SVM, Bayesian Networks, and Ensemble Methods

SVM: Support Vector Machines

- Uses a nonlinear mapping to transform the original training data into a higher dimension

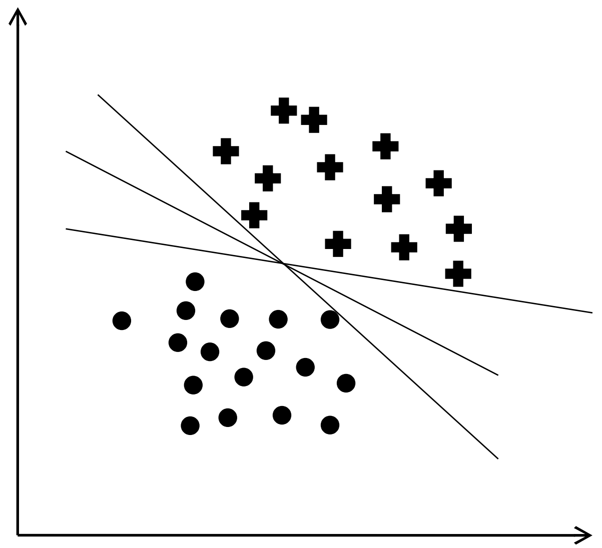
- With the new dimension, it searches for the **linear** optimal separating hyperplane (i.e., “decision boundary”)

- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane

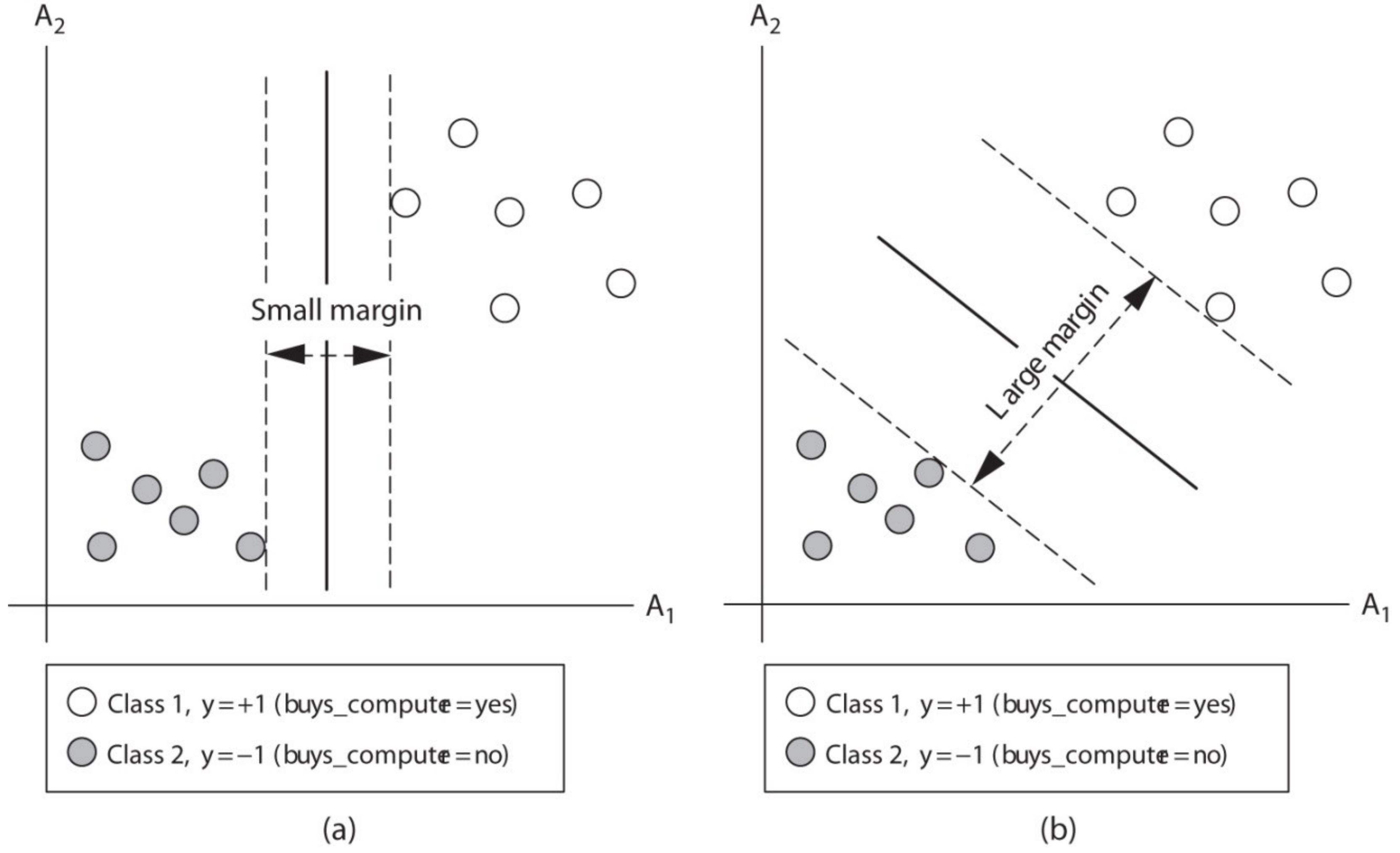
- SVM finds this hyperplane using support vectors (“essential” training tuples) and margins (defined by the support vectors)

Which Hyperplane?

