CS 188: Artificial Intelligence
Spring 2011

Final Review
5/2/2011

Pieter Abbeel – UC Berkeley

Probabilistic Reasoning

- Probability
  - Random Variables
  - Joint and Marginal Distributions
  - Conditional Distribution
  - Inference by Enumeration
  - Product Rule, Chain Rule, Bayes' Rule
  - Independence
- Distributions over LARGE Numbers of Random Variables → Bayesian networks
  - Representation
  - Inference
    - Exact: Enumeration, Variable Elimination
    - Approximate: Sampling
  - Learning
    - Maximum likelihood parameter estimation
    - Laplace smoothing
    - Linear interpolation

Probability recap

- Conditional probability $P(x|y) = \frac{P(x,y)}{P(y)}$
- Product rule $P(x,y) = P(x|y)P(y)$
- Chain rule $P(X_1, X_2, \dots X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2)\dots$
- X, Y independent iff: $\forall x, y : P(x,y) = P(x)P(y)$
  equivalently, iff: $\forall x, y \ : \ P(x|y) = P(x)$
  equivalently, iff: $\forall x, y \ : \ P(y|x) = P(y)$
- X and Y are conditionally independent given Z iff:
  $\forall x, y, z : P(x,y|z) = P(x|z)P(y|z)$
  equivalently, iff: $\forall x, y, z \ : \ P(x|y,z) = P(x|z)$
  equivalently, iff: $\forall x, y, z \ : \ P(y|x,z) = P(y|z)$

Inference by Enumeration

- P(sun)?
- P(sun | winter)?
- P(sun | winter, hot)?

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

Chain Rule → Bayes net

- Chain rule: can ***always*** write ***any*** joint distribution as an incremental product of conditional distributions

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

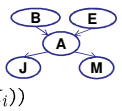
$$P(x_1, x_2, \dots x_n) = \prod_i P(x_i|x_1 \dots x_{i-1})$$

- Bayes nets: make conditional independence assumptions of the form:

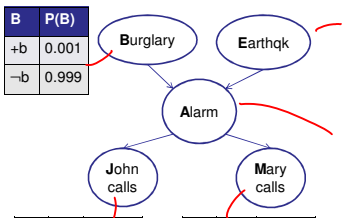
$$P(x_i|x_1 \cdots x_{i-1}) = P(x_i|parents(X_i))$$

giving us:

$$P(x_1, x_2, \dots x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i))$$



Example: Alarm Network



B	P(B)
+b	0.001
¬b	0.999

E	P(E)
+e	0.002
¬e	0.998

A	J	P(J|A)
+a	+j	0.9
+a	¬j	0.1
¬a	+j	0.05
¬a	¬j	0.95

A	M	P(M|A)
+a	+m	0.7
+a	¬m	0.3
¬a	+m	0.01
¬a	¬m	0.99

B	E	A	P(A|B,E)
+b	+e	+a	0.95
+b	+e	¬a	0.05
+b	¬e	+a	0.94
+b	¬e	¬a	0.06
¬b	+e	+a	0.29
¬b	+e	¬a	0.71
¬b	¬e	+a	0.001
¬b	¬e	¬a	0.999

Size of a Bayes' Net for $P(X_1, X_2, \ldots X_n)$

- How big is a joint distribution over N Boolean variables?
  2^N
- Size of representation if we use the chain rule
  2^N
- How big is an N-node net if nodes have up to k parents?
  $O(N * 2^{k+1})$
- Both give you the power to calculate
- BNs:
  - Huge space savings!
  - Easier to elicit local CPTs
  - Faster to answer queries

