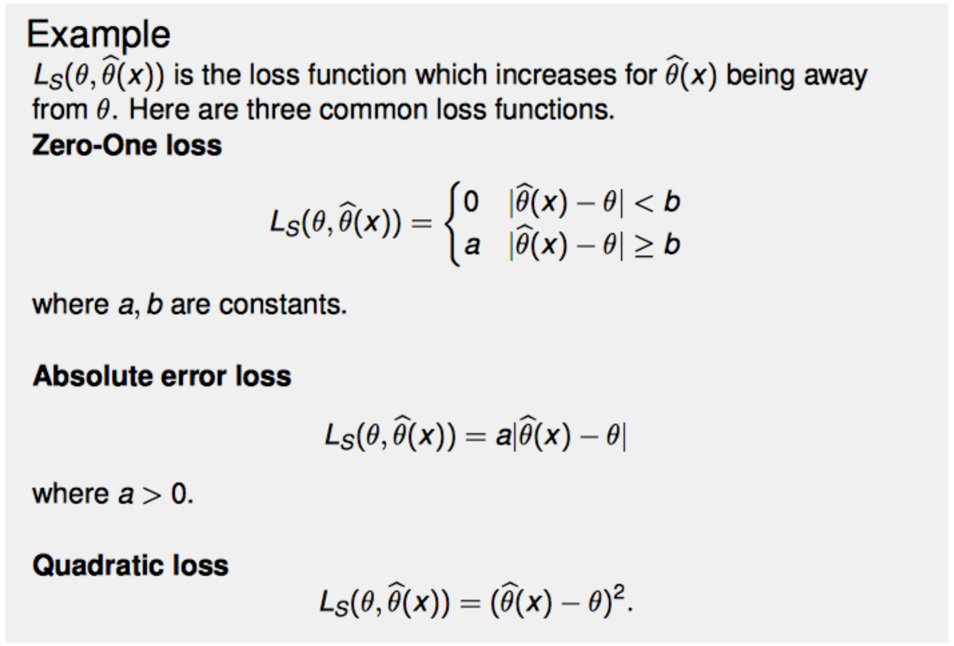
- Mean squared error

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

- Mean squared logarithm error

$$\text{MSLE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (\log_e (1 + y_i) - \log_e (1 + \hat{y}_i))^2.$$

Loss functions



Example

$L_S(\theta, \widehat{\theta}(x))$ is the loss function which increases for $\widehat{\theta}(x)$ being away from θ. Here are three common loss functions.

Zero-One loss

$$L_S(\theta, \widehat{\theta}(x)) = \begin{cases} 0 & |\widehat{\theta}(x) - \theta| < b \\ a & |\widehat{\theta}(x) - \theta| \geq b \end{cases}$$

where a, b are constants.

Absolute error loss

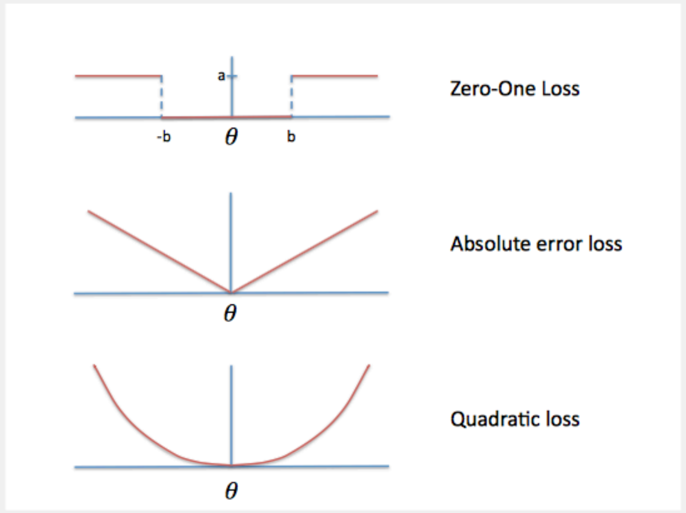
$$L_S(\theta, \widehat{\theta}(x)) = a|\widehat{\theta}(x) - \theta|$$

where $a > 0$.