- SVMs ***maximize the margin*** around the separating hyperplane
- The decision function is fully specified by a subset of training samples (<u>support vectors</u>)
- Quadratic programming problem



SVM: General Philosophy


A2
Small margin
A1
○ Class 1, y=+1 (buys_computer=yes)
● Class 2, y=−1 (buys_computer=no)
(a)
A2
Large margin
A1
○ Class 1, y=+1 (buys_computer=yes)
● Class 2, y=−1 (buys_computer=no)
(b)

Maximum Marginal Hyperplane

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 on critical training examples —they lie ***closest*** to the decision boundary (MMH)

❖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: When Data is Linearly Separable

- Let data ***D*** be ($\boldsymbol{X}_1$, y_1), …, ($\boldsymbol{X}_{|D|}$, $y_{|D|}$), where $\boldsymbol{X}_i$ is the set of training tuples associated with the class labels y_i

- 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 searches for the hyperplane with the* ***largest*** *margin*, i.e., **maximum marginal hyperplane** (MMH)

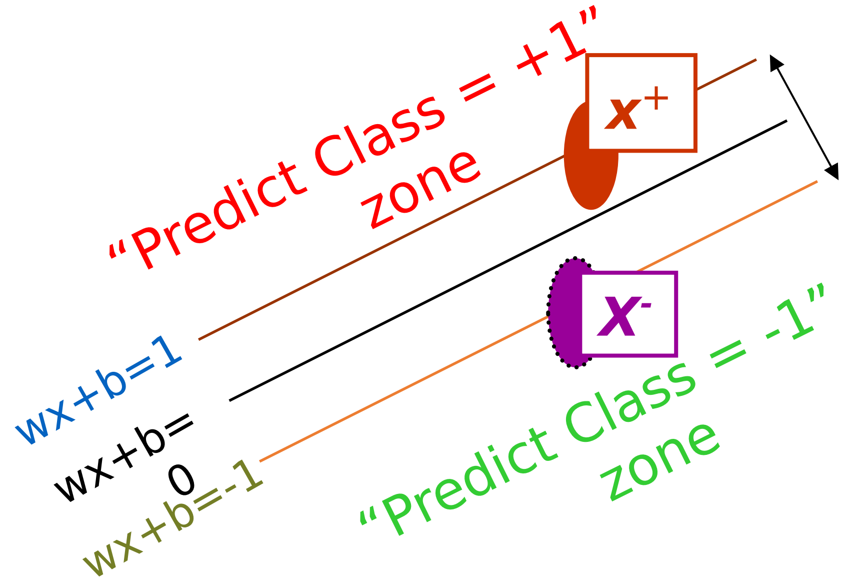
SVM: Linearly Separable

- A separating hyperplane can be written as
  - $W \bullet X + b = 0$

  where $\boldsymbol{W} = \{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D it can be written as
  - $w_0 + w_1 x_1 + w_2 x_2 = 0$

where w_0 is the bias
- Thus any point lying above the separating hyperplane satisfies:
  - $w_0 + w_1 x_1 + w_2 x_2 > 0$
- Any point lying below the separating hyperplane satisfies:
  - $w_0 + w_1 x_1 + w_2 x_2 < 0$
- Thus the hyperplane defining the sides of the margin can be written as:
  - $H1: w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1$, *and*
  - $H2: w_0 + w_1 x_1 + w_2 x_2 \leq -1 \text{ for } y_i = -1$
- Any training tuples falling on hyperplanes *H1* or *H2* (i.e., the sides defining the margin) are support vectors
- This becomes a **constrained (convex) quadratic optimization problem**: Quadratic objective function and linear constraints; Quadratic Programming (QP); Lagrangian multipliers

Linear SVM Mathematically



M = Margin Width

What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$
- $\mathbf{w} \cdot (\mathbf{x}^+ - \mathbf{x}^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: 1) Correctly classify all training data

$$\boldsymbol{wx_i + b \ge 1} \quad \text{if } y_i = +1$$

$$\boldsymbol{wx_i + b \le -1} \quad \text{if } y_i = -1$$

$$y_i(wx_i + b) \ge 1 \quad \text{for all i}$$

2) Maximize the Margin $M = \frac{2}{|w|}$

same as minimize $\frac{1}{2} w^t w$

- We can formulate a Quadratic Optimization Problem and solve for w and b

- Minimize $\Phi(w) = \frac{1}{2} w^t w$

  subject to $y_i(wx_i + b) \ge 1 \quad \forall i$

Solving the Optimization Problem

Find $\mathbf{w}$ and b such that

$\Phi(\mathbf{w}) = \frac{1}{2}\, \mathbf{w}^T\mathbf{w}$ is minimized;

and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic function* subject to *linear constraints*.
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \dots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and

(1) $\Sigma\alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

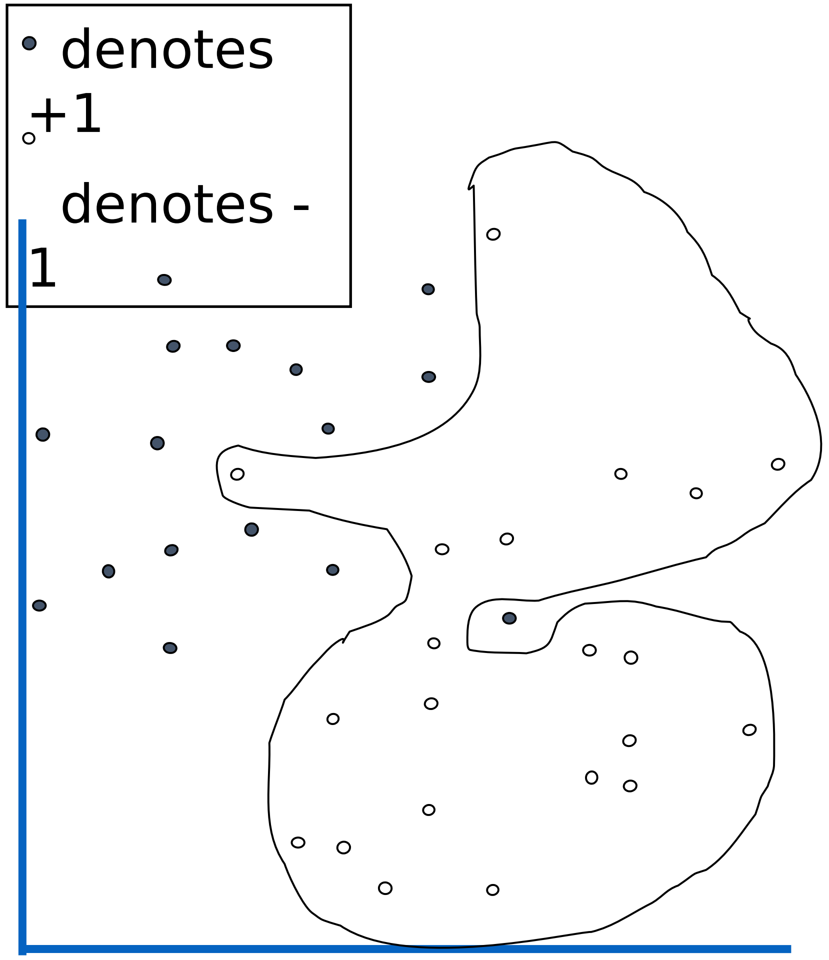
$$\boxed{\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^\mathrm{T}\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0}$$

