Advanced Introduction to Machine Learning
— Spring Quarter, Week 8 —

https://canvas.uw.edu/courses/1372141

Prof. Jeff Bilmes

University of Washington, Seattle
Departments of: Electrical & Computer Engineering, Computer Science & Engineering
http://melodi.ee.washington.edu/~bilmes

May 18th/20th, 2020
Announcements

- HW4 is due May 29th, 6:00pm via our assignment dropbox (https://canvas.uw.edu/courses/1372141/assignments).
- Virtual office hours this week, Thursday night at 10:00pm via zoom (same link as class).
- Q: should we have class on May 25th, or do a final lecture on June 8th?
Class Road Map

- **W1(3/30,4/1)**: What is ML, Probability, Coins, Gaussians and linear regression, Associative Memories, Supervised Learning
- **W2(4/6,4/8)**: More supervised, logistic regression, complexity and bias/variance tradeoff
- **W3(4/13,4/15)**: Bias/Variance, Regularization, Ridge, CrossVal, Multiclass
- **W4(4/20,4/22)**: Multiclass classification, ERM, Gen/Disc, Naïve Bayes
- **W5(4/27,4/29)**: Lasso, Regularizers, Curse of Dimensionality
- **W6(5/4,5/6)**: Curse of Dimensionality, Dimensionality Reduction, \(k\)-NN
- **W7(5/11,5/13)**: \(k\)-NN, LSH, DTs, Bootstrap/Bagging, Boosting & Random Forests, GBDTs
- **W8(5/18,5/20)**: Graphs; Graphical Models (Factorization, Inference, MRFs, BNs);
- **W9(5/27,6/1)**: Learning Paradigms; Clustering; EM Algorithm;
- **W10(6/3,6/8)**: Spectral Clustering, Graph SSL, Deep models, (SVMs, RL); The Future.

Last lecture is 6/8 since 5/25 is holiday (or we could just have lecture on 5/25).
1. Introduction
- What is ML
- What is AI
- Why are we so interested in these topics right now?

2. ML Paradigms/Concepts
- Overfitting/Underfitting, model complexity, bias/variance
- size of data, big data, sample complexity
- ERM, loss + regularization, loss functions, regularizers
- Supervised, unsupervised, and semi-supervised learning;
  - reinforcement learning, RL, multi-agent, planning/control
  - transfer and multi-task learning
  - federated and distributed learning
  - active learning, machine teaching
  - self-supervised, zero/one-shot, open-set learning

3. Dealing with Features
- dimensionality reduction, PCA, LDA, MDS, T-SNE, UMAP
- Locality sensitive hashing (LSH)
- feature selection
- feature engineering
- matrix factorization & feature engineering
- representation learning

4. Evaluation
- accuracy/error, precision/recall, ROC, likelihood/posterior, cost/utility, margin
- train/eval/test data splits
- n-fold cross validation
- method of the bootstrap

5. Optimization Methods
- Unconstrained Continuous Optimization: (stochastic) gradient descent (SGD), adaptive learning rates, conjugate gradient, 2nd order Newton
- Constrained Continuous Optimization: Frank-Wolf (conditional gradient descent), projected gradient, linear, quadratic, and convex programming
- Discrete optimization - greedy, beam search, branch-and-bound, submodular optimization.

6. Inference Methods
- probabilistic inference
- MLE, MAP
- belief propagation
- forward/backpropagation
- Monte Carlo methods

7. Models & Representation
- linear least squares, linear regression, logistic regression, sparsity, ridge, lasso
- generative vs. discriminative models
- Naive Bayes
- k-nearest neighbors
- clustering, k-means, k-mediods, EM & GMMs, single linkage
- decision trees and random forests
- support vector machines, kernel methods, max margin
- perceptron, neural networks, DNNs
- Gaussian processes
- Bayesian nonparametric methods
- ensemble methods
- the bootstrap, bagging, and boosting
- graphical models
- time-series, HMMs, DBNs, RNNs, LSTMs, Attention, Transformers
- structured prediction
- grammars (as in NLP)

8. Philosophy, Humanity, Spirituality
- artificial intelligence (AI)
- artificial general intelligence (AGI)
- artificial intelligence vs. science fiction

9. Applications
- computational biology
- social networks
- computer vision
- speech recognition
- natural language processing
- information retrieval
- collaborative filtering/matrix factorization

10. Programming
- python
- libraries (e.g., NumPy, SciPy, matplotlib, scikit-learn (sklearn), pytorch, CNTK, Theano, tensorflow, keras, H2O, etc.
- HPC: C/C++, CUDA, vector processing

11. Background
- linear algebra
- multivariate calculus
- probability theory and statistics
- information theory
- mathematical (e.g., convex) optimization

12. Other Techniques
- compressed sensing
- submodularity, diversity/homogeneity modeling

1.2 Determinant
Let A be an n x n matrix.
\[ \text{det}(A) = \prod_{i=1}^{n} \lambda_i = \text{sg}(A) \]
\[ \text{det}(A^T) = \text{det}(A), \quad \text{if } A \in \mathbb{R}^{n \times n} \]
\[ \text{det}(AB) = \text{det}(A) \text{det}(B) \]
\[ \text{det}(A^{-1}) = \frac{1}{\text{det}(A)} \]
\[ \text{det}(A^T) = \text{det}(A) \]
\[ \text{det}(I + uu^T) = 1 + u^Tu \]
Regression with a tree, constant decisions

https://medium.com/x8-the-ai-community/decision-trees-an-intuitive-introduction-86c2b39c1a6c
Regression with a tree, linear decisions

For each season:
- Spring: Rainfall decreases linearly with month, represented by the equation $-0.2 \times \text{month} + 55$.
- Summer: Rainfall increases linearly with month, represented by the equation $1.8 \times \text{month} - 40$.
- Monsoon: Rainfall increases linearly with month, represented by the equation $0.8 \times \text{month} + 20$.
- Winter: Rainfall decreases linearly with month, represented by the equation $-0.9 \times \text{month} + 200$.

For months:
- If month < 6, go to Yes.
- If month < 3, go to Yes.
- If month > 9, go to Yes.
- If month < 6, go to No.
- If month < 3, go to No.
- If month > 9, go to No.

Equation for Rainfall:
Rainfall = \begin{cases} 
-0.2 \times \text{month} + 55 & \text{if } \text{month} < 6 \\
1.8 \times \text{month} - 40 & \text{if } 3 \leq \text{month} < 6 \\
0.8 \times \text{month} + 20 & \text{if } 6 \leq \text{month} < 9 \\
-0.9 \times \text{month} + 200 & \text{if } \text{month} \geq 9
\end{cases}
Decision Trees

- Partition space into rectangular regions, each region we make a final decision (i.e., regression value, or classification).
- Regions formed by recursively splits space based on binary questions about one coordinate of input at a time (i.e., “Is $x_7 < 92$?”)

Example: (From Hastie et. al 2009 text).

Corresponds to

$$h(x) = \sum_{i=1}^{5} c_i \mathbf{1}_{\{x \in R_i\}} \quad (8.12)$$
DT regression, Greedy strategy, binary splitting

- Splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes
  $$R_1(j, s) = \{ x : x_j \leq s \} \quad R_2(j, s) = \{ x : x_j > s \}$$  \hspace{1cm} (8.15)

- Use optimization, find the $j, s$ that achieves the best (minimum) in:
  $$\min_{j, s} \left[ \sum_{x(i) \in R_1(j,s)} (y(i) - \hat{c}_1)^2 + \sum_{x(i) \in R_2(j,s)} (y(i) - \hat{c}_2)^2 \right]$$  \hspace{1cm} (8.16)

  where $\hat{c}_k = \text{average}(y(i) | x(i) \in R_k)$.

- For each $j$, can find $s$ by sorting the data via $x_j$ and scanning in order to achieve min. Select that $j, s$ as the split point.

- Can also be seen as maximizing the $\text{MSE gain}$

  $$\max_{j, s} \left( \sum_{x(i)} (y(i) - \hat{c}_0)^2 - \left[ \sum_{x(i) \in R_1(j,s)} (y(i) - \hat{c}_1)^2 + \sum_{x(i) \in R_2(j,s)} (y(i) - \hat{c}_2)^2 \right] \right)$$  \hspace{1cm} (8.17)
Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for a $x \in \mathbb{R}^m$. With $n_k = \text{count of training samples in region } k$, we have:

$$
\hat{p}(y = j | x) = \frac{1}{n_k(x)} \sum_{x(i) \in R_k(x)} 1\{y(i) = j\} = \hat{p}_k(x)(y = j) \tag{8.1}
$$

where $k(x)$ is the region containing $x$. Make decisions using $\hat{y}(x) = \arg\max_j \hat{p}(y = j | x)$. So region $k$ always decides class $\hat{y}_k$. 

With this, we can use the same greedy strategy to grow the tree from top down. How to judge each region? Several ways:

- Classification Error

$$
\frac{1}{n_k(x)} \sum_{x(i) \in R_k(x)} 1\{y(i) \neq \hat{y}_k\} \tag{8.2}
$$

- Entropy in region $k$

$$
\sum_{j=1}^J \hat{p}_k(y = j) \log \hat{p}_k(y = j) \tag{8.3}
$$
Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for $x \in \mathbb{R}^m$. With $n_k = \text{count of training samples in region } k$, we have:

$$
\hat{p}(y = j \mid x) = \frac{1}{n_k(x)} \sum_{x(i) \in R_k(x)} 1\{y(i) = j\} = \hat{p}_k(x)(y = j)
$$

where $k(x)$ is the region containing $x$. Make decisions using $\hat{y}(x) = \text{argmax}_j \hat{p}(y = j \mid x)$. So region $k$ always decides class $\hat{y}_k$.

- With this we can use the same greedy strategy to grow the tree from top down.
Classification Trees

- At each leaf, predict a class rather than a real value. Produce region-specific posterior distribution to make classification decisions for a $x \in \mathbb{R}^m$. With $n_k = \text{count of training samples in region } k$, we have:

$$\hat{p}(y = j | x) = \frac{1}{n_k(x)} \sum_{x(i) \in R_k(x)} \mathbf{1}_{\{y(i) = j\}} = \hat{p}_k(x)(y = j) \quad (8.1)$$

where $k(x)$ is the region containing $x$. Make decisions using $\hat{y}(x) = \arg \max_j \hat{p}(y = j | x)$. So region $k$ always decides class $\hat{y}_k$.

- With this we can use the same greedy strategy to grow the tree from top down.