Bayes Nets: Assumptions

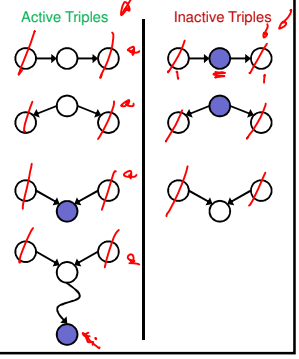
- Assumptions we are required to make to define the Bayes net when given the graph:

$$P(x_i|x_1 \cdots x_{i-1}) = P(x_i|parents(X_i))$$

- Given a Bayes net graph additional conditional independences can be read off directly from the graph
- Question: Are two nodes *necessarily* independent given certain evidence?
- If no, can prove with a counter example
  - I.e., pick a set of CPT's, and show that the independence assumption is violated by the resulting distribution
- If yes, can prove with
  - Algebra (tedious)
  - D-separation (analyzes graph)

D-Separation

- Question: Are X and Y conditionally independent given evidence vars {Z}?
  - Yes, if X and Y "separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain A → B → C where B is unobserved (either direction)
  - Common cause A ← B → C where B is unobserved
  - Common effect (aka v-structure) A → B ← C where B *or one of its descendents* is observed
- All it takes to block a path is a single inactive segment



D-Separation

- Given query $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$
- Shade all evidence nodes
- For all (undirected!) paths between and
  - Check whether path is active
    - If active return $X_i \not\perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$
- (If reaching this point all paths have been checked and shown inactive)
  - Return $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$

Example

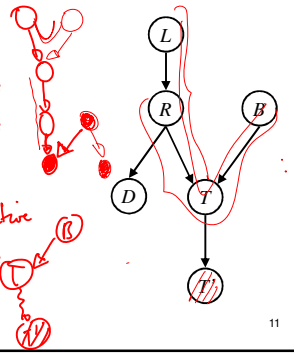
$L \perp\!\!\!\perp T' | T$ Yes

$L \perp\!\!\!\perp B$ Yes

$L \perp\!\!\!\perp B | T$

$L \perp\!\!\!\perp B | T'$

$L \perp\!\!\!\perp B | T, R$ Yes



All Conditional Independences

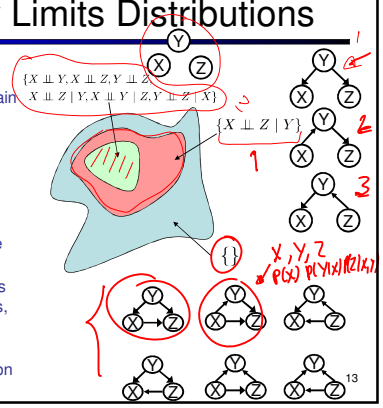
- Given a Bayes net structure, can run d-separation to build a complete list of conditional independences that are necessarily true of the form

$$X_i \perp\!\!\!\perp X_j | \{X_{k_1}, ..., X_{k_n}\}$$

- This list determines the set of probability distributions that can be represented by Bayes' nets with this graph structure

Topology Limits Distributions

- Given some graph topology G, only certain joint distributions can be encoded
- The graph structure guarantees certain (conditional) independences
- (There might be more independence)
- Adding arcs increases the set of distributions, but has several costs
- Full conditioning can encode any distribution



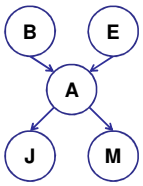
Bayes Nets Status

- ✔ Representation
- Inference
- Learning Bayes Nets from Data

Inference by Enumeration

- Given unlimited time, inference in BNs is easy
- Recipe:
  - State the marginal probabilities you need
  - Figure out ALL the atomic probabilities you need
  - Calculate and combine them
- Example:

$$P(+b|+j,+m) = \frac{P(+b,+j,+m)}{P(+j,+m)}$$



Example: Enumeration

- In this simple method, we only need the BN to synthesize the joint entries

$$P(+b,+j,+m) =$$
$$P(+b)P(+e)P(+a|+b,+e)P(+j|+a)P(+m|+a)+$$
$$P(+b)P(+e)P(-a|+b,+e)P(+j|-a)P(+m|-a)+$$
$$P(+b)P(-e)P(+a|+b,-e)P(+j|+a)P(+m|+a)+$$
$$P(+b)P(-e)P(-a|+b,-e)P(+j|-a)P(+m|-a)$$