- Each non-zero α_i indicates that corresponding $\boldsymbol{x_i}$ is a support vector.
- Then the classifying function will have the form:

$$\boxed{f(\mathbf{x}) = \Sigma \alpha_i y_i \mathbf{x_i}^\mathrm{T}\mathbf{x} + b}$$

- Notice that it relies on an *inner product* between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x_i}$.
- Also keep in mind that solving the optimization problem involves computing the inner products $\boldsymbol{x_i}^T\boldsymbol{x_j}$ between all pairs of training points.

Dataset with noise

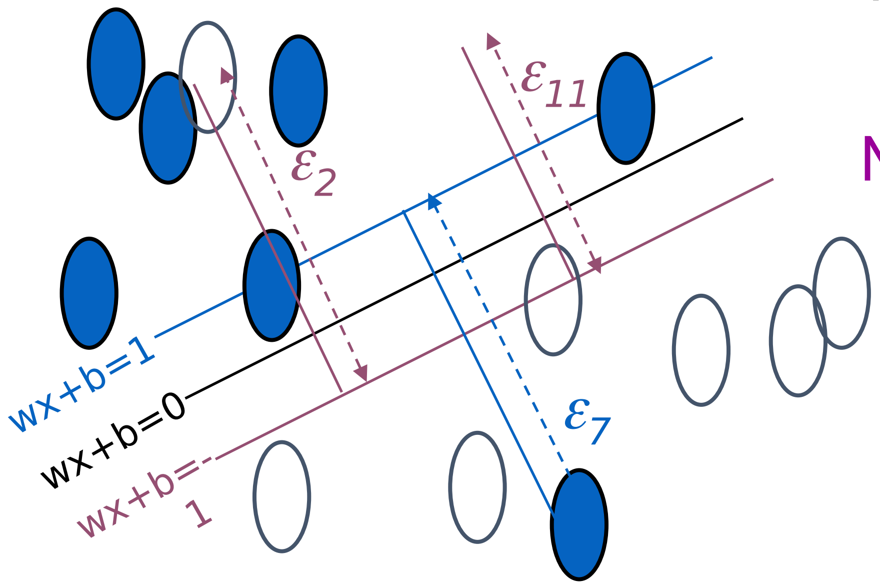


- Hard Margin: So far we require **all** data points be classified correctly
- - No training error
- What if the training set is noisy?
- - Solution 1: use very powerful kernels

OVERFITTING!

Soft Margin Classification

Slack variables *ξi* can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize $$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

Hard Margin v.s. Soft Margin

- The *old* formulation:

Find $\mathbf{w}$ and b such that

$\Phi(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{w}$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- The *new* formulation incorporating slack variables:

Find $\mathbf{w}$ and b such that

$\Phi(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ *for all* i

- Parameter C can be viewed as a way to control overfitting.

Linear SVMs: Summary

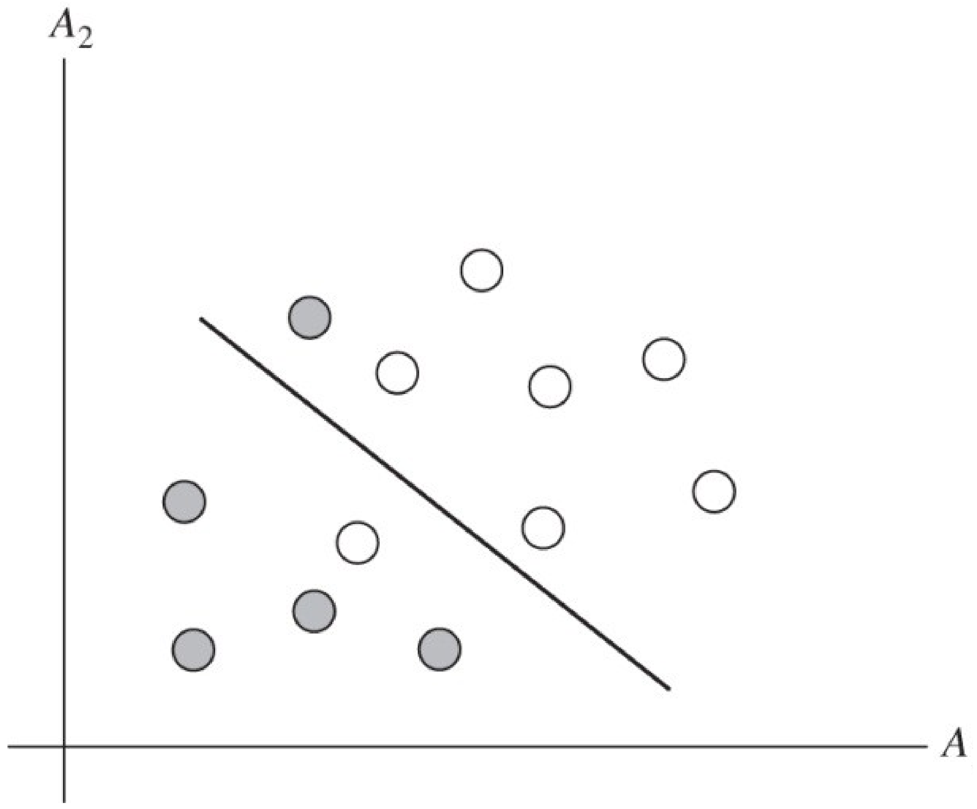
- The classifier is a *separating hyperplane*.
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points x_i are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

Find $\alpha_1 \dots \alpha_N$ such that
$Q(\alpha) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