- How to judge each region? Several ways:

  1. Classification Error
     $$\frac{1}{n_k} \sum_{i \in R_k} \mathbf{1}_{\{y(i) \neq \hat{y}_k\}} \quad (8.2)$$

  2. Entropy in region $k$
     $$- \sum_{j=1}^{\ell} \hat{p}_k(y = j) \log \hat{p}_k(y = j) \quad (8.3)$$
Definition 8.3.1 (Entropy)

Given a discrete random variable $X$ over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \left( \log \frac{1}{p(x)} \right) = - \sum_x p(x) \log p(x) \quad (8.4)$$

- $1/p(x)$ is surprise of $x$, $- \log p(x)$ is log surprise.
Entropy

Definition 8.3.1 (Entropy)

Given a discrete random variable $X$ over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = - \sum_x p(x) \log p(x) \quad (8.4)$$

- $1/p(x)$ is surprise of $x$, $- \log p(x)$ is log surprise.
- Entropy is typically units of “bits” (logs base 2) but also can be units of “nats” (logs base $e$). For optimization it doesn’t matter.
Definition 8.3.1 (Entropy)

Given a discrete random variable $X$ over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x) \quad (8.4)$$

- $1/p(x)$ is surprise of $x$, $-\log p(x)$ is log surprise.
- Entropy is typically units of “bits” (logs base 2) but also can be units of “nats” (logs base $e$). For optimization it doesn’t matter.
- Measures the degree of uncertainty in a distribution.
Definition 8.3.1 (Entropy)

Given a discrete random variable $X$ over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = - \sum_x p(x) \log p(x) \quad (8.4)$$

- $1/p(x)$ is surprise of $x$, $- \log p(x)$ is log surprise.
- Entropy is typically units of “bits” (logs base 2) but also can be units of “nats” (logs base $e$). For optimization it doesn’t matter.
- Measures the degree of uncertainty in a distribution.
- Measures the disorder or spread of a distribution.
**Definition 8.3.1 (Entropy)**

Given a discrete random variable $X$ over a finite sized alphabet, the entropy of the random variable is:

$$H(X) \triangleq E \log \frac{1}{p(X)} = \sum_x p(x) \log \frac{1}{p(x)} = - \sum_x p(x) \log p(x) \quad (8.4)$$

- $1/p(x)$ is surprise of $x$, $-\log p(x)$ is log surprise.
- Entropy is typically units of “bits” (logs base 2) but also can be units of “nats” (logs base e). For optimization it doesn’t matter.
- Measures the degree of uncertainty in a distribution.
- Measures the disorder or spread of a distribution.
- Measures the “choice” that a source has in choosing symbols according to the density (higher entropy means more choice).
Entropy Of Distributions

Low Entropy

High Entropy

In Between
Binary Entropy

- Binary alphabet, $X \in \{0, 1\}$ say.
- $p(X = 1) = p = 1 - p(X = 0)$.
- $H(X) = -p \log p - (1 - p) \log(1 - p) = H(p)$.
- As a function of $p$, we get:

Note, greatest uncertainty (value 1) when $p = 0.5$ and least uncertainty (value 0) when $p = 0$ or $p = 1$.

Note also: concave in $p$. 
Classification Trees and Entropy

- We can measure entropy in each region.

Entropy in region $k$:

$$- \sum_{j=1}^{l} \hat{p}_k(y = j) \log \hat{p}_k(y = j)$$
Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \quad R_2(j, s) = \{x : x_j > s\} \quad (8.5)$$
Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \quad R_2(j, s) = \{x : x_j > s\}$$ (8.5)

- A region’s posterior is essentially a histogram of class labels in region.
Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes

$$R_1(j, s) = \{x : x_j \leq s\} \quad R_2(j, s) = \{x : x_j > s\} \quad (8.5)$$

- A region’s posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable $j$ and split $s$ that, when made, reduces the entropy the most.
Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable $j \in [m]$ with split point $s \in \mathbb{R}$, defines two half planes
  \[
  R_1(j, s) = \{x : x_j \leq s\} \quad R_2(j, s) = \{x : x_j > s\} \quad (8.5)
  \]
  - A region’s posterior is essentially a histogram of class labels in region.
  - Starting from one large region, we find the variable $j$ and split $s$ that, when made, reduces the entropy the most.
  - Maximizing information gain, split region $R_0$ into $R_1(j, s)$ and $R_2(j, s)$

  \[
  \max_{j,s} \left[ H(p_0(y)) - (H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y))) \right] \quad (8.6)
  \]
Classification Trees and Entropy

- We can measure entropy in each region.
- Again, splitting variable \( j \in [m] \) with split point \( s \in \mathbb{R} \), defines two half planes

\[
R_1(j, s) = \{ x : x_j \leq s \} \quad R_2(j, s) = \{ x : x_j > s \}
\]

- A region’s posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable \( j \) and split \( s \) that, when made, reduces the entropy the most.
- Maximizing information gain, split region \( R_0 \) into \( R_1(j, s) \) and \( R_2(j, s) \)

\[
\max_{j,s} \left[ H(p_0(y)) - \left( H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y)) \right) \right]
\]

- Leads to regions with lower entropy, higher certainty, least diversity in each region, creating homogeneous regions.
**Classification Trees and Entropy**

- We can measure entropy in each region.
- Again, splitting variable \( j \in [m] \) with split point \( s \in \mathbb{R} \), defines two half planes

\[
R_1(j, s) = \{x : x_j \leq s\} \quad R_2(j, s) = \{x : x_j > s\} \tag{8.5}
\]

- A region’s posterior is essentially a histogram of class labels in region.
- Starting from one large region, we find the variable \( j \) and split \( s \) that, when made, reduces the entropy the most.
- Maximizing information gain, split region \( R_0 \) into \( R_1(j, s) \) and \( R_2(j, s) \)

\[
\max_{j, s} \left[ H(p_0(y)) - (H(p_{R_1(j,s)}(y)) + H(p_{R_2(j,s)}(y))) \right] \tag{8.6}
\]

- Leads to regions with lower entropy, higher certainty, least diversity in each region, creating homogeneous regions.
- **Term:** CART, for classification and regression trees.
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like $k$-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different $y$ for the same $x$). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like $k$-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different $y$ for the same $x$). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).

- Top-down Binary split tree can’t achieve all regions, ex:

![Decision Tree Diagram]

$X_1$ $X_2$
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like $k$-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different $y$ for the same $x$). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).

Top-down Binary split tree can’t achieve all regions, ex:

- DTs can be constructed using other procedures that can do this.
Trees, Expressivity, Bias, and Variance

- Decision trees are extremely flexible.
- Like $k$-NN methods, DT methods in general have high variance and low bias (they can flexibly fit any data sets). With a tall tree, a DT can perfectly fit any training data with non-conflicting labels (i.e., different $y$ for the same $x$). Like nearest neighbor, but neighborhoods are rectangular (for top-down greedy tree procedure).

- Top-down Binary split tree can’t achieve all regions, ex:

- DTs can be constructed using other procedures that can do this.
- Another advantage of trees, they lead to interpretable machine learning models since all decisions are based on original inputs, rather than based on mysterious learnt non-convex combinations thereof.
Bootstrap

- Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.
Bootstrap

- Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.

- Approach: given data set $\mathcal{D}$ of size $n$, produce $B$ new data sets of size $n$ by sampling samples from $\mathcal{D}$ with replacement leading to $\{\mathcal{D}_b\}_{b=1}^B$. 

- Useful for standard errors, confidence intervals (e.g., percentile points, proportions, odds ratio, and correlation coefficients), and also stability and sensitivity of the results. Sometimes asymptotically consistent (see van der Vaart, "Asymptotic Statistics", Chapter 23).

- Also useful to produce ensemble of models (next slide).
Bootstrap

- Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.

- Approach: given data set $D$ of size $n$, produce $B$ new data sets of size $n$ by sampling samples from $D$ with replacement leading to $\{D_b\}_{b=1}^B$.

- Any learning (or statistical) procedure can be tested on each $D_b$, and means, variances, errors, entropies, predictors, etc. can be computed.
Bootstrap

- Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.

- Approach: given data set $D$ of size $n$, produce $B$ new data sets of size $n$ by sampling samples from $D$ with replacement leading to $\{D_b\}_{b=1}^B$.

- Any learning (or statistical) procedure can be tested on each $D_b$, and means, variances, errors, entropies, predictors, etc. can be computed.

- Example: estimated error of an ML procedure

  \[
  \text{err boot} = \frac{1}{B} \sum_{b=1}^B \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, h_{\theta_b}(x^{(i)})) \tag{8.7}
  \]

  where and $\theta_b$ are the parameters learnt from $D_b$. 
Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.

Approach: given data set $\mathcal{D}$ of size $n$, produce $B$ new data sets of size $n$ by sampling samples from $\mathcal{D}$ with replacement leading to $\{\mathcal{D}_b\}_{b=1}^B$.

Any learning (or statistical) procedure can be tested on each $\mathcal{D}_b$, and means, variances, errors, entropies, predictors, etc. can be computed.

Example: estimated error of an ML procedure

$$
\text{err boot} = \frac{1}{B} \sum_{b=1}^B \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, h_{\theta_b}(x^{(i)}))
$$

(8.7)

where and $\theta_b$ are the parameters learnt from $\mathcal{D}_b$.

Useful for standard errors, confidence intervals (e.g., percentile points, proportions, odds ratio, and correlation coefficients), and also stability and sensitivity of the results. Sometimes asymptotically consistent (see van der Vaart, “Asymptotic Statistics”, Chapter 23).
Bootstrap

- Bootstrap (Efron & Tibshirani, 1998), a computational method to estimate accuracy of a statistical or machine learning procedure, and to mimic the availability of several different datasets by resampling multiple datasets from a given single dataset.

- Approach: given data set $D$ of size $n$, produce $B$ new data sets of size $n$ by sampling samples from $D$ with replacement leading to $\{D_b\}_{b=1}^B$.

- Any learning (or statistical) procedure can be tested on each $D_b$, and means, variances, errors, entropies, predictors, etc. can be computed.

- Example: estimated error of an ML procedure

$$
\text{err boot} = \frac{1}{B} \sum_{b=1}^B \frac{1}{n} \sum_{i=1}^n L(y^{(i)}, h_{\theta_b}(x^{(i)}))
$$

(8.7)

where and $\theta_b$ are the parameters learnt from $D_b$.

- Useful for standard errors, confidence intervals (e.g., percentile points, proportions, odds ratio, and correlation coefficients), and also stability and sensitivity of the results. Sometimes asymptotically consistent (see van der Vaart, “Asymptotic Statistics”, Chapter 23).

- Also useful to produce ensemble of models (next slide).
Bagging

- **Bootstrap aggregating (bagging)** is a meta-algorithm based on averaging the results of multiple bootstrap samples.
Bootstrap aggregating (bagging) is a meta-algorithm based on averaging the results of multiple bootstrap samples.