Variable Elimination

- Why is inference by enumeration so slow?
  - You join up the whole joint distribution before you sum out the hidden variables
  - You end up repeating a lot of work!
- Idea: interleave joining and marginalizing!
  - Called "Variable Elimination"
  - Still NP-hard, but usually much faster than inference by enumeration

Variable Elimination Outline

- Track objects called factors
- Initial factors are local CPTs (one per node)

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

- Any known values are selected
  - E.g. if we know $L = +\ell$, the initial factors are

$P(R)$

+r	0.1
-r	0.9

$P(T|R)$

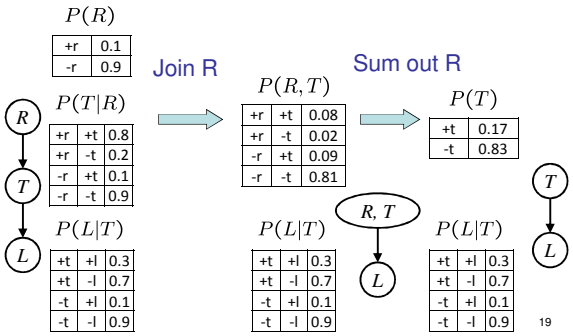
+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(+\ell|T)$

+t	+l	0.3
-t	+l	0.1

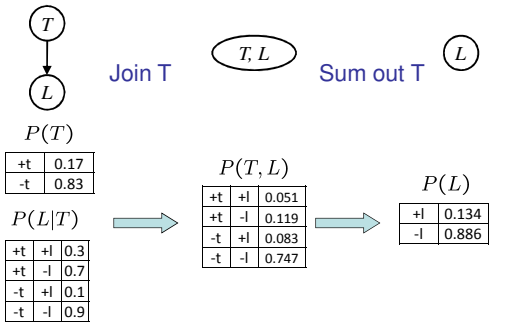
- VE: Alternately join factors and eliminate variables

Variable Elimination Example



19

Variable Elimination Example



* VE is variable elimination

Example

$P(B|j,m) \propto P(B,j,m)$

$P(B)$ $P(E)$ $P(A|B,E)$ $P(j|A)$ $P(m|A)$

Choose A

$P(A|B,E)$
$P(j|A)$
$P(m|A)$
× $P(j,m,A|B,E)$ Σ $P(j,m|B,E)$

$P(B)$ $P(E)$ $P(j,m|B,E)$

21

Example

$P(B)$ $P(E)$ $P(j,m|B,E)$

Choose E

$P(E)$
$P(j,m|B,E)$
× $P(j,m,E|B)$ Σ $P(j,m|B)$

$P(B)$ $P(j,m|B)$

Finish with B

$P(B)$
$P(j,m|B)$
× $P(j,m,B)$ Normalize $P(B|j,m)$

22

General Variable Elimination

- Query: $P(Q|E_1 = e_1, \ldots E_k = e_k)$
- Start with initial factors:
  - Local CPTs (but instantiated by evidence)
- While there are still hidden variables (not Q or evidence):
  - Pick a hidden variable H
  - Join all factors mentioning H
  - Eliminate (sum out) H
- Join all remaining factors and normalize