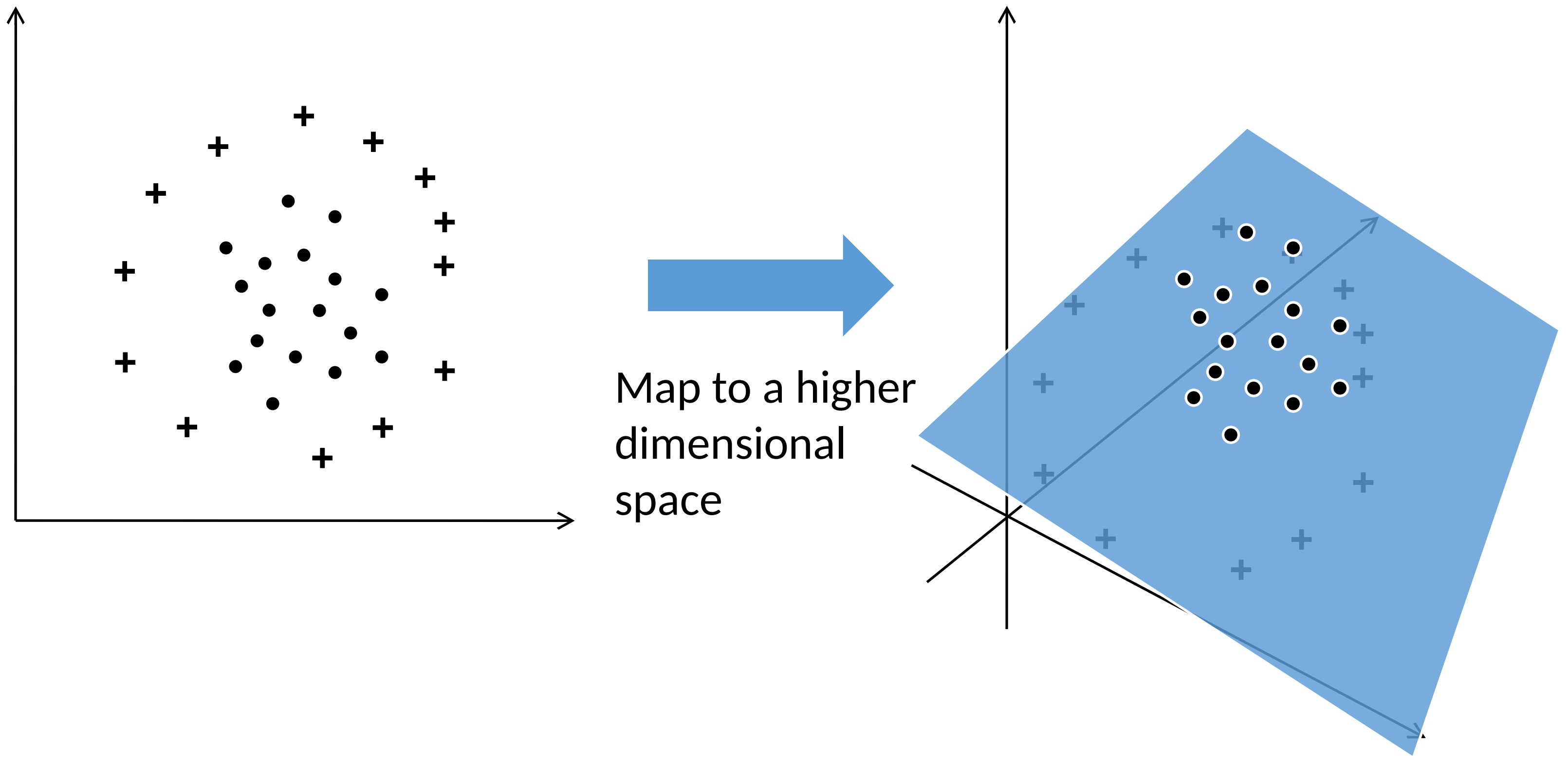
$$\boxed{f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b}$$

SVM: Linearly Inseparable

- Extend linear SVMs
- Find non-linear decision boundaries (non-linear hypersurfaces).
- *Step 1*: transform the original input data into high dimensional space
- *Step 2*: Search for linear separating hyperplane in the new space
- The linear hyperplane in high dimensional space is non-linear in the original space



SVM Linearly Inseparable



SVM: Linearly Inseparable

- Requires a ***mapping*** to higher dimensional space
  - e.g. $\varphi([x_i,x_j]) = [x_i, x_j, x_i^2+x_j^2]$
  - or $\varphi(X_i,X_j) = \varphi(X_i) \bullet \varphi(X_j)$
- Two problems:
  - How to choose a mapping?
  - Higher number of dimensions increases computational cost
    - Computing the dot product for all of the support vectors
      - One multiplication and one addition for each dimension
- Math trick:
  - Apply a *kernel function* to the original input data
  - Wherever a dot product is used, replace it with a kernel function

SVM: Kernel Function

- Equivalent to the dot product
    - $K(X_i, X_j) = \phi(X_i) \bullet \phi(X_j)$

- We can solve $K(X_i, X_j)$ without having to perform any of the mappings $\phi(X_i)$

- Provide a way to manipulate data as though it were projected into a higher dimensional space by operating on it in its original space

SVM: Kernel Function

- Instead of computing the dot product on the transformed data, it is mathematically equivalent to apply a kernel function $K(X_i, X_j)$ to the original data, i.e., $K(X_i, X_j) = \Phi(X_i)\,\Phi(X_j)$
- Typical Kernel Functions

Polynomial kernel of degree h :	$K(X_i, X_j) = (X_i \cdot X_j + 1)^h$
Gaussian radial basis function kernel :	$K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$
Sigmoid kernel :	$K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$

- After applying the kernel function, finding ***maximal margin hyperplane*** is similar to linear SVM with an upper bound on the Lagrangian multipliers
- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

Weakness of SVM

- It is sensitive to noise

  - A relatively small number of mislabeled examples can dramatically decrease the performance

- It only considers two classes

  - how to do multi-class classification with SVM?