Let $\theta_b$ be the model parameters learnt with bootstrap sample $b$, then we can produce model (regression context)

$$\hat{f}_{bagging}(x) = \frac{1}{B} \sum_{b=1}^{B} h_{\theta_b}(x)$$  \hspace{1cm} (8.8)
Bagging

- Bootstrap aggregating (bagging) is a meta-algorithm based on averaging the results of multiple bootstrap samples.
- Let $\theta_b$ be the model parameters learnt with bootstrap sample $b$, then we can produce model (regression context)

$$f_{bagging}(x) = \frac{1}{B} \sum_{b=1}^{B} h_{\theta_b}(x)$$  \hspace{1cm} (8.8)

- When model is linear in the parameters, bagging estimate $\rightarrow$ original estimate as $B \rightarrow \infty$. More interesting when ML procedure is non-linear (e.g., classification or regression trees, neural networks, etc.).
**Bagging**

- **Bootstrap aggregating (bagging)** is a meta-algorithm based on averaging the results of multiple bootstrap samples.

- Let $\theta_b$ be the model parameters learnt with bootstrap sample $b$, then we can produce model (regression context)

$$
\hat{f}_{\text{bagging}}(x) = \frac{1}{B} \sum_{b=1}^{B} h_{\theta_b}(x)
$$

- When model is linear in the parameters, bagging estimate $\to$ original estimate as $B \to \infty$. More interesting when ML procedure is non-linear (e.g., classification or regression trees, neural networks, etc.).

- Bagging can be very useful to reduce variance of an estimated quantity or function.
Boosting

- **Boosting** is an ensemble machine learning method (i.e., where multiple learnt models are combined to produce an aggregated model, e.g., by averaging).
Boosting

- Boosting is an ensemble machine learning method (i.e., where multiple learnt models are combined to produce an aggregated model, e.g., by averaging).
- Boosting has many different forms, strategies, and analyses. Basic idea: ensemble members created successively. Each next member of the ensemble is created to correct for problems in previously learnt members.

[Diagram showing single, bagging, and boosting processes]

Classic example of boosting is Adaboost ("adaptive boosting" by Freund and Schapire (1996)), although there has been much work and many variations since then. We'll describe Adaboost in detail.
Boosting

- **Boosting** is an ensemble machine learning method (i.e., where multiple learnt models are combined to produce an aggregated model, e.g., by averaging).

- Boosting has many different forms, strategies, and analyses. Basic idea: ensemble members created successively. Each next member of the ensemble is created to correct for problems in previously learnt members.

- Classic example of boosting is Adaboost (“adaptive boosting” by Freund and Schapire (1996)), although there has been much work and many variations since then.
Boosting

Boosting is an ensemble machine learning method (i.e., where multiple learnt models are combined to produce an aggregated model, e.g., by averaging).

Boosting has many different forms, strategies, and analyses. Basic idea: ensemble members created successively. Each next member of the ensemble is created to correct for problems in previously learnt members.

Classic example of boosting is Adaboost ("adaptive boosting" by Freund and Schapire (1996)), although there has been much work and many variations since then.

We’ll describe Adaboost in detail.
Algorithm 2: Classic Adaboost Algorithm

**Input:** Training data set $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^n$ size $n$

**Output:** Ensemble of models predictor trained via boosting

1. Initialize weights $\{w_i\}_{i=1}^n$ via $w_i^{(0)} = 1/n$ for all $i \in [n]$ ;
2. for $t = 1$ to $T$ do
3. Fit classifier $h_{\theta_t}$ to training data by minimizing objective $J_t(\theta_t) = \sum_{i=1}^n w_i^{(t-1)} L(y^{(i)}, h_{\theta_t}(x^{(i)}))$ where $L(\cdot, \cdot)$ is 0/1-loss, giving parameters $\theta_t^*$;
4. Compute value $\alpha_t = (1/2) \log((1 - \epsilon_t)/\epsilon_t)$ where $\epsilon_t \in (0, 1)$ and $\epsilon_t = \frac{\sum_{i=1}^n w_i^{(t-1)} L(y^{(i)}, h_{\theta_t^*}(x^{(i)}))}{\sum_{i=1}^n w_i^{(t-1)}}$ is normalized weighted avg. loss.
5. Produce new weights $w_i^t \leftarrow w_i^{t-1} \exp(-\alpha_t \cdot y^{(i)} \cdot h_{\theta_t^*}(x^{(i)}))/Z_t, \forall i$;
6. Output the ensemble of models predictor $h(x) = \left(\sum_{t=1}^T \alpha_t h_{\theta_t^*}(x)\right)$

Note, $Z_t$ ensures weights always sum to 1 and is good numerically, but we don’t mathematically need to normalize since we compute the normalized error $\epsilon_t$. $\epsilon_t \in (0, 1)$ indicates how poorly we did (weighted error) at round $t$. $\alpha_t \in \mathbb{R}$ is a goodness value, $\alpha_t \gg 0$ means we did well, $\alpha_t \ll 0$ we did poorly on average.
\( d_t = 0 \Rightarrow \) no weight updates.

\( d_t > 0 \), \( \beta_t \) model did not well than lat. \((\delta_t \leq 0.5)\)

\( y(i) - \log p(x|y(i)) \Rightarrow \) minimise \( \exp(-d) \Rightarrow \) decrease import of \( i \)

\( y(i) = \log p(x|y(i)) \Rightarrow \exp(d) \Rightarrow \) increase import of \( i \).

\( h < 0 \), \( \alpha_t \) do not poor.

Correct prediction on \( i \) \( \Rightarrow \) update \( \alpha_t(x) \) \( \Rightarrow \) increase importance of \( i \).

Incorrect prediction on \( i \) \( \Rightarrow \) update \( \exp(-d) \) \( \Rightarrow \) decrease importance of \( i \).
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
- Random forests build a ensemble of uncorrelated trees to average over. Idea: average noisy (high variance) but unbiased models, so that overall variance is reduced in the average.
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
- Random forests build an ensemble of uncorrelated trees to average over. 
  Idea: average noisy (high variance) but unbiased models, so that 
  overall variance is reduced in the average.
- If each of $B$ i.i.d. random variables has variance $\text{Var}(X_i) = \sigma^2$ for $b \in \{1, 2, \ldots, B\}$, then average is lower since 
  $\text{Var}\left(\frac{1}{B} \sum_{b=1}^{B} X_i\right) = \frac{\sigma^2}{B}$. 

Random forests are built into sklearn
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
- Random forests build a ensemble of uncorrelated trees to average over. Idea: average noisy (high variance) but unbiased models, so that overall variance is reduced in the average.
- If each of $B$ i.i.d. random variables has variance $\text{Var}(X_i) = \sigma^2$ for $b \in \{1, 2, \ldots, B\}$, then average is lower since $\text{Var}\left(\frac{1}{B} \sum_{b=1}^{B} X_i\right) = \sigma^2 / B$.
- If we do this with models, and they are (almost) i.i.d., we can reduce variance as well.

Random forests are fairly good at not overfitting. Random forests are built into sklearn https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
- Random forests build an ensemble of uncorrelated trees to average over.
  Idea: average noisy (high variance) but unbiased models, so that overall variance is reduced in the average.
- If each of $B$ i.i.d. random variables has variance $\text{Var}(X_i) = \sigma^2$ for $b \in \{1, 2, \ldots, B\}$, then average is lower since $\text{Var}(\frac{1}{B} \sum_{b=1}^{B} X_i) = \sigma^2 / B$.
- If we do this with models, and they are (almost) i.i.d., we can reduce variance as well.
- Random forests are fairly good at not overfitting.
Random Forests

- Bagging (bootstrap aggregating) + Decision Trees = Random Forests.
- Random forests build a ensemble of uncorrelated trees to average over.
  Idea: average noisy (high variance) but unbiased models, so that overall variance is reduced in the average.
- If each of $B$ i.i.d. random variables has variance $\text{Var}(X_i) = \sigma^2$ for $b \in \{1, 2, \ldots, B\}$, then average is lower since $\text{Var}(\frac{1}{B} \sum_{b=1}^{B} X_i) = \sigma^2 / B$.
- If we do this with models, and they are (almost) i.i.d., we can reduce variance as well.
- Random forests are fairly good at not overfitting.
- Random forests are built into sklearn
Random Forest Construction Algorithm

From Hastie et al. text

**Algorithm 3:** Random Forest for Regression or Classification

1. **for** $b = 1$ to $B$ **do**
2. Create a bootstrap sample of data $D_b$;
3. Grow a random-forest tree $T_b$ from $D_b$ by recursively repeating the following steps for each terminal node of the tree, until minimum node size $t_{\text{min}}$ is reached.
   i. Pick $m' < m$ variables at random from the $m$ variables
   ii. Pick the best variable/split-point among the $m'$ chosen variables.
   iii. split the node into two child nodes.
4. Output the ensemble of trees $\{T_b\}_{b=1}^B$
5. In regression case, use $\frac{1}{B} \sum_{b=1}^B T_b(x)$.
6. In classification case, let each tree vote (e.g., choose maximum posterior probability for each tree), and create final decision by democratic majority vote.
Gradient Boosted Decision Trees (GBDTs)

- This is another way to produce an ensemble of decision trees, can be used either for regression or classification.
Gradient Boosted Decision Trees (GBDTs)

- This is another way to produce an ensemble of decision trees, can be used either for regression or classification.

- Like AdaBoost, Gradient Boosting sequentially adds predictors to an ensemble, next one correcting predecessors. Here, rather than changing weights, Gradient Boosting fits each new predictor to the residual errors made by the previously learnt predictors.
Gradient Boosted Decision Trees (GBDTs)

- This is another way to produce an ensemble of decision trees, can be used either for regression or classification.
- Like AdaBoost, Gradient Boosting sequentially adds predictors to an ensemble, next one correcting predecessors. Here, rather than changing weights, Gradient Boosting fits each new predictor to the residual errors made by the previously learnt predictors.
- Gradient Boosted Decision Trees (GBDTs) do this with decision trees, each time learning a tree via standard greedy gain maximization.

Gradient Boosted Decision Trees (GBDTs)

Conditioning on the predictions of the previous \( k - 1 \) trees, the objective used to optimize tree-\( k \) is the following:

\[
F_k(\theta_k) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^k) + \Omega(\theta_k) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{k-1} + f_k(x_i)) + \Omega(\theta_k),
\]

Each tree’s prediction is based on fixing up the (additive) aggregation \( \hat{y}_i^{k-1} \) of all the previous trees predictions, and the loss is optimized to correct for previous deficiencies in the aggregate prediction.
Conditioning on the predictions of the previous $k - 1$ trees, the objective used to optimize tree-$k$ is the following:

$$F_k(\theta_k) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^k) + \Omega(\theta_k) = \sum_{i=1}^{n} \ell(y_i, \hat{y}_i^{k-1} + f_k(x_i)) + \Omega(\theta_k),$$

Each tree’s prediction is based on fixing up the (additive) aggregation $\hat{y}_i^{k-1}$ of all the previous trees predictions, and the loss is optimized to correct for previous deficiencies in the aggregate prediction.

It involves gradients since we can use a Taylor approximation and use gradients to express the above objective in terms of the regression predictions made by the $k^{th}$ tree, and then use the same greedy procedure we used above to maximize MSE gain (for regression) or information gain (for classification).
GBDTs have become popular of late because they are general, fast to train, relatively easy to get working, and perform reasonably well on many different types of data (a good alternative to Deep Neural Networks for these reasons).
GBDTs

- GBDTs have become popular of late because they are general, fast to train, relatively easy to get working, and perform reasonably well on many different types of data (a good alternative to Deep Neural Networks for these reasons).