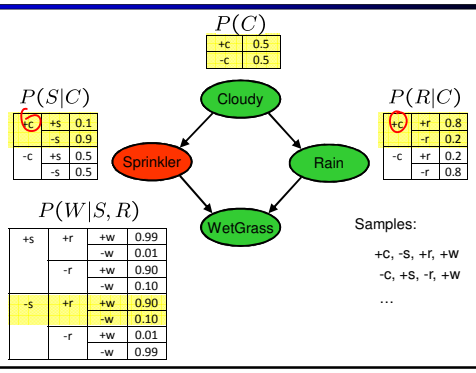
23

Approximate Inference: Sampling

- Basic idea:
  - Draw N samples from a sampling distribution S
  - Compute an approximate posterior probability
  - Show this converges to the true probability P
- Why? Faster than computing the exact answer
- Prior sampling:
  - Sample ALL variables in topological order as this can be done quickly
- Rejection sampling for query $P(Q|E_1 = e_1, \ldots E_k = e_k)$
  - = like prior sampling, but reject when a variable is sampled inconsistent with the query, in this case when a variable E_i is sampled differently from e_i
- Likelihood weighting for query $P(Q|E_1 = e_1, \ldots E_k = e_k)$
  - = like prior sampling but variables E_i are not sampled, when it's their turn, they get set to e_i, and the sample gets weighted by $P(e_i \mid$ value of parents(e_i) in current sample)
- Gibbs sampling: repeatedly samples each non-evidence variable conditioned on all other variables → can incorporate downstream evidence

24

Prior Sampling



Samples:

+c, -s, +r, +w

-c, +s, -r, +w

…

Example

- We'll get a bunch of samples from the BN:

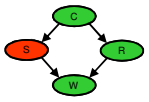
  +c, -s, +r, +w

  +c, +s, +r, +w

  -c, +s, +r, -w

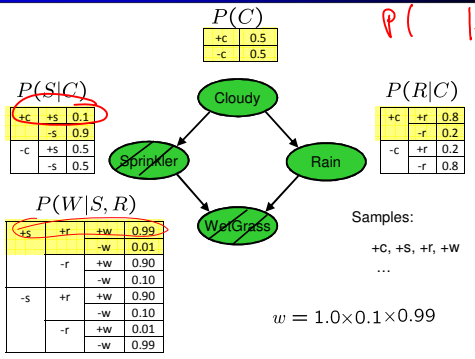
  +c, -s, +r, +w

  -c, -s, -r, +w



- If we want to know P(W)
  - We have counts <+w:4, -w:1>
  - Normalize to get P(W) = <+w:0.8, -w:0.2>
  - This will get closer to the true distribution with more samples
  - Can estimate anything else, too
  - What about P(C| +w)? P(C| +r, +w)? P(C| -r, -w)?
  - Fast: can use fewer samples if less time

Likelihood Weighting



Samples:

+c, +s, +r, +w

…

$w = 1.0 \times 0.1 \times 0.99$

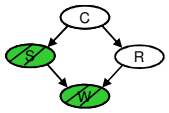
Likelihood Weighting

- Sampling distribution if z sampled and e fixed evidence

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | \text{Parents}(Z_i))$$

- Now, samples have weights

$$w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | \text{Parents}(E_i))$$



- Together, weighted sampling distribution is consistent

$$S_{WS}(\mathbf{z}, \mathbf{e}) \cdot w(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i | \text{Parents}(z_i)) \prod_{i=1}^{m} P(e_i | \text{Parents}(e_i))$$

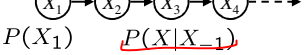
$$= P(\mathbf{z}, \mathbf{e})$$

Gibbs Sampling

- *Idea*: instead of sampling from scratch, create samples that are each like the last one.
- *Procedure*: resample one variable at a time, conditioned on all the rest, but keep evidence fixed.
- *Properties*: Now samples are not independent (in fact they're nearly identical), but sample averages are still consistent estimators!
- *What's the point*: both upstream and downstream variables condition on evidence.

Markov Models

- A Markov model is a chain-structured BN
  - Each node is identically distributed (stationarity)
  - Value of X at a given time is called the state
  - As a BN:



$P(X_1)$ $P(X|X_{-1})$

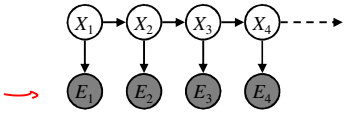
- The chain is just a (growing) BN
  - We can always use generic BN reasoning on it if we truncate the chain at a fixed length
- Stationary distributions
  - For most chains, the distribution we end up in is independent of the initial distribution
  - Called the stationary distribution of the chain

$$P_\infty(X_\infty) = \sum_x P(X_\infty | x) P(x)$$

- Example applications: Web link analysis (Page Rank) and Gibbs Sampling

Hidden Markov Models

- Underlying Markov chain over states S
- You observe outputs (effects) at each time step