  - Answer:

  1) with output arity *m*, learn *m* SVM's
    - SVM 1 learns "Output == 1" vs "Output != 1"
    - SVM 2 learns "Output == 2" vs "Output != 2"
    - :
    - SVM m learns "Output == m" vs "Output != m"

  2)To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the ***furthest*** into the positive region.

SVM vs. Logistic Regression

- Essentially they have the *same objective*
  - Binary classification with a linear decision boundary
  - Both are complex optimization problems solved using gradient descent and Lagrangian methods
- Key differences
  - *LR* uses a *logistic (sigmoid) function* to classify the data
  - SVM uses the *"kernel trick"* to find the maximum marginal boundary (defined by support vectors)
  - "Kernel trick" is difficult to implement using *LR*
  - SVM has been shown to **outperform** *LR*

SVM Applications

- SVM has been used successfully in many real-world problems

  - text (and hypertext) categorization

  - image classification

  - bioinformatics (protein classification, cancer classification)

  - hand-written character recognition

Application 1: Cancer Classification

- High Dimensional

  - p > 1000; n < 100

- Imbalanced

  - less positive samples

$$K[x,x] = k(x,x) + \lambda \frac{n^+}{N}$$

- Many irrelevant features $\longrightarrow$
- Noisy $\longrightarrow$

SVM is sensitive to noisy (mis-labeled) data

Genes				
Patients	**g-1**	**g-2**	**……**	**g-p**
P-1				
p-2				
…….				
p-n				

FEATURE SELECTION

In the linear case,
w_i^2 gives the ranking of dim i

Application 2: Text Categorization

- Task: The classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content.

  - email filtering, web searching, sorting documents by topic, etc..

- A document can be assigned to more than one category, so this can be viewed as a series of binary classification problems, one for each category

Representation of Text

IR’s vector space model (aka bag-of-words representation)

- A doc is represented by a vector indexed by a pre-fixed set or dictionary of terms
- Values of an entry can be binary or weights

$$\phi_i(x) = \frac{\mathrm{tf}_i \log(\mathrm{idf}_i)}{\kappa},$$

- Normalization, stop words, word stems
- Doc x => **φ**(*x*)

- https://www.csee.umbc.edu/~ian/irF02/lectures/07Models-VSM.pdf

Text Categorization using SVM

- The distance between two documents is $\varphi(x)\cdot\varphi(z)$

- $K(x,z) = \varphi(x)\cdot\varphi(z)$ is a valid kernel, SVM can be used with $K(x,z)$ for discrimination.

- Why SVM?

  -High dimensional input space

  -Few irrelevant features (dense concept)

  -Sparse document vectors (sparse instances)

  -Text categorization problems are linearly separable

Some Issues in SVM

- Choice of kernel

  - Gaussian or polynomial kernel is default

  - If ineffective, more elaborate kernels are needed

  - Domain experts can give assistance in formulating appropriate similarity measures

- Choice of kernel parameters

  - e.g. σ in Gaussian kernel

  - σ is the distance between closest points with different classifications

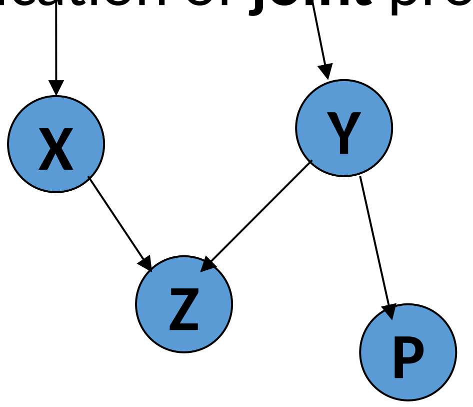
  - In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

- Optimization criterion – Hard margin v.s. Soft margin

  - a lengthy series of experiments in which various parameters are tested

Bayesian Belief Networks

- **Bayesian belief networks** (also known as **Bayesian networks**, **probabilistic networks**)
  - A simple, graphical notation for *conditional independence assertions* and hence for compact specification of full joint distributions
  - Allow *class conditional independencies* between *subsets* of variables
- A (*directed acyclic*) graphical model of causal relationships
  - Represents **dependency** among the variables
  - Gives a specification of **joint** probability distribution



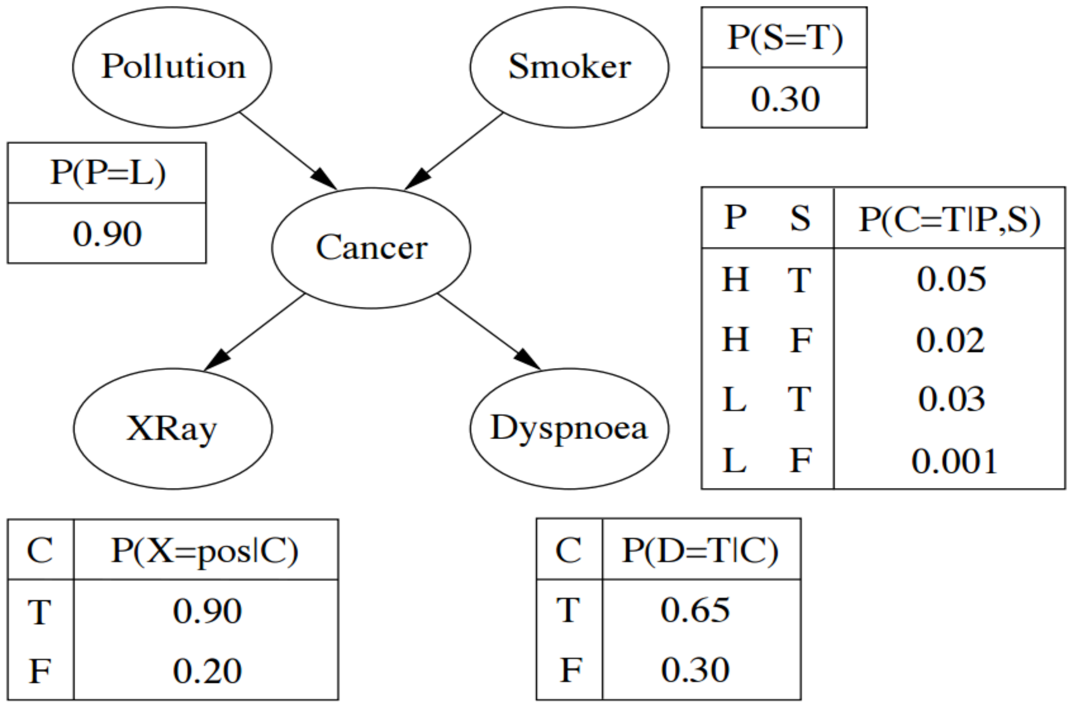
- Nodes: random variables
- Links: dependency
- X and Y are the parents of Z, and Y is the parent of P
- No dependency between Z and P
- Has no loops/cycles