- There is a detailed description of GBDTs as part of Homework 4, where you will be implementing both random forests (RFs) and GBDTs from scratch.
GBDTs have become popular of late because they are general, fast to train, relatively easy to get working, and perform reasonably well on many different types of data (a good alternative to Deep Neural Networks for these reasons).

There is a detailed description of GBDTs as part of Homework 4, where you will be implementing both random forests (RFs) and GBDTs from scratch.

See also the following interactive web graphics that allow you to play around with GBDTs to gain intuition:

Gradient Boosting explained
http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

and also Gradient Boosting Interactive Playground
http://arogozhnikov.github.io/2016/07/05/gradient_boosting_playground.html
A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.
Graphs

- A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.
- The objects might be data items/frames, or random variables, or any other object that can be represented by data.
Graphs

- A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.
- The objects might be data items/frames, or random variables, or any other object that can be represented by data.
- Relationships between objects are expressed via connectivity patterns.
A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.

The objects might be data items/frames, or random variables, or any other object that can be represented by data.

Relationships between objects are expressed via connectivity patterns.

**Definition 8.7.1 (Graph)**

A graph $G$ consists of a finite set $V$ of objects called nodes (or vertices), as well as an (ordered) binary relation $E$ on $V$ consisting of pairs of elements from $V$ called edges (or links). We say graph consists of a tuple $G = (V, E)$ where $E \subseteq V \times V$. 
Graphs

- A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.
- The objects might be data items/frames, or random variables, or any other object that can be represented by data.
- Relationships between objects are expressed via connectivity patterns.

**Definition 8.7.1 (Graph)**

A graph $G$ consists of a finite set $V$ of objects called nodes (or vertices), as well as an (ordered) binary relation $E$ on $V$ consisting of pairs of elements from $V$ called edges (or links). We say graph consists of a tuple $G = (V, E)$ where $E \subseteq V \times V$.

- We will say a graph $G$ has $|V| = n$ nodes and $|E| = m$ edges.
Graphs

- A graphical model is a general, visual, and mathematically formal description of properties of a set of relationships between objects.
- The objects might be data items/frames, or random variables, or any other object that can be represented by data.
- Relationships between objects are expressed via connectivity patterns.

**Definition 8.7.1 (Graph)**

A graph $G$ consists of a finite set $V$ of objects called nodes (or vertices), as well as an (ordered) binary relation $E$ on $V$ consisting of pairs of elements from $V$ called edges (or links). We say graph consists of a tuple $G = (V, E)$ where $E \subseteq V \times V$.

- We will say a graph $G$ has $|V| = n$ nodes and $|E| = m$ edges.
- If the edges $e \in E$ take the form $e = (u, v)$, we say it is a directed graph where $e = (u, v) = (u \rightarrow v)$ and is called a directed graph. If the edges take the form $e = \{u, v\}$, then the graph is an undirected graph.
Example Undirected Graphs
Example Directed Graphs

Earthquake → B (Burglary)
Radio Report → A
Alarm Sound → A
Call to Police → C

Diagram:
- Directed edges represent dependencies among variables:
  - Earthquake impacts Burglary.
  - Call to Police affects Earthquake.

Graphical Models illustrate complex relationships among variables using nodes and directed edges.

1. Directed Graphs
2. Graphical Models
3. DTs, Bootstrap/Bagging, Boosting & Random Forests, GBDTs

Example directed graph at bottom left:
- Nodes labeled with variables:
  - X1, X2, X3, X4, X5
  - Additional nodes with numbers:
    - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

Diagram at right:
- Various nodes and edges
- Complex network structure

Example for Earthquake, Burglary, Radio Report, Alarm Sound, Call to Police.
More on Graphs

- Graphs: model interactions among various objects.
More on Graphs

- Graphs: model interactions among various objects.
- A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.

Real-world graphs can be extremely large, containing billions or trillions of vertices and edges!!
More on Graphs

- Graphs: model interactions among various objects.
- A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.
- If $u, v \in V$ then $(u, v) \in E$ means that nodes $u$ and $v$ are connected.
More on Graphs

- Graphs: model interactions among various objects.
- A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.
- If $u, v \in V$ then $(u, v) \in E$ means that nodes $u$ and $v$ are connected.
- An edge may be given a weight $w(e) = w(\{u, v\}) \in \mathbb{R}$ (often $w(e) \in \mathbb{R}_+$) indicating the strength of the connection between $u$ and $v$. 
More on Graphs

- Graphs: model interactions among various objects.
- A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.
- If $u, v \in V$ then $(u, v) \in E$ means that nodes $u$ and $v$ are connected.
- An edge may be given a weight $w(e) = w\{u, v\} \in \mathbb{R}$ (often $w(e) \in \mathbb{R}_+$) indicating the strength of the connection between $u$ and $v$.
- Vertices $v \in V$ and edges $e \in E$ can have types and/or metadata.
Graphs: model interactions among various objects.

A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.

If $u, v \in V$ then $(u, v) \in E$ means that nodes $u$ and $v$ are connected.

An edge may be given a weight $w(e) = w(\{u, v\}) \in \mathbb{R}$ (often $w(e) \in \mathbb{R}^+$) indicating the strength of the connection between $u$ and $v$.

Vertices $v \in V$ and edges $e \in E$ can have types and/or metadata.

Almost unlimited applications of graphs: biology, social network and web analysis, communication networks, computer hardware, data mining, semi-supervised and unsupervised learning, statistics and probabilistic inference, machine learning, smart city applications, etc.
More on Graphs

- Graphs: model interactions among various objects.
- A graph $G = (V, E)$ contains vertices $V$ and edges $E$. A vertex $v \in V$ represents an object of interest, and an edge $e = (u, v)$ between two vertices $u, v \in V$ represents ability for direct interaction between the two corresponding objects.
- If $u, v \in V$ then $(u, v) \in E$ means that nodes $u$ and $v$ are connected.
- An edge may be given a weight $w(e) = w(\{u, v\}) \in \mathbb{R}$ (often $w(e) \in \mathbb{R}_+$) indicating the strength of the connection between $u$ and $v$.
- Vertices $v \in V$ and edges $e \in E$ can have types and/or metadata.
- Almost unlimited applications of graphs: biology, social network and web analysis, communication networks, computer hardware, data mining, semi-supervised and unsupervised learning, statistics and probabilistic inference, machine learning, smart city applications, etc.
- Real-world graphs can be extremely large, containing billions or trillions of vertices and edges!!
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., \( n \) nodes and 
  \[ m = n(n - 1)/2 \] edges.
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., $n$ nodes and $m = n(n - 1)/2$ edges.
- A graph is a tree if it is connected and has $n - 1$ edges (equivalently, there is a unique path between every two nodes).
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., \( n \) nodes and \( m = n(n - 1)/2 \) edges.
- A graph is a tree if it is connected and has \( n - 1 \) edges (equivalently, there is a unique path between every two nodes).
- A graph is a forest if it is composed of a bunch of trees.
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., $n$ nodes and $m = n(n - 1)/2$ edges.
- A graph is a tree if it is connected and has $n - 1$ edges (equivalently, there is a unique **path** between every two nodes).
- A graph is a forest if it is composed of a bunch of trees.
- A graph is a chain if it is a linear sequence of connected nodes.
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., $n$ nodes and $m = n(n - 1)/2$ edges.
- A graph is a tree if it is connected and has $n - 1$ edges (equivalently, there is a unique **path** between every two nodes).
- A graph is a forest if it is composed of a bunch of trees.
- A graph is a chain if it is a linear sequence of connected nodes.
- A graph is a planar if it may be 2D plotted w/o any edge crossings.
Properties of a graph

Loosely speaking, a graph may be:

- dense (many edges) or sparse (few edges) relative to the number of nodes.
- A graph is a **clique** if it is fully connected, i.e., \( n \) nodes and \( m = n(n - 1)/2 \) edges.
- A graph is a tree if it is connected and has \( n - 1 \) edges (equivalently, there is a unique **path** between every two nodes).
- A graph is a forest if it is composed of a bunch of trees.
- A graph is a chain if it is a linear sequence of connected nodes.
- A graph is a planar if it may be 2D plotted w/o any edge crossings.
- A graph is a grid if it is planar and has a consistent connectivity pattern.
Graphs can describe data flow

- Graphs can be used to describe computer networks and the flow of data through it.
Graphs can describe data flow

- Graphs can be used to describe computer networks and the flow of data through it.
- **Classic example in machine learning.** Graphs used to describe deep neural networks.
Graphs can describe data flow

- Graphs can be used to describe computer networks and the flow of data through it.
- Classic example in machine learning. Graphs used to describe deep neural networks.

Graph shows connectivity pattern, and series of matrix-vector multiplications followed by non-linearities.
Let $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ be a training set and let $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ be a distance measure (e.g., $d(x, x') = \|x - x'\|$).
Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ be a training set and let $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ be a distance measure (e.g., $d(x, x') = \|x - x'\|$).

We can represent the distances as a weighted graph. Ex: $n = 5$, fully connected graph clique. 2-NN graph. 1-NN graph.
**$k$-NN Graph**

- Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ be a training set and let $d : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ be a distance measure (e.g., $d(x, x') = \|x - x'\|$).

- We can represent the distances as a weighted graph. Ex: $n = 5$, fully connected graph **clique**. 2-NN graph. 1-NN graph.

- Note, $k$-NN graph need not be symmetric. Often, we symmetrize the $k$-NN graph, so some nodes might have more than $k$ incident edges.
Matrix representation of graphs

- **adjacency matrix**: $n \times n$ binary matrix indicating if edge $\{u, v\} \in E$, where $n = |V|$ and $G = (V, E)$. Undirected graph, symmetric matrix.

\[
\begin{pmatrix}
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 
\end{pmatrix}
\]
Matrix representation of graphs

- **adjacency matrix**: \( n \times n \) binary matrix indicating if edge \( \{u, v\} \in E \), where \( n = |V| \) and \( G = (V, E) \). Undirected graph, symmetric matrix.

  
  ![Graph with adjacency matrix](image)

  

  \[
  \begin{pmatrix}
  0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 1 & 0 \\
  0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 & 0 \\
  \end{pmatrix}
  \]

- **Adjacency matrix**, directed graph, potentially asymmetric matrix.

