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 \( P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1, X_2) \)

- \( X, Y \) 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

- \( P(\text{sun})? \)
  \[ 0.15 + 0.10 + 0.10 + 0.30 \]

- \( P(\text{sun} | \text{winter})? \)
  \[ \frac{P(\text{sun}, \text{winter})}{P(\text{winter})} = \frac{0.10}{0.40} = 0.25 \]

- \( P(\text{sun} | \text{winter}, \text{hot})? \)
  \[ \frac{P(\text{sun}, \text{winter}, \text{hot})}{P(\text{winter}, \text{hot})} \]

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>W</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>summer</td>
<td>hot</td>
<td>sun</td>
<td>0.30</td>
</tr>
<tr>
<td>summer</td>
<td>hot</td>
<td>rain</td>
<td>0.05</td>
</tr>
<tr>
<td>summer</td>
<td>cold</td>
<td>sun</td>
<td>0.10</td>
</tr>
<tr>
<td>summer</td>
<td>cold</td>
<td>rain</td>
<td>0.05</td>
</tr>
<tr>
<td>winter</td>
<td>hot</td>
<td>sun</td>
<td>0.10</td>
</tr>
<tr>
<td>winter</td>
<td>hot</td>
<td>rain</td>
<td>0.05</td>
</tr>
<tr>
<td>winter</td>
<td>cold</td>
<td>sun</td>
<td>0.15</td>
</tr>
<tr>
<td>winter</td>
<td>cold</td>
<td>rain</td>
<td>0.20</td>
</tr>
</tbody>
</table>
Chain Rule → Bayes net

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

\[
P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)
\]

\[
P(x_1, x_2, \ldots x_n) = \prod_i P(x_i|x_1 \ldots x_{i-1})
\]

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

\[
P(x_i|x_1 \ldots x_{i-1}) = P(x_i|\text{parents}(X_i))
\]

giving us:

\[
P(x_1, x_2, \ldots x_n) = \prod_{i=1}^n P(x_i|\text{parents}(X_i))
\]

---

**Example: Alarm Network**

### Joint probabilities

<table>
<thead>
<tr>
<th>B</th>
<th>P(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+b</td>
<td>0.001</td>
</tr>
<tr>
<td>¬b</td>
<td>0.999</td>
</tr>
</tbody>
</table>

### Conditional probabilities

#### B (Burglary)

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

#### E (Earthquake)

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

#### A (Alarm)

| B  | E  | 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? $O(N \times 2^{k+1})$
- 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:
  $$P(x_i|x_1 \cdots x_{i-1}) = P(x_i|\text{parents}(X_i))$$
- 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 $A \rightarrow B \rightarrow C$ where $B$ is unobserved (either direction)
  - Common cause $A \leftarrow B \rightarrow C$ where $B$ is unobserved
  - Common effect (aka v-structure) $A \rightarrow B \leftarrow C$ 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 $X_i \not\perp\!
\not\!
\perp X_j \{X_{k_1}, \ldots, X_{k_n}\}$
- Shade all evidence nodes
- For all (undirected!) paths between and
  - Check whether path is active
    - If active return $\overbrace{X_i \not\perp\!
\not\!
\perp X_j \{X_{k_1}, \ldots, X_{k_n}\}}$

- (If reaching this point all paths have been checked and shown inactive)
  - Return $\underbrace{X_i \perp\!
\perp X_j \{X_{k_1}, \ldots, X_{k_n}\}}$
Given a Bayes net structure, can run d-separation to build a complete list of conditional independences that are necessarily true of the form

\[ X_i \perp X_j \mid \{X_{k_1}, \ldots, X_{k_n}\} \]

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:

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

Example: Enumeration

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

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

$$P(\text{+b})P(\text{+e})P(\text{+a} | \text{+b}, \text{+e})P(\text{+j} | \text{+a})P(\text{+m} | \text{+a}) +$$

$$P(\text{+b})P(\text{+e})P(\text{-a} | \text{+b}, \text{+e})P(\text{+j} | \text{-a})P(\text{+m} | \text{-a}) +$$

$$P(\text{+b})P(\text{-e})P(\text{+a} | \text{+b}, \text{-e})P(\text{+j} | \text{+a})P(\text{+m} | \text{+a}) +$$

$$P(\text{+b})P(\text{-e})P(\text{-a} | \text{+b}, \text{-e})P(\text{+j} | \text{-a})P(\text{+m} | \text{-a})$$
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)

\[
\begin{align*}
\Pr(R) & : & +r & 0.1 & -r & 0.9 \\
\Pr(T|R) & : & +r & +t & 0.8 & +r & -t & 0.2 & -r & +t & 0.1 & -r & -t & 0.9 \\
\Pr(L|T) & : & +t & +l & 0.3 & +t & -l & 0.7 & -t & +l & 0.1 & -t & -l & 0.9 \\
\end{align*}
\]