- Speech recognition HMMs:
  - X_i: specific positions in specific words; E_i: acoustic signals
- Machine translation HMMs:
  - X_i: translation options; E_i: Observations are words
- Robot tracking:
  - X_i: positions on a map; E_i: range readings

Online Belief Updates

- Every time step, we start with current P(X | evidence)
- We update for time:

$$P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})$$

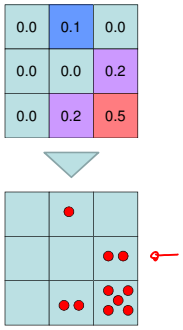
- We update for evidence:

$$P(x_t|e_{1:t}) \propto_X P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)$$

- The forward algorithm does both at once (and doesn't normalize)

Particle Filtering

- = likelihood weighting + resampling at each time slice
- Why: sometimes |X| is too big to use exact inference
- Particle is just new name for sample

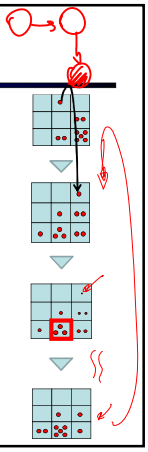


Particle Filtering

- Elapse time:
  - Each particle is moved by sampling its next position from the transition model

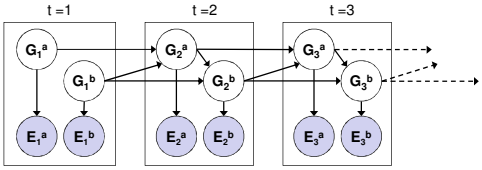
$$x' = \text{sample}(P(X'|x))$$

- Observe:
  - We don't sample the observation, we fix it and downweight our samples based on the evidence
  - This is like likelihood weighting, so we multiply with $P(e_t|x)$
- Resample:
  - Rather than tracking weighted samples, we resample
  - N times, we choose from our weighted sample distribution



Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence
- Idea: Repeat a fixed Bayes net structure at each time
- Variables from time *t* can condition on those from *t-1*



- Discrete valued dynamic Bayes nets are also HMMs

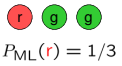
Bayes Nets Status

- ✓ Representation
- ✓ Inference
- Learning Bayes Nets from Data

Parameter Estimation

- Estimating distribution of random variables like X or X | Y
- *Empirically:* use training data
  - For each outcome x, look at the *empirical rate* of that value:

$$P_{ML}(x) = \frac{\text{count}(x)}{\text{total samples}}$$



$$P_{ML}(r) = 1/3$$

  - This is the estimate that maximizes the *likelihood of the data*

$$L(x,\theta) = \prod_i P_\theta(x_i)$$

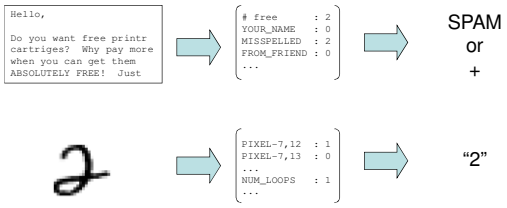
- *Laplace smoothing*
  - Pretend saw every outcome k extra times $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$
  - Smooth each condition independently: $P_{LAP,k}(x|y) = \frac{c(x,y)+k}{c(y)+k|X|}$

Bayes Nets Status

- ✔ Representation
- ✔ Inference
- ✔ Learning Bayes Nets from Data

Classification: Feature Vectors

x $\quad$ $f(x)$ $\quad$ y



Classification overview

- Naïve Bayes:
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)
- Perceptron:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate
- MIRA:
  - Like perceptron, but adaptive scaling of size of update
- SVM:
  - Properties similar to perceptron
  - Convex optimization formulation
- Nearest-Neighbor:
  - Non-parametric: more expressive with more training data
- Kernels
  - Efficient way to make linear learning architectures into nonlinear ones

