

INFO411 Lecture - Classification II

Jeremiah Deng

InfoSci, University of Otago

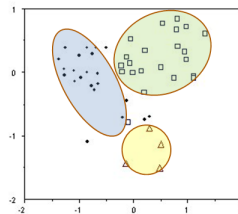
August 22, 2017

- 1 Review
 - Parametric approaches
 - Non-parametric approaches
- 2 Neural Networks
- 3 Support Vector Machine
- 4 Kernel Tricks
- 5 Decision Trees

Bayesian Classifiers

- Use a *probabilistic* discriminant function $g_i(\mathbf{x})$
- Usually we can have
 - ▶ $g_i(\mathbf{x}) = P(\omega_i|\mathbf{x})$, or
 - ▶ $g_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i)$.
- For $p(\mathbf{x}|\omega_i)$ of Gaussian (normal) density:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(\omega_i)$$
- Mixture of Gaussians: apply maximum-likelihood estimation of the Gaussian distribution for each class/each “mode”
- Conduct classification by assigning class labels according to the biggest discriminant function value

K-Nearest Neighbour (k -NN) Estimation

- Non-parametric: no need to estimate probability density ☺
- We have shown that k -NN classifier approximates the Bayes Classifier.
- Suppose we now place a cell of volume V around $\mathbf{x}$
- There are n labelled samples, of c classes
- The cell captures k samples, of which k_i belong to class i
- Then the estimate for the joint probability is

$$p(\mathbf{x}, \omega_i) = \frac{k_i/n}{V}.$$

- And a reasonable estimate for *a posteriori* probability is

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{\sum_{j=1}^c p(\mathbf{x}, \omega_j)} = \frac{k_i}{k}$$

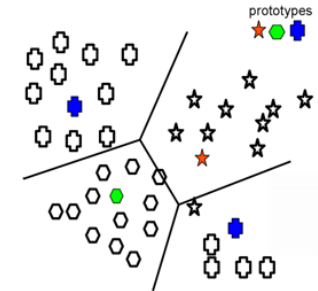
i.e., k -NN is an approximation to the Bayes Classifier.

Extending the clustering algorithms

- K-means, SOM etc. can all be augmented with a supervisory layer that tries to match to the class label of the training samples.
- For instance:
 - ▶ Linking prototypes with class labels;
 - ▶ Adopting supervised learning; or
 - ▶ Adding another layer and taking gradient descent on the weights.

Learning Vector Quantization (LVQ)

- Kohonen (1989)
- Similar to the idea of competitive learning and vector quantization
- However, since prototypes are labelled, we can make use of the supervisory learning



Algorithm

- 1 Initialize the prototype set $D^n = \{\mathbf{m}_1, \mathbf{m}_2, \dots, \mathbf{m}_n\}$.
- 2 Given an input pattern $\mathbf{x}$, find the best matching prototype $\mathbf{m}_c$.
- 3 Update the winning prototype
 - ▶ $\Delta \mathbf{m}_c = \gamma(t)(\mathbf{x} - \mathbf{m}_c)$ if $\text{Class}(\mathbf{x}) = \text{Class}(\mathbf{m}_c)$;
 - ▶ $\Delta \mathbf{m}_c = -\gamma(t)(\mathbf{x} - \mathbf{m}_c)$, otherwise.
- 4 Go back to Step 2 until convergence.

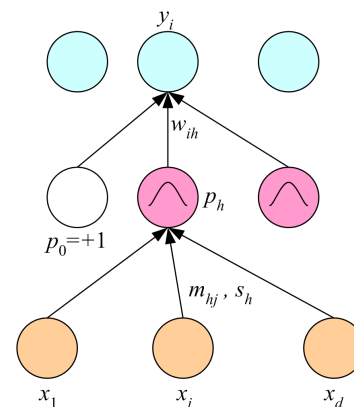
Radial Basis Functions

- Cluster data points around a few prototypes of Gaussian distribution
- $$p_h^t = \exp \left[-\frac{\|\mathbf{x}^t - \mathbf{m}_h\|^2}{2s_h^2} \right]$$
- Output is generated with a layer of “perceptron”

$$y^t = \sum_{h=1}^H w_h p_h^t + w_0$$

- Use linear regression on the membership to clusters to predict the desired output r_i^t