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

\[
\begin{align*}
\Pr(R) & : & +r & 0.1 & -r & 0.9 \\
\Pr(T|R) & : & +r & +t & 0.8 & +r & -t & 0.2 & -r & +t & 0.1 & -r & -t & 0.9 \\
\Pr(\ell|T) & : & +t & +l & 0.3 & +t & -l & 0.7 & -t & +l & 0.1 & -t & -l & 0.9 \\
\end{align*}
\]

- VE: Alternately join factors and eliminate variables
Variable Elimination Example

\[ P(R) \]

<table>
<thead>
<tr>
<th>(+r)</th>
<th>(-r)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.9</td>
</tr>
</tbody>
</table>

\[ P(T|R) \]

<table>
<thead>
<tr>
<th>(+r)</th>
<th>(+t)</th>
<th>(-t)</th>
<th>(-r)</th>
<th>(+t)</th>
<th>(-t)</th>
<th>(-r)</th>
<th>(-t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0.2</td>
<td>0.1</td>
<td>0.9</td>
<td>0.02</td>
<td>0.09</td>
<td>0.81</td>
<td>0.08</td>
</tr>
</tbody>
</table>

\[ P(L|T) \]

<table>
<thead>
<tr>
<th>(+t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0.1</td>
<td>0.9</td>
<td>0.3</td>
<td>0.7</td>
<td>0.1</td>
<td>0.9</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Join R

Sum out R

\[ P(R, T) \]

<table>
<thead>
<tr>
<th>(+r)</th>
<th>(+t)</th>
<th>(-t)</th>
<th>(-r)</th>
<th>(+t)</th>
<th>(-t)</th>
<th>(-r)</th>
<th>(-t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.08</td>
<td>0.02</td>
<td>0.09</td>
<td>0.81</td>
<td>0.17</td>
<td>0.83</td>
<td>0.08</td>
<td>0.09</td>
</tr>
</tbody>
</table>

\[ P(L|T) \]

<table>
<thead>
<tr>
<th>(+t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0.1</td>
<td>0.9</td>
<td>0.134</td>
<td>0.886</td>
<td>0.747</td>
<td>0.083</td>
<td>0.119</td>
</tr>
</tbody>
</table>

Join T

Sum out T

\[ P(T) \]

<table>
<thead>
<tr>
<th>(+t)</th>
<th>(-t)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.17</td>
<td>0.83</td>
</tr>
</tbody>
</table>

\[ P(T, L) \]

<table>
<thead>
<tr>
<th>(+t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
<th>(-t)</th>
<th>(+l)</th>
<th>(-l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.051</td>
<td>0.119</td>
<td>0.083</td>
<td>0.747</td>
<td>0.134</td>
<td>0.886</td>
<td>0.747</td>
<td>0.083</td>
<td>0.119</td>
</tr>
</tbody>
</table>

* 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

\[ P(A|B,E) \]
\[ P(j|A) \]
\[ P(m|A) \]

\[ \times \]
\[ \sum \]

\[ P(j,m,B,E) \]

\[ 2 \times 2 \]

\[ P(B) \]
\[ P(E) \]
\[ P(j,m|B,E) \]

Example

| \( P(B) \) | \( P(E) \) | \( P(j,m|B,E) \) |
|----------------|----------------|----------------|

Choose E

\[ P(E) \]
\[ P(j,m|B,E) \]

\[ \times \]
\[ \sum \]

\[ P(j,m|B) \]

\[ P(B) \]
\[ P(j,m|B) \]

Finish with B

\[ P(B) \]
\[ P(j,m|B) \]

\[ \times \]
\[ \overset{\text{Normalize}}{\times} \]

\[ P(B|j,m) \]
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

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 \( P(e_i | \text{value of parents}(e_i) \text{ in current sample}) \)

- **Gibbs sampling:** repeatedly samples each non-evidence variable conditioned on all other variables \( \Rightarrow \) can incorporate downstream evidence
Prior Sampling

![Prior Sampling Diagram]

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

26
Likelihood Weighting

\[ \mathcal{P}(\mathbf{z}) = \prod_{i=1}^{m} P(z_i | \text{Parents}(z_i)) \]

- 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

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

    \[
    P(X_1) \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow \ldots
    \]