Quadratic loss

$$L_S(\theta, \widehat{\theta}(x)) = (\widehat{\theta}(x) - \theta)^2.$$

Example



Hamming loss

The computes the average Hamming loss or Hamming distance between
Two sets of samples.

If $\hat{y}_j$ is the predicted value for the j -th label of a given sample, y_j is the corresponding true value, and n_labels is the number of classes or labels, then the Hamming loss $L_{Hamming}$ between two samples is defined as:

$$L_{Hamming}(y, \hat{y}) = \frac{1}{n_\text{labels}} \sum_{j=0}^{n_\text{labels} - 1} 1(\hat{y}_j \neq y_j)$$

Huber loss

- L(y,f(x)) = ½(y- f(x))^2 if |y-f(x)|$\leq$ δ
- = δ|y-f(x))|- 1/2 δ^2 otherwise

Jaccard similarity coefficient

The function computes the average (default) or sum of Jaccard similarity coefficients, also called the Jaccard index, between pairs of label sets.
The Jaccard similarity coefficient of the y_i-th samples, with a ground truth label set and predicted label set $\hat{y}_i$, is defined as

$$J(y_i, \hat{y}_i) = \frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|}.$$

In binary and multiclass classification, the Jaccard similarity coefficient

score is equal to the classification accuracy.

Brier score loss

- The Brier score is a proper score function that measures the accuracy of probabilistic predictions. It is applicable to tasks in which predictions must assign probabilities to a set of mutually exclusive discrete outcomes.
- The actual output is between 0 and 1 and the predicted value f_t is between 0 and 1.

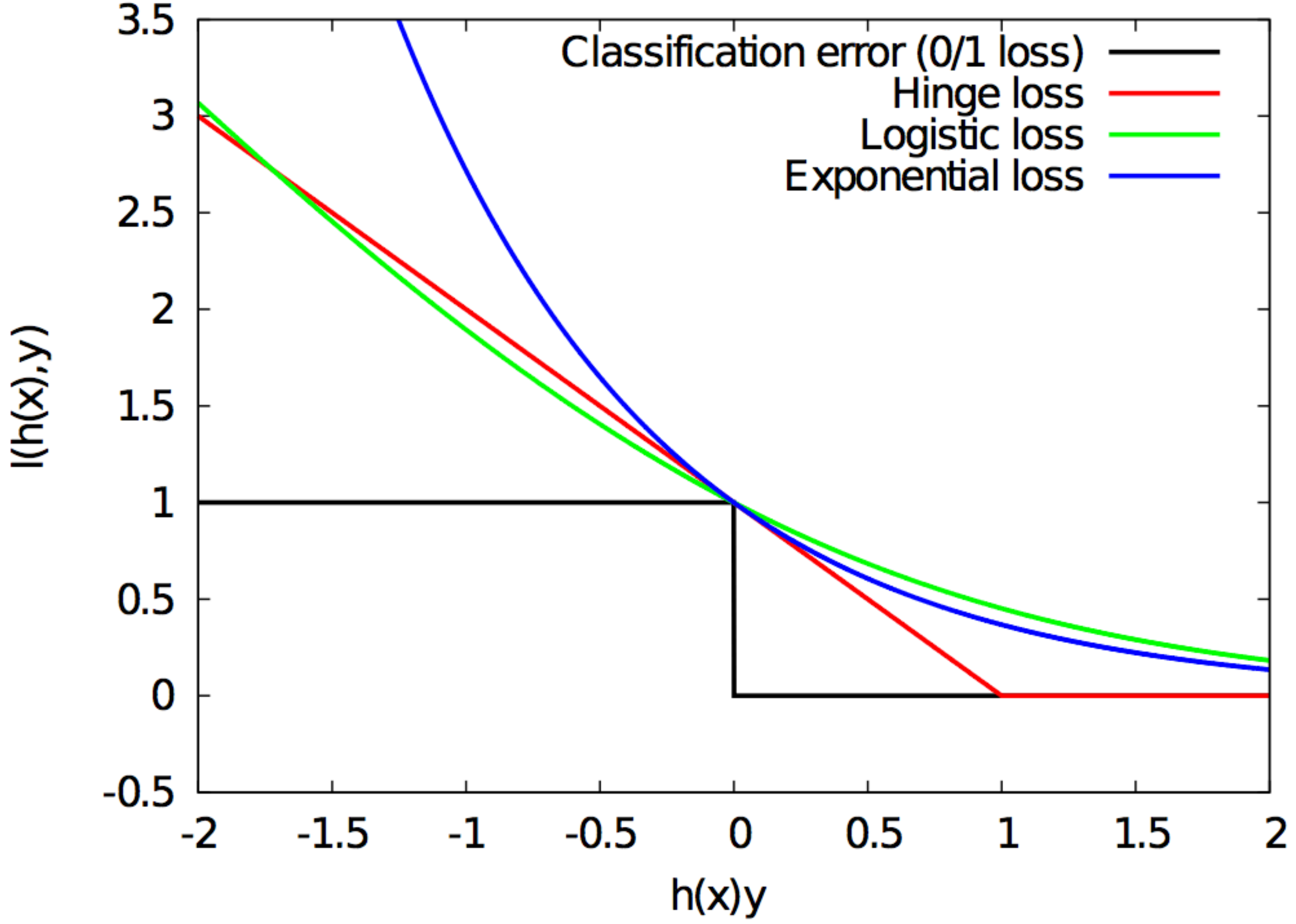
$$BS = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2$$