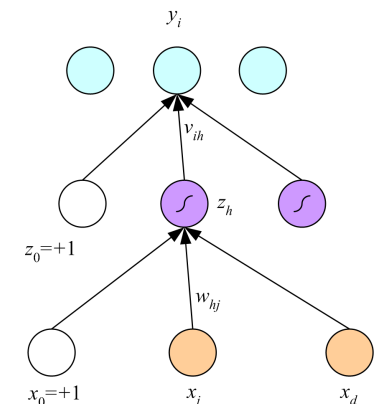
$$\Delta w_{ih} = \eta \sum_t (r_i^t - y_i^t) p_h^t$$



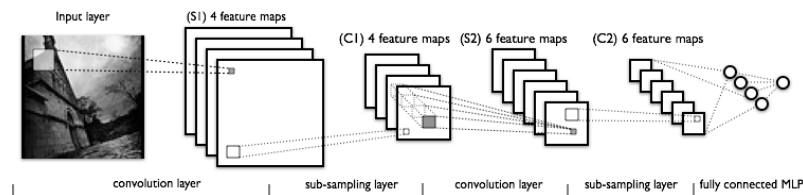
Multi-Layer Perceptron and Back Propagation

- Hidden layer employing “sigmoid” function to its weighted sum:
- $$z_h = \frac{1}{1 + \exp[-(\sum_{j=1}^d w_{hj} x_j + w_{h0})]}$$
- Output $y_i = \sum_{h=1}^H v_{ih} z_h + v_{i0}$
 - Least-squares rule to update the 2nd layer weights: $\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t$
 - Back-Propagation rule for hidden layer:

$$\begin{aligned} \Delta w_{hj} &= -\eta \frac{\delta E}{\delta w_{hj}} \\ &= -\eta \sum_t \frac{\delta E}{\delta y^t} \frac{\delta y^t}{\delta z_h^t} \frac{\delta z_h^t}{\delta w_{hj}} \\ &= \eta \sum_t (r^t - y^t) v_{ih} z_h^t (1 - z_h^t) x_j^t \end{aligned}$$



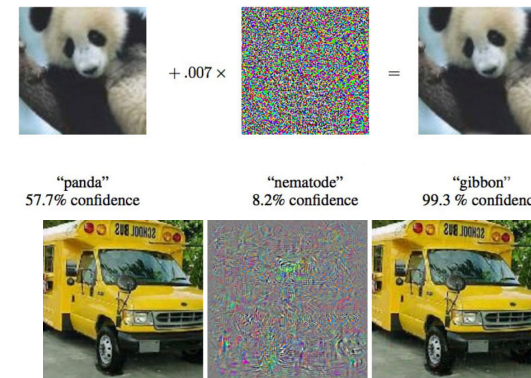
A Shallow View on Deep Learning



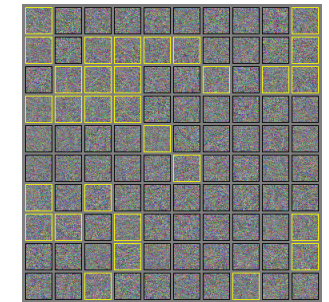
- Convolutional neural networks (CNN) are MLP-extensions inspired by biological vision.
- Dated back to Fukushima (1980), LeCun & Bengio (1998?) ...
- Multiple layers of nonlinear mechanisms: convolution, subsampling, sparsity
- Outcome: so far the best object recogniser

You see monkeys?

- CNNs remain largely *unexplainable*.
- Nguyen et al. (2014), Szegedy et al. (2014), Goodfellow et al. (2015)



Or ostrich?



The Separation Hyperplane

Consider a two-class problem, with sample $X = \{\mathbf{x}, r\}$, where $r = 1$ when $\mathbf{x} \in C_1$, and $r = -1$ if $\mathbf{x} \in C_2$.