  ![Graph with adjacency matrix](image)

  

  \[
  \begin{pmatrix}
  0 & 0 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  \end{pmatrix}
  \]
Matrix representation of graphs

- **adjacency matrix**: $n \times n$ binary matrix indicating if edge $\{u, v\} \in E$, where $n = |V|$ and $G = (V, E)$. Undirected graph, symmetric matrix.

  $$
  \begin{pmatrix}
  0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 1 & 1 & 0 \\
  1 & 1 & 0 & 1 & 0 \\
  0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 & 0 
  \end{pmatrix}
  $$

- Adjacency matrix, directed graph, potentially asymmetric matrix.

  $$
  \begin{pmatrix}
  0 & 0 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 
  \end{pmatrix}
  $$

- **incidence matrix**: $n \times m$ matrix of nodes/edges.

  $$
  \begin{pmatrix}
  1 & 0 & 1 & 1 & 0 & 0 & 0 \\
  1 & 1 & 0 & 0 & 1 & 0 & 0 \\
  0 & 1 & 1 & 0 & 0 & 0 & 1 \\
  0 & 0 & 0 & 0 & 1 & 1 & 1 \\
  0 & 0 & 0 & 1 & 0 & 1 & 0 
  \end{pmatrix}
  $$
Sparse $k$-NN graphs are good at computationally efficiently approximating a low-dimensional manifold.
Sparse $k$-NN graphs are good at computationally efficiently approximating a low-dimensional manifold.

Left: Data, Right: $k$-NN graph from the data. Nearest points determined based on geodesic distance (closest along the graph).
Sparse $k$-NN graphs are good at computationally efficiently approximating a low-dimensional manifold.

Left: Data, Right: $k$-NN graph from the data. Nearest points determined based on geodesic distance (closest along the graph).
Sparse $k$-NN graphs are good at computationally efficiently approximating a low-dimensional manifold.

Left: Data, Right: $k$-NN graph from the data. Nearest points determined based on geodesic distance (closest along the graph).

Getting the right value for $k$ crucial for proper representation.
One-dimensional spiral manifold in 2D space
One-dimensional spiral manifold in 2D space, along with PCA projection.
2D Manifold Examples in 3d Ambient Space

(A): manifolds. (B) samples from manifold. (C) Inherent flattened 2D manifolds.

From: Roweiss & Saul, 1999
Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.
Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables. Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{\text{true}}(x_1, \ldots, x_N)$.
Machine learning within restricted families

- Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.
- Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{\text{true}}(x_1, \ldots, x_N)$.
- Searching within $\mathcal{U}$ is infeasible/intractable/impossible.
Machine learning within restricted families

- Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.
- Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{true}(x_1, \ldots, x_N)$.
- Searching within $\mathcal{U}$ is infeasible/intractable/impossible.
- Desire a restricted but useful and easy to use family $\mathcal{F} \subset \mathcal{U}$.
Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.

Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{\text{true}}(x_1, \ldots, x_N)$.

Searching within $\mathcal{U}$ is infeasible/intractable/impossible.

Desire a restricted but useful and easy to use family $\mathcal{F} \subset \mathcal{U}$.

Size of $\mathcal{U}$ too large, complex, and with many local optima.
Machine learning within restricted families

- Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.
- Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{\text{true}}(x_1, \ldots, x_N)$.
- Searching within $\mathcal{U}$ is infeasible/intractable/impossible.
- Desire a restricted but useful and easy to use family $\mathcal{F} \subset \mathcal{U}$.
- Size of $\mathcal{U}$ too large, complex, and with many local optima.
- Obtainable solution in $\mathcal{F}$ better than feasible solution in $\mathcal{U}$.
Let $\mathcal{U}$ be the universe of all distributions over $n$ random variables.

Given sample data, along with domain knowledge, can learn a distribution $p(x)$ from $\mathcal{U}$ that is “close enough” to $p_{\text{true}}(x_1, \ldots, x_N)$.

Searching within $\mathcal{U}$ is infeasible/intractable/impossible.

Desire a **restricted** but useful and easy to use family $\mathcal{F} \subset \mathcal{U}$.

Size of $\mathcal{U}$ too large, complex, and with many local optima.

Obtainable solution in $\mathcal{F}$ better than feasible solution in $\mathcal{U}$

Graphical models provide one framework for specifying $\mathcal{F} \subset \mathcal{U}$
Graphical Models

- A graphical model is a visual, abstract, and mathematically formal description of properties of families of probability distributions (densities, mass functions).

There are many types of graphical models, for example (left: undirected graphical models, middle factor graphs, right Bayesian networks).

\[ G_1 \]

\[ G_2 \]

\[ G_3 \]
Definition 8.8.1 (factorization)

Given a function $f(\cdot)$ defined over $N$ variables $x_1,\ldots,x_N$, we say that $f(\cdot)$ factorizes over $x_A$ and $x_B$ if $\forall x$ there exists functions $g(x_A)$ and $h(x_B)$ such that

$$f(x_A, x_B) = g(x_A)h(x_B).$$

(8.9)

We note that $A$ and $B$ may intersect.

- Assume $A \cap B = \emptyset$. 

Definition 8.8.1 (factorization)

Given a function $f(\cdot)$ defined over $N$ variables $x_1,...,x_N$, we say that $f(\cdot)$ factorizes over $x_A$ and $x_B$ if $\forall x$ there exists functions $g(x_A)$ and $h(x_B)$ such that

$$f(x_A, x_B) = g(x_A)h(x_B). \quad (8.9)$$

We note that $A$ and $B$ may intersect.

- Assume $A \cap B = \emptyset$.
- Variables $x_a$ and $x_b$ where $a \in A$ and $b \in B$ do not directly interact with each other within $f$. 
Definition 8.8.1 (factorization)

Given a function $f(\cdot)$ defined over $N$ variables $x_1,\ldots,x_N$, we say that $f(\cdot)$ factorizes over $x_A$ and $x_B$ if $\forall x$ there exists functions $g(x_A)$ and $h(x_B)$ such that

$$f(x_A, x_B) = g(x_A)h(x_B).$$

(8.9)

We note that $A$ and $B$ may intersect.

- Assume $A \cap B = \emptyset$.
- Variables $x_a$ and $x_b$ where $a \in A$ and $b \in B$ do not directly interact with each other within $f$.
- This means, $x_b$’s influence, on the way variable $x_a$ may affect the behavior of $f$, is limited.
Definition 8.8.1 (factorization)

Given a function \( f(\cdot) \) defined over \( N \) variables \( x_1, \ldots, x_N \), we say that \( f() \) factorizes over \( x_A \) and \( x_B \) if \( \forall x \) there exists functions \( g(x_A) \) and \( h(x_B) \) such that

\[
 f(x_A, x_B) = g(x_A)h(x_B).
\]  

(8.9)

We note that \( A \) and \( B \) may intersect.

- Assume \( A \cap B = \emptyset \).
- Variables \( x_a \) and \( x_b \) where \( a \in A \) and \( b \in B \) do not directly interact with each other within \( f \).
- This means, \( x_b \)'s influence, on the way variable \( x_a \) may affect the behavior of \( f \), is limited.
- Compare \( f(x_a, x_b) = x_a x_b \) vs \( f(x_a, x_b) = \log(x_a x_b) \).
Independence and Conditional Independence

Just a form of factorization

**Definition 8.8.2 (independence)**

For any probability distribution $p$ over a set of random variables $X_V$, given any collection of two sets $A, B \subseteq V$, we say that $X_A$ is independent of $X_B$ if $p(x_A, x_B) = p(x_A)p(x_B)$ for all possible values $x_A, x_B$. This is written using the binary conditional independence relation $X_A \perp X_B$.

**Definition 8.8.3 (conditional independence)**

For any probability distribution $p$ over a set of random variables $X_V$, given any collection of three sets $A, B, C \subseteq V$, we say that $X_A$ is independent of $X_B$ given $X_C$ if $p(x_A, x_B | x_C) = p(x_A | x_C)p(x_B | x_C)$ for all possible values $x_A, x_B, x_C$. This is written using the ternary conditional independence relation $X_A \perp X_B | X_C$. Whenever $X_A \perp X_B$ (i.e., $C = \emptyset$), we call this marginal independence or just independence as above.
Lemma 8.8.4 (Conditional Independence)

For any probability distribution \( p \) over a set of random variables \( X_V \), given any collection of three sets \( A, B, C \subset V \), then \( X_A \perp X_B | X_C \) holds iff there exists non-negative functions \( g(x_A, x_C) \) and \( h(x_B, x_C) \) such that \( p(x_A, x_B, x_C) \) can be written as

\[
p(x_A, x_B, x_C) = g(x_A, x_C)h(x_B, x_C)
\]

for all values of \( x_A, x_B, x_C \) such that \( p(x_C) > 0 \).

- This is a form of conditional factorization, in that \( x_C \) is involved in both factors and under the condition that this is true, \( x_A \) and \( x_B \) factorize.
Graphical models are encodings of families of probability distributions.
Graphical models are encodings of families of probability distributions.

Encodings are done via a graph that efficiently specifies either a (potentially very large) set (conditional) independence properties (or more fundamentally, a set of factorization properties).
Graphical Models

- Graphical models are encodings of families of probability distributions.
- Encodings are done via a graph that efficiently specifies either a (potentially very large) set (conditional) independence properties (or more fundamentally, a set of factorization properties).
- This is a crucial idea to understand: a graphical model encodes a set of constraints (factorization requirements, or conditional independence properties) that all family members must obey.
Graphical Models

- Graphical models are encodings of families of probability distributions.
- Encodings are done via a graph that efficiently specifies either a (potentially very large) set (conditional) independence properties (or more fundamentally, a set of factorization properties).
- This is a crucial idea to understand: a graphical model encodes a set of constraints (factorization requirements, or conditional independence properties) that all family members must obey.
- Factorization requirements are often (but not always) identical to conditional independence requirements.
Graphical Models

- Graphical models are encodings of families of probability distributions.
- Encodings are done via a graph that efficiently specifies either a (potentially very large) set (conditional) independence properties (or more fundamentally, a set of factorization properties).
- This is a crucial idea to understand: a graphical model encodes a set of constraints (factorization requirements, or conditional independence properties) that all family members must obey.
- Factorization requirements are often (but not always) identical to conditional independence requirements.
- Factorization, in general, allows sums to be distributed into products thereby making (exact) inference quantities more efficient than if factorization properties did not exist.
Graphical Model

- Each type of graphical model requires a certain type of graph (e.g., undirected, or DAG) and a set of rules (or “Markov properties”) to define the GM.

- A graphical model is a pair \((G, \mathcal{M}) = ((V, E), \mathcal{M})\), a graph \(G\) and a set of properties \(\mathcal{M}\) that define what the graphical model means.

- Conceptually, one can think of a property \(r \in \mathcal{M}\) as a predicate on a graph and a distribution, so \(r(p, G, \mathcal{M}) \in \{\text{true, false}\}\).

- \((G, \mathcal{M})\) consists of a family of distributions over \(x_V\) where all predicates hold. That is

\[
\mathcal{F}(G, \mathcal{M}) = \{p : p \text{ is a distribution over } X_V \text{ and }, r(p, G, \mathcal{M}) = \text{true, } \forall r \in \mathcal{M}\}
\]  

(8.10)

- \(\mathcal{F}(G, \mathcal{M}) \subseteq \mathcal{U}\)
Markov Properties

- Markov properties are rules that specify what are required of every family member. Any \( p \in \mathcal{F}(G, \mathcal{M}) \) satisfies all properties/rules \( r \in \mathcal{M} \) for \( G \). Any \( p \in \mathcal{U} \setminus \mathcal{F}(G, \mathcal{M}) \) violates at least one property for \( G \).