Bayes Nets for Classification

- One method of classification:
  - Use a probabilistic model!
  - Features are observed random variables F_i
  - Y is the query variable
  - Use probabilistic inference to compute most likely Y

$$y = \text{argmax}_y \; P(y|f_1 \dots f_n)$$

- You already know how to do this inference

General Naïve Bayes

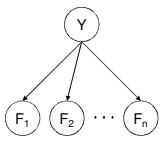
- A general *naive Bayes* model:

|Y| x |F|^n parameters

$$P(Y, F_1 \dots F_n) = P(Y) \prod_i P(F_i|Y)$$

|Y| parameters

n x |F| x |Y| parameters



- We only specify how each feature depends on the class
- Total number of parameters is *linear* in n
- Our running example: digits

Bag-of-Words Naïve Bayes

- Generative model

$$P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i|Y)$$

Word at position i, not i^th word in the dictionary!

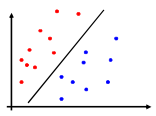
- Bag-of-words
  - Each position is identically distributed
  - All positions share the same conditional probs P(W|C)
  - → When learning the parameters, data is shared over all positions in the document (rather than separately learning a distribution for each position in the document)

- Our running example: spam vs. ham

Linear Classifier

- Binary linear classifier:

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$



- Multiclass linear classifier:
  - A weight vector for each class:

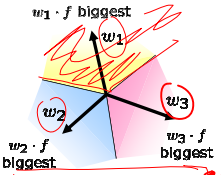
$$w_y$$

  - Score (activation) of a class y:

$$w_y \cdot f(x)$$

  - Prediction highest score wins

$$y = \arg\max_y w_y \cdot f(x)$$



Binary = multiclass where the negative class has weight zero

Perceptron = algorithm to learn weights w

- Start with zero weights
- Pick up training instances one by one
- Classify with current weights

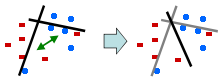
$$y = \arg\max_y w_y \cdot f(x)$$
$$= \arg\max_y \sum_i w_{y,i} \cdot f_i(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

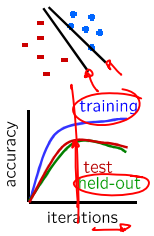
$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$

Problems with the Perceptron

- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting

Fixing the Perceptron: MIRA

- Update size that fixes the current mistake and also minimizes the change to w

Guessed y instead of y^* on example x with features $f(x)$

- Update w by solving:

$$\min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2$$
$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

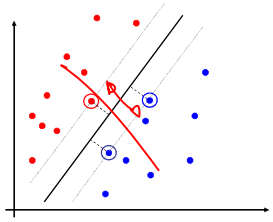
$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$
$$\tau \leq C$$

SVM

$$\min_w \frac{1}{2}||w||^2$$
$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once

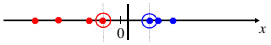


SVM

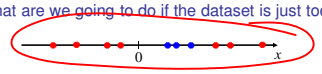
$$\min_w \frac{1}{2}||w||^2$$
$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Non-Linear Separators

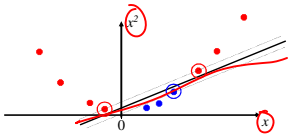
- Data that is linearly separable (with some noise) works out great:



- But what are we going to do if the dataset is just too hard?