We would like to find a hyperplane $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ such that

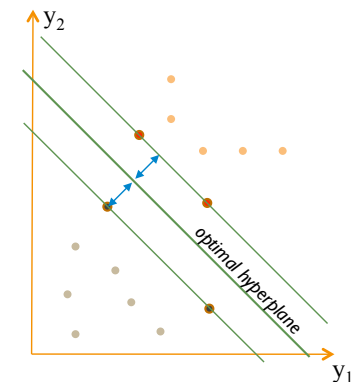
$$g(\mathbf{x}) \geq +1, \text{ for } r = +1$$

$$g(\mathbf{x}) \leq -1, \text{ for } r = -1$$

Focusing on the margins

Note there exists a margin on either side of the hyperplane, of which we wish to maximize for better generalization. The above two lines are equivalent to $rg(\mathbf{x}) = r(\mathbf{w}^T \mathbf{x} + w_0) \geq 1$. To maximize the margin, we also need to add the constraint of minimizing $\|\mathbf{w}\|$. So our optimisation goal is

$$\min \frac{1}{2} \|\mathbf{w}\|^2 \text{ subject to } r(\mathbf{w}^T \mathbf{x} + w_0) \geq 1, \forall \mathbf{x}$$



Solving SVM

Using Lagrange multipliers α_t , the optimisation criterion becomes

$$L = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{t=1}^N \alpha_t [r_t(\mathbf{w}^T \mathbf{x}_t + w_0) - 1],$$

where $\mathbf{x}_t$ and r_t denote the t -th data point and its class label respectively.

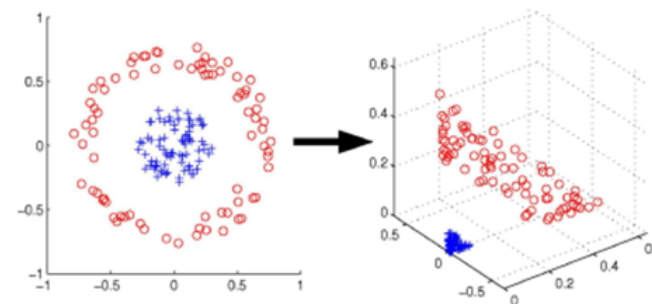
By solving this quadratic optimization problem, it is found that most α_t are zero. Those $\mathbf{x}_t$ with $\alpha_t > 0$ satisfy

$$r_t(\mathbf{w}^T \mathbf{x}_t + w_0) = 1$$

lie on the margin. These are called 'support vectors'. One can easily worked out $w_0 = r_t - \mathbf{w}^T \mathbf{x}_t$ using these support vectors.

Challenge: Linear non-separability

- Not all classification problems can be solved using linear discriminant functions.
- Solutions:
 - ▶ Use multiple linear discriminant functions (divide-and-conquer)
 - ▶ Conduct nonlinear transformation using kernel methods, projecting data into higher dimensional kernel space with linear separability



SVM as a kernel machine

- First, use a nonlinear function to map the input data to a high-dimensional space.

$$y_k = \Phi(\mathbf{x}_k)$$

- Kernel function $\Phi(\cdot)$ e.g. can be polynomials or Gaussians
- Defines a linear discriminant function in the augmented y space, hence the problem becomes more linearly separable.
- Note because of the kernel trick $\Phi(\mathbf{x}_t)^T \Phi(\mathbf{x}_s) = K(\mathbf{x}_t, \mathbf{x}_s)$, we don't need to conduct dot product in the high-dimension space.

Kernel Trick: Example

Suppose we have 2-D data points and we project it to 3-D using $\Phi(x)$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad \Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

We can easily verify that

$$\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2$$

i.e., dot product can be done in 2-D instead of 3-D.

Popular kernel functions

- Polynomials of degree q : $K(\mathbf{x}_t, \mathbf{x}) = (\mathbf{x}^T \mathbf{x}_t + 1)^q$
- Radial-basis functions define a spherical kernel as in Parzen windows with a radius s :

$$K(\mathbf{x}_t, \mathbf{x}) = \exp\left[-\frac{\|\mathbf{x}_t - \mathbf{x}\|^2}{2s^2}\right]$$

This can be extended to other definition of distance D between $\mathbf{x}$ and $\mathbf{x}_t$ (e.g., the geodesic distance used in Isomap):

$$K(\mathbf{x}_t, \mathbf{x}) = \exp\left[-\frac{\|D(\mathbf{x}_t, \mathbf{x})\|^2}{2s^2}\right]$$

- Sigmoidal functions: $K(\mathbf{x}_t, \mathbf{x}) = \tanh(2\mathbf{x}^T \mathbf{x}_t + 1)$
- ☺ Design your own?