Bayesian Belief Network: An Example

- Derivation of the conditional probability of a particular combination of values of ***X***, from conditional probability table (CPT):

$$P(x_1, x_2, \ldots, x_n) \quad = \quad \prod_i P(x_i | Parents(X_i))$$

$$\begin{aligned}
&P(X = pos \wedge D = T \wedge C = T \wedge P = low \wedge S = F) \\
&\qquad = \quad P(X = pos | D = T, C = T, P = low, S = F) \\
&\qquad\qquad \times P(D = T | C = T, P = low, S = F) \\
&\qquad\qquad \times P(C = T | P = low, S = F) P(P = low | S = F) P(S = F) \\
&\qquad = \quad P(X = pos | C = T) P(D = T | C = T) P(C = T | P = low, S = F) \\
&\qquad\qquad \times P(P = low) P(S = F)
\end{aligned}$$



P(S=T)
0.30

P(P=L)
0.90

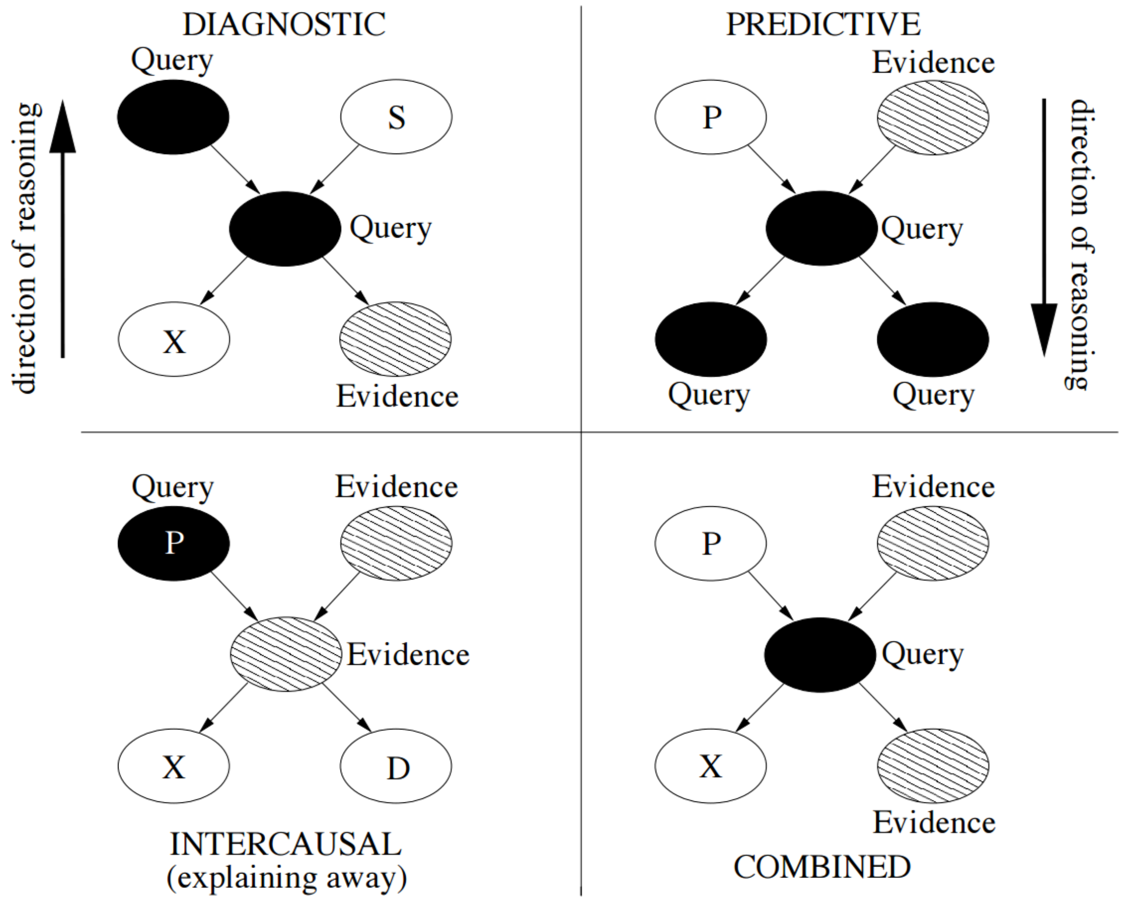
P	S	P(C=T|P,S)
H	T	0.05
H	F	0.02
L	T	0.03
L	F	0.001