- The lower BS, the better the result.

Hinge loss

- Hinge loss is used in maximal margin classifiers such as support vector machines.
- If the labels are encoded with +1 and -1, y is the true value, and w is the predicted decisions as output by decision function , then the hinge loss is defined as:

$$L_{\text{Hinge}}(y, w) = \max\{1 - wy, 0\} = |1 - wy|_+$$


Classification error (0/1 loss)
Hinge loss
Logistic loss
Exponential loss
l(h(x),y)
h(x)y
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Logistic loss

- The logistic loss function is defined as

$$V(f(x),y) = (1/\ln 2) \ln(1+e^{-yf(x)})$$

- **This function displays a similar convergence rate to the hinge loss function, and since it is continuous,** gradient descent methods can be utilized.
- However, the logistic loss function does not assign zero penalty to any points.
- Instead, functions that correctly classify points with high confidence (i.e., with high values of $|f(x)|$) are penalized less.
- This structure leads the logistic loss function to be sensitive to outliers in the data.

Log loss

- logistic regression loss or cross-entropy loss, is defined on probability estimates.
- commonly used in (multinomial) logistic regression and neural networks, as well as in some variants of expectation-maximization, and can be used to evaluate the probability outputs of a classifier instead of its discrete predictions.

- For binary classification with a true label $y \in \{0,1\}$ and a probability estimate $p = \Pr(y = 1)$, the log loss per sample is the negative log-likelihood of the classifier given the true label:

$$L_{\log}(y, p) = -\log \Pr(y|p) = -(y \log(p) + (1 - y) \log(1 - p))$$

Cross entropy loss

- The cross entropy loss is ubiquitous in modern deep neural networks.
- Using the alternative label convention $t = (1+ y)/2$, so that $t \in \{0,1\}$, the cross entropy loss is defined as

$$V(f(x), t) = -t \ln(f(x)) - (1-t) \ln(1- f(x))$$

Note: [$-p^{+}\log p^{+} - p^{-}\log p^{-}$; $y= -1$ or 1]

- **The cross entropy loss is closely related to the** Kullback-Leibler divergence between the empirical distribution and the predicted distribution.

- This function is not naturally represented as a product of the true label and the predicted value, but is convex and can be minimized using stochastic gradient descent methods.

Which loss function to use?

- Depend on real domain applications;
- The quadratic cost function is ideal for linear regression and for any system where the hypothesis (or activation function) is a linear function.
- For logistic regression and neural networks with sigmoid activation functions, however, the cross entropy cost is better.