Complications

- Often two classes are not perfectly linearly separable, and we have a soft margin instead:

$$r_t(\mathbf{w}^T \mathbf{x}_t + w_0) \geq 1 - \xi_t, \xi_t \geq 0$$

and we penalise the ξ_t terms by modifying the optimisation function, with $C > 0$ controlling the extent of penalty:

$$\min\left[\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{t=1}^N \xi_t\right]$$

- What about multi-class classification? Suppose k classes.
 - ▶ one-vs-one: build $k \times (k - 1)/2$ pair-wise classifiers; the class with the highest vote wins;
 - ▶ one-vs-rest: build k binary classifiers; the one with the highest score wins. Faster, and almost as good.

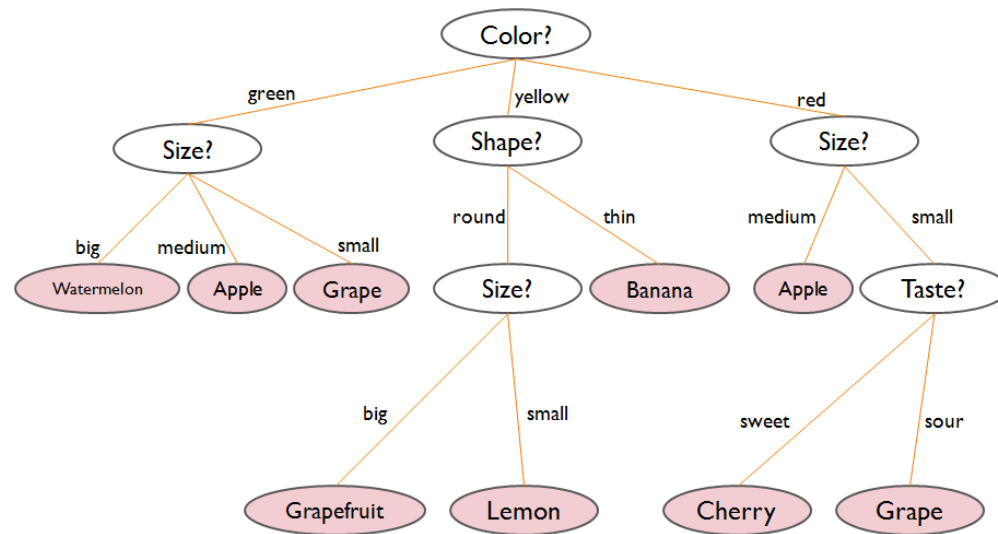
Non-metric data classification

- How can we handle non-numeric data?
- There are no distance measures, not even similarity definitions!

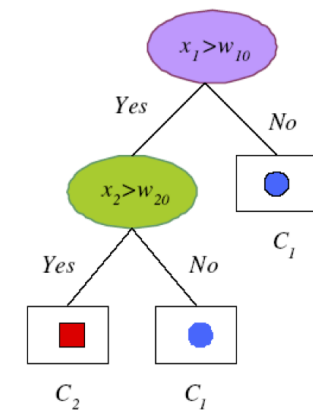
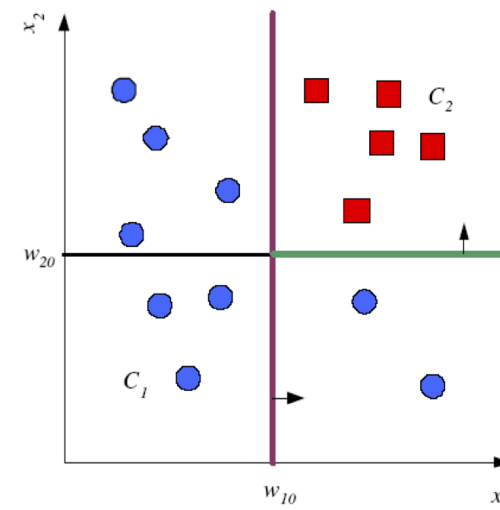
Decision Trees

- Similar to the idea of the “20-questions” game
- Useful for nonmetric data – all questions can be asked in a ‘yes/no’ or ‘true/false’ or ‘which one’ (from a finite set of values)
- The sequence of these questions (and answers) forms a decision tree, starting from the first question (root node), until it reaches the terminal or leaf nodes.
- Classification of a particular pattern begins at the root node and follows the appropriate link to a subsequent or descendent node.
- Famous algorithms: CART, C4.5 etc.
 - ▶ Can handle numeric data as well!