C	P(X=pos|C)
T	0.90
F	0.20

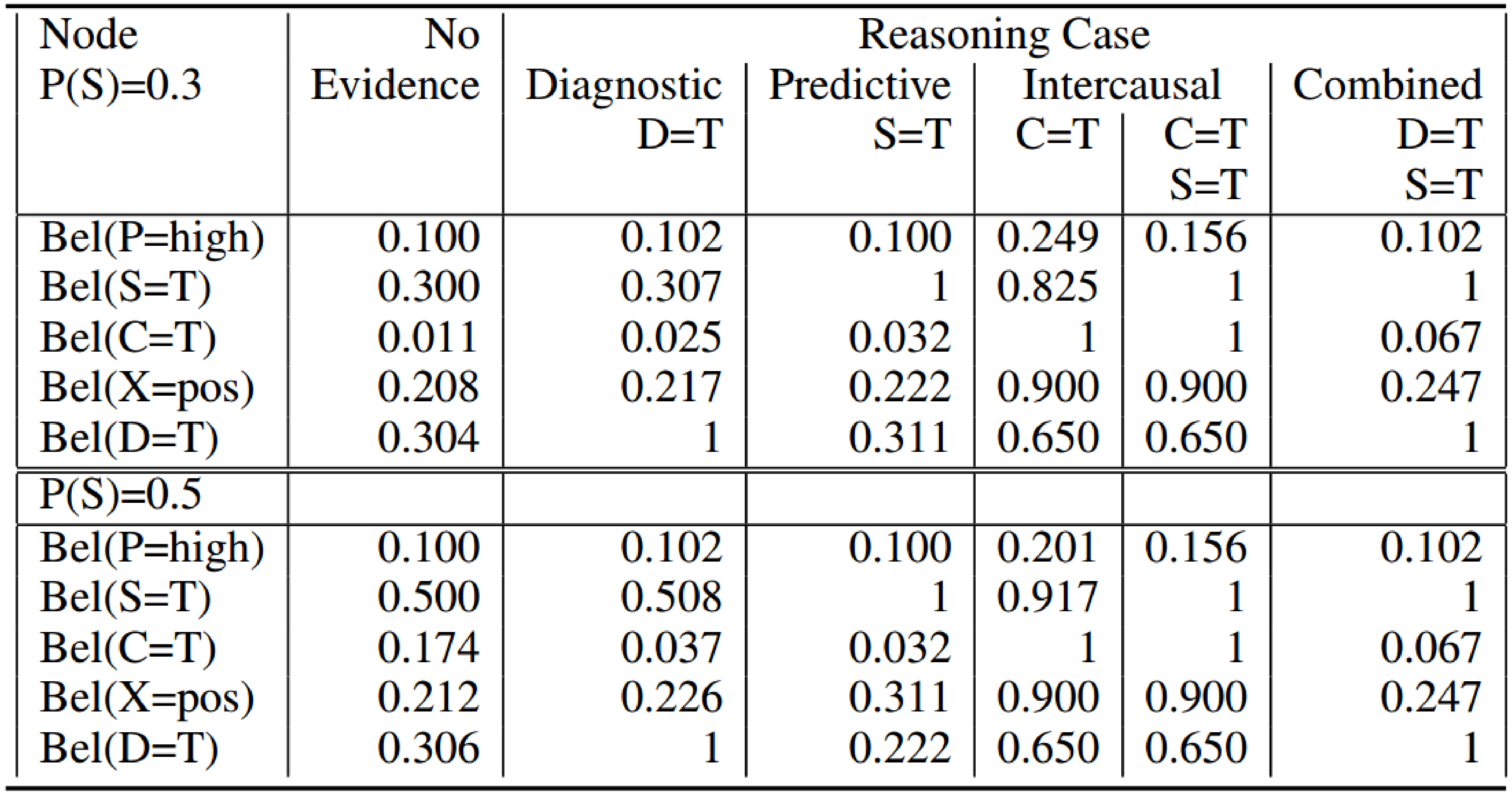
C	P(D=T|C)
T	0.65
F	0.30

Bayesian Belief Network: Types of Reasoning

- Bayesian Networks can support any direction of reasoning
- Diagnostic reasoning
  - Reasoning **from symptom to cause**
- Predictive reasoning
  - Reasoning **from new information about causes to new beliefs about effects**
- Inter-causal reasoning
  - Reasoning about the **mutual causes of a common effect**
- Combined reasoning
  - **Combining above types of reasoning** in any way



Updating Beliefs Given New Information



Node	No	Reasoning Case				
P(S)=0.3	Evidence	Diagnostic	Predictive	Intercausal		Combined
		D=T	S=T	C=T	C=T S=T	D=T S=T
Bel(P=high)	0.100	0.102	0.100	0.249	0.156	0.102
Bel(S=T)	0.300	0.307	1	0.825	1	1
Bel(C=T)	0.011	0.025	0.032	1	1	0.067
Bel(X=pos)	0.208	0.217	0.222	0.900	0.900	0.247
Bel(D=T)	0.304	1	0.311	0.650	0.650	1
P(S)=0.5						
Bel(P=high)	0.100	0.102	0.100	0.201	0.156	0.102
Bel(S=T)	0.500	0.508	1	0.917	1	1
Bel(C=T)	0.174	0.037	0.032	1	1	0.067
Bel(X=pos)	0.212	0.226	0.311	0.900	0.900	0.247
Bel(D=T)	0.306	1	0.222	0.650	0.650	1

How are Bayesian Networks Constructed?

- Subjective construction: Identification of (direct) causal structure
  - People are quite good at identifying direct causes from a given set of variables & whether the set contains all relevant direct causes
  - Markovian assumption: Each variable becomes independent of its non-effects once its direct causes are known
  - E.g., S ‹— F —› A ‹— T, path S—›A is blocked once we know F—›A
  - HMM (Hidden Markov Model): often used to model dynamic systems whose states are not observable, yet their outputs are
- **Synthesis from other specifications**
  - E.g., from a formal system design: block diagrams & info flow
- **Learning from data**
  - Learning parameters gives its structure or learning both structure and parameters
  - Maximum likelihood principle: favors Bayesian networks that maximize the probability of observing the given data set

Training Bayesian Networks

- Scenario 1: Given both the network structure and all variables observable: *compute only the CPT entries*
- Scenario 2: Network structure known, some variables hidden (missing values): ***gradient descent*** method, i.e., search for a solution along the steepest descent of a criterion function
  - Weights are initialized to random probability values
  - At each iteration, it moves towards what appears to be the best solution at the moment, without backtracking
  - Weights are updated at each iteration & converge to local optimum
- Scenario 3: Network structure unknown, all variables observable: search through the model space to *reconstruct network topology*
- Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose
- More:
  - D. Heckerman. A Tutorial on Learning with Bayesian Networks. In *Learning in Graphical Models*, M. Jordan, ed.. MIT Press, 1999.