- A \( p \in \mathcal{U} \) might have more properties. \( \mathcal{M} \) is like a filter, lets in those \( p \) that satisfy, but will let in those that satisfy more.

- Example \( r \in \mathcal{M} \) might be “if there are two nodes \( u, v \in V \) that are neither directly nor indirectly connected in \( G \) (i.e., there no path leading from \( u \) to \( v \) in \( G \)) then the corresponding random variables in \( p \) are marginally independent”
What is graphical model inference?

- Inference: computing probabilistic queries. Let $E$ be a subset, $\vec{x}_E$ a particular vector value.
What is graphical model inference?

- Inference: computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.

  1. probability of evidence (marginalize the hidden variables)

\[ p(\bar{x}_E) \quad (8.11) \]
What is graphical model inference?

- Inference: computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.
  1. probability of evidence (marginalize the hidden variables)
     \[ p(\bar{x}_E) \]  
     \[ (8.11) \]
  2. posterior probability, for $S \subseteq V \setminus E$ do
     \[ p(x_S | \bar{x}_E) \]  
     \[ (8.12) \]
What is graphical model inference?

- Inference: computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.
  1. probability of evidence (marginalize the hidden variables)
     \[ p(\bar{x}_E) \]  
     \hspace{1cm} (8.11)
  2. posterior probability, for $S \subseteq V \setminus E$ do
     \[ p(x_S | \bar{x}_E) \]  
     \hspace{1cm} (8.12)
  3. most probable assignment, for $S \subseteq V \setminus E$ do
     \[ \text{argmax}_{x_S \in \mathcal{D}_{x_S}} p(x_S, \bar{x}_E). \]  
     \hspace{1cm} (8.13)

Given a graph $G$, we want to derive this just based just on $(G, M)$ and derive this automatically. We want to understand the computational complexity of the procedure based just on $(G, M)$.
What is graphical model inference?

- **Inference**: computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.

  1. probability of evidence (marginalize the hidden variables)

     $$ p(\bar{x}_E) $$  

  2. posterior probability, for $S \subseteq V \setminus E$ do

     $$ p(x_S | \bar{x}_E) $$  

  3. most probable assignment, for $S \subseteq V \setminus E$ do

     $$ \arg\max_{x_S \in \mathcal{D}_S} p(x_S, \bar{x}_E). $$

- **Given a graph $G$, we want to derive this just based just on $(G, M)$ and derive this automatically.**
What is graphical model inference?

- **Inference:** computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.
  1. probability of evidence (marginalize the hidden variables)
     \[ p(\bar{x}_E) \]  
     \begin{equation}
     (8.11)
     \end{equation}
  2. posterior probability, for $S \subseteq V \setminus E$ do
     \[ p(x_S | \bar{x}_E) \]  
     \begin{equation}
     (8.12)
     \end{equation}
  3. most probable assignment, for $S \subseteq V \setminus E$ do
     \[ \arg\max_{x_S \in \mathcal{D}_{x_S}} p(x_S, \bar{x}_E) \]  
     \begin{equation}
     (8.13)
     \end{equation}

- Given a graph $G$, we want to derive this just based just on $(G, \mathcal{M})$ and derive this automatically.
- **We want to understand the computational complexity of the procedure based just on $(G, \mathcal{M})$.**
What is graphical model inference?

- **Inference**: computing probabilistic queries. Let $E$ be a subset, $\bar{x}_E$ a particular vector value.
  1. Probability of evidence (marginalize the hidden variables)
     \[ p(\bar{x}_E) \] (8.11)
  2. Posterior probability, for $S \subseteq V \setminus E$ do
     \[ p(x_S | \bar{x}_E) \] (8.12)
  3. Most probable assignment, for $S \subseteq V \setminus E$ do
     \[ \text{argmax} \ p(x_S, \bar{x}_E). \] (8.13)

- Given a graph $G$, we want to derive this just based just on $(G, \mathcal{M})$ and derive this automatically.
- We want to understand the computational complexity of the procedure based just on $(G, \mathcal{M})$.
- **amortization**: we want to derive a procedure that works for any $p \in \mathcal{F}(G, \mathcal{M})$ for a given rule set.
Graphical model inference diagrammatically

A particular probabilistic query

Produce Graphical Model Inference Procedure

inference\( (p) \)
an algorithm for doing Inference

Correct answer

Any \( p \in \mathcal{F}(G,M) \)

Observed Data

\((G,M)\)
Markov random fields

- $x$ is $n$-dimensional vector.
- Most often $x$ corresponds to a grid (i.e., $x$ is really a matrix or 3D-matrix).
- Ising model: $w_{ij}$ determines the interaction style of variables: if $w_{ij} = 0$ the no interaction. If $w_{ij} > 0$ then more probable for $x_i = x_j = \pm 1$. If $w_{ij} < 0$ then more probable for $x_i \neq x_j$.
- We can think of matrix $W$ and vector $s$ as a graph, $G = (V, E)$ where $s$ corresponds to $V$ and $W$ corresponds to $E$ — that is, $(i, j) = e \in E$ only when $w_{ij} \neq 0$.
- We might expect that any Ising model $p \in \mathcal{F}(G, \mathcal{M}^{(mrf)})$ for appropriately defined MRF rules.
Clique Factorization

- The “Cliques” of a graph $G = (V, E)$, or $\mathcal{C}(G)$, in a graph are the set of fully connected nodes.
- If $C \in \mathcal{C}(G)$ and $u, v \in C$ then $(u, v) \in E(G)$
- In the following graph

![Graph Diagram]

cliques are $\mathcal{C} = \{\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}, \{3, 4\}, \{2, 4\}, \{2, 3, 4\}, \{2, 5\}\}$. 
Clique Factorization

- Given graph $G$ with cliques $C(G)$, consider a probability distribution that can be represented as follows:

$$p(x_V) = \frac{1}{Z} \prod_{C \in C(G)} \phi_C(x_C)$$ (8.14)

$$Z = \sum_{x_V} \prod_{C \in C} \phi_C(x_C)$$ (8.15)

- Actually, we don’t always need $Z$ explicitly since it is a constant and can be distributed into the factors in a variety of ways, leading to:

$$p(x_V) = \prod_{C \in C(G)} \phi_C(x_C)$$ (8.16)

where only the factorization is depicted.
Trees defined

**Definition 8.8.5**

A graph $G = (V, E)$ is a *forest* if it is the case that for all $u, v \in V$, there is no more than one path that connects $u$ to $v$ in $G$. Given a forest $G$, if for all $u, v \in G$ there is a unique path connecting $u$ and $v$, then it is called a *connected forest* or just simply a *tree.*
Trees defined in many ways

**Theorem 8.8.6 (Trees, Berge)**

Let $G = (V, E)$ be an undirected graph with $|V| = n > 2$. Then each of the following properties are equivalent and each can be used to define when $G$ is a tree:

- $G$ is connected and has no cycles
- $G$ has $n - 1$ edges and has no cycles,
- $G$ is connected and contains exactly $n - 1$ edges,
- $G$ has no cycles. Exactly one cycle created if edge added to $G$.
- $G$ is connected, and if any edge is removed, the remaining graph is not connected,
- Every pair of vertices of $G$ is connected by one unique path.
- $G$ can be generated as follows: Start with $v$, repeatedly choose next vertex, and connect it with edge to exactly one previous vertex.
Trees, inference, and distributive law

- Size of any maximal clique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
Trees, inference, and distributive law

- Size of any maximal clique in tree is two. Any set $S \subseteq V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(f)})$ has factors of size at most two.
- This has important consequences for inference.
Trees, inference, and distributive law

- Size of any maximal clique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^f)$ has factors of size at most two.
- This has important consequences for inference.
- A chain is a set of nodes connected in succession.

A chain is a tree but not necessarily vice versa
Trees, inference, and distributive law

- Size of any maximal clique in tree is two. Any set \( S \subset V(T) \) with \( |S| > 2 \) induces a forest.
- Any \( p \in \mathcal{F}(T, \mathcal{M}(f)) \) has factors of size at most two.
- This has important consequences for inference.
- A \textit{chain} is a set of nodes connected in succession.

\[
\begin{array}{ccccccc}
\circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\end{array}
\]

- A chain is a tree but not necessarily vice versa
- If \( p \) factors w.r.t. a chain then

\[
p(x) = \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1}) \tag{8.17}
\]
Trees, inference, and distributive law

- Size of any maximal clique in tree is two. Any set $S \subset V(T)$ with $|S| > 2$ induces a forest.
- Any $p \in \mathcal{F}(T, \mathcal{M}^{(f)})$ has factors of size at most two.
- This has important consequences for inference.
- A **chain** is a set of nodes connected in succession.

A chain is a tree but not necessarily vice versa

- If $p$ factors w.r.t. a chain then

$$p(x) = \prod_{i=1}^{N-1} \psi_{i,i+1}(x_i, x_{i+1})$$  \hspace{1cm} (8.17)

- Suppose we wish to compute $p(x_3, x_4)$. then

$$p(x_3, x_4) = \sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \ldots, x_N)$$  \hspace{1cm} (8.18)
Trees - and inference

- Let $r \triangleq |D_{X_i}| \forall i$
- This requires $O(r^N)$ ops, as in:

1. foreach $(x_3, x_4) \in D_{X_3} \times D_{X_4}$ do
2. Compute $\sum x_1 \sum x_2 \sum x_5 \sum x_6 \cdots \sum x_N p(x_1, x_2, \ldots, x_N)$
Trees - and inference

- Let $r \triangleq |D_{X_i}| \forall i$
- This requires $O(r^N)$ ops, as in:

\begin{align*}
1 & \textbf{foreach } (x_3, x_4) \in D_{X_3} \times D_{X_4} \textbf{ do} \\
2 & \quad \text{Compute } \sum_{x_1} \sum_{x_2} \sum_{x_5} \sum_{x_6} \cdots \sum_{x_N} p(x_1, x_2, \ldots, x_N)
\end{align*}

- Very wasteful!! Does not take advantage of the distributive law in $\mathbb{R}$ (i.e., $ab + ac = a(b + c)$).

\[
\begin{align*}
\sum_{x_1, x_2, \ldots, x_N} & \left( \prod_{c \in \text{factors not involving } x_i} \psi_c \right) \left( \prod_{c \in \text{factors involving } x_i} \psi_c \right) \\
= & \sum_{x_1, \ldots, x_i-1, x_i+1, \ldots, x_N} \left( \prod_{c \in \text{factors not involving } x_i} \psi_c \right) \sum_{x_i} \left( \prod_{c \in \text{factors involving } x_i} \psi_c \right)
\end{align*}
\]
Trees - and inference

- **Goal:** compute $p(x_3, x_4)$ when
  
  $$p(x_1, \ldots, x_5) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5).$$
Trees - and inference

- **Goal:** compute \( p(x_3, x_4) \) when
  \[
p(x_1, \ldots, x_5) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5).
  \]

- We can exploit this property - move sum as far to right as possible.
  Take the case where \( N = 5 \), for example:

  \[
p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \sum_{x_1} \psi_{1,2}(x_1, x_2)
  \]

  \[
  = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \phi_{y,2}(x_2)
  \]

  \[
  \text{(8.19)}
  \]

  where \( \phi_{y,2}(x_2) \) is a function of \( x_2 \) only. The notation \( \mathcal{A} \) indicates that \( x_1 \) has been summed away.
**Trees - and inference**

- **Goal:** compute \( p(x_3, x_4) \) when
  \[
p(x_1, \ldots, x_5) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5).
  \]

- **We can exploit this property - move sum as far to right as possible.**
  Take the case where \( N = 5 \), for example:

  \[
p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \sum_{x_1} \psi_{1,2}(x_1, x_2)
  \]

  \[
  = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3)\phi_{1,2}(x_2)
  \]

  \[
  \text{(8.19)}
  \]

  where \( \phi_{1,2}(x_2) \) is a function of \( x_2 \) only. The notation \( \downarrow \) indicates that \( x_1 \) has been summed away.