A 'fruits' decision tree



Tree Uses Nodes, and Leaves



Where to make the cut: minimize impurity

A bit of DIY

- Consider a node m
- Suppose it contains N_m training samples
- Of these, N_m^i instances belong to Class C_i
- Prob. for Class C_i is

$$p_m^i = \frac{N_m^i}{N_m}$$

- Measure the 'impurity' of node m using entropy:

$$H_m = - \sum_i p_m^i \log_2 p_m^i$$

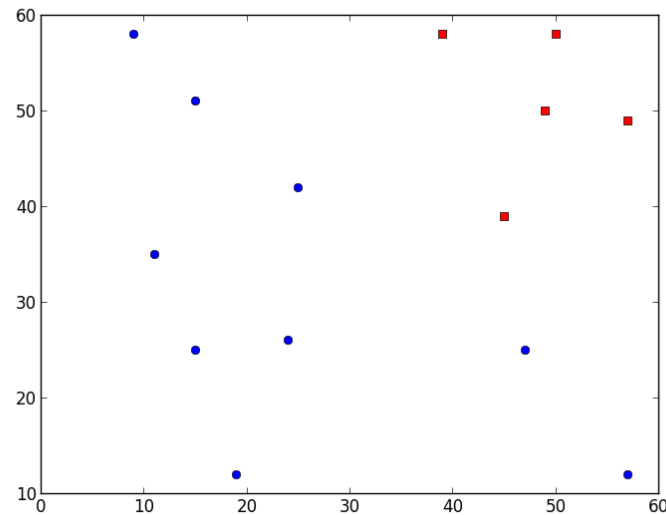
- For a 2-class problem
 - Entropy: $H = -p \log_2 p - (1-p) \log_2 (1-p)$
 - Or, using the Gini index: $2p(1-p)$

demo: finding where to make the 1st cut

```
data=np.array([[ 9.0, 58.0],
               [15, 51],
               [25, 42],
               [11, 35],
               [15, 25],
               [24, 26],
               [19, 12],
               [47, 25],
               [57, 12],
               [39, 58],
               [45, 39],
               [49, 50],
               [50, 58],
               [57, 49]])
```

```
cla=np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2])
```

```
data=np.hstack((data, cla.reshape((-1,1))))
```



Try Out the Cuts

```

dim=0
d=np.unique(data[:,dim])
cuts=[]
for i in range(len(d)-1):
    cuts.append((d[i]+d[i+1])/2) # cut btw two values

def impurity(a):
    labels=a[:, -1]
    p=(np.sum(labels==1)+0.)/len(a)
    if min(p,1-p)>1e-10:
        impu=-p*np.log2(p)-(1-p)*np.log2(1-p)
    else:
        impu=0.0
    return impu

print 'Imp_orig.', impurity(data)
for th in cuts:
    print 'Cut@', th, 'Imp:',
    left=data[:,dim]<=th
    right=data[:,dim]>th
    p1=np.sum(data[left]==1)/(len(data)+0.)
    print impurity(data[left])*p1+impurity(data[right])*(1-p1)

```

Impurity from cuts

Original impurity: 0.940285958671

Outcome for Dimension #0

Cut@	10.0	Impu	0.892576847243
Cut@	13.0	Impu	0.839887505701
Cut@	17.0	Impu	0.714285714286
Cut@	21.5	Impu	0.637120324182
Cut@	24.5	Impu	0.545390858814
Cut@	32.0	Impu	0.431560284283
Cut@	42.0	Impu	0.730930138627
Cut@	46.0	Impu	0.867577550482
Cut@	48.0	Impu	0.760220964704
Cut@	49.5	Impu	0.876613035522
Cut@	53.5	Impu	0.953311905174