Ensemble Methods: Increasing Accuracy

- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of *k* learned models, M_1, M_2, …, M_k, with the aim of creating an **improved** model M^*

- Popular ensemble methods
  - **Bagging**: averaging the prediction over a collection of classifiers
  - **Boosting**: weighted vote with a collection of classifiers
  - **Random Forests**
  - **Ensemble**: combining a set of heterogeneous classifiers

Bagging: Bootstrap Aggregation

- **Analogy: Diagnosis based on multiple doctors' majority vote**
- Training
  - Given a set *D* of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample ***X***
  - Each classifier M_i returns its class prediction
  - The bagged classifier M^* counts the votes and assigns the class with the **most votes** to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from *D*
  - For noise data: not considerably worse, more **robust**
  - Proved improved accuracy in prediction

Boosting

- Analogy: Consult several doctors, based on a combination of weighted diagnoses - weight assigned based on the previous diagnosis accuracy
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of *k* classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final classifier M^* **combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Random Forest (Breiman 2001)

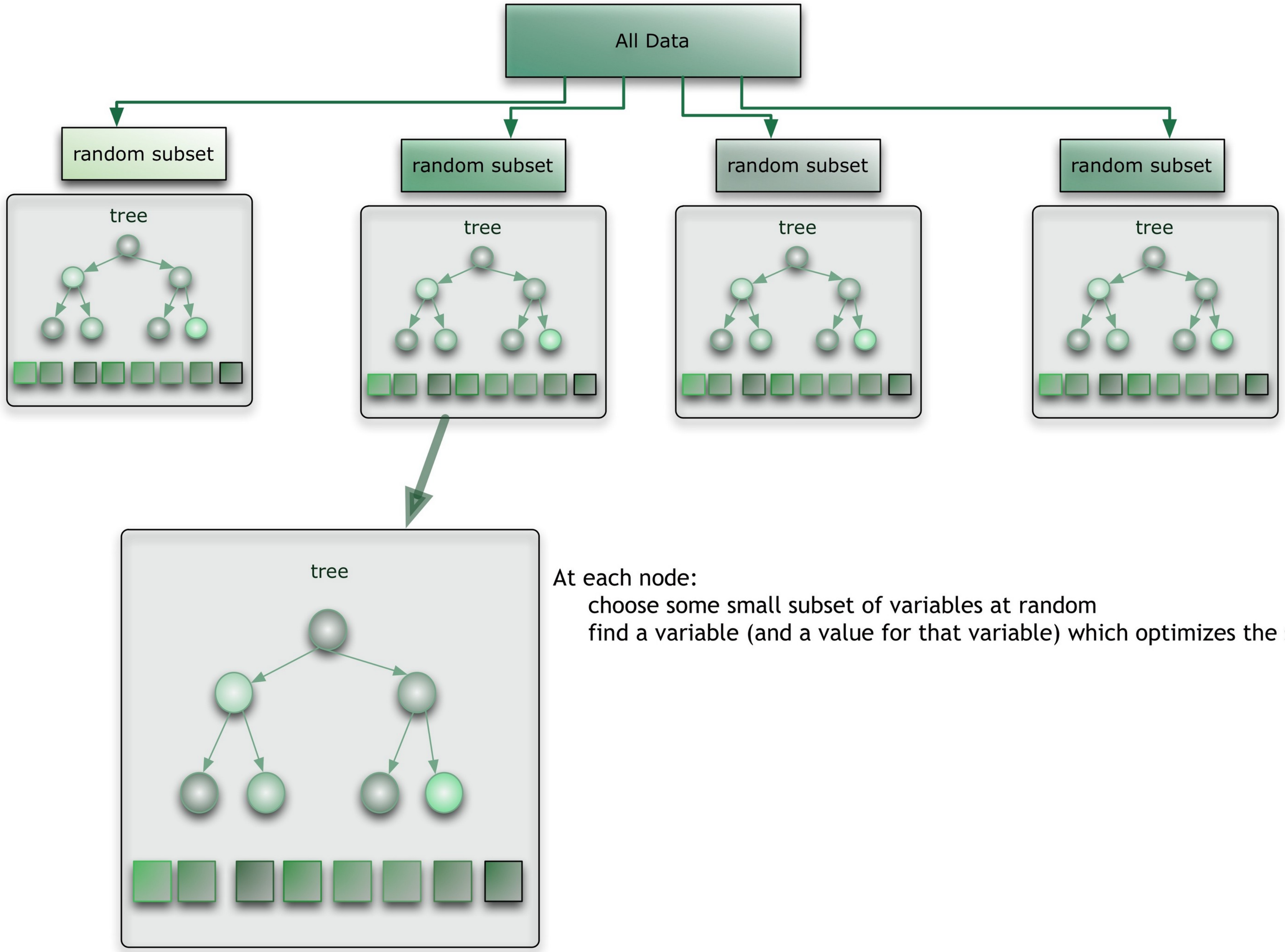
- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a **random selection of attributes at each node** to determine the split
  - During classification, *each tree votes and the most popular class is returned*

- Two Methods to construct Random Forest:
  - **Forest-RI** (*r**andom** **i**nput selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - **Forest-RC** (*r**andom linear** **c**ombinations*): Creates new attributes (or features) that are a ***linear*** combination of the existing attributes (reduces the correlation between individual classifiers)

- Comparable in accuracy to boosting, but **more robust to errors and outliers**
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

General Random Forest Algorithm

- Each tree is constructed using the following algorithm:
  1. *N*: the number of training samples, *M*: the number of variables in the classifier.
  2. *m*: the number of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*.
  3. Choose a training set for this tree by choosing *n* times with replacement from all *N* available training cases (i.e.: bootstrap). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
  4. For each node of the tree, randomly choose *m* variables on which to base the decision at that node. Calculate the best split based on these *m* variables in the training set.
  5. Each tree is fully grown and not pruned.
- Prediction: a new sample is assigned the label of the training sample in the terminal node it ends up in. This procedure is iterated over all trees in the ensemble, and *the average vote of all trees is reported as random forest prediction*.



At each node:
- choose some small subset of variables at random
- find a variable (and a value for that variable) which optimizes the split

References

- https://scikit-learn.org/stable/modules/svm.html
- https://towardsdatascience.com/bbn-bayesian-belief-networks-how-to-build-them-effectively-in-python-6b7f93435bba
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html