- 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

    \[
    \pi(x_{\infty}) = \sum_{x} P(x_{\infty} \mid x) P(x)
    \]

- **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

\[ X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow \]
\[ E_1 \rightarrow E_2 \rightarrow E_3 \rightarrow E_4 \]

- 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 $P(X | \text{evidence})$
- 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:
  \[ P(x_t | e_{1:t}) \propto \prod_{X} P(x_t | e_{1:t-1}) \cdot P(e_t | x_t) \]
- The forward algorithm does both at once (and doesn’t normalize)

\[ L_0 = \text{variable depends on} \quad X_1 \ldots X_n \]
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

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.1</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>0.0</td>
<td>0.2</td>
<td>0.5</td>
</tr>
</tbody>
</table>

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 \( p(x_t | y_t) \)

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 $X \mid Y$

- **Empirically**: use training data
  - For each outcome $x$, look at the empirical rate of that value:
    \[
    P_{ML}(x) = \frac{\text{count}(x)}{\text{total samples}}
    \]
    - This is the estimate that maximizes the likelihood of the data

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

Bayes Nets Status

- **Representation**
- **Inference**
- **Learning Bayes Nets from Data**
Hello,

Do you want free prinr

cartriges? Why pay more

when you can get them

ABSOLUTELY FREE! Just

\[ x \rightarrow f(x) \rightarrow y \]

\# free : 2
YOUR_NAME : 0
MISSPELLED : 2
FROM_FRIEND : 0
...

+ SPAM

PIXEL-7,12 : 1
PIXEL-7,13 : 0
...
NUM_LOOPS : 1
...

"2"

---

**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 $Y$

$$y = \arg\max_y P(y|f_1 \ldots f_n)$$

- You already know how to do this inference

General Naïve Bayes

- A general \textit{naive Bayes} model:

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

- We only specify how each feature depends on the class
- Total number of parameters is \textit{linear} in $n$
- Our running example: digits
**Bag-of-Words Naïve Bayes**

- **Generative model**
  \[ P(Y, W_1 \ldots W_n) = P(Y) \prod_i P(W_i|Y) \]
  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

$$w_y = w_y - f(x)$$  \quad \text{if } f(x) \leq w_y \\ 
\text{if } f(x) \geq w_y$$

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:

  \[
  \min_w \frac{1}{2} \sum_y ||w_y - w'_y||^2
  \]

  Update $w$ by solving:

  \[
  w_y \cdot f(x) \geq w_y \cdot f(x) + 1
  \]

  \[
  w_y = w'_y - \tau f(x)
  \]

  \[
  w_y^* = w'^*_y + \tau f(x)
  \]

  \[
  \tau \leq C
  \]

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

\[
\min_w \frac{1}{2} ||w||^2
\]

\[
\forall i, y \ w_y \cdot f(x_i) \geq w_y \cdot f(x_i) + 1
\]
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

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

Φ: x → φ(x)
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:  
  \[ K(x, x') = (x \cdot x' + 1)^2 = \phi(x) \cdot \phi(x') \]
  \[ = \sum x_i x_j x'_i x'_j + 2 \sum x_i x'_i + 1 \]

For \( x \in \mathbb{R}^3 \):

\[ \phi(x) = [x_1 x_1 x_2 x_2 x_3 x_3 x_1 x_2 x_2 x_3 x_3 x_1 x_3 x_2 x_2 x_3 x_3 \sqrt{2x_1} \sqrt{2x_2} \sqrt{2x_3} 1] \]

Some Kernels (2)

- Polynomial kernel:  
  \[ K(x, x') = (x \cdot x' + 1)^d \]
  \[ = \phi(y) \cdot \phi(y') \]

For \( x \in \mathbb{R}^3 \):

\[ \phi(x) = [x_1^d x_2^d x_3^d \sqrt{dx_1^{d-1}} x_2 \sqrt{dx_2^{d-1}} x_3 \ldots \sqrt{dx_1} \sqrt{dx_2} \sqrt{dx_3} 1] \]

For \( x \in \mathbb{R}^n \) 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)

<table>
<thead>
<tr>
<th>Examples</th>
<th>2 Examples</th>
<th>10 Examples</th>
<th>100 Examples</th>
<th>10000 Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truth</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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

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

---

55
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: $k, \alpha$ (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
Feature-Based Ranking

\[ x = \text{“Apple Computers”} \]

\[ f(x, \cdot) = [0.3 \ 5 \ 0 \ 0 \ \ldots] \]

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):
    \[ w = w + f(x, y^*) - f(x, y) \]