Outcome for Dimension #1

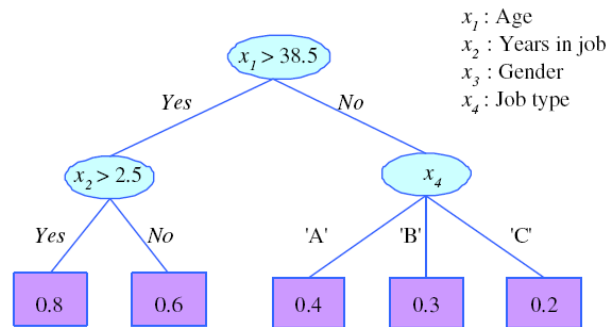
Cut@	18.5	Impu	0.839887505701
Cut@	25.5	Impu	0.714285714286
Cut@	30.5	Impu	0.637120324182
Cut@	37.0	Impu	0.545390858814
Cut@	40.5	Impu	0.816561554269
Cut@	45.5	Impu	0.730930138627
Cut@	49.5	Impu	0.867577550482
Cut@	50.5	Impu	0.940645449615
Cut@	54.5	Impu	0.876613035522

Divide and Conquer

- Internal decision nodes
 - ▶ Univariate: Uses a single attribute, x_i
 - Numeric x_i : Binary split at w_m that reduces impurity the most
 - Discrete x_i : n -way split for n possible values
 - ▶ Multivariate: Uses all attributes, $\mathbf{x}$
- Leaves
 - ▶ Classification: Class labels, or proportions
 - ▶ Regression: Numeric; r average, or local fit
- Learning is greedy; find the best split recursively (Breiman et al, 1984; Quinlan, 1986, 1993)
- When to stop: use cross-validation, or thresholding on impurity reduction

Rule Extraction from Trees

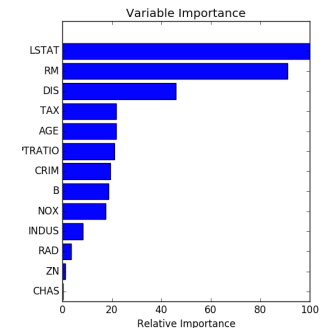
C4.5 rules (Quinlan 1993)



- R1: IF (age > 38.5) AND (years-in-job > 2.5) THEN $y = 0.8$
 R2: IF (age > 38.5) AND (years-in-job ≤ 2.5) THEN $y = 0.6$
 R3: IF (age ≤ 38.5) AND (job-type = 'A') THEN $y = 0.4$
 R4: IF (age ≤ 38.5) AND (job-type = 'B') THEN $y = 0.3$
 R5: IF (age ≤ 38.5) AND (job-type = 'C') THEN $y = 0.2$

Interpretation of trees

- Every time a split of a node is made on a variable results in reduction of the Gini impurity.
- Adding up the impurity reduction reveals the importance of variables.
- However, a single tree tends to be highly sensitive to noises in data ☹
- In practice, **random forests** are preferred for their better generalisation ability.
- Adding up the Gini decreases for each individual variable over all trees in the forest gives the importance measure.
- Variable importance can also be found through permutation.



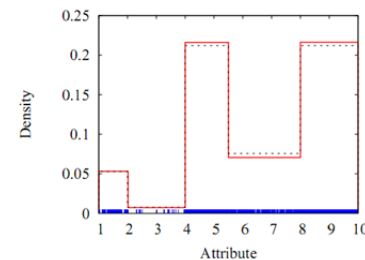
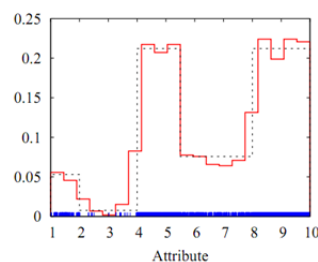
TUBE: Using Decision-Tree for Histograms

Recap

- Tree-based unsupervised bin estimator (Schmidberger et al., 2005)
- Where to cut: maximize the likelihood based on the training data

$$L = n_l \log \frac{n_l}{w_l N} + n_r \log \frac{n_r}{w_r N}$$

- 10-fold cross validation used to decide number of cuts
- Compare Equal-Width with TUBE (dotted line is true density):



- Other classification methods: MLP, LVQ, RBF ...
 - ▶ Alpaydin Ch.11, "Multilayer Perceptrons"
 - ▶ Alpaydin Ch.12 "Local Models"
- SVM and kernel method (Alpaydin Ch.13)
- Decision Trees (Ch. 9)
- Hands-on: Lab 5 (after the break)