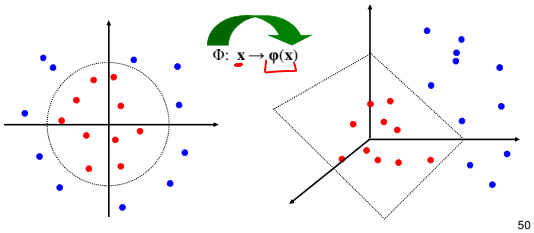
- How about… mapping data to a higher-dimensional space:



This and next few slides adapted from Ray Mooney, UT

Non-Linear Separators

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Some Kernels

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back
- Linear kernel: $K(x,x') = x \cdot x' = \sum_i x_i x'_i$

$$\phi(x) = x$$

- Quadratic kernel: $K(x,x') = (x \cdot x' + 1)^2$

$$= \sum x_i x_j x'_i x'_j + 2\sum x_i x'_i + 1$$

For $x \in \Re^3$:

$$\phi(x) = [x_1x_1\ x_1x_2\ x_1x_3\ x_2x_1\ x_2x_2\ x_2x_3\ x_3x_1\ x_3x_2\ x_3x_3\ \sqrt{2}x_1\ \sqrt{2}x_2\ \sqrt{2}x_3\ 1]$$

Some Kernels (2)

- Polynomial kernel: $K(x,x') = (x \cdot x' + 1)^d$

For $x \in \Re^3$:

$$\phi(x) = [x_1^d\ x_2^d\ x_3^d\ \sqrt{d}x_1^{d-1}x_2\ \sqrt{d}x_1^{d-1}x_3\ \ldots\ \sqrt{d}x_1\ \sqrt{d}x_2\ \sqrt{d}x_3\ 1]$$

For $x \in \Re^n$ the d-order polynomial kernel's implicit feature space is $\binom{n+d}{d}$ dimensional.

By contrast, computing the kernel directly only requires $O(n)$ time.

Why and When Kernels?

- Can't you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
- Kernels let us compute with these features implicitly
  - Example: implicit dot product in polynomial, Gaussian and string kernel takes much less space and time per dot product
- When can we use kernels?
  - When our learning algorithm can be reformulated in terms of only inner products between feature vectors
    - Examples: perceptron, support vector machine

K-nearest neighbors

- 1-NN: copy the label of the most similar data point
- K-NN: let the k nearest neighbors vote (have to devise a weighting scheme)

Truth

2 Examples

10 Examples

100 Examples

10000 Examples

- Parametric models:
  - Fixed set of parameters
  - More data means better settings
- Non-parametric models:
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit
- (K)NN is non-parametric

Basic Similarity

- Many similarities based on feature dot products:

$$\text{sim}(x, x') = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x')$$

- If features are just the pixels:

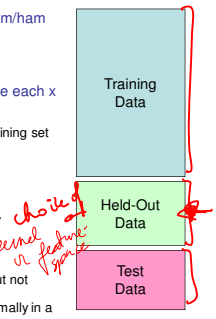
$$\text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

- Note: not all similarities are of this form

55

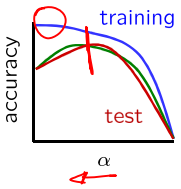
Important Concepts

- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Learn parameters (e.g. model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test set
  - Very important: never "peek" at the test set!
- Evaluation
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - We'll investigate overfitting and generalization formally in a few lectures



Tuning on Held-Out Data

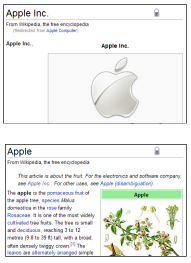
- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(Y|X), P(Y)
  - Hyperparameters:
    - Amount of smoothing to do: k, α (naïve Bayes)
    - Number of passes over training data (perceptron)
- Where to learn?
  - Learn parameters from training data
  - Must tune hyperparameters on different data
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data



Extension: Web Search

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR
- Web search: not exactly classification, but rather ranking

x = "Apple Computers"



Feature-Based Ranking

x = "Apple Computers"

$f($  $) = [0.3\ 5\ 0\ 0\ \ldots]$

$f($  $) = [0.8\ 4\ 2\ 1\ \ldots]$

Perceptron for Ranking

- Inputs x
- Candidates y
- Many feature vectors: $f(x, y)$
- One weight vector: w
  - Prediction:

$$y = \arg\max_y w \cdot f(x, y)$$

  - Update (if wrong):

$$w = w + f(x, y^*) - f(x, y)$$