- **Node \( x_1 \) has been “eliminated” since once marginalized, it never appears in future summations.**


**Goal:** compute \( p(x_3, x_4) \) when
\[
p(x_1, \ldots, x_5) = \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5).
\]

We can exploit this property - move sum as far to right as possible. Take the case where \( N = 5 \), for example:

\[
p(x_3, x_4) = \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \sum_{x_1} \psi_{1,2}(x_1, x_2)
\]

\[
= \sum_{x_2} \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\psi_{2,3}(x_2, x_3) \phi_{1,2}(x_2)
\]

\[ (8.19) \]

where \( \phi_{1,2}(x_2) \) is a function of \( x_2 \) only. The notation \( \lambda \) indicates that \( x_1 \) has been summed away.

Node \( x_1 \) has been “eliminated” since once marginalized, it never appears in future summations.

Computing \( \phi_{1,2}(x_2) \) costs only \( O(r^2) \).
Trees - and inference

- We have expression that does not involve $x_1$, let's next sum away $x_2$.

$$p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4) \psi_{4,5}(x_4, x_5) \sum_{x_2} \psi_{2,3}(x_2, x_3) \phi_{y,2}(x_2)$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4) \psi_{4,5}(x_4, x_5) \phi_{y,2,3}(x_3)$$

(8.20)
- We have expression that does not involve $x_1$, lets next sum away $x_2$.

$$p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \sum_{x_2} \psi_{2,3}(x_2, x_3)\phi_{y,2}(x_2)$$

$$= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5)\phi_{y,2,3}(x_3)$$  \hspace{1cm} (8.20)

$$\phi_{y,2,3}(x_3)$$ - both $x_1$ and $x_2$ are eliminated, only function of $x_3$.  \hspace{1cm} (8.21)
We have expression that does not involve \( x_1 \), let's next sum away \( x_2 \).

\[
p(x_3, x_4) = \sum_{x_5} \psi_{3,4}(x_3, x_4) \psi_{4,5}(x_4, x_5) \sum_{x_2} \psi_{2,3}(x_2, x_3) \phi_{Y,2}(x_2) \]

\[
= \sum_{x_5} \psi_{3,4}(x_3, x_4) \psi_{4,5}(x_4, x_5) \phi_{Y,2,3}(x_3)
\]

\( \phi_{Y,2,3}(x_3) \) - both \( x_1 \) and \( x_2 \) are eliminated, only function of \( x_3 \).

Again, only \( O(r^2) \)
Next, we sum away (eliminate) $x_5$ (moving sums in as far as possible).

\[
p(x_3, x_4) = \psi_{3,4}(x_3, x_4) \phi_{Y,\gamma,3}(x_3) \sum_{x_5} \psi_{4,5}(x_4, x_5) \tag{8.22}
\]

\[
= p(x_3, x_4) = \psi_{3,4}(x_3, x_4) \phi_{Y,\gamma,3}(x_3) \phi_{5,4}(x_4) \tag{8.23}
\]
Next, we sum away (eliminate) \( x_5 \) (moving sums in as far as possible).

\[
p(x_3, x_4) = \psi_{3,4}(x_3, x_4) \phi_{\mathcal{V}, 3}(x_3) \sum_{x_5} \psi_{4,5}(x_4, x_5) \]

\[
\phi_{\mathcal{V}^*, 4}(x_4)
\]

\[
= p(x_3, x_4) = \psi_{3,4}(x_3, x_4) \phi_{\mathcal{V}, 3}(x_3) \phi_{\mathcal{V}^*, 4}(x_4)
\]

Again, only \( O(r^2) \) to produce \( \phi_{\mathcal{V}^*, 4}(x_4) \)
Next, we sum away (eliminate) $x_5$ (moving sums in as far as possible).

$$p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{Y,2,3}(x_3) \sum_{x_5} \psi_{4,5}(x_4, x_5)$$

$$\phi_{5,4}(x_4)$$

$$= p(x_3, x_4) = \psi_{3,4}(x_3, x_4)\phi_{Y,2,3}(x_3)\phi_{5,4}(x_4)$$

Again, only $O(r^2)$ to produce $\phi_{5,4}(x_4)$

Entire computation is $O(r^2)$

Length $N$ chain can be done in $O(N r^2)$. 
Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$
Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$
- Other orders also have $O(r^2)$, such as $(5, 1, 2)$ or $(1, 5, 2)$ and would still obtain $p(x_3, x_4)$. 
Trees - and inference

- Get $O(r^2)$ if we eliminate variables in order $(1, 2, 5)$
- Other orders also have $O(r^2)$, such as $(5, 1, 2)$ or $(1, 5, 2)$ and would still obtain $p(x_3, x_4)$.
- Not all orders have same efficiency, consider order $(2, 1, 5)$.

\[
p(x_3, x_4) = \sum_{x_1, x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \sum_{x_2} \psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3) \phi_{\mathcal{P},1,3}(x_1, x_3)
\]

(8.24)

\[
= \sum_{x_5} \psi_{3,4}(x_3, x_4)\psi_{4,5}(x_4, x_5) \sum_{x_1} \phi_{\mathcal{P},1,3}(x_1, x_3)
\]

(8.25)

\[
= \psi_{3,4}(x_3, x_4)\phi_{\mathcal{P},1,3}(x_3) \sum_{x_5} \psi_{4,5}(x_4, x_5)
\]

(8.26)

\[
= \psi_{3,4}(x_3, x_4)\phi_{\mathcal{P},1,3}(x_3)\phi_{\mathcal{P},4,4}(x_4)
\]

(8.27)
Problem: Sum over $x_2$ in Eq. 8.24 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$. 
Problem: Sum over $x_2$ in Eq. 8.24 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.

Some orders inextricably couple together factors, others don't.
Problem: Sum over $x_2$ in Eq. 8.24 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.

Some orders inextricably couple together factors, others don’t.

How do we ensure the best (fastest) elimination order? Graph tells us.
Summing, Marginalization, and variable elimination

- Problem: Sum over $x_2$ in Eq. 8.24 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don’t.
- How do we ensure the best (fastest) elimination order? Graph tells us.
- Key Problem: there exist no functions $g(a)$ and $h(c)$ that constitute a factorization of a sum as in:

$$g(a)h(c) = \sum_{b} f_1(a, b)f_2(b, c)$$

(8.28)
Summing, Marginalization, and variable elimination

- Problem: Sum over $x_2$ in Eq. 8.24 has cost $O(r^3)$. Total complexity is $O(r^3)$ which is unboundedly worse than $O(r^2)$.
- Some orders inextricably couple together factors, others don’t.
- How do we ensure the best (fastest) elimination order? Graph tells us.
- Key Problem: there exist no functions $g(a)$ and $h(c)$ that constitute a factorization of a sum as in:

$$g(a)h(c) = \sum_b f_1(a, b)f_2(b, c) \quad (8.28)$$

- In general, for disjoint variables $A, B, C \subseteq V$, the function

$$f(x_A, x_C) = \sum_{x_B} f_1(x_A, x_B)f_2(x_B, x_C) \quad (8.29)$$

does not factor, exists no $g, h$ such that $f(x_A, x_C) = g(x_A)h(x_C)$. 

Prof. Jeff Bilmes
Bayesian networks (BNs)

- Bayesian networks are another type of graphical model.
- It has distinct specificity from the other types.
- Nothing to do with “Bayesian statistics”, \( p(x|\theta)p(\theta|\eta)p(\eta|\nu)p(\nu|\varsigma) \ldots \)
  There are Bayesian and non-Bayesian Bayesian networks.

- \( G = (V, E) \) is a directed model, \( E \) is a set of directed edges. I.e.,
  \( e \in E \) means \( e = (u, v) \) ordered, so \( (u, v) \in E \) does not imply \( (v, u) \in E \) like before.
Bayesian networks (BNs)

- $G$ is directed acyclic (DAG) – no directed cycles anywhere.

- Each node $x_i$ has a set of parents $\text{pa}(x_i)$. We use $\text{pa}(i)$ to be the index set of the parents of nodes $i$, we use $\text{pa}(x_i)$ to refer to the actual node parents themselves.
Bayesian networks (BNs)

Alarm Example

- Alarm network in LA
- Nodes are random variables
- Edges encode “potential dependence”
- More of a causal semantics, but true causality?

\[ p(B, E, A, C, R) = p(B)p(E)p(A|B, E)p(R|E)p(C|A) \]
Consider the following family defined relative to a DAG $G$:

$$
\mathcal{F}(G, \mathcal{M}^{(df)}) = \left\{ p : p(x) = \prod_{v \in V} p(x_v | x_{\text{pa}(v)}) \right\}
$$

(8.30)

- this is the “directed factorization” rule that defines a family for DAGs.

Why is any such $p \in \mathcal{F}(G, \mathcal{M}^{(df)})$ a valid probability distribution?
Bayesian networks (BNs)

- Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ be an ordering (permutation) of the variables indices.
- Then $p(x_1, x_2, \ldots, x_N) = p(x_{\sigma_1}, x_{\sigma_2}, \ldots, x_{\sigma_N})$ for any $\sigma$.
- We have $\forall p \in \mathcal{U}$ (without any assumptions)

\[
p(x) = p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i|x_1, \ldots, x_{i-1}) \quad \text{(8.31)}
\]

\[
= \prod_{i=1}^{N} p(x_{\sigma_i}|x_{\sigma_1}, \ldots, x_{\sigma_{i-1}}) \quad \text{(8.32)}
\]
Bayesian networks (BNs)

Directed Factorization Rule for BNs

- Let \( p \in \mathcal{F}(G, \mathcal{M}^{(df)}) \) so that

\[
p(x) = \prod_{i=1}^{N} p(x_i | \text{pa}(x_i))
\]  

(8.33)

- Then, \( \exists \) ordering \( \sigma \) so that \( \text{pa}(x_i) \subseteq \{x_{\sigma_1}, \ldots, x_{\sigma_{i-1}}\} \).

- How many such orderings are there?
Bayesian networks (BNs)

Directed Factorization Rule for BNs

- Let $p \in \mathcal{F}(G, \mathcal{M}^{(df)})$ so that

$$p(x) = \prod_{i=1}^{N} p(x_i | \text{pa}(x_i))$$

(8.33)

- Then, $\exists$ ordering $\sigma$ so that $\text{pa}(x_i) \subseteq \{x_{\sigma_1}, \ldots, x_{\sigma_i-1}\}$.
- How many such orderings are there?
- Depends. Consider topological sort of the graph (which only exists when DAG). Chains, trees, etc.
- Also, intuitively $\{x_{\sigma_1}, \ldots, x_{\sigma_i-1}\} \setminus \text{pa}(x_i)$, is some form of conditional independence for each $\sigma$. I.e., $X_i \perp \{X_{\sigma_1}, \ldots, X_{\sigma_i-1}\} \setminus \text{pa}(X_i) | \text{pa}(X_i)$. 
Bayesian networks (BNs)

Relationship to MRF Factorization Rule

- Starting from a BN-DAG, transform so that each variable and its parents form a clique.
- For each $v \in V$, form clique $C_v = \{v\} \cup \text{pa}(v)$.
- Set of cliques become $\mathcal{C} = \bigcup_{v \in V} C_v$.
- Construct undirected graph $G_m = (V, E_m)$ with $\mathcal{C}$ as its cliques — the cliques define the edges, $e \in E_m$ iff $\exists C \in \mathcal{C}$ s.t. $e \in C$.
- Let $m(G) = G_m$ be the process of forming undirected $G_m$ from DAG $G$. 
Bayesian networks (BNs)
Relationship to MRF Factorization Rule and $m(G)$

- What is $m(G)$ when seen graphically?

Original factorization of $G$ is

$$p(a)p(b)p(c|a)p(d|b)$$
$$p(e|c)p(f|d)p(g|b)p(h)p(i|e, f)p(j|h, i)p(k|i, g)$$

- maximal cliques in $m(G)$

$$\{\{a, c\}, \{c, e\}, \{d, f\}, \{b, d\}, \{g, b\}, \{e, f, i\}, \{h, i, j\}, \{i, g, k\}\}$$
Bayesian networks (BNs)
Relationship to MRF Factorization Rule and $m(G)$

- What is $m(G)$ when seen graphically?

- $G \mapsto m(G) :$ for every variable $v \in V$, connect together all the unconnected parents of $v$ and then drop the arrow connections.
Bayesian networks (BNs)

Relationship to MRF Factorization Rule and $m(G)$

- What is $m(G)$ when seen graphically?

- $G \mapsto m(G)$: for every variable $v \in V$, connect together all the unconnected parents of $v$ and then drop the arrow connections.

- This process is known as “moralization” — “marry” all parents.

- It ensures that the resulting MRF family of $m(G)$ will “cover” a given $p \in \mathcal{F}(G, \mathcal{M}^{(df)})$. 
Bayesian networks (BNs)
Practical consequences of local normalization

- each local factor $p(v|\text{pa}(v))$ may be “produced” separately of the others and merged into one model later.
- separate training
- In an MRF, factors only make sense in concert with each other — one factor can’t be adjusted w/o considering the entire rest of the model.
- Each local factor $p(v|\text{pa}(v))$ may have quite different implementations (e.g., Neural network, SVM, conditional probability table, etc.)
- When doing $G \mapsto m(G)$, is anything lost? Local normalization. Might this mean that something else is lost w.r.t. the family?
Bayesian networks (BNs)

Separation

- In MRF - global Markov property, separation defined the conditional independence statement.
- \( A \) and \( B \) being separated by \( C \) means that all paths from anywhere in \( A \) to anywhere in \( B \) are “blocked” by \( C \).
- Separation property in BNs is similar, but we need a new definition of “blocked”
  - We define the notion of *directional separation*, or *d-separation* that uses the notion of *d-blocked*.
- Obviously, the arrow directions must matter.
Bayesian networks (BNs)

d-blocked

Definition 8.8.7 (d-blocked)

A path is \textit{d-blocked} by $C \subset V$ in a Bayesian network if there exists a node $v$ on the path such that either of the following two cases hold:

1. **Case 1:** $v \in C$, and along the path we have at $v$ either serial arrows $\rightarrow v \rightarrow$, or $\leftarrow v \leftarrow$, or divergent arrows $\leftarrow v \rightarrow$

2. **Case 2:** We have at $v$ convergent arrows $\rightarrow v \leftarrow$ along the path, and neither $v$ nor any of $v$’s descendants are in $C$.

If a path is not d-blocked by $C \subset V$, it is called \textit{d-active}.
Bayesian networks (BNs)

d-Blocked figure

Paths Blocked by the indicated nodes.

Paths not blocked by the indicated node.
Bayesian networks (BNs)

\textbf{d-separation}

\begin{definition}[d-Separation]
A set of nodes $A \subseteq V$ is \textit{d-separated} from $B \subseteq V$ by $C \subseteq V$ if all paths from any node in $A$ to any node in $B$ are d-blocked.
\end{definition}

This definition means that if there exists even one path between a node in $A$ and a node in $B$ that is d-active by nodes in $C$, then d-separation does not hold — all paths must be d-blocked for d-separation to hold.
Consider the following family of models:

\[ \mathcal{F}(G, \mathcal{M}^{(ds)}) = \{ p : X_A \independent X_B | X_C \text{ in } p \text{ whenever } C \text{ d-separates } A \text{ and } B \text{ in } G \} \tag{8.35} \]

This family is defined in an analogous way to the global Markov property of MRFs except here d-separation is used rather than the usual (undirected) graph separation.
The d-blocked property of d-separation determines separation based on
1. a particular node (center in this case),
2. the arrows adjacent at that node, and
3. whether or not the node is in the set $C$ (shown above as shaded when it is in $C$ and unshaded when not).

- Being in $C$ corresponds to conditioning.
- The above are three canonical examples of what a BN can express.
Bayesian networks (BNs)

Three examples with possible labels

1. SUVs → Greenhouse Gasses → Global Warming
2. Lung Cancer ↔ Smoking → Bad Breath
3. Genetics → Cancer ↔ Smoking
Bayesian networks (BNs)

Example 1: Markov chain

- We have a Markov chain \( X_1 \to X_2 \to X_3 \).
- Left case: When \( C = \emptyset \), nothing d-separates anything else, or no marginal independence properties for any \( p \in \mathcal{F}(G, \mathcal{M}^{\text{(ds)}}) \).
- Right case: \( X_2 \) d-separates \( X_1 \) and \( X_3 \) implying that any \( p \in \mathcal{F}(G, \mathcal{M}^{\text{(ds)}}) \) must have \( X_1 \perp \!\!\!\!\!\perp X_3 | X_2 \).
- On the other hand, if we take \( p \in \mathcal{F}(G, \mathcal{M}^{\text{(df)}}) \), then

\[
p(x_1, x_2, x_3) = p(x_1)p(x_2|x_1)p(x_3|x_2) = p(x_1, x_2)p(x_3|x_2), \text{ and}
\]

\[
p(x_3|x_1, x_2) = \frac{p(x_1, x_2, x_3)}{p(x_1, x_2)} = p(x_3|x_2)
\]  

(8.36)

also requiring \( X_1 \perp \!\!\!\!\!\perp X_3 | X_2 \). Coincidence?
Bayesian networks (BNs)

Example 2: Diverging arrows

We arrows diverging from $X_2$ in $X_1 \leftarrow X_2 \rightarrow X_3$.

Left case: When $C = \emptyset$, nothing d-separates anything else, or no marginal independence properties for any $p \in \mathcal{F}(G, \mathcal{M}^{(ds)})$.

Right case: $X_2$ d-separates $X_1$ and $X_3$ implying that any $p \in \mathcal{F}(G, \mathcal{M}^{(ds)})$ must have $X_1 \perp X_3 | X_2$.

On the other hand, if we take $p \in \mathcal{F}(G, \mathcal{M}^{(df)})$, then

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

and

$$p(x_3 | x_1, x_2) = \frac{p(x_1, x_2, x_3)}{p(x_1, x_2)} = p(x_3 | x_2)$$

requiring $X_1 \perp X_3 | X_2$. Another coincidence?
Bayesian networks (BNs)
Example 3: converging arrows, V-structure

- We have arrows converging on $X_2$ in $X_1 \rightarrow X_2 \leftarrow X_3$.
- Left case: When $C = \emptyset$, $X_1$ and $X_3$ are d-separated, so for any $p \in \mathcal{F}(G, \mathcal{M}^{(ds)})$, $X_1 \perp X_3$.
- Right case: When $C = \{X_2\}$, nothing d-separates anything else. So it is not the case that $X_1 \perp X_3 | X_2$.
- On the other hand, if we take $p \in \mathcal{F}(G, \mathcal{M}^{(df)})$, then
  
  $$p(x_1, x_2, x_3) = p(x_1)p(x_3)p(x_2 | x_3, x_2) \quad \text{and}$$

  $$p(x_1, x_3) = \sum_{x_2} p(x_1, x_2, x_3) = p(x_1)p(x_3) \quad (8.38)$$

  requiring $X_1 \perp X_3$. Still another coincidence?

---

Prof. Jeff Bilmes
EE511/Spring 2020/Adv. Intro ML - Week 8 - May 18th/20th, 2020
Some BNs can’t represent MRFs. Some MRFs can’t represent BNs.

Consider V-structure for BNs or consider 4-cycle for MRFs.
Bayesian networks (BNs)

BNs vs. MRFs examples

- Some BNs can’t represent MRFs. Some MRFs can’t represent BNs.
- Consider V-structure for BNs or consider 4-cycle for MRFs.
Graphical Models: In Sum

- Useful and powerful way of understanding properties of families of probability distributions.
Graphical Models: In Sum

- Useful and powerful way of understanding properties of families of probability distributions.
- Good example of using graphs in machine learning.
Graphical Models: In Sum

- Useful and powerful way of understanding properties of families of probability distributions.
- Good example of using graphs in machine learning.
- Enormous literature on how to do probabilistic inference very fast, often approximately (when exact inference is intractable).
Useful and powerful way of understanding properties of families of probability distributions.

Good example of using graphs in machine learning.

Enormous literature on how to do probabilistic inference very fast, often approximately (when exact inference is intractable).

Can be used together with other ML methods (e.g., image segmentation graphs to post-process output of deep neural networks).